Стюарт
Рассел

Питер
Норвиг

Искусственный интеллект
Современный подход

Четвертое издание

Том 3
Обучение, восприятие и действие

Искусственный интеллект
Современный подход

Четвертое издание

Том 3
Обучение, восприятие и действие

Artificial Intelligence

A Modern Approach

Fourth Edition

Stuart J. Russel and Peter Norvig

Pearson

Искусственный интеллект
Современный подход

Четвертое издание

Том 3
Обучение, восприятие и действие

С. Рассел, П. Норвиг

Диалектика

Москва · Санкт-Петербург
2022

ББК 32.813
Р24
УДК 004.8

ООО "Диалектика"

Перевод с английского и редакция *А.В. Слепцова*

По общим вопросам обращайтесь в издательство "Диалектика"
по адресу: info@dialektika.com, http://www.dialektika.com

Рассел, Стюарт, **Норвиг**, Питер.

Р24 Искусственный интеллект: современный подход, 4-е издание, том 3. Обучение, восприятие и действие. : Пер. с англ. — СПб. : ООО "Диалектика", 2022. — 640 с. — Парал. тит. англ.

ISBN 978-5-907365-27-8 (рус., том 3)

ISBN 978-5-907365-24-7 (рус., многотом.)

ББК 32.813

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Pearson Education, Inc.

Copyright © 2022 by Dialektika Computer Publishing.

Original English edition Copyright © 2021 by Pearson Education, Inc.

All rights reserved including the right of reproduction in whole or in part in any form. This translation is published by arrangement with Pearson Education, Inc.

Научно-популярное издание

Стюарт Рассел, Питер Норвиг

Искусственный интеллект: современный подход
4-е издание, том 3
Обучение, восприятие и действие

Подписано в печать 27.01.2022. Формат 70x100/16
Усл. печ. л. 51,6. Уч.-изд. л. 38,7
Тираж 400 экз. Заказ № 1159

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1
Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8(499) 270-73-59

ООО "Диалектика", 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907365-27-8 (рус., том 3)
ISBN 978-5-907365-24-7 (рус., многотом.)
ISBN 978-0-13-461099-3 (англ.)

© ООО "Диалектика", 2022, перевод, оформление, макетирование
© 2021, 2010, 2003 by Pearson Education, Inc.

Оглавление

ЧАСТЬ V. МАШИННОЕ ОБУЧЕНИЕ 11

Глава 19. Обучение на основе наблюдений 13

Глава 20. Статистические методы обучения 121

Глава 21. Глубокое обучение 167

Глава 22. Обучение с подкреплением 227

ЧАСТЬ VI. ОБЩЕНИЕ, ВОСПРИЯТИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ 281

Глава 23. Обработка естественного языка 283

Глава 24. Глубокое обучение при обработке естественного языка 341

Глава 25. Компьютерное зрение 381

Глава 26. Робототехника 447

ЧАСТЬ VII. ЗАКЛЮЧЕНИЕ 535

Глава 27. Философия, этика и безопасность искусственного интеллекта 537

Глава 28. Будущее искусственного интеллекта 587

Приложение А. Математические основы 605

Приложение Б. Сведения о языках и алгоритмах, используемых в книге 616

Предметный указатель 621

Содержание

ЧАСТЬ V. МАШИННОЕ ОБУЧЕНИЕ 11

Глава 19. Обучение на основе наблюдений 13

19.1. Формы обучения 14
19.2. Обучение с учителем 16
19.3. Метод деревьев решений 22
19.4. Выбор модели и оптимизация 35
19.5. Теория вычислительного обучения 45
19.6. Линейная регрессия и классификация 51
19.7. Непараметрические модели 65
19.8. Ансамблевые методы машинного обучения 79
19.9. Разработка систем машинного обучения 91
Резюме 106
Библиографические и исторические заметки 108
Упражнения 116

Глава 20. Статистические методы обучения 121

20.1. Статистическое обучение 121
20.2. Обучение при полных данных 126
20.3. Обучение с использованием скрытых переменных: алгоритм EM 145
Резюме 159
Библиографические и исторические заметки 160
Упражнения 163

Глава 21. Глубокое обучение 167

21.1. Простые сети с прямой связью 169
21.2. Графы вычислений для глубокого обучения 176
21.3. Сверточные сети 181
21.4. Алгоритмы обучения 188
21.5. Обобщение 193
21.6. Рекуррентные нейронные сети 200
21.7. Обучение без учителя и трансферное обучение 204

21.8. Применение глубокого обучения на практике 213
Резюме 217
Библиографические и исторические заметки 217
Упражнения 223

Глава 22. Обучение с подкреплением 227
22.1. Обучение посредством вознаграждения 227
22.2. Пассивное обучение с подкреплением 230
22.3. Активное обучение с подкреплением 239
22.4. Обобщение в обучении с подкреплением 248
22.5. Поиск стратегии 259
22.6. Обучение на демонстрации и обратное обучение с подкреплением 262
22.7. Применение методов обучения с подкреплением 267
Резюме 271
Библиографические и исторические заметки 273
Упражнения 278

ЧАСТЬ VI. ОБЩЕНИЕ, ВОСПРИЯТИЕ И ОСУЩЕСТВЛЕНИЕ ДЕЙСТВИЙ 281

Глава 23. Обработка естественного языка 283
23.1. Модели языка 284
23.2. Грамматика 299
23.3. Синтаксический анализ 302
23.4. Расширенные грамматики 310
23.5. Сложности реального естественного языка 317
23.6. Задачи обработки естественного языка 322
Резюме 325
Библиографические и исторические заметки 326
Упражнения 332

Глава 24. Глубокое обучение при обработке естественного языка 341
24.1. Метод встраивания слов 342
24.2. Рекуррентные нейронные сети для задач обработки естественного языка 347
24.3. Модели “от последовательности к последовательности” 352
24.4. Архитектура “трансформер” 360

24.5. Предобучение и трансферное обучение 363
24.6. Современное состояние разработок 369
Резюме 374
Библиографические и исторические заметки 375

Глава 25. Компьютерное зрение 381
25.1. Введение 381
25.2. Формирование изображения 383
25.3. Простые свойства изображения 392
25.4. Классификация изображений 402
25.5. Обнаружение объектов 407
25.6. Трехмерный мир 411
25.7. Использование компьютерного зрения 418
Резюме 436
Библиографические и исторические заметки 437
Упражнения 444

Глава 26. Робототехника 447
26.1. Роботы 447
26.2. Аппаратное обеспечение роботов 448
26.3. Решение задач в робототехнике 454
26.4. Восприятие в робототехнике 456
26.5. Планирование и управление 467
26.6. Планирование движений в условиях неопределенности 492
26.7. Обучение с подкреплением в робототехнике 496
26.8. Люди и роботы 499
26.9. Альтернативные архитектуры роботов 510
26.10. Области применения робототехники 514
Резюме 518
Библиографические и исторические заметки 520
Упражнения 528

ЧАСТЬ VII. ЗАКЛЮЧЕНИЕ 535

Глава 27. Философия, этика и безопасность искусственного интеллекта 537

27.1. Ограничения ИИ 537
27.2. Могут ли машины действительно мыслить? 542
27.3. Этика ИИ 545
Резюме 576
Библиографические и исторические заметки 576
Упражнения 584

Глава 28. Будущее искусственного интеллекта 587

28.1. Компоненты ИИ 588
28.2. Архитектуры систем ИИ 597

Приложение А. Математические основы 605

А.1. Анализ сложности и нотация O() 605
А.2. Векторы, матрицы и линейная алгебра 609
А.3. Распределения вероятностей 611
Библиографические и исторические заметки 615

Приложение Б. Сведения о языках и алгоритмах, используемых в книге 616

Б.1. Определение языков с помощью формы Бэкуса–Наура 616
Б.2. Описание алгоритмов с помощью псевдокода 617
Б.3. Дополнительный материал в Интернете 619

Предметный указатель 621

Часть V
Машинное обучение

Обучение на основе наблюдений 13
Статистические методы обучения 121
Глубокое обучение 167
Обучение с подкреплением 227

ГЛАВА 19

Обучение на основе наблюдений

В данной главе рассматриваются агенты, способные усовершенствовать свое поведение благодаря тщательному изучению полученного ранее опыта и собственных прогнозов о будущем.

Агент **обучается**, если он улучшает свою производительность в результате наблюдения за миром. Обучение может варьировать от тривиального, такого как беглый набросок списка необходимых покупок, до глубокого, как в случае Альберта Эйнштейна, разработавшего новую теорию вселенной. Когда агентом является компьютер, этот процесс называют ▶**машинным обучением**: компьютер наблюдает за поступающими данными, строит на основе этих данных **модель** и использует свою модель и как **гипотезу** о мире, и как часть программного обеспечения, способного решать задачи.

Почему нам требуется, чтобы машина могла обучаться? Почему бы просто не запрограммировать ее необходимым образом с самого начала? На это есть две основные причины. Во-первых, создатели машины не могут предвидеть все ситуации, возможные в будущем. Например, робот, назначение которого — успешно проходить через лабиринты, должен изучить схему каждого нового лабиринта, в который он попадает, а программа прогнозирования цен на фондовом рынке должна научиться адаптироваться к изменению условий от бума до спада. Во-вторых, иногда разработчики просто не имеют точного представления о том, как можно запрограммировать требуемое поведение системы. Большинство людей хорошо распознают лица членов своей семьи, но делают это подсознательно, из-за чего даже лучшие программисты не знают, как жестко запрограммировать компьютер, чтобы он успешно справлялся с подобной задачей, и поэтому единственный выход в данном случае — использование алгоритмов машинного обучения.

В этой главе мы будем чередовать обсуждение различных моделей обучения — деревья принятия решений (раздел 19.3), линейные модели (раздел 19.6), непараметрические модели, такие как модель ближайших соседей (раздел 19.7), ансамблевые модели, такие как случайный лес (раздел 19.8) — с практическими советами по созданию систем машинного обучения (раздел 19.9) и обсуждением основ теории машинного обучения (разделы 19.1–19.5).

19.1. Формы обучения

Любой компонент агентской программы может быть улучшен посредством машинного обучения. Улучшения и методы, используемые для их достижения, зависят от следующих факторов.

- Какой именно *компонент* должен быть улучшен.
- Какие *предварительные знания* имеет агент и что влияет на *модель*, которую он строит.
- Какие доступны *данные* и *обратная связь* на эти данные.

В главе 2 было описано несколько конструкций агентов. К **компонентам** этих агентов относится следующее.

1. Средства прямого отображения условий текущего состояния в действия.
2. Средства логического вывода релевантных свойств мира из последовательности результатов восприятия.
3. Информация о путях развития мира и результатах возможных действий, которые агент может предпринять.
4. Информация о полезности, указывающая, насколько желательными являются состояния мира.
5. Информация о ценности действий, указывающая на желательность действий.
6. Цели, описывающие наиболее желательные состояния.
7. Генератор проблем, критик и обучающий компонент, позволяющие улучшить систему.

Каждый из этих компонентов можно обучать. Рассмотрим, например, агента, управляющего автомобилем в автономном режиме, который учится вождению, наблюдая за человеком-водителем. Всякий раз, когда инструктор тормозит, агент может изучить очередное правило "условие–действие" в отношении того, когда следует тормозить (компонент 1). Рассматривая множество изображений с камеры, на которых, как ему сказано, имеются автобусы, он может научиться распознавать автобусы (компонент 2). Пробуя выполнять действия и наблюдая за результатами (например, резко затормозив на мокрой дороге), он может изучать результаты своих действий (компонент 3). Позднее, когда начнут поступать жалобы от пассажиров, которых сильно трясло во время поездки, агент сможет обучить соответствующий компонент своей общей функции полезности (компонент 4).

Технологии машинного обучения уже стали стандартным элементом программной инженерии. Каждый раз, когда создается система программного обеспечения, даже если она не воспринимается ее создателями как агент ИИ, ее компоненты потенциально могут быть улучшены за счет машинного обучения. Например, за счет использования модели машинного обучения скорость работы программного

обеспечения для анализа изображений галактик с учетом гравитационных линз была увеличена примерно в 10 млн раз (Хезаве и др. [1019], 2017), а расход энергии на охлаждение центров обработки данных был снижен на 40% благодаря другой модели машинного обучения (Гао [811], 2014). Лауреат премии Тьюринга Дэвид Паттерсон и руководитель департамента Google AI Джефф Дин объявили о наступлении "золотого века" для компьютерной архитектуры — благодаря машинному обучению (Дин и др. [565], 2018).

Мы уже встречались с несколькими примерами моделей для компонентов агентов: атомарными, развернутыми и реляционными моделями, основанными на логике или вероятности, и т.д. Для всех этих моделей уже были разработаны соответствующие алгоритмы обучения.

В этой главе предполагается лишь небольшое ▶**предварительное знание** со стороны агента: он начинает с нуля и учится лишь на основе полученных данных. В разделе 21.7.2 рассматривается **трансферное обучение**, когда знания из одной проблемной области передаются в новую проблемную область, благодаря чему обучение может протекать быстрее и с меньшим количеством используемых данных. Однако мы также предполагаем, что разработчик системы выбирает ту структуру модели, которая может обеспечить эффективное обучение.

Переход от конкретного множества наблюдений к общему правилу называется **индукцией**: из наблюдений, что в прошлом солнце вставало каждый день, можно предположить, что солнце взойдет и завтра. Этот подход отличается от **дедукции**, обсуждавшейся в главе 7, поскольку индуктивные выводы могут быть неверными, тогда как дедуктивные выводы гарантированно верны, если предпосылки правильны.

В этой главе мы сосредоточимся на задачах, в которых входные данные имеют **развернутое представление** (*factored representation*) — в виде вектора значений атрибутов. Также возможно, что на вход будут поступать данные любой другой структуры, включая атомарные и реляционные.

Если выходные данные выбираются из известного конечного множества значений (например, *sunny*/*cloudy*/*rainy* или *true*/*false*), то задачу обучения называют ▶**классификацией**. Если результат является числовым значением (например, как завтрашняя температура, представленная как целое или вещественное число), задача обучения получает название (по общему признанию, малопонятное[1]) ▶**регрессия**.

---

[1] Лучшим названием было бы *функция приближения* или *числовое предсказание*. Однако в 1886 году Фрэнсис Гальтон написал влиятельную статью о концепции *регрессии к среднему* (например, дети высоких родителей будут, вероятно, иметь рост выше среднего, но будут не так высоки, как их родители). Гальтон представил графики, которые он назвал "линии регрессии", и читатели в конечном счете стали ассоциировать слово "регрессия" со статистическим методом построения функций аппроксимации, а не с темой регрессии к среднему значению.

Существует три типа ▶**обратной связи**, связанной с входными данными, которые определяют три основных типа обучения.

- При ▶**обучении с учителем** (*supervised learning*) агент наблюдает за парами данных "ввод-вывод" (или "стимул-реакция") и изучает функцию, которая отображает входные данные на выходные. Например, входными данными могут быть изображения, каждое из которых сопровождается соответствующим выходным определением, скажем, "автобус", "пешеход" и т.д. Такое определение называется ▶**меткой** (*label*). Агент изучает функцию, которая при получении нового изображения предсказывает соответствующую ему метку. В случае действия "торможение" (компонент 1 выше) ввод — это текущее состояние (скорость и направление движения автомобиля, состояние дороги), а выход — расстояние, которое было пройдено при торможении до полной остановки. В этом случае набор выходных значений может быть получен агентом из его собственного восприятия (по факту); окружение — это учитель, а агент изучает функцию, которая отображает состояния на расстояние, пройденное до остановки.
- При ▶**обучении без учителя** (*unsupervised learning*) агент изучает поступающие на вход шаблоны без какой-либо явной обратной связи. Наиболее распространенная задача обучения без учителя — **кластеризация**, т.е. выявление во входных примерах потенциально полезных закономерностей. Например, когда агенту предъявляют миллионы изображений, взятых из Интернета, система компьютерного зрения может идентифицировать большой кластер схожих изображений, которые человек назвал бы "кошки".
- При ▶**обучении с подкреплением** (*reinforcement learning*) агент учится на сериях подкреплений: наград и наказаний. Например, в конце шахматной игры агенту говорят, что он выиграл (награда) или проиграл (наказание). На агента возлагается задача принять решение, какие из действий до получения подкрепления в наибольшей степени ответственны за достигнутый результат, и изменить свои действия так, чтобы нацелиться на большее вознаграждение в будущем.

19.2. Обучение с учителем

Более формально задача обучения с учителем формулируется следующим образом.

При наличии ▶**обучающего набора данных** в виде пар входных и выходных значений (или *прецедентов*)

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N),$$

сгенерированных неизвестной функцией $y = f(x)$, найти функцию h, аппроксимирующую истинную функцию f.

Функция h называется **гипотезой** о мире. Она выбирается из ▶**пространства гипотез** $\mathcal{H}$ возможных функций. Например, пространство гипотез может представлять собой множество многочленов третьей степени или набор функций языка JavaScript, или множество формул 3-SAT булевой логики.

Используя альтернативный словарь, можно сказать, что h — это **модель** данных, взятых из ▶**класса моделей** $\mathcal{H}$, или же **функция**, взятая из **класса функций**. Выходное значение y_i называют ▶**истинным значением** (*ground truth*) — это тот самый правильный ответ, который модель должна предоставить в ответ на запрос.

Как выбирается пространство гипотез? Могут иметь место некоторые предварительные знания о том процессе, который генерировал данные. Если этого нет, можно выполнить ▶**предварительный анализ данных**: исследовать их с помощью статистических тестов и визуализаций — гистограмм, диаграмм рассеяния, коробчатых диаграмм — чтобы получить некоторое представление о данных и сделать те или иные выводы в отношении того, какое пространство гипотез может быть подходящим. Или можно просто проверить на опыте несколько вариантов пространств гипотез и оценить, какой из них подходит наилучшим образом.

Как выбирается хорошая гипотеза из определенного пространства гипотез? Можно надеяться подобрать ▶**согласованную гипотезу**, когда функция h будет такой, что для каждого стимула x_i в обучающем наборе она будет давать реакцию $h(x_i) = y_i$. В случае непрерывной области значений результатов не следует ожидать точного соответствия реакции истинному значению; целесообразнее искать **наиболее подходящую функцию**, для которой каждое значение $h(x_i)$ будет достаточно близким к истинному y_i (в том смысле, который будет формально определен в разделе 19.4.2).

Степень истинности гипотезы определяется не тем, насколько точно она работает на обучающем наборе, а скорее тем, насколько хорошо она справляется с теми входными данными (стимулами), которые ей ранее не встречались. Это можно оценить с помощью второго набора пар значений (x_i, y_i), который принято называть ▶**тестовым набором**. Говорят, что функция h хорошо ▶**обобщает** данные, если она точно предсказывает отклики из тестового набора.

На рис. 19.1 демонстрируется, что функция h, подобранная обучающим алгоритмом, зависит от рассматриваемого пространства гипотез $\mathcal{H}$ и заданного обучающего набора данных. При построении каждого из четырех графиков, расположенных на рисунке в верхнем ряду, использовался один и тот же обучающий набор из 13 точек данных в плоскости (x, y). Четыре графика в нижнем ряду рисунка были построены на основе другого набора из 13 точек данных, но при этом оба набора представляют одну и ту же неизвестную функцию $f(x)$. В каждом из столбцов приведены наиболее подходящие гипотезы h из определенного пространства гипотез.

- **Столбец 1.** Пространство гипотез — прямые линии, т.е. функции вида $h(x) = w_1 x + w_0$. В нем отсутствуют линии, представляющие согласованную гипотезу для данного набора точек.
- **Столбец 2.** Пространство гипотез — синусоидальные функции вида $h(x) = w_1 x + \sin(w_0 x)$. Этот вариант не вполне согласован, но соответствует наборам данных достаточно хорошо.
- **Столбец 3.** Пространство гипотез — кусочно-линейные функции, каждый линейный сегмент которых соединяет две соседние точки данных. Подобные функции всегда являются согласованными.
- **Столбец 4.** Пространство гипотез — полиномы 12-й степени вида $h(x) = \sum_{i=0}^{12} w_i x^i$. В данном случае они также являются согласованными: всегда можно подобрать полином 12-й степени, идеально соответствующий 13 различным точкам данных. Однако тот факт, что гипотеза является согласованной, вовсе не означает, что это правильный выбор.

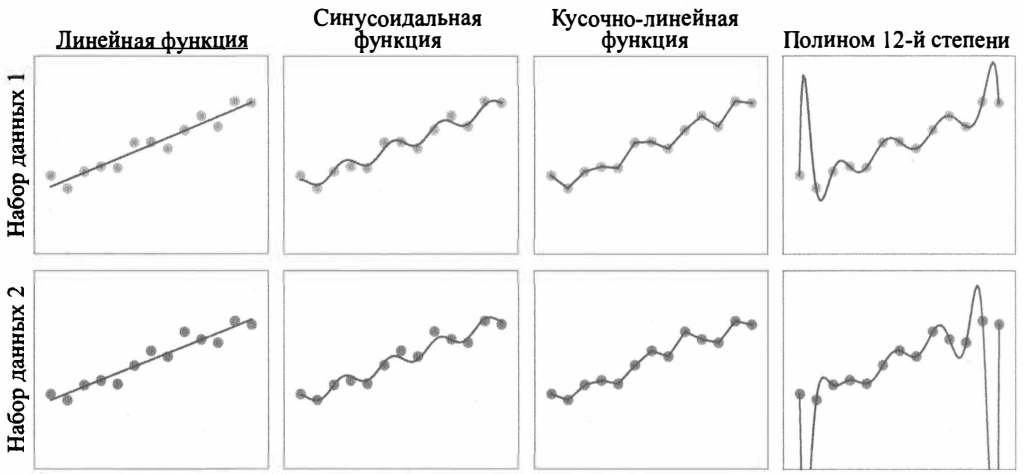


Рис. 19.1. Подбор гипотез для соответствия данным. **Верхний ряд**: четыре графика функций из четырех различных пространств гипотез, наиболее подходящих для данных из обучающего набора 1. **Нижний ряд**: функций тех же четырех типов, подобранные на несколько ином обучающем наборе данных (оба обучающих набора представляют одну и ту же неизвестную функцию $f(x)$)

Один из способов анализа пространств гипотез — оценить смещение, которое они налагают (независимо от обучающего набора данных), и дисперсию, которую они производят (от одного обучающего набора к другому).

Под ▶**смещением** (*bias*) имеется в виду (в широком смысле) тенденция предсказывающей гипотезы давать отклонение от ожидаемого значения при

усреднении по различным обучающим наборам. Смещение часто является результатом ограничений, налагаемых пространством гипотез. Например, пространство гипотез линейных функций вызывает сильное смещение, поскольку оно допускает только функции, представляющие прямые линии. Если в данных есть какие-либо шаблоны, отличные от общего наклона прямой, линейная функция не сможет представить такие шаблоны. Говорят, что гипотеза проявляет ▶**недообучение** (*underfitting*), если она не позволяет обнаружить шаблоны, существующие в данных. С другой стороны, кусочно-линейная функция обладает низким смещением, поскольку ее форма полностью определяется самими данными.

Под ▶**дисперсией** подразумевается количество изменений в гипотезе вследствие разброса значений в обучающих наборах. На рис. 19.1 две строки представляют наборы данных, каждый из которых был сформирован из значений одной и той же функции $f(x)$. Эти наборы оказались немного различающимися. Для первых трех столбцов небольшие различия в наборе данных приводят к небольшим различиям в гипотезах. Подобную ситуацию называют низкой дисперсией. Однако для многочленов 12-й степени в четвертом столбце имеет место высокая дисперсия: посмотрите, насколько различны две функции на обоих концах оси x. Очевидно, что хотя бы один из этих многочленов должен быть плохим приближением к истинной функции $f(x)$. Говорят, что функция является ▶**переобученной** (*overfitting*), если она уделяет слишком много внимания конкретным значениям данных определенного обучающего набора и как следствие плохо работает с ранее неизвестными ей значениями данных.

Часто необходимо найти ▶**компромисс "смещение–дисперсия"**, т.е. сделать выбор между более сложными гипотезами с небольшим смещением, хорошо соответствующими данным обучающего набора, и более простыми гипотезами с низкой дисперсией, обеспечивающими лучшее обобщение. В 1933 году Альберт Эйнштейн сказал: "Высшая цель любой теории состоит в том, чтобы сделать неподдающиеся сокращению основные элементы как можно более простыми и в наименьшем количестве, не отказываясь от адекватного представления единой точки отсчета, опыта". Другими словами, Эйнштейн рекомендует выбрать простейшую гипотезу, соответствующую данным. Этот принцип в глубь времен можно проследить вплоть до английского философа XIV века Уильяма Оккема.[2] Его принцип "не следует привлекать новые сущности без крайней на то необходимости" называют **бритвой Оккама**, поскольку он используется для "сбривания" сомнительных объяснений.

Определить, какая гипотеза проще, а какая сложнее, обычно нелегко. По-видимому, вполне очевидно, что полином 2-й степени с двумя параметрами проще, чем полином 13-й степени с тринадцатью параметрами. Это общее представление

[2] Это имя часто пишется с ошибкой как "Оккам", однако в действительности этот принцип получил свое название в честь английского философа Уильяма *из* Оккема, — это название города, в котором работал философ. Впоследствии его имя стали записывать как "Оккам", следуя французской транскрипции: "Guillaume d'Occam" (Гийом из Оккама).

будет уточнено в разделе 19.3.4. Однако в главе 21 будет показано, что модели глубоких нейронных сетей часто могут довольно хорошо обобщать, даже если они очень сложные, — у некоторых из них могут быть миллиарды параметров. Таким образом, количество параметров само по себе не является хорошим показателем пригодности модели. Возможно, следует стремиться к "соответствию", а не "простоте" модельного класса. Подробнее этот вопрос рассматривается в разделе 19.4.1.

Какая гипотеза будет лучшей из числа представленных на рис. 19.1? Этого нельзя сказать с определенностью. Если бы было известно, что входные данные представляют, например, количество посещений веб-сайта, которое растет со дня на день, но также циклически зависит от времени суток, то имело бы смысл предпочесть синусоидальную функцию. С другой стороны, если бы мы знали, что данные, определенно, не являются циклическими, но имеют высокий уровень шума, то можно было бы отдать предпочтение линейной функции.

В некоторых случаях аналитика может интересовать не просто, является ли гипотеза возможной или невозможной, а скорее насколько она вероятна. Обучение с учителем может быть выполнено путем выбора гипотезы h^*, которая наиболее вероятна с учетом имеющихся данных:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}} \; P(h \mid data).$$

По правилу Байеса это эквивалентно

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmax}} \; P(data \mid h) P(h).$$

Тогда можно сказать, что априорная вероятность $P(h)$ будет выше для гладкого полинома 1- или 2-й степени и ниже для полинома 12-й степени с большими острыми пиками. Необычно выглядящие функции допускаются, когда данные указывают, что эти функции действительно необходимы, но их выбор осложняется за счет присвоения им низкой априорной вероятности.

Почему бы не позволить пространству гипотез $\mathcal{H}$ представлять собой просто множество всех компьютерных программ или всех машин Тьюринга? Проблема заключается в том, что ➡ *существует компромисс между выразительностью пространства гипотез и вычислительной сложностью нахождения хорошей гипотезы в этом пространстве*. Например, подгонка прямой линии к данным реализуется простыми вычислениями, подгонка полиномов высокой степени будет уже несколько сложнее, а подгонка машин Тьюринга представляет собой задачу неразрешимую. Вторая причина предпочтения простых пространств гипотез состоит в том, что, вероятно, мы захотим использовать функцию h после ее обучения, а вычисление значения $h(x)$, когда функция h является линейной, гарантированно будет быстрым, тогда как в случае вычисления произвольной машинной программы Тьюринга не гарантируется, что оно вообще может быть завершено.

По этим причинам до нынешнего времени большая часть исследовательских работ по машинному обучению фокусировалась на простых способах представления. В последние годы проявился большой интерес к глубокому обучению (см. главу 21), в котором представления уже не столь просты, но вычисление значения $h(x)$ все еще требует *ограниченного количества этапов* вычислений при наличии соответствующего оборудования.

Ниже будет показано, что найти компромисс между выразительностью и сложностью не так-то просто. Как было показано в главе 8 на примере логики первого порядка, часто бывает так, что выразительный язык позволяет создать *простую* гипотезу, согласующуюся с данными, тогда как ограничение выразительности языка приводит к тому, что любая согласованная гипотеза становится очень сложной.

19.2.1. Пример задачи: целесообразность ожидания в ресторане

Давайте подробно рассмотрим методику обучения с учителем на следующем примере: проблеме принятия решения, стоит ли ожидать освобождения столика в ресторане. На протяжении всей главы эта задача будет использоваться для демонстрации различных классов моделей. В данной задаче выходные данные y представляются логической переменной *WillWait* (*следует ждать*), которой присваивается значение *true*, когда принимается решение ожидать освобождения столика. Входные данные представляются вектором $\mathbf{x}$, включающим значения десяти атрибутов, каждый из которых характеризуется собственным множеством допустимых дискретных значений.

1. *Alternate* (альтернатива). Есть ли другой подходящий ресторан поблизости.
2. *Bar* (бар). Если ли в ресторане удобный бар, в котором можно было бы подождать.
3. *Fri/Sat* (уик-энд). Этот атрибут принимает значение *true* по пятницам и субботам.
4. *Hungry* (голод). Голоден ли посетитель в данный момент.
5. *Patrons* (посетители). Сколько посетителей находится сейчас в ресторане; допустимые значения — *None* (ресторан пуст), *Some* (есть несколько посетителей) и *Full* (ресторан полон).
6. *Price* (цены). Ценовая категория ресторана (допустимые значения — $, $$, $$$).
7. *Raining* (дождь). Идет ли на улице дождь.
8. *Reservation* (бронирование). Было ли забронировано место.
9. *Type* (тип). Тип ресторана; допустимые значения — *French* (французская кухня), *Italian* (итальянская кухня), *Thai* (тайская кухня) и *Burger* (закусочная).
10. *WaitEstimate* (оценка времени ожидания). Продолжительность ожидания, как ее оценивает метрдотель. Допустимые значения — 0–10, 10–30, 30–60 и более 60 минут.

Обучающий набор из 12 примеров, взятых из личного опыта одного из авторов (Стюарта Рассела), приведен на рис. 19.2. Обратите внимание, насколько скудным является этот набор: существует $2^6 \times 3^2 \times 4^2 = 9216$ возможных комбинаций значений входных атрибутов, но исходно результирующая функция определена только для 12 из них; а для каждого из оставшихся 9204 вариантов она может принимать либо значение *true*, либо значение *false* — больше об этом ничего не известно. Данная ситуация характеризует суть индукции: необходимо дать наилучшее предсказание в отношении этих недостающих 9204 значений выходной функции, имея для доказательства лишь 12 примеров.

Пример	Входные атрибуты										Выход
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Да*	*Нет*	*Нет*	*Да*	*Some*	*$$$*	*Нет*	*Да*	*French*	0–10	y_1 = *Да*
$\mathbf{x}_2$	*Да*	*Нет*	*Нет*	*Да*	*Full*	*$*	*Нет*	*Нет*	*Thai*	30–60	y_2 = *Нет*
$\mathbf{x}_3$	*Нет*	*Да*	*Нет*	*Нет*	*Some*	*$*	*Нет*	*Нет*	*Burger*	0–10	y_3 = *Да*
$\mathbf{x}_4$	*Да*	*Нет*	*Да*	*Да*	*Full*	*$*	*Да*	*Нет*	*Thai*	10–30	y_4 = *Да*
$\mathbf{x}_5$	*Да*	*Нет*	*Да*	*Нет*	*Full*	*$$$*	*Нет*	*Да*	*French*	> 60	y_5 = *Нет*
$\mathbf{x}_6$	*Нет*	*Да*	*Нет*	*Да*	*Some*	*$$*	*Да*	*Да*	*Italian*	0–10	y_6 = *Да*
$\mathbf{x}_7$	*Нет*	*Да*	*Нет*	*Нет*	*None*	*$*	*Да*	*Нет*	*Burger*	0–10	y_7 = *Нет*
$\mathbf{x}_8$	*Нет*	*Нет*	*Нет*	*Да*	*Some*	*$$*	*Да*	*Да*	*Thai*	0–10	y_8 = *Да*
$\mathbf{x}_9$	*Нет*	*Да*	*Да*	*Нет*	*Full*	*$*	*Да*	*Нет*	*Burger*	> 60	y_9 = *Нет*
$\mathbf{x}_{10}$	*Да*	*Да*	*Да*	*Да*	*Full*	*$$$*	*Нет*	*Да*	*Italian*	10–30	y_{10} = *Нет*
$\mathbf{x}_{11}$	*Нет*	*Нет*	*Нет*	*Нет*	*None*	*$*	*Нет*	*Нет*	*Thai*	0–10	y_{11} = *Нет*
$\mathbf{x}_{12}$	*Да*	*Да*	*Да*	*Да*	*Full*	*$*	*Нет*	*Нет*	*Burger*	30–60	y_{12} = *Да*

Рис. 19.2. Примеры для проблемной области задачи ожидания в ресторане

19.3. Метод деревьев решений

►**Дерево решений** является средством представления функции, отображающей вектор значений атрибутов на единственное выходное значение — "решение". Алгоритм дерева решений приходит к решению за счет выполнения последовательности тестов (проверок), начиная от корня и следуя по соответствующей ветви, пока не будет достигнуто листовое значение. Каждый внутренний узел в дереве соответствует проверке значения одного из входных атрибутов, ветви от узла помечены возможными значениями тестируемого атрибута, а листовые (конечные) узлы определяют, какое значение должно быть возвращено функцией.

В общем случае входные атрибуты и выходные значения могут быть дискретными или непрерывными, но на данный момент мы будем рассматривать только случаи, когда входные данные представлены исключительно дискретными

значениями, а выходом являются либо значение *true* (►**положительный** пример), либо значение *false* (►**отрицательный** пример). Такой тип задач называется **булевой классификацией**. Будем использовать переменную j для индексации примеров ($\mathbf{x}_j$ — входной вектор для j-го примера, а y_j — выходное значение), а обозначение $x_{j,i}$ — для ссылок на i-й атрибут j-го примера.

Дерево решений, представляющее функцию решения, которую Стюарт Рассел обычно использует для решения задачи ожидания в ресторане, приведено на рис. 19.3. Следуя по его ветвям, можно увидеть, что пример с атрибутами *Patrons* = *Full* и *WaitEstimate* = 0–10 будет классифицирован как положительный (т.е. ответ — *да*, Рассел будет ждать освобождения столика).

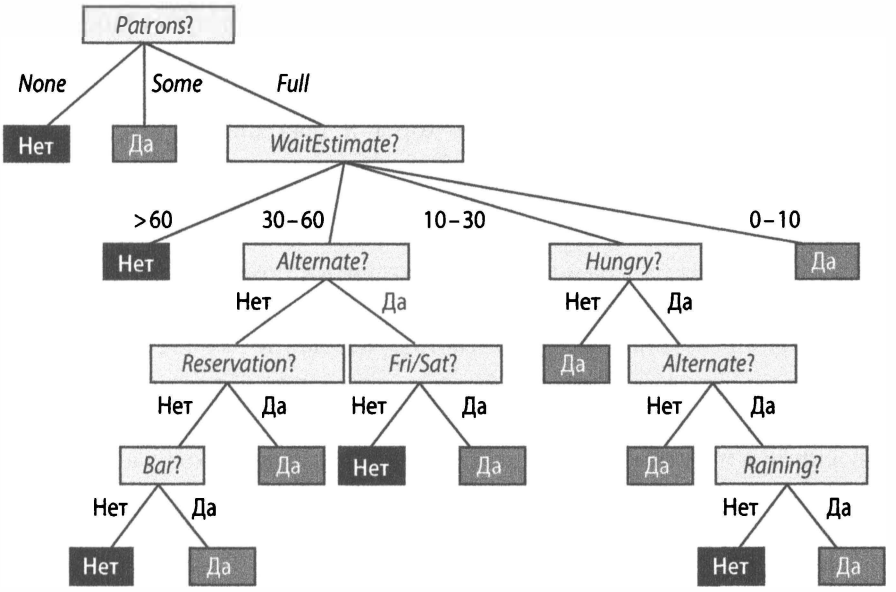


Рис. 19.3. Дерево решений для принятия решения о том, следует ли подождать, пока освободится столик в ресторане

19.3.1. Выразительность деревьев решений

Булево дерево решений эквивалентно логическому утверждению следующей формы:

$$Output \Leftrightarrow (Path_1 \vee Path_2 \vee \ldots),$$

где каждое условие $Path_i$ представляет собой конъюнкцию проверок значений атрибутов ($A_m = v_x \wedge A_n = v_y \wedge \ldots$), соответствующих пути от корня дерева к листу с результатом *true*. Следовательно, все выражение находится в дизъюнктивной

нормальной форме, а это означает, что любая функция логики высказываний может быть представлена в виде дерева решений.

Для многих задач формат дерева решений дает хороший, лаконичный и понятный результат. Действительно, многие руководства "Как..." (например, по ремонту автомобилей) построены в форме деревьев решений. Однако некоторые функции невозможно представить кратко. Например, мажоритарная функция, возвращающее значение *true* тогда и только тогда, когда более половины входных значений являются истинными, требует экспоненциально большого дерева решений, как и функция четности, возвращающая значение *true* тогда и только тогда, когда количество истинных входных атрибутов является четным. В случае вещественных атрибутов функцию $y > A_1 + A_2$ также будет сложно представить в виде дерева решений, поскольку границей решений здесь является диагональная линия, а все проверки в деревьях решений последовательно делят пространство на прямоугольные, выровненные по осям координат участки. Потребуется нагромоздить множество таких участков, чтобы достаточно приблизиться к диагональной линии. Другими словами, деревья решений хороши для одних видов функций и совсем не походят для других.

Существует ли *какое-либо* представление, которое будет эффективным для *всех* видов функций? К сожалению, ответ на этот вопрос — *нет*, существует слишком много функций, чтобы их можно было представить с помощью небольшого количества битов. Даже если рассматривать только булевы функции с n булевыми атрибутами, их таблицы истинности будут иметь 2^n строк, где каждая строка может давать результат *true* или *false*, поэтому будет существовать 2^{2^n} различных функций. При 20 атрибутах имеется $2^{1\,048\,576} \approx 10^{300\,000}$ функций, поэтому, если ограничиться представлением в миллион битов, представить все эти функции будет невозможно.

19.3.2. Обучение деревьев решений на примерах

Допустим, что необходимо построить дерево решений, которое будет соответствовать примерам, приведенным на рис. 19.2, и иметь как можно меньшие размеры. К сожалению, очень трудно найти гарантированно наименьшее согласованное дерево. Но при использовании некоторой простой эвристики можно эффективно отыскать такое дерево, которое будет достаточно близко к наименьшему. В алгоритме LEARN-DECISION-TREE принята жадная стратегия "разделяй и властвуй": первым всегда проверять наиболее важный атрибут, а затем аналогичным образом рекурсивно решать более мелкие подзадачи, которые будут определяться возможными результатами проверок. Здесь под "наиболее важным атрибутом" подразумевается тот, который имеет наибольшее значение для классификации примера. В результате можно надеяться получить правильную классификацию с относительно небольшим количеством тестов, а это означает, что все пути в дереве будут короткими, а дерево в целом будет небольшим.

На рис. 19.4, *а* показано, что атрибут *Type* не является подходящим на роль наиболее важного, поскольку после его проверки остаются четыре возможных результата, каждый из которых содержит одинаковое количество положительных и отрицательных примеров. С другой стороны, из рис. 19.4, *б* следует, что атрибут *Patrons* вполне можно рассматривать как важный, поскольку если его значение равно *None* или *Some*, то остаются такие множества примеров, для которых можно сразу же получить определенный ответ (*Да* и *Нет* соответственно). А если значением является *Full*, то остается смешанное множество примеров, — ниже описаны четыре случая, которые следует рассматривать при анализе подобных рекурсивных подзадач.

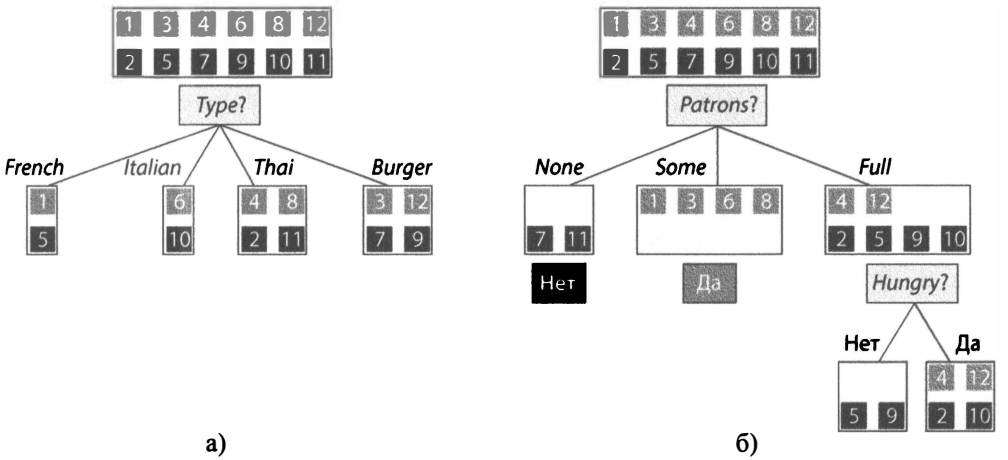


Рис. 19.4. Разбиение множества примеров на подмножества путем проверки атрибутов. Для каждого узла приведены положительные (светлые прямоугольники) и отрицательные (темные прямоугольники) оставшиеся примеры. **а)** Разбиение по атрибуту *Type* не позволяет приблизиться к решению задачи проведения различий между положительными и отрицательными примерами. **б)** Разбиение по атрибуту *Patrons* позволяет добиться большего в отношении разделения положительных и отрицательных примеров. После разбиения по атрибуту *Patrons* довольно неплохим кандидатом для второй проверки становится атрибут *Hungry*

1. Если все оставшиеся примеры являются положительными (или все они отрицательные), то конечный результат достигнут: можно дать окончательный ответ *Да* или *Нет*. На рис. 19.4, *б* соответствующие примеры имеются в ветвях *None* и *Some*.
2. Если имеются некоторые положительные и отрицательные примеры, то для выполнения нового разбиения следует выбрать очередной наилучший атрибут. На рис. 19.4, *б* показано, что для разбиения оставшихся примеров можно использовать атрибут *Hungry*.

3. Если не осталось ни одного примера, то это означает, что соответствующие примеры не наблюдались и следует возвратить наиболее часто встречающееся выходное значение для множества примеров, которые использовались при построении узла, являющегося родительским для данного узла.
4. Если не осталось ни одного атрибута, но имеются и положительные и отрицательные примеры, то это означает, что данные примеры имеют полностью одинаковое описание, но классифицированы по-разному. Такое происходит, если в данных есть ошибки или в них присутствует ▶ **шум** вследствие того, что данная проблемная область является недетерминированной, либо того, что не наблюдаются атрибуты, позволяющие провести различие между примерами. Наилучшее, что можно сделать в подобной ситуации, — это возвратить наиболее часто встречающееся выходное значение среди оставшихся примеров.

Алгоритм LEARN-DECISION-TREE приведен на рис. 19.5. Обратите внимание, что на входе алгоритма присутствует множество примеров, но они нигде не представлены в дереве, возвращаемом алгоритмом. Это дерево состоит из тестов атрибутов во внутренних узлах, значений атрибутов — в ветвях и выходных значений — в листовых узлах. Детали построения функции IMPORTANCE будут приведены в разделе 19.3.3. Результат работы данного обучающего алгоритма для нашего обучающего набора данных представлен на рис. 19.6. Очевидно, что построенное им дерево заметно отличается от исходного дерева, приведенного на рис. 19.3. Казалось бы, можно сделать вывод, что этот алгоритм обучения не очень хорошо справляется с изучением истинной функции. Однако делать такие выводы будет неправильно. Обучающий алгоритм рассматривает предоставленные ему *примеры*, а не истинную функцию, и фактически его гипотеза (см. рис. 19.6) не только соответствует всем предоставленным ему примерам, но и значительно проще, чем исходное дерево! При немного различающихся примерах это дерево могло бы оказаться совершенно иным, но функция, которую оно представляет, была бы той же самой.

В процессе работы обучающего алгоритма не было обнаружено причин для включения проверок значений атрибутов *Raining* и *Reservation*, поскольку алгоритм смог классифицировать все примеры и без них. Кроме того, алгоритм обнаружил интересную и ранее не наблюдавшуюся особенность: Стюарт Рассел готов ждать, чтобы есть тайскую пищу по уик-эндам. Также можно отметить, что построенное алгоритмом дерево способно выдавать ошибочные решения для случаев, которые не наблюдались в предоставленных примерах. Например, в обучающих примерах отсутствует ситуация, в которой потребовалось бы ждать от 0 до 10 минут, когда ресторан полон. В подобной ситуации дерево выводит ответ, что ждать не следует, когда атрибут *Hungry* имеет значение *false*, тогда как сам Стюарт Рассел, безусловно, согласился бы подождать. При предоставлении дополнительных обучающих примеров обучаемая программа сможет исправить эту ошибку.

function LEARN-DECISION-TREE(*examples*, *attributes*, *parent_examples*) **returns** дерево решений

if множество *examples* пусто **then return** PLURALITY-VALUE(*parent_examples*)
else if все примеры в *examples* имеют одну и ту же классификацию
  then return классификация
else if множество *attributes* пусто **then return** PLURALITY-VALUE(*examples*)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
  tree ← новое дерево решений с результатом проверки корневого узла A
  for each значние v из A **do**
    $exs \leftarrow \{e : e \in examples$ **and** $e.A = v\}$
    subtree ← LEARN-DECISION-TREE(*exs*, *attributes* − A, *examples*)
    добавить в дерево *tree* ветвь с меткой $(A = v)$ и поддерево *subtree*
  return *tree*

Рис. 19.5. Алгоритм обучения дерева решений. Функция IMPORTANCE подробно описывается в разделе 19.3.3. Функция PLURALITY-VALUE выбирает выходное значение, чаще всего встречающееся в множестве примеров, случайным образом разрывая связи

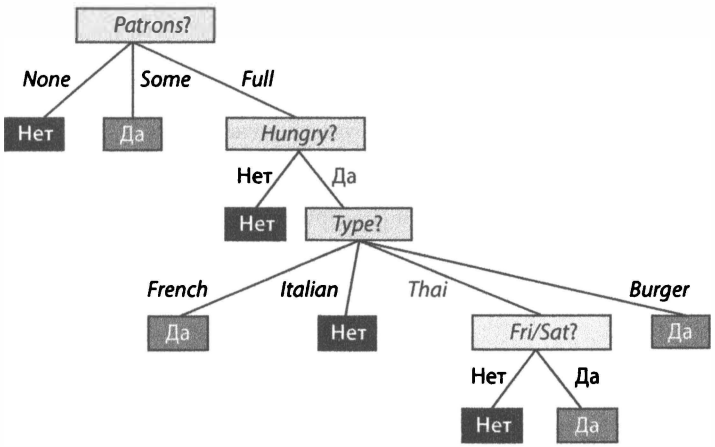


Рис. 19.6. Дерево решений, сформированное на основании обучающего набора с 12 примерами

Производительность алгоритма обучения можно оценить с помощью ►**кривой обучения**, как показано на рис. 19.7. Для построения этого рисунка использовалось 100 примеров, имевшихся в нашем распоряжении, которые случайным

образом разделялись на обучающий набор и тестовый набор. Гипотеза h изучалась на обучающем наборе, а ее точность измерялась с помощью тестового набора. Предусматривалось проводить этот эксперимент, начиная с обучающего набора размером в 1 пример и увеличивая его раз за разом на 1 пример до 99 примеров. В экспериментах для каждого размера обучающего набора процедура случайного разделения примеров на обучающие и тестовые данные повторялась 20 раз, а полученные 20 результатов испытаний усреднялись. Полученная кривая показывает, что с ростом размера обучающего набора точность предсказания увеличивается. (По этой причине подобные кривые обучения также называют ►**счастливыми графиками**.) На данном графике была достигнута точность предсказаний на уровне 95%, и похоже, что кривая могла бы и дальше демонстрировать возрастание точности, если бы у нас было больше данных.

Рис. 19.7. Кривая обучения для алгоритма обучения дерева решений на основании 100 случайным образом сформированных примеров в проблемной области задачи ожидания в ресторане. Каждая точка данных представляет собой среднее значение для 20 испытаний

19.3.3. Выбор проверок атрибутов

Алгоритм обучения дерева решений выбирает атрибут с наивысшим значением, возвращаемым функцией IMPORTANCE (важность). Теперь мы покажем, как можно измерить эту важность, воспользовавшись понятием получения информации, определяемой в терминах ►**энтропии**, являющейся фундаментальной величиной в теории информации (Шеннон и Уивер [2039], 1949).

Энтропия — это мера неопределенности случайной величины: чем больше информации, тем меньше энтропия. Случайная переменная, имеющая только одно возможное значение — скажем, монета, которая всегда падает орлом вверх, —

не имеет неопределенности, и поэтому ее энтропия определяется как нулевая. Обыкновенная монета при подбрасывании с равной вероятностью может упасть вверх орлом или решкой, и скоро будет показано, что это можно рассматривать как "1 бит" энтропии. Бросок правильной *четырехсторонней* кости имеет 2 бита энтропии, поскольку в этом случае существует 2^2 одинаково вероятных вариантов. Если обратиться к монете со смещенным центром тяжести, которая в 99% случаев падает вверх орлом, то интуитивно понятно, что результат ее подбрасывания связан с меньшей неопределенностью, чем в случае обычной монеты — если мы догадываемся, что орел не выпадет только в 1% бросков, — поэтому желательно, чтобы такая монета была связана с количеством энтропии, близким к нулю, но положительным. В общем случае энтропия случайной переменной V с возможными значениями v_k, имеющими вероятность $P(v_k)$, определяется как

$$H(V) = \sum_K P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k).$$

Теперь можно убедиться в том, что энтропия результата подбрасывания обычной монеты (обозначим его как *Fair*) действительно равняется 1 биту:

$$H(Fair) = -(0{,}5 \log_2 0{,}5 + 0{,}5 \log_2 0{,}5) = 1.$$

А энтропия броска четырехсторонней кости (*Die4*) составляет 2 бита:

$$H(Die4) = -(0{,}25 \log_2 0{,}25 + 0{,}25 \log_2 0{,}25 + 0{,}25 \log_2 0{,}25 + 0{,}25 \log_2 0{,}25) = 2.$$

Наконец, для монеты со смещенным центром тяжести, которая в 99% случаев падает вверх орлом, энтропия результата ее подбрасывания (*Loaded*) будет равна

$$H(Loaded) = -(0{,}99 \log_2 0{,}99 + 0{,}01 \log_2 0{,}01) \approx 0{,}08 \text{ бита}.$$

Также полезно будет определить $B(q)$ как энтропию булевой случайной переменной, которая принимает значение *true* с вероятностью q:

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q)).$$

Таким образом, $H(Loaded) = B(0{,}99) \approx 0{,}08$. А теперь вернемся к обучению дерева решений. Если обучающий набор содержит p положительных примеров и n отрицательных примеров, то энтропия выходной переменной *Output* на всем наборе будет равна

$$H(Output) = B\left(\frac{p}{p+n}\right).$$

Обучающий набор для задачи с рестораном, приведенный на рис. 19.2, включает одинаковое количество положительных и отрицательных примеров $p = n = 6$, поэтому соответствующая энтропия будет равна $B(0{,}5)$, или ровно 1 биту. Результат проверки атрибута A даст некоторую информацию, тем самым снижая общую

энтропию на некоторую величину. Это уменьшение можно измерить, оценив энтропию, оставшуюся после проверки атрибута.

Любой атрибут A, имеющий d различных значений, делит обучающий набор E на подмножества $E_1,\ldots,E_d$. Каждое подмножество E_k включает p_k положительных примеров и n_k отрицательных примеров, поэтому после перехода по ветви этого атрибута потребуется дополнительно $B(p_k/(p_k+n_k))$ битов информации, чтобы получить ответ на вопрос. Случайно выбранный пример из обучающего набора содержит k-е значение данного атрибута (т.е. находится в подмножестве E_k с вероятностью $(p_k+n_k)/(p+n)$), поэтому после проверки атрибута A ожидаемая остаточная энтропия (*Remainder*) будет равна

$$Remainder(A)=\sum_{k=1}^{d}\frac{p_k+n_k}{p+n}B(\frac{p_k}{p_k+n_k}).$$

► **Приращение информации**, полученное в результате проверки атрибута A, — это ожидаемое снижение энтропии:

$$Gain(A)=B(\frac{p}{p+n})-Remainder(A).$$

Фактически $Gain(A)$ — это как раз то, что необходимо для реализации функции IMPORTANCE. Возвращаясь к атрибутам, показанным на рис. 19.4, получаем следующее:

$$Gain(Patrons)=1-\left[\frac{2}{12}B(\frac{0}{2})+\frac{4}{12}B(\frac{4}{4})+\frac{6}{12}B(\frac{2}{6})\right]\approx 0{,}541 \text{ бит},$$

$$Gain(Type)=1-\left[\frac{2}{12}B(\frac{1}{2})+\frac{2}{12}B(\frac{1}{2})+\frac{4}{12}B(\frac{2}{4})+\frac{4}{12}B(\frac{2}{4})\right]=0 \text{ бит},$$

что подтверждает интуитивное предположение о том, что для первого разбиения атрибут *Patrons* подходит лучше. В действительности атрибут *Patrons* позволяет получить наибольшее приращение информации по сравнению с любыми другими атрибутами и, следовательно, должен быть выбран алгоритмом обучения дерева решений в качестве корневого.

19.3.4. Обобщение и переобучение

Необходимо, чтобы обучающие алгоритмы были способны находить гипотезу, соответствующую обучающему набору данных, но что еще более важно, необходимо, чтобы они также были способны хорошо обобщать данные, с которыми ранее не встречались. На рис. 19.1 видно, что полином высокой степени может соответствовать всем известным данным, но при этом имеет огромные пики, которые не подтверждаются данными: он подходит, но может и чрезмерно переоценивать данные. Такая ситуация называется *переобучением*, и ее вероятность возрастает по мере увеличения количества атрибутов и уменьшается с увеличением

количества обучающих примеров. Бо́льшие пространства гипотез (т.е. деревья решений с бо́льшим количеством узлов или многочлены более высокой степени) обладают как большей способностью обеспечивать соответствие, так и большей склонностью к переобучению, при этом одни классы моделей более склонны к переобучению, чем другие.

Для деревьев решений борьба с переобучением осуществляется с помощью метода, получившего название ▶**обрезка деревьев решений**. Обрезка работает посредством удаления узлов, релевантность которых неочевидна. Процедура начинается с полного дерева, сгенерированного алгоритмом LEARN-DECISION-TREE. Далее выполняется поиск тестовых узлов, имеющих в качестве потомков только листовые узлы. Если тест такого узла не имеет отношения к делу — т.е. позволяет обнаружить только шум в данных, — то этот тестовый узел исключается и заменяется листовым узлом. Этот процесс повторяется, и анализируется каждый тестовый узел с только листовыми потомками, пока каждый такой узел не будет или обрезан, или принят, как есть.

Основной вопрос состоит в том, как обнаружить, что в данном узле тестируется нерелевантный атрибут. Предположим, что рассматривается узел, включающий p положительных и n отрицательных примеров. Если данный атрибут является нерелевантным, то можно ожидать, что он будет разделять примеры на такие подмножества, что каждое из них будет содержать положительные примеры приблизительно в той же пропорции, что и во всем обучающем наборе, $p/(p+n)$, поэтому приращение информации будет близким к нулю.[3] Следовательно, низкое приращение информации является хорошим признаком того, что данный атрибут является нерелевантным. Теперь вопрос сводится к тому, насколько большим должно быть это приращение, чтобы имело смысл выполнять разбиение по какому-то определенному атрибуту?

Получить ответ на этот вопрос можно посредством ▶**проверки статистической значимости**. Такая проверка начинается с принятия предположения о том, что в данных нет никаких скрытых закономерностей (это предположение называется ▶**нуль-гипотезой**). Далее проводится анализ фактических данных для определения того, в какой степени они отклоняются от случая абсолютного отсутствия закономерностей. Если вычисленную степень отклонения можно считать статистически маловероятной (обычно это вероятность 5% или меньше), то это рассматривается как надежное свидетельство наличия значимых закономерностей в данных. Вероятности вычисляются на основании стандартных распределений величины отклонения, которые предположительно можно обнаружить в случайной выборке.

В данном случае нуль-гипотеза состоит в том, что атрибут является нерелевантным, а следовательно, приращение информации для бесконечно большой выборки

[3] Это приращение всегда будет строго положительным за исключением маловероятного случая, когда все эти пропорции будут абсолютно одинаковыми (см. упражнение 19.7).

будет равно нулю. Необходимо вычислить вероятность того, что после принятия нуль-гипотезы выборка размером $v = n + p$ будет показывать наблюдаемое отклонение от ожидаемого распределения положительных и отрицательных примеров. Такое отклонение можно измерить, сравнивая фактические количества положительных и отрицательных примеров в каждом подмножестве, p_k и n_k, с ожидаемыми количествами, $\hat{p}_k$ и $\hat{n}_k$, при условии, что принято предположение об истинной нерелевантности атрибута:

$$\hat{p}_k = p \times \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \times \frac{p_k + n_k}{p + n}.$$

Удобный критерий суммарного отклонения определяется следующей формулой:

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}.$$

Согласно нуль-гипотезе, значение Δ распределяется в соответствии с распределением χ^2 (хи-квадрат) с $d - 1$ степенями свободы. Статистическая функция χ^2 используется здесь для того, чтобы определить, подтверждает или опровергает конкретное значение Δ нуль-гипотезу. Например, в задаче об ожидании в ресторане рассмотрим атрибут *Type*, имеющий четыре значения, а следовательно, три степени свободы. Значение $\Delta = 7{,}82$ или более будет указывать на необходимость отклонить нулевую гипотезу на уровне 5% (а значение $\Delta = 11{,}35$ или более укажет на необходимость отклонить нулевую гипотезу на уровне 1%). Меньшие значения Δ позволяют принять гипотезу о нерелевантности атрибута, а это означает, что связанная с ним ветвь дерева решений может быть обрезана. Этот метод известен под названием ▶"χ^2-**обрезка**".

Такой метод обрезки позволяет справиться с шумом в примерах. Ошибки в метках примеров (т.е. пример классифицирован как (**x**, *да*), а должно быть (**x**, *нет*)) приводят к линейному увеличению ошибки предсказания, тогда как ошибки в описаниях примеров (т.е. указано, что атрибут *Price* = \$, тогда как в действительности должно быть *Price* = \$\$) оказывают асимптотическое влияние, которое становится все более ярко выраженным по мере того, как дерево все больше сокращается до меньших наборов. Обрезанные деревья работают значительно лучше необрезанных, когда данные содержат большое количество шума. Кроме того, обрезанные деревья часто бывают намного меньше, а следовательно, их структура проще для понимания, а работа — эффективнее.

И еще одно последнее предупреждение: можно решить, что обрезка χ^2 и приращение информации схожи, так почему бы не объединить их с использованием подхода, называемого ▶ **ранней остановкой**, — позволить алгоритму дерева решений прекратить генерацию узлов, когда больше не осталось хороших атрибутов для разделения, вместо того чтобы продолжать генерацию проблемных узлов, которые затем будут обрезаны? Серьезная проблема ранней остановки заключается

в том, что она блокирует возможность выявления ситуации, когда уже нет ни одного хорошего атрибута, но еще существуют такие их комбинации, которые будут информативны. Например, рассмотрим функцию XOR двух двоичных атрибутов. Если для всех четырех возможных комбинаций входных значений имеется примерно одинаковое количество примеров, то ни один из атрибутов не будет информативным, и правильным решением в этой ситуации будет разделить один из атрибутов (не имеет значения, какой именно), а затем, на следующем уровне, найти такие разделения, которые окажутся очень информативными. Ранняя остановка приведет к пропуску этого результата, тогда как подход "сгенерируй, а затем обрежь" позволит принять правильное решение в данной ситуации.

19.3.5. Расширение области применения деревьев решений

Область применения деревьев решений можно значительно расширить, если должным образом справиться со следующими сложностями.

- **Недостающие данные**. Во многих проблемных областях значения не всех атрибутов могут быть определены для каждого примера. Необходимые значения могут быть просто не зарегистрированы или их получение оказывается слишком дорогостоящим. Такая ситуация приводит к возникновению двух проблем. Во-первых, если дано полное дерево решений, то как следует классифицировать некоторый объект, для которого не задан один из проверяемых атрибутов? Во-вторых, как следует модифицировать формулу приращения информации, если в некоторых примерах значения данного атрибута неизвестны? Эти вопросы рассматриваются в упражнении 19.12.
- **Непрерывные и многозначные атрибуты**. Для непрерывных атрибутов, таких как *Height* (высота), *Weight* (вес) и *Time* (время), может оказаться, что каждый пример будет иметь собственное, отличное от других значение атрибута. Критерий приращения информации придаст оценке полезности такого атрибута очень высокую, зачастую не отвечающую действительности значимость, что приведет к построению обширного дерева с этим атрибутом в корне и поддеревьями с единственным примером для каждого из его возможных значений. Однако все это не поможет при поступлении нового примера с таким значением этого атрибута, которое еще не встречалось ранее.

  Лучшим способом справиться с непрерывными значениями является проверка относительно ▶**точки разбиения** (*split point*) — проверка значения атрибута на выполнение неравенства. Например, в определенном узле дерева может оказаться, что наибольший объем информации позволяет получить проверка по условию $Weight > 160$. Существуют эффективные методы нахождения хороших точек разделения: начните с сортировки значений атрибута, а затем рассмотрите только те точки разделения, которые находятся между двумя примерами в отсортированном порядке, имеющими различные

классификации, отслеживая при этом промежуточное число положительных и отрицательных примеров на каждой стороне точки разделения. Разделение — наиболее затратная часть реальных обучающих приложений на основе дерева решений.

Для атрибутов, которые не являются непрерывными и не позволяют применить какое-либо осмысленное упорядочение, но при этом имеют очень большое количество возможных значений — например, *Zipcode* (почтовый индекс) или *CreditCardNumber* (номер кредитной карточки), — можно применить показатель, называемый **коэффициентом приращения информации** (см. упражнение 19.13), позволяющий избежать разбиения узла на множество поддеревьев с единственным примером. Еще один полезный подход — применить **проверку равенства** в виде $A = v_k$. Например, тест *Zipcode* = 10002 можно использовать для отбора большой группы людей, проживающих в том районе Нью-Йорка, которому присвоен этот почтовый индекс, и направления всех остальных в поддерево *другое*.

- **Выходные атрибуты с непрерывными значениями.** Если предпринимается попытка предсказать некоторое числовое значение, такое как оценка стоимости жилья, то необходимо использовать ▶**дерево регрессии**, а не дерево классификации. В каждом листовом узле дерева регрессии задано не единственное значение, а линейная функция от некоторого подмножества числовых атрибутов. Например, ветвь, которая относится к квартирам с двумя спальнями, может оканчиваться линейной функцией от общей площади и количества ванных. Обучающий алгоритм должен выработать решение о том, когда следует прекратить разбиение и приступить к применению метода линейной регрессии (см. раздел 19.6) с использованием оставшихся атрибутов. Для обозначения обоих классов деревьев используется аббревиатура ▶**CART** — *Classification And Regression Trees*.

Система обучения на основе деревьев решений для реальных приложений должна быть способна справиться со всеми этими проблемами. Особенно важной является обработка переменных с непрерывными числовыми значениями, поскольку числовые данные широко применяются, например, в физических и финансовых процессах. Было разработано несколько коммерческих пакетов, соответствующих этим критериям, которые использовались для создания тысяч проблемно-ориентированных систем. Во многих областях промышленности и торговли деревья решений обычно становятся первым методом, к которому пытаются прибегнуть, когда из некоторого набора данных необходимо извлечь соответствующий ему метод классификации.

Деревья решений имеют много полезных качеств: простота понимания, масштабируемость для больших наборов данных и универсальность в обработке дискретных и непрерывных входных данных, а также в классификации и регрессии. Однако они могут иметь неоптимальную точность (в основном из-за жадного

поиска), и если деревья оказываются очень глубокими, то получение прогноза для нового примера может потребовать значительных затрат времени. Деревья решений также ▶**нестабильны** в том смысле, что добавление только одного нового примера может привести к изменению проверки в корне и как следствие к изменению всего дерева. В разделе 19.8.2 будет показано, что **модель случайного леса** позволяет справиться с некоторыми из этих проблем.

19.4. Выбор модели и оптимизация

В машинном обучении главной целью является выбор такой гипотезы, которая будет оптимально соответствовать будущим примерам. Чтобы достичь в этом необходимой точности, прежде всего следует дать определение понятиям "будущий пример" и "оптимальное соответствие".

Сначала сделаем предположение, что будущие примеры будут похожи на прежние. Назовем это предположение ▶**стационарностью**, — без него все дальнейшие рассуждения бесполезны. Будем полагать, что каждый пример E_j имеет одно и то же априорное распределение вероятностей

$$\mathbf{P}(E_j) = \mathbf{P}(E_{j+1}) = \mathbf{P}(E_{j+2}) = \ldots$$

и не зависит от предыдущих примеров:

$$\mathbf{P}(E_j) = \mathbf{P}(E_j \mid E_{j-1}, E_{j-2}, \ldots).$$

Примеры, которые удовлетворяют этим уравнениям, являются *независимыми и одинаково распределенными* или ▶**i.i.d.** — *independent and identically distributed*.

Теперь дадим определение "оптимальному соответствию". С этого момента будем считать, что оптимальное соответствие (*optimal fit*) — это гипотеза, минимизирующая ▶**частоту появления ошибок** (*error rate*): доля случаев, когда $h(x) \neq y$ для примера (x, y). (Позже это определение будет расширено так, чтобы позволить разным ошибкам иметь разную оценку, что даст некоторое преимущество ответам, которые можно считать "почти" правильными.) Частоту появления ошибок гипотезы можно оценить, просто проверив ее: эффективность гипотезы измеряется на **тестовом наборе** примеров. Для гипотезы (или студента) будет нечестно узнать ответы к тестам, прежде чем сдавать их. Самый простой способ добиться, чтобы этого не произошло, — разделить все имеющиеся примеры на две группы: **обучающий набор** для создания гипотезы и **тестовый набор** для ее оценки.

Если предполагается выработать только одну гипотезу, то такого подхода будет достаточно. Однако чаще обучение приводит к созданию нескольких гипотез: может потребоваться сравнить две совершенно разные модели машинного обучения или будет необходимо настроить различные "винтики" в одной модели. Например, можно было бы попробовать разные пороговые значения для χ^2-обрезки деревьев решений или разные степени для полиномов. Принято называть

эти "винтики" ▶**гиперпараметрами** — параметрами *класса* модели, а не конкретной модели.

Предположим, что исследователь создает гипотезу с одним значением настройки гиперпараметра χ^2-обрезки, измеряет частоту появления ошибок на тестовом наборе, а затем проверяет другие значения гиперпараметра. Ни одна отдельная гипотеза не встречалась с данными тестового набора, но *процесс* в целом — да, через исследователя, который мог наблюдать его неоднократно.

Чтобы избежать такой ситуации, нужно *действительно* полностью отказаться от обращения к тестовому набору — заблокировать его до тех пор, пока не будут полностью завершены все процедуры обучения, экспериментов, настройки гиперпараметров, повторного обучения и т.д. А это означает, что необходимо иметь *три* набора данных.

1. **Обучающий набор** (*training set*), предназначенный для обучения моделей-кандидатов.
2. ▶**Проверочный набор** (*validation set*), также известный как **набор разработки** (*development set* или *dev set*), предназначенный для оценки моделей-кандидатов и выбора лучшей из них.
3. **Тестовый набор** (*test set*), предназначенный для проведения окончательной объективной оценки самой лучшей модели.

Но что делать, если нет достаточного количества данных, чтобы можно было сформировать все три указанных выше набора? Можно попробовать выжать больше из имеющихся данных, используя метод, получивший название ▶***k*-блочная перекрестная проверка** (*k-fold cross-validation*), — в русскоязычной литературе также используются названия "кросс-валидация" и "скользящий контроль". Идея состоит в том, что каждый пример используется двояко — и как обучающий, и как проверочный, но не в одно и то же время. Сначала данные разбиваются на k равных подмножеств. Затем выполняется k раундов обучения, причем на каждом раунде $1/k$ часть данных резервируется как проверочный набор, а остальные примеры используются в качестве обучающего набора. Можно считать, что средняя оценка для тестовых наборов за k раундов, очевидно, должна быть лучше, чем единственная оценка. Популярными значениями для k являются 5 и 10 — этого достаточно, чтобы получить оценку, которая статистически вероятно будет точной, заплатив за это в 5 или 10 раз бо́льшими затратами вычислительного времени. Крайний случай, когда $k = n$, также называют **перекрестной проверкой по отдельным объектам** (*leave-one-out cross-validation* — ▶**LOOCV**). Но даже при использовании перекрестной проверки все еще требуется отдельный тестовый набор.

На рис. 19.1 видно, что на одном и том же наборе данных линейная функция недообучает модель, а полином высокой степени переобучает ее. Можно рассматривать задачу поиска хорошей гипотезы как состоящую из двух подзадач: ▶**выбора**

модели,[4] предполагающей выбор хорошего пространства гипотез, и ▶**оптимизации** (также называемой **обучением**), предполагающей поиск лучшей гипотезы в пределах выбранного пространства.

Подзадача выбора модели является качественной и субъективной, например можно предпочесть полиномы деревьям решений, исходя из того, что уже известно о решаемой задаче. И одновременно эта подзадача является количественной и эмпирической: в пределах класса полиномов можно остановиться на полиномах 2-й степени, потому что они лучше всего работают на проверочном наборе данных.

19.4.1. Выбор модели

На рис. 19.8 представлен простой алгоритм MODEL-SELECTION. На входе в качестве аргумента он принимает обучающий алгоритм *Learner* (например, это может быть алгоритм LEARN-DECISION-TREE), имеющий один гиперпараметр, который в алгоритме представлен переменной *size*. Для деревьев решений гиперпараметр *size* может определять количество узлов в дереве, а для полиномов он может определять их степень. Работа алгоритма MODEL-SELECTION начинается с наименьшим значением гиперпараметра *size*, что предполагает простейшую модель (которая, вероятно, будет склонна к недообучению) с последующей итерацией по возрастающим значениям *size*, рассматривая все более сложные модели. В конце работы алгоритм MODEL-SELECTION выбирает модель, имеющую самую низкую среднюю частоту ошибок на основании данных о результатах выполненных проверок.

На рис. 19.9 приведены два типичных шаблона динамики показателей, встречающихся при выборе модели. В обоих случаях (*а* и *б*) ошибка на обучающем наборе монотонно уменьшается (с небольшими случайными колебаниями) по мере возрастания сложности модели. На рис. 19.9, *а* сложность измеряется количеством узлов дерева решений, а на рис. 19.9, *б* она определяется количеством нейронных сетевых параметров (w_i). Для многих классов моделей по мере увеличения их сложности ошибка на обучающем наборе достигает нуля.

Эти два случая заметно различаются поведением ошибки на проверочном наборе. На рис. 19.9, *а* мы видим типичную U-образную кривую: некоторое время ошибка уменьшается по мере возрастания сложности модели, но после достижения точки, в которой начинает проявляться переобучение модели, ошибка на проверочном наборе начинает возрастать. Алгоритм MODEL-SELECTION выбирает значение в нижней части U-образной кривой ошибки: в данном случае — дерево

[4] Хотя название "выбор модели" широко используется, лучшим названием было бы "выбор *класса* модели" или "выбор пространства гипотез". Дело в том, что слово "модель" используется в литературе для ссылок на три различных уровня конкретизации: широкое пространство гипотез (например, "полиномы"), пространство гипотез с заполненными гиперпараметрами (например, "полиномы 2-й степени") и конкретная гипотеза со всеми заполненными параметрами (например, $5x^2 + 3x - 2$).

размером 7. Это точка, в которой наблюдается наилучший баланс между недообучением и переобучением модели. На рис. 19.9, *б* в начале также имеется U-образная кривая, как и на рис. 19.9, *а*, но затем в некоторый момент ошибка вновь начинает уменьшаться и достигает наименьшего значения в конечной точке графика, при 1 000 000 параметров.

```
function MODEL-SELECTION(Learner, examples, k) returns
                                        пара значений (гипотеза, частота ошибок)

  err ← массив, индексированный по size и предназначенный для хранения
                                        частот ошибок проверочных наборов
  training_set, test_set ← разделение множества примеров examples на два набора
  for size = 1 to ∞ do
      err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)
      if err начинает существенно возрастать then
          best_size ← значение size с минимальным err[size]
          h ← Learner(best_size, training_set)
          return h, ERROR-RATE(h, test_set)

function CROSS-VALIDATION(Learner, size, examples, k) returns частота ошибок
  N ← количество примеров в множестве examples
  errs ← 0
  for i = 1 to k do
      validation_set ← examples[(i − 1) × N/k:i × N/k]
      training_set ← examples − validation_set
      h ← Learner(size, training_set)
      errs ← errs + ERROR-RATE(h, validation_set)
  return errs/k          // Средняя частота ошибок для проверочных наборов
                         // по всей процедуре k-блочной перекрестной проверки
```

Рис. 19.8. Алгоритм выбора модели с наименьшей ошибкой на проверочных данных. Алгоритм строит ряд моделей с возрастающей сложностью и выбирает из них одну с лучшей эмпирической частотой ошибок *err* на проверочном наборе данных. Функция *Learner* (*size*, *examples*) возвращает гипотезу, сложность которой определяется параметром *size*, обученную на наборе *examples*. В функции CROSS-VALIDATION на каждой итерации цикла **for** выбирается разная группа примеров в качестве проверочного набора, а оставшиеся примеры множества *examples* используются как обучающий набор. Функция возвращает среднюю ошибку проверочного набора по всем итерациям. После определения, какое значение параметра *size* является наилучшим, алгоритм MODEL-SELECTION возвращают модель (т.е. *learner*/*hypothesis*) этого размера, обученную на всех обучающих примерах, вместе с ее частотой ошибок на тестовых примерах

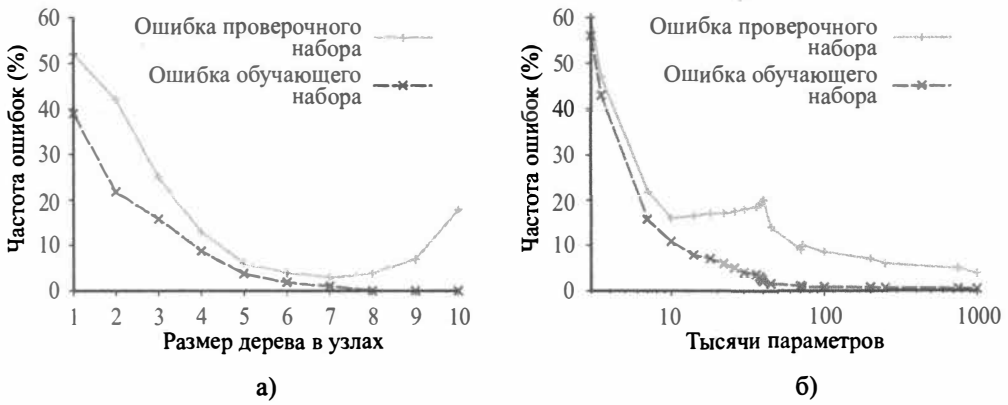


Рис. 19.9. Частота ошибок на обучающем наборе (нижняя темная линия) и на проверочном наборе данных (верхняя светлая линия) моделей различной сложности для двух различных задач. Алгоритм MODEL-SELECTION выбирает значение гиперпараметра с наименьшей ошибкой на проверочном множестве. **а)** На этом рисунке класс модели — деревья решений, а гиперпараметр — количество их узлов. Данные взяты из определенной версии задачи об ожидании в ресторане. Оптимальный размер дерева — 7. **б)** На этом рисунке класс модели представляет собой сверточные нейронные сети (см. раздел 21.3), а гиперпараметр определяет количество регулярных параметров в сети. Данные — это набор данных MNIST, включающий изображения цифр, а задача заключается в идентификации каждой цифры. Оптимальное количество параметров — 1 000 000 (обратите внимание, что на рисунке используется логарифмическая шкала)

Почему одни кривые ошибки на проверочном наборе выглядят так, как на рис. 19.9, *а*, а другие — как на рис. 19.9, *б*? Все сводится к тому, как различные классы моделей используют их избыточную емкость и насколько хорошо это соответствует рассматриваемой задаче. По мере того как растет емкость класса модели, часто достигается точка, когда в модели все обучающие примеры могут быть представлены идеально. Например, для обучающего набора с n различными примерами всегда существует дерево решений с n конечными узлами, способное точно представить все примеры набора.

Говорят, что модель, точно соответствующая всем данным обучения, ►**интерполирует** данные.[5] Классы моделей, как правило, начинают переобучаться по мере приближения их емкости к точке интерполяции. Похоже, это происходит потому, что бо́льшая часть емкости модели сосредоточена на обучающих примерах, а оставшаяся емкость распределяется довольно случайным образом, не представляющим шаблоны в проверочном наборе данных. Одни классы моделей

[5] Некоторые авторы говорят, что модель "запомнила" данные.

никогда не оправляются от подобного переобучения, подобно деревьям решений на рис. 19.9, *а*. Но для других классов моделей повышение их емкости означает, что появляется больше функций-кандидатов и некоторые из них естественным образом лучше подходят для шаблонов данных, присутствующих в истинной функции $f(x)$. Чем выше становится емкость модели, тем больше будет таких подходящих представлений и выше будет вероятность того, что механизм оптимизации сможет их обнаружить и остановиться на одном из них.

Глубокие нейронные сети (глава 21), ядерные машины опорных векторов (раздел 19.7.5), случайные леса (раздел 19.8.2) и бустинг ансамблей (раздел 19.8.4) — все эти модели обладают свойством уменьшения ошибки на проверочном наборе по мере увеличения их емкости, как на рис. 19.9, *б*.

Алгоритм выбора модели можно расширять различными способами, например сравнивать совершенно различные классы моделей, вызвав функцию MODEL-SELECTION с алгоритмом DECISION-TREE-LEARNER в качестве аргумента, а затем — с алгоритмом POLYNOMIAL-LEARNER и посмотрев, какой из них работает лучше. Также можно разрешить наличие нескольких гиперпараметров, хотя это и означает, что потребуется более сложный алгоритм оптимизации, такой как сеточный поиск (см. раздел 19.4.4), а не линейный поиск.

19.4.2. От частот ошибок к потерям

До сих пор обсуждалось лишь, как минимизировать частоту появления ошибок. Это, очевидно, лучше, чем максимизировать эту частоту, но далеко еще не все. Обратимся к задаче классификации сообщений электронной почты как спам или не-спам. Гораздо хуже классифицировать не-спам как спам (и, следовательно, потенциально пропустить важное сообщение), чем классифицировать спам как не-спам (и, следовательно, испытать лишь несколько секунд раздражения). А это означает, что классификатор с частотой ошибок 1%, у которого почти все ошибки — классификация спама как не-спама, будет лучше, чем классификатор с частотой ошибок всего 0,5%, но при этом большинство ошибок — классификация не-спама как спама. В главе 16 было показано, что лица, принимающие решения, должны максимизировать ожидаемую полезность, а полезность — это то, что учащиеся также должны максимизировать. Однако в машинном обучении этот аспект традиционно выражают в отрицательном ракурсе: минимизировать ▶**функцию потерь**, а не максимизировать функцию полезности. Функция потерь $L(x, y, \hat{y})$ определяется как количество потерянной полезности за счет выдачи прогноза $h(x) = \hat{y}$, когда в действительности правильным ответом будет $f(x) = y$.

$$L(x, y, \hat{y}) = \textit{Utility}\ (\text{результата использования } y \text{ при стимуле } x) - \textit{Utility}\ (\text{результата использования } \hat{y} \text{ при стимуле } x)$$

Это самая общая формулировка функции потерь. Чаще используется упрощенная версия $L(y, \hat{y})$, которая не зависит от x. В остальной части данной главы будет

использоваться упрощенная версия, и это означает, что нельзя будет сказать, что хуже: неправильно классифицировать письмо от мамы или неправильно классифицировать письмо от раздражающего вас двоюродного брата, но можно будет сказать, что классифицировать не-спам как спам в 10 раз хуже, чем наоборот:

$$L(spam, nospam) = 1, \qquad L(nospam, spam) = 10.$$

Обратите внимание, что значение $L(y, y)$ всегда равно нулю, — по определению потерь нет, если предсказание точно. Для функций с дискретными выходами можно перечислить значения потерь для каждой возможной неправильной классификации, но невозможно перечислить все возможности для данных в виде реальных чисел. Если $f(x)$ равняется 137,035999, можно рассматривать прогноз $h(x) = 137{,}036$ как удачный, но насколько он удачный? В общем случае маленькая ошибка лучше, чем большая, и две функции, реализующие данную идею, — это абсолютное значение разности (называемое потерей L_1) и квадрат этой разности (называемый потерей L_2, — здесь 2 можно воспринимать как математический знак возведения в квадрат). Для дискретных выходов, если нас устраивает идея минимизации частоты ошибок, можно использовать функцию потерь $L_{0/1}$, которая принимает значение 1 для любого неправильного ответа.

Абсолютное значение потери:	$L_1(y, \hat{y}) = \lvert y - \hat{y} \rvert$
Квадратичная оценка потери:	$L_2(y, \hat{y}) = (y - \hat{y})^2$
0/1 (бинарная) потеря:	$L_{0/1}(y, \hat{y}) = 0$, если $y = \hat{y}$, иначе 1

Теоретически обучающийся агент максимизирует свою ожидаемую полезность, выбирая гипотезу, которая минимизирует ожидаемые потери по всем парам ввода-вывода, которые он наблюдает. Для вычисления этого ожидания необходимо определить априорное распределение вероятностей $\mathbf{P}(X, Y)$ для примеров. Пусть $\mathcal{E}$ — это множество всех возможных примеров ввода-вывода. Тогда ожидаемые ▶**потери при обобщении** (*generalization loss*) для гипотезы h (по отношению к функции потерь L) можно представить как

$$GenLoss_L(h) = \sum_{(x,y)\in\mathcal{E}} L(y, h(x))P(x, y),$$

и лучшая гипотеза h^* — это гипотеза с минимальными ожидаемыми потерями при обобщении:

$$h^* = \operatorname*{argmin}_{h\in\mathcal{H}} GenLoss_L(h).$$

Поскольку распределение $P(x, y)$ в большинстве случаев неизвестно, обучающийся агент может *оценивать* потери обобщения только по ▶**эмпирическим потерям** на множестве примеров E размером N:

$$EmpLoss_{L,E}(h) = \sum_{(x,y)\in E} L(y, h(x)) \frac{1}{N}.$$

Тогда предполагаемой лучшей гипотезой $\hat{h}^*$ будет та, эмпирические потери которой минимальны:

$$\hat{h}^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, EmpLoss_{L,E}(h).$$

Существует четыре причины, по которым функция $\hat{h}^*$ может отличаться от истинной функции f: нереализуемость, дисперсия, шум и сложность вычислений.

Что касается первой, говорят, что задача обучения ▶**реализуема**, если пространство гипотез $\mathcal{H}$ действительно содержит истинную функцию f. Если пространство гипотез $\mathcal{H}$ является множеством линейных функций, а истинная функция f является квадратичной функцией, то никакое количество обучающих данных не восстановит истинную функцию f. Вторая причина, **дисперсия**, означает, что алгоритм обучения в общем случае будет генерировать разные гипотезы для разных наборов примеров. Если задача является реализуемой, то дисперсия уменьшается к нулю, по мере увеличения количества примеров в обучающем наборе. Третья причина указывает на то, что функция f может быть недетерминированной или ▶**зашумленной**, т.е. она может возвращать разные значения $f(x)$ для одного и того же x. По определению шум нельзя предсказать, — его можно только охарактеризовать. И наконец, если $\mathcal{H}$ представляет сложную функцию в большом пространстве гипотез, она может оказаться **вычислительно неразрешимой** в отношении выполнения систематического поиска всех возможностей. В этом случае поиск может исследовать лишь часть пространства и вернуть достаточно хорошую гипотезу, но не всегда гарантированно лучшую.

В первые годы исследования в области машинного обучения (как, впрочем, и традиционные методы в статистике) сосредоточивались на ▶**мелкомасштабном обучении**, когда число примеров в обучающих наборах варьировало от десятков до первых тысяч. В подобных случаях потери при обобщении в основном происходят из-за ошибок аппроксимации — отсутствия истинной функции f в пространстве гипотез, а также ошибок оценки, возникающих по причине отсутствия такого количества обучающих примеров, которое необходимо для ограничения дисперсии.

В последние годы акцент сместился на ▶**крупномасштабное обучение** с миллионами примеров. В этом случае потери при обобщении могут определяться преимущественно вычислительными ограничениями: имеется достаточное количество данных и можно отыскать достаточно богатую модель, для которой гипотеза h будет очень близка к истинной функции f, но вычисления, необходимые для ее отыскания, слишком сложны, поэтому приходится соглашаться на приближение, а не на точный результат.

19.4.3. Регуляризация

В разделе 19.4.1 было показано, как осуществляется выбор модели с перекрестной проверкой. Альтернативным подходом является поиск гипотезы, которая точно минимизирует взвешенную сумму эмпирических потерь и сложности гипотезы, которую мы будем называть общей стоимостью:

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h)$$

$$\hat{h}^* = \operatorname*{argmin}_{h \in \mathcal{H}} Cost(h).$$

Здесь λ — это гиперпараметр, положительное число, которое служит коэффициентом пересчета между потерями и сложностью гипотезы. Если значение λ выбрано правильно, эмпирические потери простой функции хорошо уравновешиваются с тенденцией сложной функции к переобучению.

Такой процесс явного наказания сложных гипотез называется ▶**регуляризацией**: цель состоит в отыскании более регулярных функций. Обратите внимание, что теперь необходимо сделать два выбора: выбрать функцию потерь (L_1 или L_2) и меру сложности, называемую ▶**регуляризационной функцией**. Выбор регуляризационной функции зависит от пространства гипотез. Например, для полиномов хорошей регуляризационной функцией является сумма квадратов коэффициентов — стремление сохранить эту сумму небольшой удерживает от обращения к замысловатым полиномам 12-й степени, подобных показанному на рис. 19.1. Пример такого типа регуляризации приведен в разделе 19.6.3.

Другой способ упрощения моделей — уменьшение размерностей, с которыми работают модели. Процесс ▶**отбора признаков** заключается в отбрасывании атрибутов, которые кажутся нерелевантными. В частности, χ^2-обрезка — это один методов отбора признаков.

В действительности есть возможность получить эмпирические потери и сложность измеренными в одном и том же масштабе, без коэффициента пересчета λ: и то, и другое можно измерить в битах. Сначала закодируйте гипотезу как программу машины Тьюринга и подсчитайте количество битов. Затем подсчитайте количество битов, необходимых для кодирования данных, правильно спрогнозированный пример которых стоит нуль битов, а стоимость неправильно спрогнозированного примера зависит от того, насколько велика ошибка. ▶**Минимальная длина описания**, или MDL — *minimum description length*, минимизирует общее количество требуемых битов. Этот подход хорошо работает в пределе, но для небольших задач выбор кодировки для программы — как лучше всего закодировать дерево решений в виде строки битов — влияет на результат. В главе 20 описывается вероятностная интерпретация подхода MDL.

19.4.4. Настройка гиперпараметра

В разделе 19.4.1 было показано, как выбрать лучшее значение гиперпараметра *size*, применяя перекрестную проверку к каждому возможному значению до тех пор, пока частота ошибок не начнет возрастать. Это хороший подход, когда есть только один гиперпараметр с небольшим количеством возможных значений. Но когда есть несколько гиперпараметров или когда они имеют непрерывные значения, выбрать хорошие значения будет труднее.

Пожалуй, самым простым подходом к настройке гиперпараметра является ▶**ручная настройка**: исходя из прошлого опыта, приблизительно выбрать определенные значения параметра, обучить модель, измерить ее производительность на проверочных данных, проанализировать результаты и, полагаясь на собственную интуицию, предложить новые значения параметра. Процедура повторяется до тех пор, пока не будет достигнута удовлетворительная производительность (или не закончится отведенное на настройку время, выделенный бюджет или терпение).

Если есть лишь несколько гиперпараметров, каждый из которых имеет небольшое количество возможных значений, то в этом случае будет уместен более систематический подход, называемый ▶**сеточным поиском** (*grid search*): попробуйте использовать все комбинации значений и посмотрите, какие из них лучше других работают с проверочными данными. Различные комбинации могут параллельно проверяться на разных машинах, так что, если у вас достаточно вычислительных ресурсов, подобная процедура необязательно будет слишком длительной, хотя известны случаи, когда выбор модели мог потребовать нескольких дней непрерывной работы кластеров из тысяч компьютеров.

Стратегии и методы поиска, обсуждавшиеся в главах 3 и 4, также вполне применимы. Например, если два гиперпараметра независимы друг от друга, их можно оптимизировать по отдельности.

Если существует слишком много комбинаций возможных значений, то можно обратиться к ▶**случайному поиску**, равномерно делая выборки из полного множества всех возможных значений гиперпараметров и повторяя эту процедуру до тех пор, пока есть возможность тратить на это время и вычислительные ресурсы. Случайная выборка также является хорошим способом работы с непрерывными значениями.

Когда каждый обучающий прогон требует достаточно много времени, может оказаться полезным постараться извлечь полезную информацию из каждого. В методе ▶**байесовской оптимизации** проблема выбора хорошего значения гиперпараметра сама по себе рассматривается как задача машинного обучения. Имеется в виду, что вектор значений гиперпараметров $\mathbf{x}$ принимается как входное значение, а полные потери на проверочном наборе для модели, построенной с этими значениями гиперпараметров, — как выходное значение y, и наша задача — найти такую функцию $y = f(\mathbf{x})$, которая минимизирует потери y. После выполнения каждого обучающего прогона мы получаем новую пару $(y, f(\mathrm{x}))$, которую можно использовать для обновления текущего представления о форме функции f.

Идея состоит в поиске компромисса между эксплуатацией (выбором таких значений параметров, которые близки к предыдущему хорошему результату) и разведкой (испытанием новых значений параметров). Это тот же компромисс, который уже рассматривался при обсуждении поиска по дереву методом Монте-Карло (раздел 5.4), и в действительности идея верхних доверительных границ также используется здесь, чтобы свести к минимуму сожаление. Если принять допущение, что функцию f можно аппроксимировать **гауссовым процессом**, то математические методы обновления текущего представления о функции f отлично работают. Сноек и соавт. в работе [2106] (2013) подробно прокомментировали математические выкладки и дали практическое руководство по этому подходу, показав, что он позволяет достичь лучших результатов в сравнении с ручной настройкой параметров, даже если она будет выполнена экспертами.

Альтернативой байесовской оптимизации является метод ▶**обучения популяции** (*population-based training* — **PBT**). Процедура PBT начинается с использования случайного поиска для обучения (параллельно) некоторой совокупности моделей, каждая из которых получает разные значения гиперпараметров. Затем обучается второе поколение моделей, значения гиперпараметров которых выбираются на основе успешных значений предыдущего поколения, с добавлением некоторых случайных мутаций, как в генетических алгоритмах (раздел 4.1.4). Таким образом, в методе обучения популяции преимущества метода случайного поиска — когда много прогонов может выполняться параллельно — комбинируются с преимуществами байесовской оптимизации (или ручной настройки хорошим специалистом), заключающимися в использовании информации из предыдущих прогонов для оптимизации последующих.

19.5. Теория вычислительного обучения

Как можно достичь уверенности в том, что обученная гипотеза будет хорошо предсказывать ранее неизвестные ей входы? Иначе говоря, как можно узнать, что гипотеза h близка к целевой функции f, если неизвестно, что собой представляет f? Эти вопросы обдумывались на протяжении многих веков Оккамом, Юмом и др. В последние десятилетия возникли и другие вопросы: сколько примеров необходимо, чтобы получить хорошую гипотезу h? Какое пространство гипотез лучше использовать? Если пространство гипотез очень сложное, можно ли вообще найти лучшую гипотезу h, или же следует удовлетвориться локальным максимумом? Насколько сложной должна быть гипотеза h? Как избежать переобучения? Все эти вопросы рассматриваются в данном разделе.

Начнем с вопроса о том, сколько примеров необходимо для обучения. Из кривой обучения дерева решений для задачи ожидания в ресторане (см. рис. 19.7) можно видеть, что его точность повышается с увеличением количества данных в обучающем наборе. Кривые обучения полезны, но они будут специфическими для

конкретного алгоритма обучения по конкретной задаче. Существуют ли более общие принципы, регулирующие количество необходимых примеров?

Такие вопросы решаются с помощью ▶**теории вычислительного обучения** (или просто теории обучения) — научной области, лежащей на стыке областей искусственного интеллекта, статистики и теоретических компьютерных наук. Основополагающий принцип состоит в том, что любая гипотеза, которая серьезно ошибочна, почти наверняка будет "обнаружена" с высокой вероятностью после небольшого числа примеров, поскольку сделает неправильное предсказание. Следовательно, любая гипотеза, согласованная с достаточно большим набором обучающих примеров, едва ли будет серьезно неправильной, т.е. она должна быть ▶**вероятностно приблизительно корректной** — или **ВПК** (*probably approximately correct* — ▶**PAC**).

Любой алгоритм обучения, возвращающий гипотезы, которые вероятностно приблизительно корректны, называется алгоритмом ▶**ВПК-обучения**. Этот подход можно использовать для установки границ производительности различных обучающих алгоритмов.

Теоремы ВПК-обучения, как и все другие теоремы, являются логическими следствиями аксиом. Если теоремы (в отличие, скажем, от политиков), утверждая что-то о будущем, всегда опираются на известное прошлое, то аксиомы должны предоставить "основы", позволяющие установить эту связь. Для ВПК-обучения эти основы обеспечиваются предположением о стационарности, введенном в разделе 19.4, в котором говорится, что будущие примеры всегда будут соответствовать тому же самому фиксированному распределению $\mathbf{P}(E) = \mathbf{P}(X, Y)$, что и предыдущие примеры. (Обратите внимание: нет никакой необходимости знать, какое это распределение; достаточно того, что оно *не меняется*.) Кроме того, для простоты будем предполагать, что истинная функция f является детерминированной и входит в состав рассматриваемого пространства гипотез $\mathcal{H}$.

Простейшие ВПК-теоремы имеют дело с булевыми функциями, которым соответствуют потери $L_{0/1}$. **Частота ошибки** гипотезы h, определенная выше неформально, здесь формально определяется как ожидаемая ошибка обобщения *error* на примерах, взятых из стационарного распределения:

$$error(h) = GenLoss_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x))P(x, y).$$

Другими словами, $error(h)$ — это вероятность того, что гипотеза h неверно классифицирует новый пример. Это та самая величина, которая ранее измерялась экспериментально посредством изучения кривых обучения.

Гипотеза h называется **приблизительно корректной**, если $error(h) \leq \epsilon$, где ϵ — небольшая константа. Покажем, что можно найти такое N, что после обучения на N примерах все согласованные гипотезы с высокой вероятностью станут приблизительно корректными. Приблизительно корректная гипотеза может рассматриваться как "близкая" к истинной функции в пространстве гипотез: она находится

внутри так называемого ▶ ϵ-**шара**, окружающего истинную функцию f. Пространство гипотез вне этого шара обозначают как $\mathcal{H}_{bad}$.

Получить оценку вероятности того, что "серьезно ошибочная" гипотеза $h_b \in \mathcal{H}_{bad}$ будет согласована с первыми N примерами, можно следующим образом. Нам известно, что $error(h_b) > \epsilon$. Следовательно, вероятность того, что эта гипотеза согласуется с заданным примером, равна по меньшей мере $1 - \epsilon$. Поскольку все примеры независимы, граничное значение этой вероятности для N примеров будет равно

$$P(h_b \text{ согласована с } N \text{ примерами}) \le (1-\epsilon)^N.$$

Вероятность того, что пространство $\mathcal{H}_{bad}$ содержит хотя бы одну согласованную гипотезу, ограничивается суммой отдельных вероятностей:

$$P(\mathcal{H}_{bad} \text{ содержит согласованную гипотезу}) \le |\mathcal{H}_{bad}|\,(1-\epsilon)^N \le |\mathcal{H}|\,(1-\epsilon)^N,$$

где учитывается тот факт, что $\mathcal{H}_{bad}$ является подмножеством $\mathcal{H}$, и, следовательно, $|\mathcal{H}_{bad}| \le |\mathcal{H}|$. Желательно уменьшить вероятность этого события так, чтобы она не превышала некоторого небольшого числа δ:

$$P(\mathcal{H}_{bad} \text{ содержит согласованную гипотезу}) \le |\mathcal{H}|\,(1-\epsilon)^N \le \delta.$$

С учетом того, что $1 - \epsilon \le e^{-\epsilon}$, этой цели можно добиться, предоставив алгоритму возможность обработать следующее количество примеров:

$$N \ge \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|\mathcal{H}|\right). \tag{19.1}$$

Таким образом, с вероятностью по крайней мере $1 - \delta$ после обучения на таком множестве примеров обучающий алгоритм вернет гипотезу, имеющую ошибку не больше ϵ. Другими словами, она будет вероятностно приблизительно корректной. Количество требуемых примеров, таких как функция от ϵ и δ, называется ▶ **выборочной сложностью** (*sample complexity*) обучающего алгоритма.

Как было показано выше, если $\mathcal{H}$ — множество всех булевых функций от n атрибутов, то $|\mathcal{H}| = 2^{2^n}$. Следовательно, выборочная сложность пространства возрастает как 2^n. Поскольку количество возможных примеров также равно 2^n, то можно утверждать, что ВПК-обучение в классе всех булевых функций требует просмотра всех или почти всех возможных примеров. Причину этого раскрывает следующее простое наблюдение: пространство гипотез $\mathcal{H}$ содержит достаточно гипотез для классификации любого заданного множества примеров всеми возможными способами. В частности, для любого набора из N примеров множество гипотез, согласованных с этими примерами, будет содержать столько же гипотез, предсказывающих, что x_{N+1} будет положительным, сколько и гипотез, предсказывающих, что x_{N+1} будет отрицательным.

Таким образом, чтобы получить реальное обобщение еще не встречавшихся примеров, кажется, нужно каким-то образом ограничить пространство гипотез $\mathcal{H}$.

Но, конечно, вводя ограничения на это пространство, можно заодно полностью исключить из него и истинную функцию. Есть три способа избежать этой дилеммы.

Во-первых, для устранения этой проблемы можно воспользоваться предварительными знаниями.

Во-вторых, можно воспользоваться способом, представленным в разделе 19.4.3 и заключающимся в принятии условия, что алгоритм должен возвращать не просто какую-либо согласованную гипотезу, а предпочтительно простую (как это делается при обучении дерева решений). В случаях, когда найти простые согласованные гипотезы достаточно легко, результаты выборки по сложности обычно лучше, чем при анализе, основанном только на согласованности.

В-третьих, можно обратиться к способу, который подробно рассматривается ниже и состоит в том, что нужно сосредоточиться лишь на тех подмножествах всего пространства гипотез булевых функций, которые доступны для обучения. Этот подход основан на предположении, что ограниченное пространство гипотез содержит гипотезу h, достаточно близкую к истинной функции f. Преимущество заключается в том, что ограниченное пространство гипотез позволяет эффективно обобщать и, как правило, в нем проще вести поиск. А теперь рассмотрим одно из таких ограниченных пространств гипотез более подробно.

19.5.1. Пример ВПК-обучения: обучение списков решений

В этом разделе показано, как можно применить методы ВПК-обучения к новому пространству гипотез: ▶ **спискам решений**. Список решений состоит из серии проверок, каждая из которых представляет собой конъюнкцию литералов. Если проверка, применяемая к описанию примера, завершается успешно, то список решений задает возвращаемое значение. Если же проверка оканчивается неудачей, то обработка продолжается со следующей проверки в списке. Списки решений напоминают деревья решений, но их общая структура проще: они разветвляются только в одном направлении. И наоборот, отдельные проверки в них намного сложнее. На рис. 19.10 приведен список решений, представляющий следующую гипотезу:

$$WillWait \Leftrightarrow (Patrons = Some) \vee (Patrons = Full \wedge Fri/Sat).$$

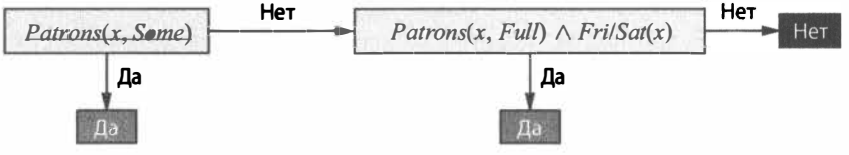


Рис. 19.10. Список решений для задачи ожидания в ресторане

Если допустить наличие проверок произвольного размера, то списки решений позволят представить любую булеву функцию (см. упражнение 19.18). С другой стороны, если ограничить размер каждой проверки, самое большее k литералами, то обучающий алгоритм сможет успешно выполнять обобщение на основе небольшого количества примеров. Будем использовать нотацию k-DL (здесь DL — *Decision List*, список решений) для списков решений с не более чем k литералами. Пример, приведенный на рис. 19.10, относится к классу 2-DL. Можно легко показать (упражнение 19.18), что класс k-DL включает в качестве подмножества ►**класс** k-DT (здесь DT — *Decision Tree*, дерево решений), представляющий множество всех деревьев решений глубиной не более k. Важно помнить, что конкретный язык, к которому применяется обозначение k-DL, зависит от атрибутов, используемых для описания примеров. Ниже обозначение k-DL(n) будет использоваться для списков k-DL, в которых используется n булевых атрибутов.

Первая задача состоит в том, чтобы показать, что класс k-DL обучаем, т.е. что любая функция в классе k-DL может быть точно аппроксимирована после обучения на приемлемом количестве примеров. Для этого необходимо вычислить количество гипотез, которые могут быть представлены в этом классе. Обозначим множество конъюнкций самое большее k литералов, в которых используется n атрибутов, как $Conj(n, k)$. Поскольку любой список решений состоит из проверок, а каждая проверка может быть связана с результатом *Да* или *Нет* или же отсутствовать в списке решений, то может существовать не более $3^{|Conj(n,k)|}$ различных множеств проверок. Каждое из этих множеств проверок может быть задано в любом порядке, поэтому справедливо следующее соотношение:

$$|k\text{-DL}(n)| \leq 3^c c! \quad \text{где} \quad c = |Conj(n,k)|.$$

Количество конъюнкций из не более чем k литералов с n атрибутами может быть определено как

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k).$$

Следовательно, после некоторых преобразований можно получить следующее:

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}.$$

Это соотношение можно подставить в уравнение (19.1), чтобы показать, что количество примеров, необходимое для ВПК-обучения некоторой функции k-DL(n), определяется полиномиальной зависимостью от n:

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right).$$

Таким образом, любой алгоритм, возвращающий согласованный список решений, обеспечит ВПК-обучение некоторой функции k-DL с использованием приемлемого количества примеров при небольших значениях k.

Следующая задача состоит в том, чтобы найти эффективный алгоритм, возвращающий согласованный список решений. Будем использовать жадный алгоритм под названием "DECISION-LIST-LEARNING", который раз за разом находит проверку, точно согласующуюся с некоторым подмножеством обучающего набора. Найдя такую проверку, алгоритм добавляет ее к создаваемому списку решений и удаляет соответствующие примеры из обучающего набора. Далее алгоритм формирует оставшуюся часть списка решений, используя лишь оставшиеся примеры. Этот цикл повторяется до тех пор, пока в обучающем наборе не останется ни одного примера. Данный алгоритм приведен на рис. 19.11.

---

function DECISION-LIST-LEARNING(*examples*) **returns** список решений или индикатор отказа *failure*

if обучающее множество *examples* пусто **then return** тривиальный список решений *Нет*
$t \leftarrow$ проверка, соответствующая непустому подмножеству $examples_t$ из обучающего набора *examples*, — такая, что все члены $examples_t$ являются положительными или все они являются отрицательными
if таких проверок t больше не существует **then return** *failure*
if все примеры в $examples_t$ положильны **then** $o \leftarrow$ *Да* **else** $o \leftarrow$ *Нет*
return список решений с исходной проверкой t, результатом o и оставшимися проверками, определяемыми выражением DECISION-LIST-LEARNING($examples - examples_t$)

Рис. 19.11. Алгоритм построения списков решений

В этом алгоритме не определен метод выбора следующей проверки для добавления к списку решений. Хотя формально приведенные выше результаты не зависят от метода выбора, кажется разумным отдавать предпочтение небольшим проверкам, согласующимся с большими множествами однородно классифицируемых примеров, что позволит сделать общий список решений максимально компактным. Простейшая стратегия достижения этой цели состоит в том, чтобы искать наименьшую проверку t, согласующуюся с любым однородно классифицируемым подмножеством, независимо от размера этого подмножества. Как показано на рис. 19.12, даже такой простейший подход действует вполне успешно. Для конкретной задачи, представленной на этом рисунке, дерево решений обучается немного быстрее, чем список решений, но для дерева процесс характеризуется большими колебаниями. Оба метода после обработки 100 обучающих примеров достигают точности более 90%.

Рис. 19.12. Кривая обучения по данным задачи об ожидании в ресторане при использовании алгоритма DECISION-LIST-LEARNING. Для сравнения приведена кривая, соответствующая алгоритму DECISION-TREE-LEARNING, — деревья решений немного лучше справляются с этой конкретной задачей

19.6. Линейная регрессия и классификация

Теперь перейдем от деревьев и списков решений к другому пространству гипотез, используемому уже на протяжении сотен лет: классу ▶**линейных функций** с непрерывными значениями входных параметров. Начнем с простейшего случая: одномерной линейной регрессии, иначе называемой "подгонкой прямой линии". В разделе 19.6.3 рассматривается случай с несколькими переменными. В разделах 19.6.4 и 19.6.5 показано, как использовать линейные функции в качестве классификаторов за счет применения жестких и мягких порогов.

19.6.1. Одномерная линейная регрессия

Простая линейная функция (прямая линия) с входным параметром x и выходным значением y имеет вид $y = w_1 x + w_0$, где w_0 и w_1 являются вещественными коэффициентами, которые подвергаются обучению. Здесь буква w используется для обозначения коэффициентов потому, что они рассматриваются как ▶**весовые коэффициенты** (*weights*), — значение y изменяется посредством изменения относительного веса одного или другого терма. Определим $\mathbf{w}$ как вектор весов (w_0, w_1), что позволит записать гипотезу в виде линейной функции с этими весовыми коэффициентами как

$$h_{\mathbf{w}}(x) = w_1 x + w_0.$$

На рис. 19.13, *а* приведен пример обучающего набора из n точек в плоскости x, y, где каждая точка представляет размер площади в квадратных футах и цену дома, выставленного на продажу. Задача нахождения гипотезы h_w, которая наилучшим образом соответствует этим данным, называется ▶**линейной регрессией**. Чтобы подогнать прямую линию к данным, все, что нужно сделать, — это найти такие значения весов (w_0, w_1), которые минимизируют эмпирические потери $Loss(h_w)$. Согласно традиции (восходящий еще к Гауссу[6]), для этого используется квадратичная функция потерь L_2, суммируемая по всем примерам в обучающем наборе:

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2.$$

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - w_1 x_j + w_0))^2 = 0 \quad \text{и} \quad \frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0. \tag{19.2}$$

Эти уравнения имеют уникальное решение:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N. \tag{19.3}$$

Например, на рис. 19.13, *а* оптимальное решение имеет вид $w_1 = 0{,}232$; $w_0 = 246$; линия с этими весами показана на рисунке пунктиром.

Многие формы обучения включают в себя корректировку весов с целью минимизации потерь, поэтому будет полезно мысленно представить картину того, что происходит в ▶**пространстве весов** — пространстве, определяемом всеми возможными настройками весовых коэффициентов. Для одномерной линейной регрессии весовое пространство, определяемое весами w_0 и w_1, является двухмерным, поэтому можно построить график потерь как функцию от w_0 и w_1 на трехмерном графике (см. рис. 19.13, *б*). Можно видеть, что функция потерь является **выпуклой** в том смысле, как это было определено в конце раздела 4.2. Эта характеристика будет истинной для *любой* задачи линейной регрессии с функцией потерь L_2, и это означает, что локальные минимумы отсутствуют. В некотором смысле это "конец истории" для линейных моделей, — если требуется подогнать линию к данным, достаточно просто применить уравнение (19.3).[7]

---

[6] Гаусс показал, что если значения y_j имеют нормально распределенный шум, то наиболее вероятные значения w_1 и w_0 можно получить с использованием потерь L_2, сводя к минимуму сумму квадратов ошибок. (Если значения имеют шум, подчиняющийся распределению Лапласа (двойная экспонента), то более подходящими будут потери L_1.)

[7] С некоторыми оговорками: функция потерь L_2 будет уместна только тогда, когда независимая переменная x характеризуется нормально распределенным шумом, все результаты отвечают предположению о стационарности и т.д.

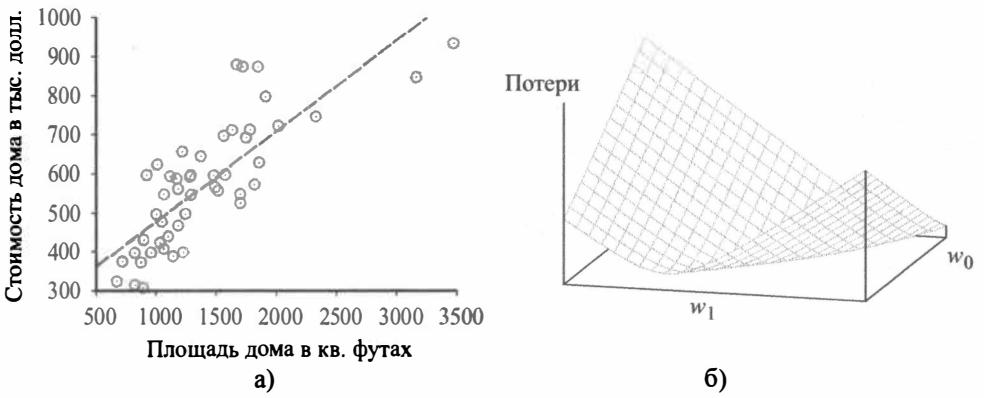


Рис. 19.13. а) Точки данных цены в сравнении с площадью домов, выставленных на продажу в Беркли, Калифорния, в июле 2009 года, вместе с гипотезой линейной функции, минимизирующей квадратичную функцию потерь: $y = 0{,}232x + 246$. **б)** График функции потерь $\Sigma_j(y_j - w_1x_j + w_0)^2$ для различных значений w_0, w_1. Обратите внимание, что функция потерь является выпуклой, с единственным глобальным минимумом

19.6.2. Градиентный спуск

Одномерная линейная модель имеет приятное свойство — несложно найти оптимальное решение, для которого частичные производные являются нулевыми. Но так бывает не всегда, поэтому ниже представлен метод минимизации потерь, не требующий решения задачи поиска нулевых значений производных, который может быть применен к любой сколь угодно сложной функции потерь.

Как обсуждалось в разделе 4.2, поиск в непрерывном весовом пространстве можно проводить путем постепенного изменения параметров. В том разделе такой алгоритм назывался **восхождением к вершине**, но в данном случае нам необходимо свести к минимуму потери, а не максимизировать прибыль, поэтому мы воспользуемся термином ►**градиентный спуск**. Выбираем любую начальную точку в пространстве весов — здесь это точка на плоскости (w_0, w_1), — а затем вычисляем оценку градиента и немного смещаемся вниз в направлении самого крутого спуска, повторяя эти действия до тех пор, пока в весовом пространстве не будет достигнута точка с (локально) минимальными потерями. Алгоритм выглядит так.

w ← любая точка в пространстве параметров
while not сошлось **do**
  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w}) \tag{19.4}$$

Параметр α, который в разделе 4.2 назывался **размером шага**, в ситуации, когда предпринимаются попытки минимизировать потери в задаче обучения, обычно называют ▶**скоростью обучения**. Он может представлять собой фиксированную константу или может уменьшаться со временем по ходу процесса обучения.

В случае одномерной регрессии потери являются квадратичными, поэтому частная производная будет линейной. (Единственная формула, которую вам нужно знать, — это ▶**цепное правило**: $\partial g(f(x))/\partial x = g'(f(x))\partial f(x)/\partial x$, плюс тот факт, что $\frac{\partial}{\partial x}x^2 = 2x$ и $\frac{\partial}{\partial x}x = 1$.) Сначала выразим частные производные — наклоны — для простейшего случая только одного обучающего примера (x, y):

$$\frac{\partial}{\partial w_i}Loss(\mathbf{w}) = \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) = \\ = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)). \tag{19.5}$$

Применив эту формулу и к w_0, и к w_1, получим

$$\frac{\partial}{\partial w_0}Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)); \qquad \frac{\partial}{\partial w_1}Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x.$$

Подставив эти выражения в уравнение (19.4) и заменив 2 неопределенной скоростью обучения α, получим следующее правило обучения для весовых коэффициентов:

$$w_0 \leftarrow w_0 + \alpha(y - h_{\mathbf{w}}(x)); \quad w_1 \leftarrow w_1 + \alpha(y - h_{\mathbf{w}}(x)) \times x.$$

Эти обновления имеют интуитивно понятный смысл: если $h_{\mathbf{w}}(x) > y$ (т.е. отклик слишком велик), немного уменьшаем w_0 и также уменьшаем w_1, если стимул x был положительным, но если x был отрицательным, значение w_1 следует увеличить.

Приведенные выше уравнения относятся лишь к обучающему набору с единственным примером. Для обучающего набора с N примерами следует минимизировать сумму индивидуальных потерь для каждого примера. Производная суммы — это сумма производных, поэтому получим

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)); \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j.$$

Эти результаты составляют правило обучения методом ▶**серийного спуска по градиенту** для одномерной линейной регрессии (также называемого *детерминированным градиентным спуском*). Поверхность графика потерь выпуклая, а это означает, что в данном случае не существует локальных минимумов, в которых алгоритм может застрять, и сходимость к глобальному минимуму в данном случае гарантирована (если только не будет выбрано настолько большое значение α, что поиск будет перепрыгивать через искомый минимум). Однако этот процесс может быть очень медленным: на каждом этапе необходимо просуммировать результаты всех

N примеров обучающей выборки, и таких этапов может быть много. Проблема усугубляется, если N превосходит объем памяти процессора. Каждый этап расчетов, на котором рассматриваются все примеры обучающей выборки, называют ▶**эпохой**.

Более быстрый вариант этого метода называется ▶**стохастическим градиентным спуском** или ▶**SGD** — *stochastic gradient descent*. В этом случае на каждом этапе случайным образом выбирается небольшое количество обучающих примеров и выполняются обновления в соответствии с уравнением (19.5). В оригинальной версии метода **SGD** на каждом этапе из обучающего набора выбирался только один пример, но в настоящее время чаще используют вариант, когда выбирается ▶**мини-пакет** (*minibatch*) размером m примеров из обучающей выборки размером N. Предположим, что $N = 10\,000$ примеров и используется мини-пакет размером $m = 100$. Тогда на каждом этапе объем вычислений сокращается в 100 раз; но поскольку стандартная ошибка оценки среднего градиента пропорциональна квадратному корню из числа примеров, стандартная ошибка увеличивается только с множителем 10. Поэтому, даже если в этом случае до достижения сходимости потребуется выполнить в 10 раз больше этапов, метод SGD с мини-пакетами все же будет работать в 10 раз быстрее, чем SGD с полным пакетом.

Для некоторых архитектур ЦПУ или ГПУ можно выбрать такое значение m, которое позволит использовать преимущества параллельного выполнения векторных операций и выполнять этап с обработкой m примеров почти так же быстро, как этап с обработкой только одного примера. С этой точки зрения значение m можно рассматривать как гиперпараметр, который следует настраивать для каждой задачи машинного обучения отдельно.

Сходимость метода SGD с мини-пакетами строго не гарантируется — возможна ситуация практически бесконечных колебаний вокруг минимума. В разделе 19.6.4 показано, что уменьшение скорости обучения α по некоторому графику (как при методе имитации отжига) гарантирует схождение.

Метод SGD может быть полезным в динамическом окружении, когда новые данные поступают по одному прецеденту за раз и предположение о стационарности может нарушаться. (На самом деле метод SGD также известен как ▶**градиентный спуск в динамическом режиме**.) При хорошем выборе скорости обучения α модель будет медленно развиваться, запоминая кое-что из того, чему она научилась в прошлом, но также адаптируясь к изменениям, представленным новыми данными.

Метод SGD широко применяется для моделей, отличных от линейной регрессии, в частности для нейронных сетей. Даже когда поверхность потерь не является выпуклой, этот подход доказал свою эффективность при поиске хороших локальных минимумов, достаточно близких к глобальному минимуму.

19.6.3. Множественная линейная регрессия

Полученные результаты можно легко распространить на задачи ▶ **множественной линейной регрессии**, в которых каждый пример $\mathbf{x}_j$ представлен n-элементным вектором.[8] В подобных случаях пространство гипотез представляет собой множество функций вида

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \ldots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}.$$

Здесь терм w_0, свободный член, выделяется среди всех других. Эту ситуацию можно исправить посредством введения фиктивного входного атрибута $x_{j,0}$, который определяется как всегда равный 1. Тогда гипотеза h — это просто точечное произведение весовых коэффициентов и входного вектора (или, что эквивалентно, матричное произведение транспонированной матрицы весов и входного вектора):

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i}.$$

Наилучший вектор весов, $\mathbf{w}^*$, минимизирует квадратичные потери по всем примерам:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j).$$

На самом деле множественная линейная регрессия ненамного сложнее, чем одномерный случай, который только что был рассмотрен. Метод градиентного спуска позволяет достичь (единственного) минимума функции потерь. Соответствующее уравнение обновления для каждого из весов w_i имеет следующий вид:

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}. \qquad (19.6)$$

Также, применив инструменты линейной алгебры и векторного исчисления, можно найти аналитическое решение для такого $\mathbf{w}$, которое сводит потери к минимуму. Пусть $\mathbf{y}$ является вектором выходных значений для всех примеров обучающего множества, а $\mathbf{X}$ является ▶ **матрицей факторов**, т.е. матрицей входных значений, каждая строка которой представляет один n-мерный пример. В этом случае вектором откликов, т.е. предсказанных выходных значений, будет $\hat{\mathbf{y}} = \mathbf{Xw}$, а квадратичную функцию потерь по всему обучающему набору можно определить как

$$L(\mathbf{w}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{Xw} - \mathbf{y}\|^2.$$

[8] При необходимости краткое описание линейной алгебры можно найти в приложении А. Также обратите внимание, что здесь термин "множественная регрессия" используется для указания на то, что входные данные представляют собой вектор из нескольких значений, однако выходные данные представлены одной переменной. Для тех случаев, когда выходные данные тоже представляют собой вектор из нескольких переменных, мы будем использовать термин "многомерная регрессия". Однако другие авторы часто используют оба термина взаимозаменяемо.

Устанавливаем градиент на нуль:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0.$$

Выполнив перестановку, находим, что вектор весов минимальных потерь определяется как

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}. \tag{19.7}$$

Выражение $(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top$ называют ▶ **псевдообратной** матрицей факторов, а уравнение (19.7) — ▶ **нормальным уравнением**.

В случае одномерной линейной регрессии не было необходимости беспокоиться о переобучении. Однако при множественной линейной регрессии в многомерных пространствах вполне возможно, что некоторые размерности, которые на самом деле являются нерелевантными, случайно принимаются за полезные, что приводит к переобучению модели.

По этой причине, чтобы избежать переобучения, общим правилом является использование **регуляризации** множественных линейных функций. Напомним, что с помощью регуляризации минимизируются общая стоимость гипотезы с учетом как эмпирических потерь, так и сложности гипотезы:

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h).$$

Для линейных функций сложность можно определить как функцию весов. Рассмотрим семейство функций регуляризации:

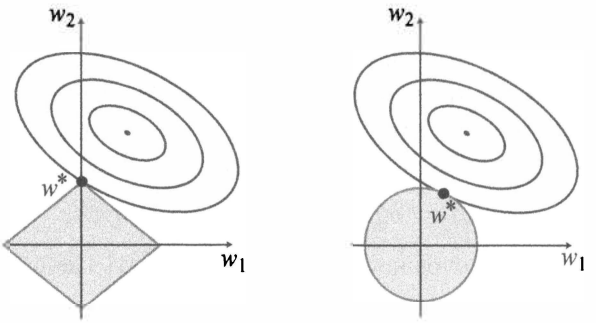
$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q.$$

Как и в случае функций потерь, при $q = 1$ имеет место регуляризация L_1,[9] минимизирующая сумму абсолютных значений, а при $q = 2$ — регуляризация L_2, минимизирующая сумму квадратов. Какой тип функции регуляризации следует выбрать? Это зависит от конкретной задачи, но регуляризация L_1 имеет важное преимущество: тенденцию к получению ▶ **разреженной модели** (*sparse model*). Имеется в виду, что в этом случае часто множество весов устанавливается равным нулю, фактически объявляя соответствующие атрибуты полностью нерелевантными, как это делается в алгоритме LEARN-DECISION-TREE (хотя и с помощью другого механизма). Гипотезы, в которых многие атрибуты отбрасываются, обычно понятнее для человека и с меньшей вероятностью приводят к переобучению.

На рис. 19.14 приведено интуитивно понятное объяснение того, почему регуляризация L_1 часто приводит к весам, равным нулю, а регуляризация L_2 — нет. Обратите внимание, что минимизация выражения $Loss(\mathbf{w}) + \lambda Complexity(\mathbf{w})$

[9] Может вводить в заблуждение, что обозначения L_1 и L_2 используются как для функций потерь, так и для функций регуляризации. Следует понимать, что нет необходимости использовать одинаковый тип для обеих функций: можно использовать потери L_2 при регуляризации L_1 и наоборот.

эквивалентна минимизации *Loss*(**w**) при условии выполнения ограничения *Complexity*(w) ≤ *c* для некоторой константы *c*, связанной с λ. Далее, на рис. 19.14, *а* ромбовидная фигура представляет собой множество точек **w** в двухмерном пространстве весов, для которых сложность L_1 будет меньше *c*. Искомое решение должно находиться где-то в пределах этой фигуры. Концентрические овалы представляют контуры функции потерь с общим минимумом потерь в центре. Необходимо найти точку в пределах фигуры, которая будет ближе всего находиться к этому минимуму, — на рисунке видно, что для любого положения минимума потерь и его овальных контуров ближе всего к нему всегда будет одна из угловых точек фигуры. При этом очевидно, что вершина любого угла фигуры — это точка, имеющая нулевое значение одного из измерений.



а) б)

Рис. 19.14. Наглядное пояснение, почему регуляризация L_1 имеет тенденцию к созданию разреженной модели. **а)** При регуляризации L_1 (ромбическая фигура) минимально достижимые потери $\mathbf{w}^*$ (пересечение с концентрическими контурами) чаще будут находиться на оси, а это означает, что соответствующий весовой коэффициент равен нулю. **б)** При регуляризации L_2 (круг) минимальные потери $\mathbf{w}^*$ с одинаковой вероятностью могут находиться в любой точке окружности, не давая предпочтения точкам на осях с нулевыми весовыми коэффициентами

На рис. 19.14, *б* приведена аналогичная схема для меры сложности L_2, которая в этом случае представлена кругом, а не ромбом. Здесь можно видеть, что в общем случае нет причин для преимущественного нахождения пересечения на одной из осей, — следовательно, регуляризация L_2 не обладает тенденцией к предпочтению нулевых весовых коэффициентов. В результате количество примеров, необходимых для нахождения хорошей гипотезы *h*, линейно относительно количества несущественных признаков для регуляризации L_2, но является логарифмическим при регуляризации L_1. Эмпирические данные по многим задачам подтверждают этот анализ.

Еще одна особенность, которую стоит отметить, состоит в том, что регуляризация L_1 серьезно относится к осям размерностей, тогда как регуляризация L_2 рассматривает их как произвольные. Функция L_2 — сферическая, что делает ее инвариантной к вращению: представьте множество точек на плоскости, заданных их координатами x и y. Теперь представьте, что координатные оси на этой плоскости были повернуты на 45°, — получится уже другое множество значений координат (x', y'), представляющих те же самые точки. Если применяется регуляризация L_2, то и до, и после поворота вы получите одну и ту же точку на плоскости в качестве ответа (хотя эта точка будет представлена новыми значениями координат (x', y')). Такой вариант будет наиболее подходящим, если выбор осей координат на самом деле является произвольным, когда совершенно неважно, являются ли два используемых измерения направлениями на север и на восток или на северо-восток и юго-восток. В случае регуляризации L_1 после поворота осей полученный ответ будет иным, так как функция L_1 не является инвариантной к вращению. Она уместна, когда оси размерностей не являются взаимозаменяемыми: нет смысла поворачивать "количество ванных комнат" на 45° относительно "размера участка земли".

19.6.4. Линейные классификаторы с жестким порогом

Линейные функции могут использоваться для классификации точно так же, как и для регрессии. Например, на рис. 19.15, *а* приведены точки данных двух классов: представляющие землетрясения (представляют интерес для сейсмологов) и подземные ядерные взрывы (представляют интерес для специалистов по контролю за вооружениями). Каждая точка задается двумя входными значениями, x_1 и x_2, представляющими величину объемных и поверхностных волн соответственно, вычисленную на основании поступившего сейсмического сигнала. При наличии этого обучающего набора данных задача классификации состоит в изучении гипотезы h, которая будет принимать новые точки данных (x_1, x_2) и возвращать либо значение 0 для землетрясений, либо значения 1 для ядерных взрывов.

►**Граница решения** — это линия (или поверхность, в более высоких измерениях), разделяющая два класса. На рис. 19.15, *а* граница решения является прямой линией. Линейную границу решения называют ►**линейным разделителем**, а данные, которые допускают такое разделение, называют ►**линейно разделимыми**. Линейный разделитель в этом случае определяется как

$$x_2 = 1{,}7x_1 - 4{,}9 \quad \text{или} \quad -4{,}9 + 1{,}7x_1 - x_2 = 0.$$

Точки ядерных взрывов, которые следует классифицировать выходным значением 1, находятся ниже и правее этой линии, — они являются точками, для которых $-4{,}9 + 1{,}7x_1 - x_2 > 0$, тогда как для точек землетрясений $-4{,}9 + 1{,}7x_1 - x_2 < 0$. Можно упростить работу с этим уравнением, переписав его в форме точечного векторного произведения: при $x_0 = 1$ мы получим

$$-4{,}9x_0 + 1{,}7x_1 + x_2 = 0,$$

и теперь можно определить вектор весов как

$$\mathbf{w} = \langle -4{,}9;\ 1{,}7;\ -1 \rangle$$

и записать гипотезу классификации в виде

$$h_{\mathbf{w}}(\mathbf{x}) = 1 \quad \text{если} \quad \mathbf{w} \cdot \mathbf{x} \geq 0 \quad \text{и} \quad = 0 \text{ в противном случае.}$$

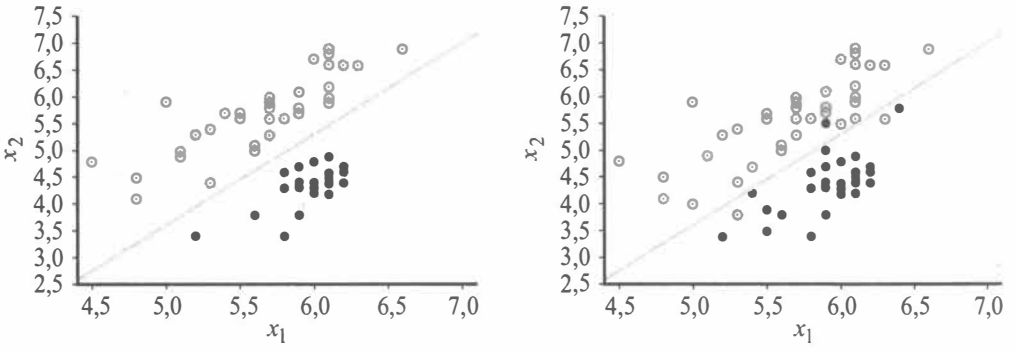


Рис. 19.15. а) Точки данных, представляющие два сейсмических параметра: величины объемной волны x_1 и поверхностной волны x_2 для землетрясений (светлые окружности) и ядерных взрывов (темные точки), произошедших в период с 1982 по 1990 год в Азии и на Среднем Востоке (Кебиси и др. [1211], 1998). Дополнительно приведена граница решения между двумя классами. **б)** Тот же обучающий набор с дополнительными точками данных. Точки, представляющие землетрясения и ядерные взрывы, уже не являются линейно разделимыми

В качестве альтернативы можно понимать функцию h как результат пропуска линейной функции $\mathbf{w} \cdot \mathbf{x}$ через ▶ **пороговую функцию** *Threshold*:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathit{Threshold}(\mathbf{w} \cdot \mathbf{x}),$$
$$\text{где } \mathit{Threshold}(z) = 1 \text{ если } z \geq 0 \text{ и } = 0 \text{ в противном случае.}$$

(Вид этой пороговой функции показан ниже, на рис. 19.17, *а*.)

Теперь, когда гипотеза $h_{\mathbf{w}}(\mathbf{x})$ приобрела четко определенную математическую форму, можно обратиться к задаче выбора вектора весов $\mathbf{w}$, минимизирующего потери. В разделах 19.6.1 и 19.6.3 эта задача решалась как в закрытой форме (установкой градиента на нуль и решением уравнения для весов), так и градиентным спуском в пространстве весов. Здесь у нас нет возможности применить ни один из этих способов, поскольку градиент в весовом пространстве является нулевым почти везде, за исключением тех точек, где $(\mathbf{w} \cdot \mathbf{x}) = 0$, а в этих точках градиент не определен.

Однако существует простое правило обновления веса, которое сходится к решению — т.е. к линейному разделителю, классифицирующему данные идеально, — при условии, что эти данные являются линейно разделимыми. Для единственного примера $(\mathbf{x}, y)$ имеем

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i, \tag{19.8}$$

что по существу идентично уравнению (19.6) — правилу обновления для линейной регрессии! Это правило называется ▶**правилом обучения перцептрона** по причинам, которые станут понятны в главе 21. Однако, поскольку рассматривается задача классификации 0/1, наше поведение будет несколько иным. Как истинным значением y, так и выходным значением гипотезы $h_{\mathbf{w}}(\mathbf{x})$ могут быть либо 0, либо 1, поэтому возможны три варианта действий.

- Если выходное значение является правильным (т.е. $y = h_{\mathbf{w}}(\mathbf{x})$), то веса не изменяются.
- Если истинное значение y равно 1, а $h_{\mathbf{w}}(\mathbf{x}) = 0$, то значение весового коэффициента w_i следует *увеличить*, если соответствующее входное значение x_i является положительным, и *уменьшить*, если x_i является отрицательным. Это имеет смысл, потому что требуется сделать значение $\mathbf{w} \cdot \mathbf{x}$ бо́льшим настолько, чтобы выходное значение $h_{\mathbf{w}}(\mathbf{x})$ стало равным 1.
- Если истинное значение y равно 0, а $h_{\mathbf{w}}(\mathbf{x}) = 1$, то значение весового коэффициента w_i следует *уменьшить*, если соответствующее входное значение x_i является положительным, и *увеличить*, если x_i является отрицательным. Это имеет смысл, потому что требуется сделать значение $\mathbf{w} \cdot \mathbf{x}$ меньшим настолько, чтобы выходное значение $h_{\mathbf{w}}(\mathbf{x})$ стало равным 0.

Типичное правило обучения предусматривает применение по одному примеру за раз и выбор этих примеров случайным образом (как при стохастическом градиентном спуске). На рис. 19.16, *а* показана ▶**кривая обучения** (*training curve*, чаще *learning curve*) для подобного правила обучения, примененного к данным о землетрясениях и взрывах, представленных на рис. 19.15, *а*. Кривая обучения характеризует производительность классификатора на фиксированном обучающем наборе по мере выполнения обучающего процесса по одному обновлению за раз на данном обучающем наборе. Кривая показывает, что правило обновления сходится к линейному разделителю с нулевой ошибкой. Процесс "схождения" не слишком гладок, но работает всегда. Этот конкретный прогон потребовал выполнения 657 этапов до момента схождения для обучающего набора данных с 63 примерами, поэтому каждый пример в этом случае был представлен системе в среднем около 10 раз. Как правило, различия между отдельными прогонами достаточно велики.

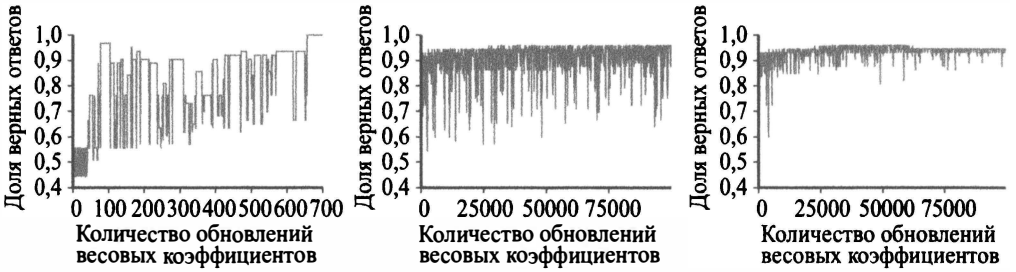


Рис. 19.16. а) График общей точности на обучающем наборе в зависимости от количества итераций по нему для правила обучения перцептрона на основании данных о землетрясениях и взрывах, приведенных на рис. 19.15, *а*. **б)** Такой же график для зашумленных неразделимых данных, приведенных на рис. 19.15, *б*; обратите внимание на изменение масштаба по оси *x*. **в)** Тот же случай, что и на рис. 19.15 *б*, но при графике скорости обучения $\alpha(t) = 1000/(1000 + t)$

Выше уже говорилось, что правило обучения перцептрона сходится к идеальному линейному разделителю, когда точки данных линейно разделимы, но что если это требование не выполняется? Подобная ситуация весьма распространена в реальном мире. Например, на рис. 19.15, *б* к прежним данным добавлены новые, не включенные Кебиси и соавторами, когда они отбирали примеры, приведенные на рис. 19.15, *а*. На рис. 19.16, *б* показано, что на этих данных правило обучения перцептрона не сходится даже после 10 000 этапов: несмотря на то что решение с минимально возможной ошибкой (три ошибки) было неоднократно найдено, алгоритм вновь и вновь возвращается к процедуре изменения весов. В общем случае правило обучения перцептрона может не обеспечить схождения к стабильному решению при фиксированной скорости обучения α, но если значение α уменьшается как $O(1/t)$, где t является номером итераций, то правило может демонстрировать сходимость к решению с наименьшей ошибкой, когда примеры представляются в случайном порядке.[10] Кроме того, можно показать, что задача нахождения решения с минимальной ошибкой является NP-трудной, поэтому можно ожидать, что для достижения схождения потребуется большое количество представлений примеров. На рис. 19.16, *в* показано, что для процесса обучения при графике изменения скорости обучения $\alpha(t) = 1000/(1000 + t)$ сходимость все еще остается несовершенной и после 100 000 итераций, но это все же намного лучше, чем при фиксированном значении α.

[10] Технически требуется, чтобы $\sum_{t=1}^{\infty} \alpha(t) = \infty$ и $\sum_{t=1}^{\infty} \alpha^2(t) < \infty$. Скорость обучения $\alpha(t) = O(1/t)$ удовлетворяет этим условиям. Часто также используется $c/(c + t)$ для некоторой довольно большой константы c.

19.6.5. Линейная классификация с логистической регрессией

Выше уже было показано, что пропуск выхода линейной функции через пороговую функцию (рис. 19.17, *а*) позволяет создать линейный классификатор, однако жесткий характер пороговой функции вызывает некоторые проблемы: гипотеза $h_{\mathbf{w}}(\mathbf{x})$ является недифференцируемой и фактически разрывной функцией от значений ее входов и весовых коэффициентов. Это делает обучение по правилу перцептрона весьма непредсказуемым занятием. Более того, линейный классификатор всегда выдает категорический прогноз — это 1 или 0 даже для примеров, которые очень близки к границе решения. Было бы лучше, если бы он мог классифицировать одни примеры как четкие 0 или 1, а другие — как не вполне ясные пограничные случаи.

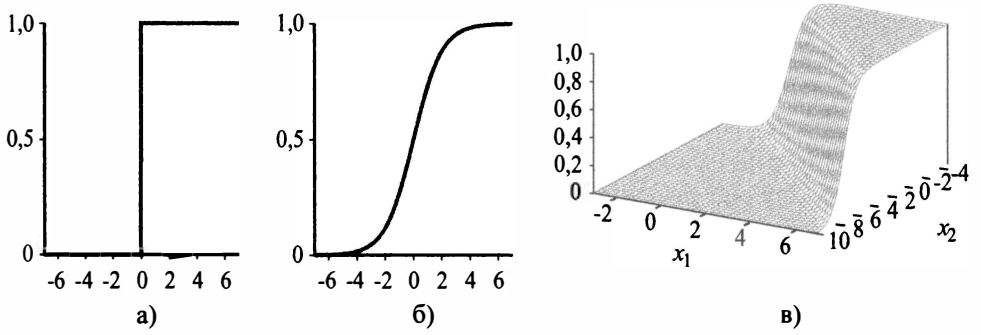


Рис. 19.17. а) Жесткая пороговая функция $Threshold(z)$ с выходными значениями 0/1. Обратите внимание, что эта функция недифференцируема при $z = 0$. **б)** Логистическая функция $Logistic(z) = \frac{1}{1+e^{-z}}$, также известная как *сигмоида*. **в)** График логистической регрессии гипотезы $h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})$ для данных, приведенных на рис. 19.15, *б*

Все эти проблемы можно в значительной степени устранить посредством смягчения пороговой функции — аппроксимацией жесткого порога непрерывной дифференцируемой функцией. В главе 13 (раздел 13.2.3) уже были представлены две функции, которые выглядят, как мягкие пороги: интеграл от стандартного нормального распределения (используется в пробит-модели) и логистической функции (используются в логит-модели). Хотя эти две функции очень схожи по своей форме, логистическая функция

$$Logistic(z) = \frac{1}{1+e^{-z}}$$

имеет более удобные математические свойства. Эта функция показана на рис. 19.17, *б*. Если пороговую функцию заменить логистической, то получится следующее уравнение:

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.$$

Пример такой гипотезы для задачи о землетрясениях и ядерных взрывах с двумя входными атрибутами приведен на рис. 19.17, *в*. Обратите внимание, что выходные данные, представляющие собой числа в диапазоне от 0 до 1, можно интерпретировать как *вероятность* принадлежности к классу с меткой 1. Гипотеза формирует мягкую границу в пространстве входных значений и выдает вероятность 0,5 для любых входных данных в центре граничной области, постепенно приближаясь к 0 или 1 по мере удаления от границы.

Процесс подбора весов этой модели с целью минимизации потерь в наборе данных называется ▶**логистической регрессией**. Для этой модели не существует простого решения в замкнутой форме, позволяющего аналитически найти оптимальное значение **w**, но вычисления для метода градиентного спуска достаточно просты. Поскольку гипотезы в такой модели больше не возвращают только 0 или 1, будем использовать функцию потерь L_2. Кроме того, чтобы сохранить читабельность формул, для обозначения логистической функции далее будем использовать нотацию g, где g' — ее производная.

Для одного обучающего примера $(\mathbf{x}, y)$ вывод градиента будет таким же, как для линейной регрессии (уравнение (19.5)), до момента, когда в уравнение подставляется фактическая форма гипотезы h. (Для этого вывода нам вновь потребуется цепное правило.) Итак, мы имеем

$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 = \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) = \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x} = \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i.
\end{aligned}$$

Поскольку производная g' логистической функций удовлетворяет уравнению $g'(z) = g(z)(1 - g(z))$, имеем

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

и поэтому процедура обновления веса для минимизации потерь делает шаг в направлении разности между входным значением и прогнозом, $(y - h_{\mathbf{w}}(\mathbf{x}))$, а длина этого шага зависит от константы α и производной g':

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i. \qquad (19.9)$$

Если повторить представленные на рис. 19.16 эксперименты, но с использованием логистической регрессии вместо линейного порогового классификатора, получим результаты, приведенные на рис. 19.18. Рис. 19.18, *а*, — это вариант с линейно разделяемыми данными: логистическая регрессия сходится несколько медленнее, но ведет себя гораздо более предсказуемо. На рис. 19.18, *б* и *в*, где данные являются зашумленными и неразделяемыми, логистическая регрессия сходится гораздо быстрее и надежнее. Эти преимущества, как правило, сохраняются и в реальных приложениях, поэтому логистическая регрессия стала одним из наиболее популярных методов классификации для задач в области медицины, маркетинга, анализа результатов опросов, оценки кредитоспособности, здравоохранения и других приложений.

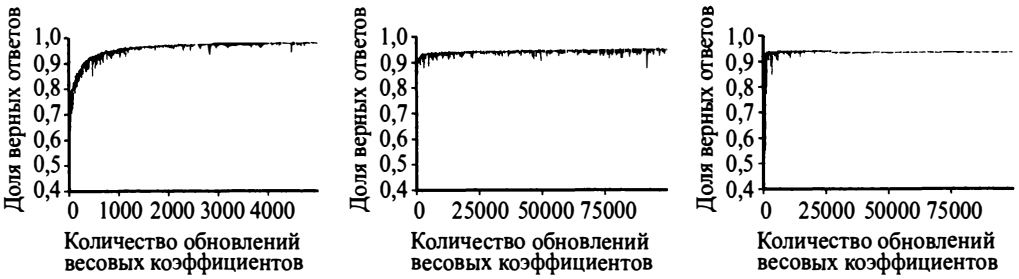


Рис. 19.18. Повторение экспериментов, показанных ранее, на рис. 19.16, но с использованием логистической регрессии. График *а* представляет 5000 итераций вместо 700, тогда как графики *б* и *в* приведены в том же масштабе

19.7. Непараметрические модели

В линейной регрессии обучающий набор данных используется для оценки фиксированного множества параметров $\mathbf{w}$. На этом основании определяется гипотеза $h_{\mathbf{w}}(\mathbf{x})$, после чего обучающий набор данных больше не нужен, поскольку все они уже обобщены в $\mathbf{w}$. Модель обучения, в которой данные обобщаются с использованием множества параметров фиксированного размера (независимо от количества обучающих примеров), называется ▶ **параметрической моделью**.

Если наборы данных невелики, имеет смысл наложить на допустимые гипотезы строгие ограничения, чтобы избежать переобучения. Но когда доступны миллионы или миллиарды примеров, на которых можно проводить обучение, кажется лучшей идеей позволить этим данным говорить самим за себя, а не заставлять их проявлять свои свойства через крошечный вектор параметров. Если данные говорят о том, что правильным ответом является функция с весьма извилистым

графиком, не следует ограничиваться подбором линейных или слегка волнистых функций.

Модель, которую нельзя охарактеризовать посредством ограниченного множества параметров, называется ▶**непараметрической моделью**. Например, кусочно-линейная функция на рис. 19.1 сохраняет все точки данных как часть модели. Методы обучения, которые позволяют реализовать такой подход, также были описаны как ▶**обучение на примерах** (*instance-based learning*) или **обучение на основе памяти** (*memory-based learning*). Самым простым методом обучения на примерах является ▶**поиск по таблице**: возьмите все обучающие примеры и поместите их в поисковую таблицу, а затем, в ответ на запрос о $h(\mathbf{x})$, проверьте, есть ли $\mathbf{x}$ в таблице, и, если есть, верните соответствующий y.

Проблема этого метода заключается в том, что он плохо обобщает: если $\mathbf{x}$ отсутствует в таблице, у нас нет информации о вероятном значении.

19.7.1. Метод ближайших соседей

Поиск по таблице можно улучшить за счет внесения небольшого изменения: по запросу $\mathbf{x}_q$ вместо поиска примера, точно соответствующего $\mathbf{x}_q$, отыскивается k примеров, ближайших к $\mathbf{x}_q$. Этот подход называется ▶**методом k-ближайших соседей**. Мы будем использовать нотацию $NN(k, \mathbf{x}_q)$ для обозначения множества из k соседей, ближайших к $\mathbf{x}_q$.

Для выполнения классификации выполняется поиск множества соседей $NN(k, \mathbf{x}_q)$ и в нем выбирается наиболее общее выходное значение, например если $k = 3$ и выходными значениями являются (*Да*, *Нет*, *Да*), то результатом классификации будет значение *Да*. Чтобы при бинарной классификации избежать "ничьих", для k обычно выбирается нечетное значение.

Для выполнения регрессии можно взять среднее значение или медиану для k соседей либо решить задачу линейной регрессии на множестве соседних узлов. Кусочно-линейная функция, приведенная на рис. 19.1, позволяет решить (тривиальную) задачу линейной регрессии для двух точек данных справа и слева от $\mathbf{x}_q$. (Когда точки данных x_i расположены на одинаковом расстоянии, это будут два ближайших соседа.)

На рис. 19.19 показаны границы решений для задачи классификации методом k-ближайших соседей при $k = 1$ и $k = 5$ на множестве данных о землетрясениях и ядерных взрывах, представленном на рис. 19.15, *в*. Непараметрические методы также подвержены недообучению и переобучению, как и параметрические методы. В данном случае при использовании метода с 1 ближайшим соседом очевидно переобучение: слишком сильная реакция модели на выбросы для черных точек в верхнем правом углу и для белой точки (5,4; 3,7). Граница решения для варианта с 5 ближайшими соседями достаточно хороша, при большем значении k может начать проявляться недообучение. Как обычно, для выбора наилучшего значения k может использоваться перекрестная проверка.

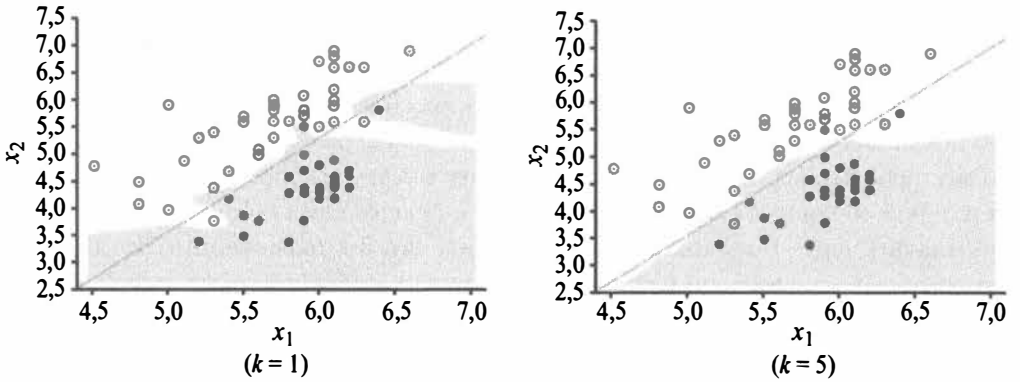


Рис. 19.19. а) Модель k-ближайших соседей представляет распределение данных класса ядерных взрывов для обучающего набора, приведенного на рис. 19.15, *в*, при $k = 1$. Переобучение очевидно. **б)** При $k = 5$ свидетельства переобучения для этого обучающего набора исчезают

Само слово "ближайших" подразумевает применение некоторой метрики расстояния. Каким же образом можно измерить расстояние от точки запроса $\mathbf{x}_q$ до точки примера $\mathbf{x}_j$? Как правило, подобные расстояния измеряются с помощью ►**расстояния** (метрики) **Минковского** или L^p нормы, определяемой как

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p}.$$

При $p = 2$ это евклидово расстояние, а при $p = 1$ — манхэттенское расстояние. При булевых значениях атрибутов количество атрибутов, которыми различаются две точки, называется ►**расстоянием Хемминга**. Евклидово расстояние чаще используется, если измеряемые размерности имеют схожие свойства, например это длина, ширина и высота деталей; а если они различны, например возраст, вес и пол пациента, то используется манхэттенское расстояние. Обратите внимание: когда для каждого измерения используются исходные, необработанные значения, то общее расстояние изменяется при изменении единицы измерения в любом из измерений. Иначе говоря, если для *высоты* единицу измерения "метр" изменить на футы, сохранив при этом неизменными единицы измерения для *длины* и *ширины*, то будет получено иное множество ближайших соседей. А как можно сравнивать разницу в возрасте с разницей в весе? Общим подходом является применение ►**нормализации** к результатам измерения в каждом измерении. Можно вычислить среднее значение μ_i и стандартное отклонение σ_i значений в каждом измерении, а затем масштабировать их так, что $x_{j,i}$ превращается в $(x_{j,i} - \mu_i)/\sigma_i$. Более сложная метрика, известная как ►**расстояние Махаланобиса**, учитывает ковариацию между измерениями.

В низкоразмерных пространствах с большим количеством данных метод ближайших соседей работает очень хорошо: весьма вероятно, что в любом случае будет достаточно точек данных поблизости, чтобы получить хороший ответ. Но с ростом числа измерений это становится проблемой: ближайшие соседи в многомерных пространствах обычно не очень близки! Рассмотрим работу метода k-ближайших соседей на наборе данных из N точек, равномерно распределенных во внутренней области n-мерного единичного гиперкуба. Определим k-окрестность некоторой точки как наименьший гиперкуб, содержащий k-ближайших соседей. Обозначим через ℓ среднюю длину стороны такой k-окрестности. Тогда объем окрестности (содержащей k точек) будет равен ℓ^n, а объем полного куба (содержащего N точек) будет равен 1. Следовательно, в среднем $\ell^n = k/N$. Извлекая корень n-степени из обеих сторон уравнения, получаем $\ell = (k/N)^{1/n}$.

Для конкретности пусть $k = 10$, а $N = 1\,000\,000$. При двух измерениях ($n = 2$; единичная область представляет собой квадрат), средняя k-окрестность будет иметь $\ell = 0{,}003$, очень небольшую часть стороны единичного квадрата, а при трех измерениях ℓ составит уже 2% от длины ребра единичного куба. Если увеличить размерность до 17, то ℓ будет равно целой половине длины ребра единичного гиперкуба, а при 200 измерениях это будет уже 94%. Данная проблема получила название ►**проклятие размерности**.

Есть и другой способ взглянуть на данную проблему: рассмотрим точки, которые попадают в тонкую оболочку, составляющую внешний 1% единичного гиперкуба. Это выбросы, и в общем случае будет трудно найти для них хорошее значение, потому что потребуется экстраполяция, а не интерполяция. При одном измерении эти выбросы составляют лишь 2% точек на единичной линии (те точки, в которых $x < 0{,}01$ или $x > 0{,}99$), но при 200 измерениях более 98% точек попадает в пределы этой тонкой оболочки, т.е. почти все имеющиеся точки являются выбросами. Пример того, как плохо метод ближайших соседей подходит для работы с выбросами, можно увидеть, если заглянуть вперед, на рис. 19.20, *б*.

Функция $NN(k, \mathbf{x}_q)$ концептуально является тривиальной: при заданном наборе из N примеров и запросе $\mathbf{x}_q$ выполнить итерацию по всем примерам, измеряя расстояние до $\mathbf{x}_q$ от каждого из них и сохраняя лучшие k результатов. Если нас вполне устроит реализация, время выполнения которой пропорционально $O(N)$, то это конец истории. Однако методы на основе анализа экземпляров предназначены для очень больших наборов данных, поэтому хотелось бы иметь что-то, что работает быстрее. В следующих двух разделах показано, как деревья и хеш-таблицы можно использовать для ускорения вычислений.

19.7.2. Поиск ближайших соседей с помощью k-d-деревьев

Сбалансированное бинарное дерево для данных с произвольным числом измерений называют ►***k*-d-деревом** или k-мерным деревом. Конструкция k-d-дерева подобна конструкции сбалансированного двоичного дерева. Начинаем с полного

множества примеров и в корневом узле делим его по i-му измерению путем проверки, является ли $x \leq m$, где m — медиана примеров по i-му измерению. В результате половина примеров будет находиться в левой ветви дерева, а другая половина — в правой. Затем рекурсивно строятся деревья для левого и правого множеств примеров, — процесс останавливается, когда останется меньше двух примеров. В отношении выбора измерения для разделения примеров в каждом из узлов дерева можно просто выбирать измерение i mod n на уровне i всего дерева. (Обратите внимание, что, возможно, потребуется несколько раз выполнить разделение по любому заданному измерению по мере продвижения вниз по дереву.) Другой стратегией может быть выполнение каждого очередного разделения по измерению, которое имеет самый широкий спектр значений.

Точный поиск по k-d-дереву выполняется так же, как поиск в двоичном дереве (с небольшим осложнением, поскольку дополнительно требуется обращать внимание на то, по какому измерению следует выполнять тестирование в каждом их узлов). Однако поиск ближайших соседей в данном случае будет более сложным. По мере продвижения вниз по ветвям, разделяющим примеры на две половины, в некоторых случаях вторую половину примеров можно будет просто игнорировать. Но не всегда. Иногда та точка, поиск которой выполняется, окажется расположенной очень близко к границе разделения, в результате чего сама точка запроса может находиться слева от границы, а один или несколько из ее k-ближайших соседей в действительности будут располагаться справа от нее.

Необходимо учитывать эту возможность, вычисляя расстояние от точки запроса до границы раздела, а затем выполняя поиск по обеим ее сторонам, если с нужной стороны не удастся найти k примеров, находящихся ближе, чем на этом расстоянии. Из-за этой проблемы k-d-деревья будут эффективны только тогда, когда примеров гораздо больше, чем измерений, — предпочтительно не менее 2^n примеров. Таким образом, k-d-деревья будут хорошим выбором для задач с размерностью примерно до 10, когда доступны тысячи примеров, и размерностью до 20 — при миллионах примеров.

19.7.3. Локально-чувствительное хеширование

Потенциал хеш-таблиц настолько велик, что они способны обеспечить более быстрый поиск, чем бинарные деревья. Но как можно найти ближайших соседей с помощью хеш-таблицы, если хеш-коды полагаются на *точное* соответствие? Хеш-коды случайным образом распределяют значения по корзинам, но мы хотим, чтобы точки, расположенные поблизости, были сгруппированы в одной корзине, т.е. нам требуется ▶**локально-чувствительное хеширование** (*locality-sensitive hash* — ▶**LSH**).

Нельзя использовать хеши для точного решения задачи $NN(k, \mathbf{x}_q)$, но при умном использовании рандомизированных алгоритмов можно найти *приблизительное* решение. Сначала определяем задачу ▶**приближенно ближайших соседей**: при

заданных наборе данных точек примеров и точке запроса $\mathbf{x}_q$ найти точку (или точки) примера, которая с высокой вероятностью находится вблизи $\mathbf{x}_q$. Говоря точнее, выдвигается требование, чтобы, если существует такая точка $\mathbf{x}_j$, которая находится в пределах радиуса r от точки $\mathbf{x}_q$, то с высокой вероятностью алгоритм найдет точку $\mathbf{x}_{j'}$, которая находится на расстоянии cr от точки $\mathbf{x}_q$. Если в пределах радиуса r нет точек примеров, то алгоритму разрешается сообщить об ошибке. Значения c и "высокая вероятность" являются гиперпараметрами этого алгоритма.

Для решения задачи приближенно ближайших соседей необходима хеш-функция $g(\mathbf{x})$, обладающая таким свойством, что для любых двух точек $\mathbf{x}_j$ и $\mathbf{x}_{j'}$ вероятность того, что они имеют одинаковый хеш-код, мала, если расстояние между ними больше cr, и высока, если это расстояние меньше r. Для простоты будем рассматривать каждую точку как битовую строку. (Любые функции, которые не являются булевыми, могут быть закодированы в набор булевых функций.)

Здесь мы полагаемся на интуитивное предположение, что если две точки расположены близко друг к другу в n-мерном пространстве, то они обязательно будут близки и при проецировании вниз на одномерное пространство (линию). На практике эту линию всегда можно дискретизировать на сегменты — хеш-корзины — так, что с высокой вероятностью соседние точки будут спроецированы вниз в один и тот же сегмент. Точкам, которые находятся далеко друг от друга, будет свойственна тенденция проецироваться вниз в разные сегменты, но всегда будет несколько проекций, которые случайно спроецируют удаленные точки в один сегмент. Таким образом, хеш-козина для точки $\mathbf{x}_q$ будет содержать много точек (но не все), находящихся близко к $\mathbf{x}_q$, а также может содержать и несколько точек, находящихся далеко от нее.

Главная хитрость LSH состоит в создании *множества* случайных проекций и их комбинировании. Случайная проекция — это просто случайное подмножество строки битового представления. Выбирается ℓ разных случайных проекций, и создается ℓ хеш-таблиц, $g_1(\mathbf{x}),\ldots, g_\ell(\mathbf{x})$. Далее все имеющиеся примеры вводятся в каждую из созданных хеш-таблиц. Затем, когда задается точка запроса $\mathbf{x}_q$, извлекается множество точек из корзины $g_i(\mathbf{x}_q)$ в каждой хеш-таблице, и эти ℓ множеств объединяются в единое множество точек-кандидатов, C. Для каждой из точек в множестве C вычисляется фактическое расстояние до точки запроса $\mathbf{x}_q$ и возвращается k-ближайших из них. С высокой вероятностью каждая из точек, близких к $\mathbf{x}_q$, будет представлена как минимум в одной из корзин, и хотя некоторые отдаленные точки также будут в них присутствовать, их можно просто игнорировать. Для больших реальных задач, таких как поиск ближайших соседей в наборе данных из 13 млн веб-изображений с использованием 512 размерностей (Торральба и др. [2222], 2008), метод локально-чувствительного хеширования при поиске ближайших соседей позволяет ограничиться изучением только нескольких тысяч изображений из 13 млн, — тысячекратное ускорение в сравнении с исчерпывающим поиском или поиском по k-d-дереву.

19.7.4. Непараметрическая регрессия

Теперь рассмотрим непараметрические подходы к *регрессии*, а не к классификации. На рис. 19.20 представлены примеры для некоторых различных моделей. На рис. 19.20, *а* демонстрируется, пожалуй, самый простой метод из всех, неофициально известный как "соединение точек", а в теории — как "кусочно-линейная непараметрическая регрессия". В этой модели создается некоторая функция $h(x)$ и, когда вводится запрос x_q, рассматриваются примеры обучения непосредственно слева и справа от x_q с последующей интерполяцией между ними. При низком уровне шума этот тривиальный метод на самом деле оказывается не таким уж плохим, и по этой причине он является стандартной функцией программного обеспечения для создания диаграмм в электронных таблицах. Однако, когда данные зашумлены, результирующая функция становится "колючей", что сильно затрудняет обобщение.

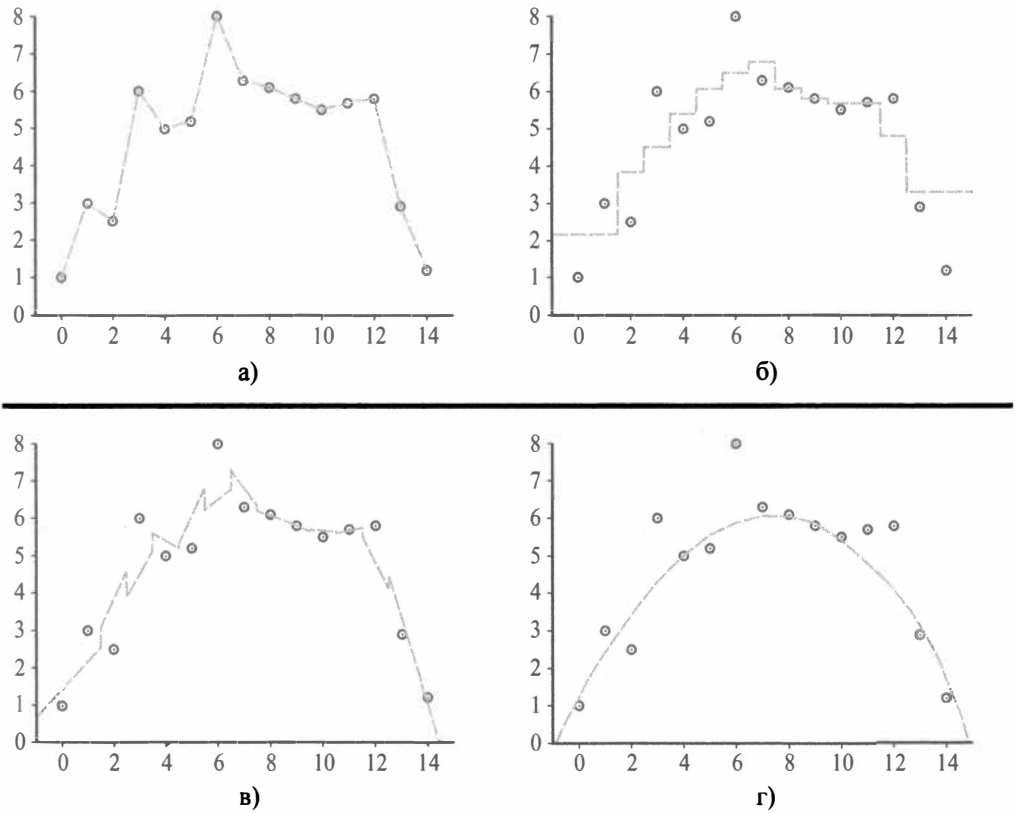


Рис. 19.20. Непараметрические регрессионные модели. **а)** Соединение точек. **б)** Усреднение по 3 ближайшим соседям. **в)** Линейная регрессия по 3 ближайшим соседям. **г)** Локально взвешенная регрессия с квадратичным ядром шириной 10

Метод ▶**регрессии по *k*-ближайшим соседям** является улучшением метода соединения точек. Вместо только двух примеров слева и справа от точки запроса x_q здесь используются ее *k*-ближайших соседей. (На рис. 19.20 используется $k = 3$.) При увеличении значения *k* отмечается тенденция к сглаживанию острых выбросов, хотя результирующая функция может иметь разрывы. На рис. 19.20 показаны две версии регрессии по k-ближайшим соседям. На рис. 19.20, *б* показан вариант усреднения по *k*-ближайшим соседям: функция $h(x)$ вычисляет среднее значение для *k* точек: $\Sigma y_j/k$. Обратите внимание, что для крайних точек вблизи $x = 0$ и $x = 14$ оценки оказались плохими, поскольку все свидетельства приходят с одной стороны (изнутри), но эта тенденция игнорируется. На рис. 19.20, *в* демонстрируется линейная регрессия по *k*-ближайшим соседям, выполняющая поиск наилучшей прямой, проходящей через *k* примеров. Такой подход позволяет лучше захватывать тенденции на выбросах, но график все еще остается прерывистым. Для обоих вариантов, *б* и *в*, осталось найти ответ на вопрос о выборе наилучшего значения *k*. Ответом, как обычно, является перекрестная проверка.

Метод ▶**локально взвешенной регрессии** (см. рис. 19.20, *г*) позволяет использовать преимущества нахождения ближайших соседей, полностью исключая возможность разрывов. Чтобы избежать появления разрывов в функции $h(x)$, необходимо исключить разрывы в множестве примеров, используемых для оценки $h(x)$. Основная идея локально взвешенной регрессии заключается в том, что для каждой точки запроса x_q близким к ней примерам придается большой вес, тем примерам, которые находятся дальше от нее, — меньший вес, а совсем далекие примеры вовсе не учитываются. Уменьшение веса с расстоянием обычно происходит постепенно, а не внезапно.

Решение о том, насколько большой вес присваивается каждому примеру, принимается с помощью функции, называемой ▶**ядром**, на вход которой подается расстояние между точкой запроса и примером. Функция ядра $\mathcal{K}$ является понижающей функцией в зависимости от расстояния (*distance*) с максимумом в 0, так что функция $\mathcal{K}(Distance(\mathbf{x}_j, \mathbf{x}_q))$ даст более высокий вес такому примеру $\mathbf{x}_j$, который будет ближе к точке запроса $\mathbf{x}_q$, для которой требуется выдать прогноз значения функции. Интеграл значений функции ядра по всему входному пространству для $\mathbf{x}$ должен быть конечным, и если принять его значение равным 1, то некоторые вычисления упростятся.

График на рис. 19.20, *г* был построен с использованием квадратичного ядра, $\mathcal{K}(d) = \max(0, 1 - (2|d|/w)^2)$ с шириной ядра $w = 10$. Часто используются и другие формы, такие как гауссианы. Как правило, ширина ядра имеет большее значение, чем точная форма: это такой гиперпараметр модели, который лучше выбирать путем перекрестной проверки. Если ядра слишком широкие, результатом будет недообучение модели, а если слишком узкие, модель будет переобучена. На рис. 19.20, *г* ширина ядра 10 позволяет получить более гладкую кривую, которая выглядит примерно правильной.

Понять, как выполняется локально взвешенная регрессия с ядрами, теперь должно быть просто. Для заданной точки запроса $\mathbf{x}_q$ необходимо решить следующую задачу взвешенной регрессии:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(Distance(\mathbf{x}_q, \mathbf{x}_j))(y_j - \mathbf{w} \cdot \mathbf{x}_j)^2,$$

где *Distance* — любая метрика расстояния из числа обсуждавшихся выше, при рассмотрении метода k-ближайших соседей. В этом случае ответом (прогнозом) будет $h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$.

Обратите внимание, что здесь требуется решить новую задачу регрессии для *каждой* точки запроса — вот что на практике означает быть *локальным.* (В методе обычной линейной регрессии глобальная задача регрессии решалась один раз, а затем одно и то же значение h_w использовалось для любой точки запроса.) Возможность смягчения этой дополнительной работы заключается в том, что каждую задачу регрессии решать будет проще, поскольку она включает только примеры с ненулевым весом — примеры, которые находятся в пределах ширины ядра запроса. Когда ширина ядра мала, это множество может состоять всего из нескольких точек.

Большинство непараметрических моделей имеют то преимущество, что для них легко выполнить перекрестную проверку по отдельным объектам без необходимости пересчитывать все. В модели k-ближайших соседей, например, при заданном тестовом примере $(\mathbf{x}, y)$ выборка k-ближайших соседей выполняется один раз, для них вычисляются потери на пример $L(y, h(\mathbf{x}))$ и записываются как результат перекрестной проверки по отдельным объектам для каждого примера, который не входит в число соседей. Затем делается выборка $k+1$ ближайших соседей и записываются отличающиеся результаты для исключения каждого из k соседей. При N примерах весь процесс характеризуется сложностью $O(k)$, а не $O(kN)$.

19.7.5. Метод опорных векторов

В самом начале 2000-х годов ▶**метод опорных векторов** (*support vector machine* — ▶**SVM**) был наиболее популярным вариантом при выборе класса моделей готовых систем обучения с учителем в тех случаях, когда отсутствуют какие-либо специальные предварительные знания о проблемной области. В настоящее время их место заняли сети глубокого обучения и случайные леса, но метод SVM по-прежнему обладает тремя привлекательными свойствами.

1. Метод SVM предусматривает построение ▶**оптимально разделяющего классификатора** (*maximum margin separator*) — границы решения с максимально возможным расстоянием до точек примеров, что способствует хорошему обобщению.
2. В методе SVM создается *линейная* разделяющая гиперплоскость, но имеется возможность встраивать данные в многомерное пространство большей размерности, используя так называемый ▶**ядерный трюк** (*kernel trick*). Часто

данные, которые не являются линейно разделимыми в исходном пространстве входных данных, легко разделяются в пространстве с большей размерностью.

3. Метод SVM является непараметрическим — разделяющая гиперплоскость определяется набором точек примеров, а не набором значений параметров. Но если для модели по методу *k*-ближайших соседей необходимо хранить все примеры, то SVM-модель поддерживает только те примеры, которые находятся ближе всего к разделяющей плоскости, — обычно это лишь небольшая постоянная, умноженная на число размерностей. Следовательно, метод SVM сочетает в себе преимущества непараметрических и параметрических моделей: модели обладают гибкостью, достаточной для представления сложных функций, но при этом они устойчивы к переобучению.

На рис. 19.21, *а* приведена задача двоичной классификации с тремя возможными границами решения, каждая из которых представляет собой линейный разделитель. Любая из них соответствует всем примерам, поэтому с точки зрения 0/1 потери все они будут одинаково хороши. Логистическая регрессия сможет найти некоторую разделительную линию, но ее точное расположение будет зависеть от *всех* точек задачи. Ключевая идея метода SVM состоит в том, что одни примеры являются более важными, чем другие, и если уделить им дополнительное внимание, то это может привести к лучшему обобщению.

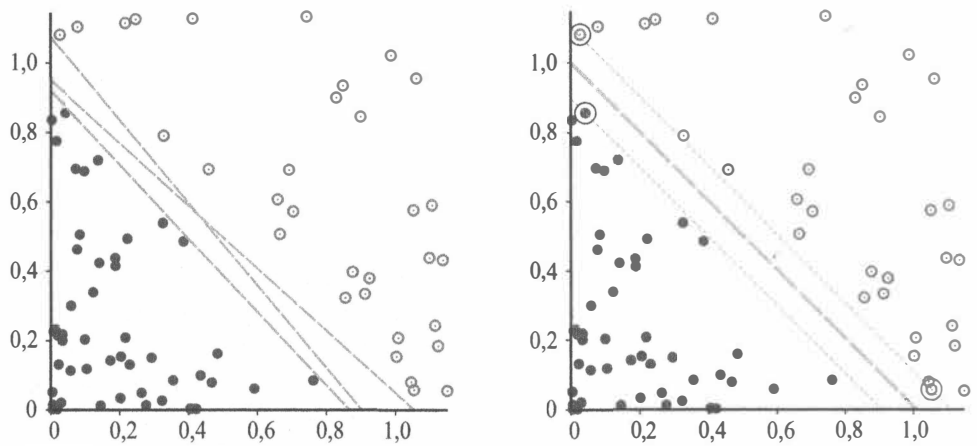


Рис. 19.21. Классификация по методу опорных векторов. **а)** Два класса точек (светлые кружки и сплошные темные точки) и три возможных линейных разделителя. **б)** Оптимально разделяющий классификатор (широкая линия) проходит в середине **зазора** (область между двумя тонкими пунктирными линиями). **Векторы поддержки** (точки, обведенные черными окружностями) являются примерами, ближайшими к классификатору, — в данном случае их три

Рассмотрим первый снизу из трех линейных разделителей на рис. 19.21, *а*. Он проходит очень близко к пяти примерам, представленным темными точками. Хотя он правильно классифицирует все имеющиеся примеры и, следовательно, сводит потери к минимуму, он все же вызывает некоторое беспокойство, так как довольно много примеров находятся очень близко к этой линии и может случиться, что новые примеры, относящиеся к данной категории, окажутся не с той стороны от этой линии.

Метод SVM позволяет решить эту проблему: вместо того чтобы минимизировать ожидаемые *эмпирические потери* на обучающих данных, в методе опорных векторов предпринимается попытка минимизировать ожидаемые потери при *обобщении*. Нельзя сказать, где окажутся еще неизвестные нам точки, но при вероятностном допущении, что они будут подчиняться тому же распределению, что и предыдущие примеры, теория вычислительного обучения (раздел 19.5) предлагает некоторые аргументы в пользу того, что можно минимизировать потери при обобщении, выбрав разделитель, находящийся как можно дальше от уже известных нам примеров. Такой разделитель, показанный на рис. 19.21, *б*, в методе SVM называют ▶**оптимально разделяющим классификатором** (*maximum margin separator*). Здесь ▶**зазор** (*margin*) представляет ширину области, ограниченной на рисунке тонкими пунктирными линиями, — это удвоенное расстояние от разделителя до ближайшей точки примера.

А как можно найти такой разделитель? Прежде чем обратиться к уравнениям, нужно сделать несколько замечаний о принятых обозначениях. Традиционно в теории метода SVM используется соглашение, что класс метки обозначается как +1 и −1 вместо +1 и 0, как было принято ранее в этой книге. Кроме того, хотя выше мы помещали свободный член в вектор весов $\mathbf{w}$ (и соответствующее фиктивное значение 1 — в $x_{j,0}$), в теории метода SVM так не делают — свободный член представлен здесь как отдельный параметр, b.

С учетом всего этого разделитель определяется как множество точек $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$. Можно было бы выполнить поиск в пространстве $\mathbf{w}$ и b методом градиентного спуска, чтобы найти значения параметров, максимизирующих зазор при правильной классификации всех примеров.

Однако было установлено, что существует и другой подход к решению этой проблемы. Здесь мы не будем углубляться в детали, а лишь укажем, что существует альтернативное представление, называемое двойственным, в котором оптимальное решение может быть найдено решением уравнения

$$\underset{\alpha}{\operatorname{argmax}} \sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j \cdot \mathbf{x}_k) \qquad (19.10)$$

с учетом ограничений $\alpha_j \geq 0$ и $\Sigma_j \alpha_j y_j = 0$. Это ▶**квадратичное программирование** оптимизационной задачи, для выполнения которого имеются хорошие пакеты программного обеспечения. Как только вектор α будет найден, можно будет вернуться

к **w** с помощью уравнения $\mathbf{w} = \Sigma j\, \alpha_j y_j x_j$ или остаться в двойственном представлении. Есть три важных свойства уравнения (19.10). Во-первых, это выражение является выпуклым и имеет единственный глобальный максимум, который может быть эффективно найден. Во-вторых, ➡ *данные входят в это выражение только в виде точечных произведений пар точек*. Это, второе, свойство будет справедливо и для уравнения самого разделителя: как только оптимальное α_j будет вычислено, это уравнение примет вид[11]

$$h(\mathbf{x}) = \text{sign}\left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x}_j) - b\right). \tag{19.11}$$

И в-третьих, веса α_j, связанные с каждой точкой данных, будут *нулевыми* для всех точек, кроме ►**опорных векторов** (*support vectors*), — точек, наиболее близких к разделителю. (Их называют "опорными" векторами, потому что они как бы "держат" разделительную плоскость.) Поскольку опорных векторов, как правило, гораздо меньше, чем обучающих примеров, метод SVM имеет определенные преимущества перед параметрическими моделями.

Но что если множество обучающих примеров не является линейно разделимым? На рис. 19.22, *а* представлено входное пространство, образованное атрибутами $\mathbf{x} = (x_1, x_2)$, с положительными примерами ($y = +1$) внутри круговой области и отрицательными примерами ($y = -1$) — вне ее. Понятно, что для подобной задачи линейного разделителя не существует. Теперь предположим, что входные данные были переопределены, т.е. что каждый входной вектор **x** был отображен на новый вектор признаков значений $F(\mathbf{x})$. В частности, в новом векторе будут использоваться три признака:

$$f_1 = x_1^2, \quad f_2 = x_2^2, \quad f_3 = \sqrt{2}\, x_1 x_2. \tag{19.12}$$

Вскоре будет показано, откуда они взялись, но сейчас просто посмотрим на то, что в результате произошло. На рис. 19.22, *б* приведены те же данные, но уже в новом, трехмерном пространстве, определенном с помощью трех указанных выше функций: в этом пространстве данные являются *линейно разделимыми*! Это явление на самом деле довольно общее: если данные отобразить на пространство достаточно высокой размерности, то они почти всегда будут линейно разделимыми, — если смотреть на множество точек с достаточного числа направлений, можно найти способ заставить их выстроиться в линию. В данном случае использовались три размерности,[12] а в упражнении 19.21 вам будет предложено показать, что четырех измерений будет достаточно для линейного разделения окружности, распо-

[11] Функция sign(x) возвращает +1 для положительного x и −1 — для отрицательного x.

[12] Внимательный читатель может заметить, что при отображении достаточно было бы использовать только функции f_1 и f_2, но трехмерное отображение лучше иллюстрирует обсуждаемую идею.

ложенной в любом месте плоскости координат (а не только в начале координат), а пяти размеров будет достаточно, чтобы найти линейное разделение для любого эллипса. В общем (за исключением некоторых особых случаев), если имеется N точек данных, они всегда будут линейно разделимыми в пространстве с $N-1$ измерениями или более (упражнение 19.24).

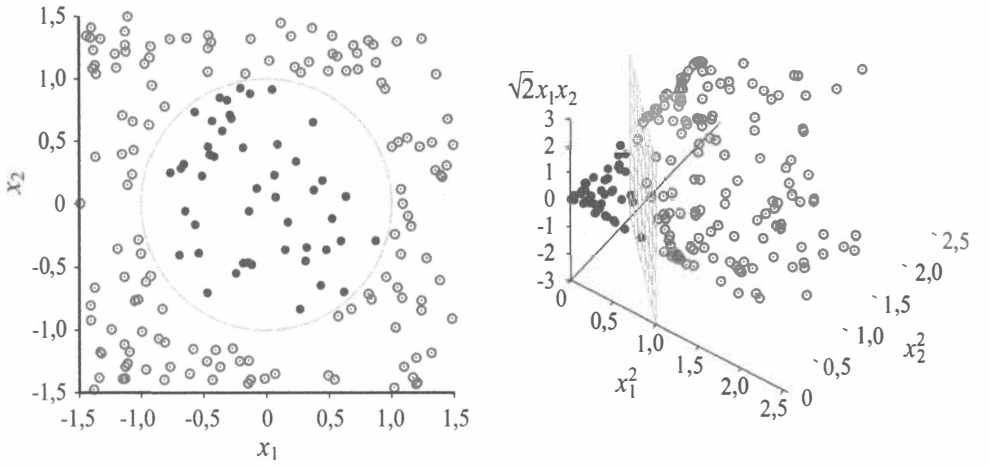


Рис. 19.22. а) Двухмерный обучающий набор с положительными примерами (темные точки) и отрицательными примерами (светлые кружки). Окружностью представлена истинная граница решения $x_1^2 + x_2^2 \leq 1$. **б)** Те же данные после отображения в трехмерное входное пространство (x_1^2 , x_2^2 , $\sqrt{2}\,x_1x_2$). Круговая граница решения на рисунке *а* в трех измерениях становится линейной границей решения, — на рис. 19.21, *б* приведено некоторое приближение к этому разделителю

Теперь уже не потребуется предпринимать попыток отыскать линейный разделитель в исходном пространстве **x**, — можно найти линейные разделители в пространстве признаков $F(\mathbf{x})$ более высокой размерности просто путем замены $\mathbf{x}_j \cdot \mathbf{x}_k$ в уравнении (19.10) на $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$. Само по себе это не является чем-то замечательным — замена **x** на $F(\mathbf{x})$ в *любом* обучающем алгоритме даст тот же эффект, — а вот точечное произведение имеет некоторые особые свойства. Как оказалось, значение $F(\mathbf{x}_j) \cdot F(\mathbf{x}_k)$ часто можно вычислить без предварительного вычисления F для каждой точки. В нашем трехмерном пространстве признаков, задаваемом уравнением (19.12), небольшие алгебраические преобразования показывают, что

$$F(\mathbf{x}_j) \cdot F(\mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2.$$

(Вот почему в f_3 присутствует $\sqrt{2}$.) Выражение $(\mathbf{x}_j \cdot \mathbf{x}_k)^2$ называют ▶**ядерной функцией**[13] и обычно записывают в виде $K(\mathbf{x}_j, \mathbf{x}_k)$. Ядерная функция может быть применена к парам входных данных для оценки точечных произведений в некотором соответствующем пространстве признаков. Таким образом, можно найти линейные разделители в многомерном пространстве признаков $F(\mathbf{x})$ просто путем замены $\mathbf{x}_j \cdot \mathbf{x}_k$ в уравнении (19.10) ядерной функцией $K(\mathbf{x}_j, \mathbf{x}_k)$. В результате обучение можно будет проводить в пространстве более высокой размерности, но при этом вычислять только ядерные функции, а не полный перечень функций для каждой точки данных.

На следующем этапе необходимо убедиться, что в ядре $K(\mathbf{x}_j, \mathbf{x}_k) = (\mathbf{x}_j \cdot \mathbf{x}_k)^2$ нет ничего особенного. Оно соответствует определенному пространству признаков высокой размерности, тогда как другие ядерные функции будут соответствовать другим пространствам признаков. Достоверный результат в математике, ▶**теорема Мерсера** (1909), говорит, что любая "разумная"[14] функция ядра соответствует *некоторому* пространству признаков. Эти пространства признаков могут быть очень большими даже для вполне безобидно выглядящих ядер. Например, ▶**полиномиальные ядра**, $K(\mathbf{x}_j, \mathbf{x}_k) = (1 + \mathbf{x}_j \cdot \mathbf{x}_k)^d$, соответствуют пространству признаков, размерность которого экспоненциальна относительно d. Наиболее общий вариант ядра является гауссианом: $K(\mathbf{x}_j, \mathbf{x}_k) = e^{-\gamma|x_j - x_k|^2}$.

19.7.6. Ядерный трюк

В этом и состоит хитрый ▶**ядерный трюк**: при подстановке рассмотренных выше ядер в уравнение (19.10) оптимальные линейные сепараторы могут быть эффективно найдены в пространствах признаков с миллиардами (или даже бесконечно большим числом) измерений. Полученные линейные разделители при обратном отображении в исходное входное пространство могут быть представлены произвольно извилистыми, нелинейными границами решения между положительными и отрицательными примерами.

В случае зашумленных данных линейный разделитель в некотором многомерном пространстве может не потребоваться. Скорее было бы желательно найти поверхность принятия решения в пространстве с более низким числом измерений, которая не строго разделяет классы, а хорошо отражает реалии зашумленных данных. Это можно сделать с помощью ▶**мягкого разделяющего классификатора** (*soft margin*), позволяющего примерам попадать не на свою сторону от границы решения, но при этом назначающего им штраф, пропорциональный тому расстоянию, на которое потребуется их переместить для возвращения на правильную сторону.

---

[13] Данное использование термина "ядерная функция" немного отличается от ядер в локально-взвешенной регрессии. Некоторые ядра SVM также являются метриками расстояния, но не все.

[14] Здесь определение "разумный" означает, что матрица $\mathbf{K}_{jk} = K(\mathbf{x}_j, \mathbf{x}_k)$ является положительно определенной.

Ядерный метод можно применять не только к алгоритмам обучения, находящим оптимальные линейные разделители, но и к любым другим алгоритмам, которые можно переформулировать для работы только с точечными произведениями пар точек данных, как в уравнениях (19.10) и (19.11). Как только это удастся сделать, скалярное произведение можно будет заменить ядерной функцией, в результате чего будет получена ▶**ядерная версия** данного алгоритма.

19.8. Ансамблевые методы машинного обучения

Выше рассматривались методы обучения, в которых для получения прогнозов использовалась единственная гипотеза. Идея ▶**ансамблевого обучения** состоит в выборе некоторого набора или **ансамбля** гипотез $h_1, h_2, \ldots, h_n$ и комбинировании их прогнозов усреднением, голосованием или посредством другого уровня машинного обучения. В этом случае отдельные гипотезы называют ▶**базовыми моделями** (*base models*), а их комбинацию — ▶**ансамблевой моделью** (*ensemble model*).

Для выбора такого подхода имеются две причины. Во-первых, чтобы уменьшить смещение. Пространство гипотез базовой модели может быть слишком ограничительным, что станет причиной сильного смещения (например, смещение линейной границы решения в логистической регрессии). Ансамбль может оказаться более выразительным, а следовательно, иметь меньшее смещение, чем базовые модели. На рис. 19.23 показано, что ансамбль из трех линейных классификаторов может успешно представлять треугольную область, которая не может быть представлена одним линейным классификатором. Ансамбль из n линейных классификаторов позволяет реализовать больше функций за счет дополнительных затрат в виде в n раз большего объема вычислений. Часто это лучше, чем остановиться на совершенно общем пространстве гипотез, что может потребовать экспоненциально больших объемов вычислений.

Вторая причина состоит в уменьшении дисперсии. Рассмотрим ансамбль из $K = 5$ бинарных классификаторов, результаты которых комбинируются методом большинства голосов. Чтобы ансамбль неправильно классифицировал новый пример, *по крайней мере три из пяти классификаторов должны неправильно его классифицировать*. Можно надеяться, что это менее вероятно, чем одна ошибка классификации единственного классификатора. Чтобы количественно оценить это утверждение, предположим, что был обучен один классификатор, дающий правильный прогноз в 80% случаев. Теперь создадим ансамбль из 5 классификаторов, каждый из которых был обучен на ином подмножестве этих же обучающих данных, так что они являются независимыми. Допустим, что это приводит к некоторому снижению качества и что каждый отдельный классификатор дает правильный прогноз лишь в 75% случаев. Но при совместной их работе большинство голосов в ансамбле будет правильным в 89% случаев (и в 99% — при 17 классификаторах) в предположении, что их независимость истинна.

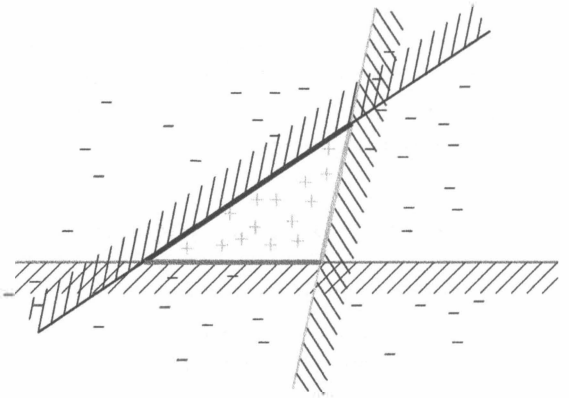


Рис. 19.23. Иллюстрация повышенной выразительной мощности, достигаемой за счет использования ансамблевого обучения. Были выбраны три линейные пороговые гипотезы, каждая из которых классифицирует положительно на незаштрихованную сторону относительно своей границы решения. Весь ансамбль классифицирует как положительный любой пример, который классифицирован положительно всеми тремя базовыми гипотезами. Результирующая треугольная область является гипотезой, не выражаемой линейными разделителями в исходном пространстве гипотез

На практике предположение о независимости классификаторов является необоснованным — отдельные классификаторы используют одни и те же данные и предположения, а следовательно, не являются полностью независимыми и им будет свойственна определенная склонность к одним и тем же ошибкам. Но если классификаторы компонентов, по крайней мере в чем то, не коррелированны, то ансамблевое обучение приведет к уменьшению количества ошибочных классификаций. Ниже будут рассмотрены четыре способа создания ансамблей: бэггинг, случайные леса, стекинг и бустинг.

19.8.1. Бэггинг

Способ создания ансамблей под названием ▶**бэггинг** (*bagging*)[15] предполагает генерацию K различных обучающих наборов посредством выборки с возвращением примеров из исходного обучающего набора (выборка с возвращением означает, что выбранный пример сохраняется в исходном наборе). Иначе говоря, из исходного обучающего набора случайным образом выбирается N примеров, причем любой из них может оказаться примером, который уже выбирался ранее. Затем

[15] Замечание по терминологии. В статистике используется метод, называемый "bootstrap sampling" (~ *вариация загрузочной выборки*). В машинном обучении реализацию этой идеи называют "bootstrap aggregating" (или сокращенно "bagging), что можно понимать как *объединение результатов при различных загрузках.*

на этих N примерах запускается алгоритм машинного обучения с целью получения гипотезы. Этот процесс повторяется K раз для получения K разных гипотез. В дальнейшем, когда потребуется сделать прогноз для нового входного значения, прогнозы от всех K гипотез объединяются. Для задач классификации это означает принятие результата множественного голосования (большинство голосов при двоичной классификации). Для задач регрессии конечный результат является средним значением:

$$h(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} h_i(\mathbf{x}).$$

Бэггинг имеет тенденцию к уменьшению дисперсии и представляет собой стандартный подход, когда имеется ограниченное количество данных или когда базовая модель кажется переобученной. Бэггинг можно применять к моделям любого класса, но чаще всего он используется для деревьев решений. Этот подход можно считать целесообразным, поскольку деревья решений нестабильны: несколько иной набор обучающих примеров может привести к построению совершенно иного дерева. Бэггинг позволяет сгладить этот разброс. Если имеется доступ к нескольким компьютерам, то бэггинг будет особенно эффективным, поскольку гипотезы можно будет вычислять параллельно.

19.8.2. Случайные леса

К сожалению, применение метода бэггинга к деревьям решений часто заканчивается получением K деревьев, которые очень сильно коррелируют между собой. Если есть один атрибут с очень высоким информативным вкладом, то он, вероятно, будет корнем большинства деревьев. Модель ▶**случайного леса** — это особая форма бэггинга деревьев решений, в которой предпринимаются дополнительные меры для того, чтобы сделать ансамбль из K деревьев более разнообразным, что позволит уменьшить дисперсию. Случайные леса могут использоваться для задач классификации или регрессии.

Ключевая идея этого подхода состоит в том, чтобы случайным образом варьировать *выбор атрибутов* (а не обучающих примеров). При построении дерева решений в каждой точке разбиения случайным образом формируется выборка атрибутов, после чего вычисляется, какой из них обеспечивает самый высокий прирост информации. Если есть n атрибутов, чаще всего по умолчанию при каждом разбиении случайным образом выбирается $\sqrt{n}$ атрибутов для задач классификации или $n/3$ атрибутов для задач регрессии.

Дальнейшее улучшение заключается в использовании случайности при выборе *значения* точки разделения: для каждого выбранного атрибута случайным образом выбирается несколько значений-кандидатов из равномерного распределения по всему диапазону значений атрибута, после чего выбирается значение, дающее

наибольший информационный прирост. Это повышает вероятность того, что каждое дерево в лесу будет отличаться от других. Деревья, построенные таким образом, называются ▶**чрезвычайно рандомизированными деревьями** (*extremely randomized trees*) или просто ▶**ExtraTrees**.

Создание случайных лесов является достаточно эффективным. Можно возразить, что для создания ансамбля из K деревьев потребуется в K раз больше времени, но это не так уж плохо по трем причинам. Во-первых, обработка каждой точки разбиения выполняется быстрее, поскольку рассматривается меньше атрибутов. Во-вторых, для каждого отдельного дерева можно пропустить этап обрезки, поскольку ансамбль как целое уменьшает переобучение. И в-третьих, если доступно K компьютеров, есть возможность строить все деревья ансамбля параллельно. Например, Адель Катлер сообщает, что для задачи со 100 атрибутами при наличии только трех процессоров можно построить случайный лес из $K = 100$ деревьев за такое же время, которое необходимо для создания единственного дерева решений на одном процессоре.

Для случайных лесов с помощью перекрестной проверки можно обучить все их гиперпараметры: количество деревьев K, число примеров, используемых при построении каждого дерева N (часто выражается в процентах от полного набора данных), количество атрибутов, используемых при обработке каждой точки разбиения (часто выражается в виде функции от общего числа атрибутов, такой как $\sqrt{n}$), и число проверяемых значений для случайных точек разбиения, если используется подход ExtraTrees. Вместо обычной стратегии перекрестной проверки можно измерить ошибку, получившую название ▶**ошибка "out-of-bag"** (*вне выборки*), — это усредненная ошибка на каждом примере с использованием только тех деревьев, при обучении которых набор примеров не включал данный конкретный пример.

Выше уже указывалось, что более сложные модели могут быть склонны к переобучению, и было показано, что это так для деревьев решений, и затем было доказано, что **обрезка** является хорошим способом предотвращения переобучения. Случайные леса — это сложные, необрезанные модели, и все же они устойчивы к переобучению. По мере увеличения емкости модели путем добавления к случайному лесу новых деревьев они проявляют тенденцию к улучшению в отношении частоты ошибок на проверочном наборе. Кривая обычно выглядит так, как на рис. 19.9, *б*, а не как на рис. 19.9, *а*.

Брейман ([299], 2001) дал математическое доказательство того, что (почти во всех случаях) при добавлении в лес новых деревьев ошибка сходится, — она не возрастает. Один из способов обосновать это состоит в том, что случайный выбор атрибутов приводит к разнообразию деревьев, тем самым уменьшая дисперсию, а поскольку обрезка деревьев не требуется, они могут охватить пространство входа целиком и с более высоким разрешением. Какое-то количество деревьев может охватывать уникальные случаи, появляющиеся в данных лишь несколько раз, и их голоса могут оказаться решающими, хотя они могут быть и в меньшинстве, когда

не применяются. Тем не менее случайные леса нельзя считать полностью защищенными от переобучения. Хотя в пределе ошибка не может увеличиваться, это не означает, что она будет равна нулю.

Применение случайных лесов оказалось очень успешным подходом в широком спектре прикладных задач. В научных соревнованиях на платформе Kaggle они были наиболее популярным методом команд-победителей с 2011 по 2014 год и остаются распространенным подходом по сей день (хотя **глубокое обучение** и **градиентный бустинг** стали даже более популярным выбором среди недавних победителей). Безусловным фаворитом был пакет *randomForest* на языке R. В области финансов случайные леса использовались для прогнозирования дефолта кредитной карты, прогноза доходов в домашнем хозяйстве и выбора цен опционов. Технические применения включают диагностику неисправностей машины и дистанционное зондирование. Биоинформационные и медицинские применения включают диабетическую ретинопатию, количественный анализ экспрессии генов в микроматрицах, анализ экспрессии белков в масс-спектрометрии, обнаружение биомаркеров и предсказание межбелкового взаимодействия.

19.8.3. Стекинг

В то время как бэггинг предполагает комбинирование нескольких базовых моделей одного и того же модельного класса, но обученных на разных данных, метод ▶**стекового обобщения** (*stacked generalization*), или ▶**стекинг** для краткости, объединяет несколько базовых моделей разных классов, обученных на одних и тех же данных. Например, предположим, что имеется набор данных для задачи об ожидании в ресторане, первая строка которого имеет следующий вид:

$$\mathbf{x}_1 = Да, Нет, Нет, Да, Some, \$\$\$, Нет, Да, French, 0–10; y_1 = Да.$$

Сначала исходные данные разделяются на обучающий, проверочный и тестовый наборы, после чего обучающий набор используется для обучения, скажем, трех различных базовых моделей — модели SVM, модели логистической регрессии и модели дерева решений.

На следующем этапе берется проверочный набор и каждая его строка дополняется прогнозами, сделанными для нее тремя базовыми моделями. В результате его строки будут теперь выглядеть так (здесь предсказания выделены полужирным шрифтом):

$$\mathbf{x}_2 = Да, Нет, Нет, Да, Full, \$, Нет, Нет, Thai, 30–60, \mathbf{Да, Нет, Нет}; y_2 = Нет.$$

Далее этот проверочный набор используется для обучения новой ансамблевой модели, скажем класса логистической регрессии (это необязательно должен быть один из классов, использованных для базовых моделей). Ансамблевая модель может использовать прогнозы и исходные данные, как считает нужным. Она может изучить взвешенное среднее для базовых моделей, например что прогнозы должны

быть взвешены в соотношении 50%:30%:20%. Или она может изучить нелинейные взаимодействия между данными и прогнозами — возможно, доверяя SVM-модели больше, когда время ожидания велико. При обучении каждой из базовых моделей мы применяли один и тот же обучающий набор данных, а затем использовали отложенные проверочные данные (плюс прогнозы) для обучения ансамблевой модели. При желании также можно обратиться к перекрестной проверке.

Метод называется "стековым" потому, что данную схему можно рассматривать как слой из базовых моделей с лежащей поверх них ансамблевой моделью, работающей на выходных данных базовых моделей. В действительности можно укладывать в стопку несколько слоев, каждый из которых будет работать на выходных данных предыдущего слоя. Стекинг уменьшает смещение и, как правило, обеспечивает производительность, превышающую производительность любой из отдельных базовых моделей. Стекинг часто используется командами-победителями в научных соревнованиях по обработке данных (таких, как платформа Kaggle или кубок KDD, потому что отдельные участники команды могут работать независимо, оттачивая собственную базовую модель, а затем объединиться для построения финальной стекинговой ансамблевой модели.

19.8.4. Бустинг

Наиболее популярный ансамблевый метод называется ▶**бустингом** (*boosting* — улучшение). Чтобы понять, как он работает, сначала нужно познакомиться с идеей ▶**взвешенного обучающего набора**, когда каждый пример в наборе имеет ассоциированный с ним вес $w_j \geq 0$, определяющий, сколько раз этот пример должен использоваться во время обучения. Например, если первому из примеров присвоен вес 3, а все остальные примеры имеют вес 1, то это эквивалентно тому, что в обучающем наборе имеется 3 копии первого примера.

Бустинг начинается с присвоения одинаковых весов $w = 1$ всем примерам в обучающем наборе. На этом наборе генерируется первая гипотеза h_1. В общем случае гипотеза h_1 классифицирует некоторые примеры из обучающего набора правильно, а некоторые — неправильно. Желательно, чтобы следующая гипотеза лучше работала с теми примерами, которые были классифицированы неверно, поэтому их веса увеличиваются при одновременном уменьшении весов всех примеров, которые были классифицированы верно.

На базе нового взвешенного обучающего набора генерируется гипотеза h_2. Процесс аналогичным образом продолжается до тех пор, пока не будет выработано K гипотез, где K является входным параметром алгоритма бустинга. Примеры, которые сложно поддаются классификации, будут получать более крупный вес то тех пор, пока алгоритм не будет вынужден создать гипотезу, которая сможет классифицировать их правильно. Обратите внимание, что это жадный алгоритм в том смысле, что он не делает возвратов: выбрав однажды гипотезу h_i, он никогда не отменит этот выбор, но будет добавлять все новые и новые гипотезы. Также этот

алгоритм является последовательным, поскольку нельзя вычислить все K гипотез параллельно, как это было возможно в случае бэггинга.

Полученный в конечном счете ансамбль принимает результат голосования гипотез, — аналогично бэггингу, за исключением того, что каждая гипотеза обладает взвешенным количеством голосов: те гипотезы, которые лучше работали на соответствующих им взвешенных обучающих наборах, получают больший вес при голосовании. Для задач регрессии или двоичной классификации имеем

$$h(\mathbf{x}) = \sum_{i=1}^{K} z_i h_i(\mathbf{x}),$$

где z_i — вес при голосовании, присвоенный i-й гипотезе. (Это взвешивание гипотез отличается от взвешивания примеров.)

На рис. 19.24 показано, как этот алгоритм работает на концептуальном уровне. Есть много вариантов базовой идеи бустинга с различными способами корректировки весов примеров и объединения гипотез. Однако все эти варианты следуют общей идее, что сложные примеры должны получать больший вес при переходе от одной гипотезы к следующей. Как и в методе байесовского обучения, который будет рассматриваться в главе 20, во всех вариантах дается больше веса при голосовании более точным гипотезам.

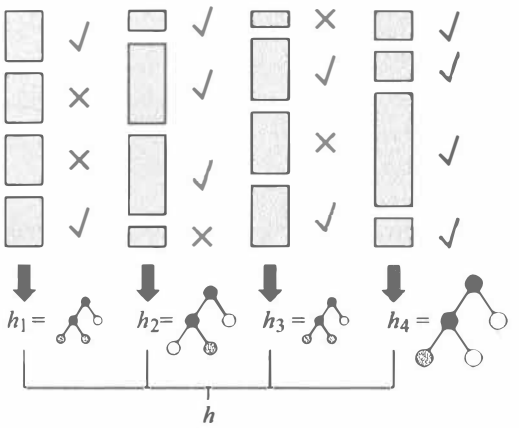


Рис. 19.24. Концептуальная схема работы алгоритма бустинга. Каждый затененный прямоугольник соответствует примеру, причем высота прямоугольника соответствует весу этого примера. Галочки и крестики указывают, как данный пример был классифицирован согласно текущей гипотезе — правильно или неправильно. Размер дерева решения указывает на вес соответствующей гипотезы при голосовании в полученном ансамбле

Один конкретный алгоритм, называемый "AdaBoost", представлен на рис. 19.25. Обычно его применяют для деревьев решений в качестве гипотезы компонентов; часто эти деревья ограничены в размерах. Алгоритм AdaBoost имеет очень важное свойство: если входной обучающий алгоритм L является алгоритмом ▶ **слабого обучения** — а это означает, что алгоритм L всегда возвращает гипотезу с точностью на обучающем наборе, которая лишь немного лучше, чем при случайном угадывании (т.е. 50% + ϵ для булевой классификации), — то AdaBoost возвращает гипотезу, которая *идеально классифицирует данные обучающего набора* при достаточно больших K. Таким образом, алгоритм *повышает* (*boost*) точность исходного алгоритма обучения на обучающих данных.

---

function AdaBoost(*examples*, L, K) **returns** комбинация гипотез
  inputs: *examples*, множество из N помеченных примеров $(x_1, y_1), \ldots, (x_N, y_N)$
    L, обучающий алгоритм
    K, количество гипотез в ансамбле
  local variables: **w**, вектор из N весов примеров, первоначально все равны $1/N$
    h, вектор из K гипотез
    z, вектор из K весов гипотез

  $\epsilon \leftarrow$ небольшое положительное число, используемое для предотвращения деления на нуль
  for $k = 1$ **to** K **do**
    h[k] $\leftarrow L$(*examples*, **w**)
    error $\leftarrow 0$
    for $j = 1$ **to** N **do** // Вычисление общей ошибки для **h**[k]
      if h[k](x_j) $\neq y_j$ **then** *error* $\leftarrow$ *error* + **w**[j]
    if *error* $> 1/2$ **then break** из цикла
    error $\leftarrow \min(error, 1 - \epsilon)$
    for $j = 1$ **to** N **do** // Присвоение больших весов ошибочным примерам в **h**[k]
      if h[k](x_j) $= y_j$ **then w**[j] $\leftarrow$ **w**[j] $\cdot$ *error*/(1 − *error*)
    w $\leftarrow$ NORMALIZE(**w**)
    z[k] $\leftarrow \frac{1}{2} \log((1 - error)/error)$ // Присвоение больших весов правильным гипотезам в **h**[k]
  return WEIGHTED-MAJORITY(**h**, **z**)

Рис. 19.25. Алгоритм ADABOOST — вариант метода бустинга для ансамблевого обучения. Он формирует гипотезы, последовательно переопределяя весовые коэффициенты обучающих примеров. Функция WEIGHTED-MAJORITY формирует гипотезу, которая возвращает выходное значение с наивысшими результатами голосования гипотез в векторе **h**, результаты голосования которого взвешиваются с помощью вектора **z**. Для задач регрессии или бинарной классификации с двумя классами −1 и 1 это Σ_k **h**[k]**z**[k]

Другими словами, бустинг может преодолеть смещение в базовой модели, если базовая модель будет на ϵ лучше, чем случайное угадывание. (В приведенном псевдокоде алгоритм прекращает генерацию гипотез, если обнаружена такая, которая окажется хуже случайного угадывания.) Такой результат остается в силе независимо от того, насколько невыразительным является первоначальное пространство гипотез и насколько сложна изучаемая функция. Точные формулы для весов на рис. 19.25 (с *error*/(1 – *error* и т.д.) выбраны так, чтобы упростить доказательство этого свойства (см. Фреунд и Шапир [781], 1996). Безусловно, это свойство не гарантирует точности на ранее неизвестных примерах.

Давайте посмотрим, насколько хорошо бустинг покажет себя на данных задачи об ожидании в ресторане. Выберем в качестве исходного пространства гипотез класс ►**пней решения** (*decision stumps*), представляющих собой одноузловые деревья решений с единственной проверкой, выполняемой в их корне. На рис. 19.26, *а* нижняя кривая показывает, что без бустинга пни решений не очень эффективны для этого набора данных и достигают в прогнозах производительности лишь 81% на 100 обучающих примерах. В случае применения бустинга (с $K = 5$) производительность заметно улучшается, достигая 93% для 100 обучающих примеров.

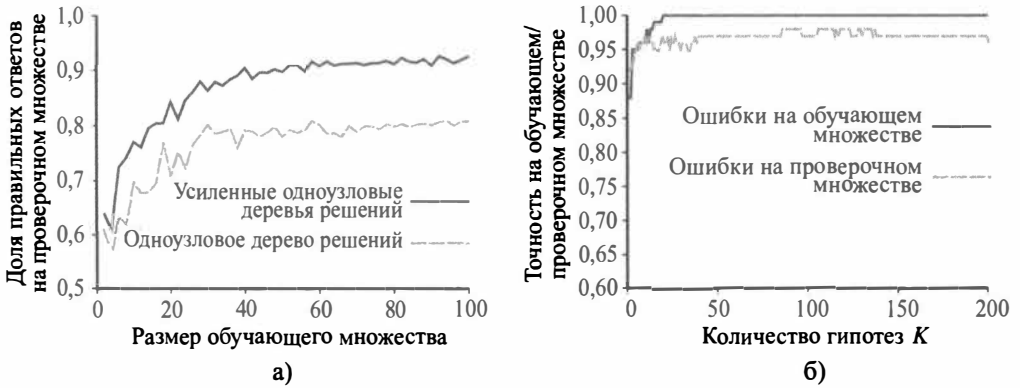


Рис. 19.26. а) График, показывающий эффективность бустинга пней решений при $K = 5$ по сравнению с эффективностью пня решений без бустинга на данных задачи ожидания в ресторане. **б)** Пропорция правильных ответов на обучающем наборе и тестовом наборе как функция от K — количества гипотез в ансамбле. Обратите внимание, что с определенного момента точность на тестовом наборе улучшается незначительно даже после того, как точность на обучающем наборе достигает 1, т.е. после того, как ансамбль будет подогнан под обучающие данные совершенно точно

Интересная особенность проявляется по мере увеличения размера ансамбля K. На рис. 19.26, *б* показана производительность ансамбля на обучающем наборе

(из 100 примеров) как функция от K. Обратите внимание, что ошибка на этом наборе достигает нуля, уже когда K равно 20, т.е. взвешенной мажоритарной комбинации из 20 пней решений достаточно для точного соответствия 100 обучающим примерам — это точка интерполяции. При дальнейшем добавлении новых пней в ансамбль число ошибок по-прежнему остается нулевым. На этом же рисунке показано, что ➡ *производительность ансамбля на тестовом наборе продолжает увеличиваться еще долгое время после того, как ошибка на обучающем наборе уже достигнет нулевого значения*. При $K = 20$ производительность на тестовом наборе составляет 0,95 (или ошибка 0,05) и эта производительность продолжает увеличиваться до 0,98 при K = 137, после чего постепенно снижается до значения 0,95.

Эта особенность, которая достаточно надежно проявляется для различных множеств данных и пространств гипотез, оказалась полной неожиданностью, когда впервые была замечена. Правило бритвы Оккама утверждает, что не следует делать гипотезы более сложными, чем это необходимо, но приведенный выше график указывает, что прогнозы *улучшаются* по мере того, как ансамбль гипотез становится более сложным! По этому поводу были предложены различные объяснения. Согласно одному из них бустинг аппроксимирует **байесовское обучение** (см. главу 20), в отношении которого можно показать, что оно является оптимальным алгоритмом обучения, и эта аппроксимация улучшается по мере добавления гипотез. Другое возможное объяснение состоит в том, что добавление дополнительных гипотез позволяет ансамблю получить больше уверенности в своей способности различать положительные и отрицательные примеры, и это помогает ему, когда дело доходит до классификации новых примеров.

19.8.5. Градиентный бустинг

Для задач регрессии и классификации табличных данных в развернутом представлении метод ▶ **градиентного бустинга**, иногда называемый *машинами градиентного бустинга* (gradient boosting machines — GBM) или *градиентным бустингом регрессионных деревьев* (gradient boosted regression trees — GBRT), стал очень популярным. Как следует из названия, градиентный бустинг является формой бустинга, в которой используется метод градиентного спуска. Напомним, что в алгоритме AdaBoost работа начиналась с одной гипотезы $\mathbf{h}_1$, которая затем улучшалась последовательностью гипотез, и особое внимание при этом обращалось на примеры, которые предыдущие гипотезы обрабатывали неправильно. В градиентном бустинге также добавляются новые улучшающие гипотезы, однако внимание в них уделяется не конкретным примерам, а **градиенту** между правильными ответами и ответами, предоставленными предыдущими гипотезами.

Как и в других алгоритмах, использующих градиентный спуск, обсуждение начнем с выбора дифференцируемой функции потерь, — можно использовать квадратичную ошибку для регрессии или логарифмические потери для классификации. Далее, как и в алгоритме AdaBoost, строится дерево решений.

В разделе 19.6.2 градиентный спуск использовался для минимизации параметров модели, — рассчитывались потери и параметры обновлялись в направлении уменьшения потерь. Однако в градиентном бустинге обновляются параметры не уже существующей модели, а построения следующего дерева, причем делать это следует так, чтобы потери уменьшались за счет перемещения в правильном направлении вдоль градиента.

Как и в тех моделях, которые обсуждались в разделе 19.4.3, предотвратить переобучение можно с помощью **регуляризации**. Она может принять форму ограничений на количество деревьев или их размер (в терминах глубины или числа узлов). Можно ввести коэффициент скорости обучения α, определяющий, насколько далеко разрешено продвинуться в направлении градиента, — обычно используются значения в пределах диапазона от 0,1 до 0,3. И чем меньше будет скорость обучения, тем больше деревьев потребуется включить в ансамбль.

Градиентный бустинг реализован в популярном пакете XGBoost (*eXtreme Gradient Boosting*), который обычно используется как для крупномасштабных приложений в промышленности (для задач с миллиардами примеров), так и победителями научных соревнований по обработке данных (в 2015 году этот пакет использовался каждой командой в первой десятке участников кубка KDD). В пакете XGBoost градиентный бустинг реализован с использованием обрезки и регуляризации, обеспечивается высокая эффективность работы, тщательно контролируется доступ к памяти во избежание потери кеша и допускается параллельное выполнение вычислений на нескольких машинах.

19.8.6. Динамическое обучение

Все, что обсуждалось в этой главе до данного раздела, строилось на предположении, что данные являются независимыми и подчиняются одному и тому же распределению. С одной стороны, это вполне разумное предположение: если будущее не имеет сходства с прошлым, то как можно что-нибудь предсказать? С другой стороны, это слишком строгое предположение: известно, что существует корреляция между прошлым и будущим, а в сложных случаях очень маловероятно, что удастся получить все данные, которые сделали бы будущее независимым от прошлого с учетом имеющихся данных.

В этом разделе обсуждается, что делать, когда данные этим предположениям не соответствуют, например, если они могут изменяться во времени. В этом случае имеет значение, *когда* был сделан прогноз, поэтому мы примем точку зрения, называемую ▶**динамическим обучением** (*online learning*): агент получает извне исходные данные x_j и прогнозирует соответствующее y_j, а затем получает правильный ответ. Затем процесс повторяется с $x_j + 1$ и т.д. Кто-то может подумать, что эта задача безнадежна: если окружение враждебно, все предсказания могут быть неверными. Тем не менее выяснилось, что существуют некоторые гарантии, что подобное сделать можно.

Давайте рассмотрим ситуацию, когда входные данные — это прогнозы, полученные от группы экспертов. Например, каждый день K экспертов предсказывают подъем или падение фондового рынка, и наша задача состоит в том, чтобы объединить эти прогнозы и сделать собственный. Один из возможных способов состоит в том, чтобы отслеживать, насколько хорошо работает каждый из экспертов, и выбирать уровень доверия к нему в соответствии с прежними результатами его деятельности. Такой подход называется ▶**алгоритмом рандомизированного взвешенного большинства**. Более формально этот алгоритм формулируется следующим образом.

Множество весов $\{w_1,\ldots, w_K\}$ инициализируется значениями 1.
for each решаемая задача **do**

1. Получить от экспертов прогнозы $\{\hat{y}_1, \ldots, \hat{y}_K\}$.
2. Случайным образом выбрать эксперта k^* пропорционально его весу: $P(k) = w_k$.
3. **yield** $\hat{y}_{k^*}$ в качестве ответа для данной задачи.
4. Получить правильный ответ y.
5. Для каждого эксперта k, такого, что $\hat{y}_k \neq y$, обновить $w_k \leftarrow \beta w_k$.
6. Нормализовать веса таким образом, что $\Sigma_k w_k = 1$.

Здесь β — это число, $0 < \beta < 1$, определяющее, насколько следует оштрафовать эксперта за каждую ошибку.

Успех этого алгоритма измеряется в терминах ▶**сожаления** (*regret*), которое определяется как количество дополнительно совершенных ошибок по сравнению с экспертом, который, оглядываясь назад, имел наилучшие записи прогнозов. Пусть M^* — количество ошибок, допущенных лучшим экспертом. Тогда количество ошибок M, допущенных алгоритмом рандомизированного взвешенного большинства, будет ограничено выражением[16]

$$M < \frac{M^* \ln(1/\beta) + \ln K}{1-\beta}.$$

Эта оценка справедлива для *любой* последовательности примеров, даже тех, которые были выбраны противниками, старающимися причинить максимальный вред. Если говорить конкретнее, то если есть $K = 10$ экспертов, то при выборе $\beta = 1/2$ количество ошибок будет ограничено функцией $1{,}39M^* + 4{,}6$, а если выбрать $\beta = 3/4$, то $1{,}15M^* + 9{,}2$. В общем случае, если значение β близко к 1, то алгоритм чутко реагирует на изменения в долгосрочной перспективе: если нынешнего лучшего эксперта заменит другой, это будет замечено в ближайшее время. Однако в самом начале за это придется заплатить "штраф"; начиная работу с одинаковым доверием ко всем, можно слишком долго принимать советы плохих экспертов. Когда β находится ближе к 0, эти два фактора становятся обратными. Обратите

[16] Блум в работе [233] (1996) дает элегантное доказательство этого утверждения.

внимание, что значение β можно выбрать так, что в долгосрочной перспективе M становится асимптотически близким M^*: это называют ▶**беспроигрышным обучением** (*no-regret learning*), поскольку среднее количество сожаления в суде по каждому процессу, как правило, стремится к нулю по мере увеличения общего количества судебных процессов.

Динамическое обучение полезно, когда данные могут быстро меняться во времени. Применение его также будет целесообразно для приложений, включающих большой постоянно растущий набор данных, даже если изменения в них будут постепенными. Например, работая с набором данных из миллионов веб-изображений, будет нежелательно переучивать систему с нуля каждый раз, когда добавляется новое изображение. Гораздо практичнее использовать в системе динамический алгоритм, позволяющий постепенно добавлять в нее изображения. Для большинства обучающих алгоритмов, основанных на минимизации потерь, существует динамическая версия, построенная на минимизации сожаления. Многие из этих динамических алгоритмов обеспечивают гарантированные границы сожаления.

Может показаться удивительным, что существуют подобные жесткие границы того, настолько хорошо можно работать в сравнении с группой экспертов. Еще более удивителен тот факт, что, когда такие группы собирают для прогнозирования результатов политических дискуссий или спортивных событий, зрители весьма охотно прислушиваются к их прогнозам и совершенно безразличны к тому, чтобы выяснить, как часто они ошибаются.

19.9. Разработка систем машинного обучения

Выше в этой главе обсуждалась лишь *теория* машинного обучения. *Практика* использования машинного обучения для решения реальных задач представляет собой отдельную дисциплину. За последние 50 лет отрасль разработки программного обеспечения уже выработала методологию, позволяющую повысить вероятность того, что создаваемый (традиционный) программный продукт будет успешным. Однако мы все еще находимся на ранних этапах определения методологии для проектов машинного обучения: инструменты и методы здесь еще не столь хорошо развиты. Ниже приводится схематический набросок типичной последовательности этапов подобного процесса.

19.9.1. Формулировка задачи

Первый этап — тщательно проанализировать ситуацию и понять, какую именно задачу предстоит решить. Этот процесс можно разделить на части. Прежде всего нужно ответить на вопрос "Какая именно проблема будущих пользователей должна быть решена?" Ответ, подобный "Упростить пользователям организацию хранения и доступ к фотографиям" будет слишком расплывчатым. Вариант "Помочь пользователю найти все фотографии, соответствующие конкретному

условию, такому как *Париж*" будет уже лучше. Затем дайте ответ на вопрос "Какую часть (части) этой проблемы можно решить средствами машинного обучения?" Скажем, можно остановиться на варианте "Изучить функцию, отображающую фотографию на множество меток, а затем, получив некоторую метку в качестве запроса, выбрать все фотографии, имеющие эту метку".

Говоря конкретнее, нужно определить функцию потерь для компонента машинного обучения, возможно, измеряя точность системы в прогнозировании правильной метки. Эта цель должна коррелировать с истинными целями, но, как правило, будут существовать различия, — истинной целью может быть максимизация числа пользователей, которых сможет заинтересовать и позволит удержать создаваемая система, а также доходов, которые она принесет. Это те метрики, которые должны отслеживаться разработчиками, но не обязательно те, для которых можно непосредственно построить модель машинного обучения.

После разделения общей задачи на компоненты часто обнаруживается, что многие из них можно реализовать традиционными методами разработки программного обеспечения, без обращения к машинному обучению. Например, для тех пользователей, которые ищут "лучшие фотографии", можно реализовать простые процедуры сортировки изображений по количеству одобрительных оценок и просмотров. Когда разработка системы в целом достигнет момента, когда она уже станет жизнеспособной, можно будет вернуться назад и начать ее оптимизацию, заменяя простые компоненты более сложными моделями машинного обучения.

Также частью постановки задачи является определение того, на каком типе машинного обучения лучше остановиться — обучении с учителем, обучении без учителя или обучении с подкреплением. Различия не всегда бывают очевидными. При ▶**обучении с частичным привлечением учителя** (*semisupervised learning*) обучение проводится на небольшом количестве примеров, снабженных метками, которые используются для извлечения дополнительной информации из большого числа примеров без меток. Сейчас уже стало обычным делом появление новых компаний, задача которых состоит в быстром снабжении метками некоторых примеров с целью помочь системам машинного обучения лучше использовать множество примеров, оставшихся неразмеченными.

Иногда возможен выбор, какой из подходов лучше использовать. Рассмотрим систему выдачи клиентам рекомендаций в отношении песен или кинофильмов. Можно рассматривать ее как задачу обучения с учителем, в которой входными данными будут представленные клиентам рекомендации, а выходом с меткой — сведения о том, понравилась она им или нет. Или же можно выбрать подход обучения с подкреплением, когда система выдает клиентам серии рекомендаций и иногда получает от них вознаграждения за предоставление хорошего предложения.

Метки сами по себе могут не являться теми абсолютными истинами, на которые можно было бы надеяться. Представим, что предпринята попытка создать систему угадывания возраста человека по фотографии. Было найдено и отобрано несколько помеченных примеров, когда люди предоставляли свои фотографии

с указанием возраста. Это обучение с учителем. Но в действительности некоторые из этих людей солгали, намеренно неверно указав свой возраст. Этот факт нельзя рассматривать как случайный шум в данных; такие неточности являются систематическими, и их обнаружение можно рассматривать как задачу обучения без учителя, включающую изображения, указанный людьми возраст и их истинный (неизвестный) возраст. В результате как шум, так и отсутствие меток создают непрерывную среду, некоторый континуум между обучением с учителем и обучением без учителя. В области ▶**обучения с незначительным привлечением учителя** (*weakly supervised learning*) основное внимание уделяется использованию меток зашумленных, неточных или предоставленных неспециалистами.

19.9.2. Сбор данных, их оценка и манипуляция ими

Каждый проект машинного обучения нуждается в данных. В случае проекта по идентификации фотографий в свободном доступе имеются наборы данных изображений, такие как ▶**ImageNet**, содержащий более 14 млн фотографий, снабженных примерно 20 тыс. различных меток. Иногда может потребоваться подготовить собственные данные, что можно сделать собственными силами или посредством **краудсорсинга** с участием оплачиваемых работников или неоплачиваемых добровольцев, работающих через интернет-сервис. Иногда данные поступают от потенциальных пользователей. Например, навигационная служба Waze поощряет пользователей загружать данные о пробках и использует эти данные для предоставления актуальных навигационных указаний всем пользователям. Когда имеется недостаточно собственных данных, можно использовать трансферное обучение (см. раздел 21.7.2): начать с наборов данных общего назначения, находящихся в открытом доступе (или с модели, которая была предварительно обучена на этих данных), а затем добавить конкретные данные от пользователей и провести переобучение.

Когда система будет развернута и станет доступна пользователям, они неизбежно предоставят обратную связь, — возможно, просто щелкая на одних элементах и игнорируя другие. Здесь понадобится выбрать стратегию работы с этими данными. Это предполагает получение рецензии от экспертов по конфиденциальности (см. раздел 27.3.2), подтверждающей, что имеется надлежащее разрешение для сбора таких данных, есть средства обеспечения целостности пользовательских данных и что пользователи понимают, для чего эти данные будут использоваться. Также следует убедиться, что процессы накопления и обработки информации обратной связи справедливы и беспристрастны (см. раздел 27.3.3). Если имеются данные, которые можно считать слишком деликатными для сбора, но которые были бы полезны для модели машинного обучения, следует рассмотреть возможность интегрированного подхода к обучению, при котором данные остаются на устройстве пользователя, а параметры модели разделяются таким образом, что частная информация не раскрывается.

Хорошей практикой является сохранение информации о ▶**происхождении данных** для всех используемых данных. Для каждого столбца в наборе данных следует знать точное определение, откуда поступают данные, каковы возможные значения и кто работал над этим. Имеются ли периоды времени, когда подача данных была прервана? Меняется ли определение какого-либо источника данных со временем? Это необходимо знать, если желательно сравнивать результаты по периодам времени.

Это особенно актуально, если приходится полагаться на данные, предоставленные кем-то другим, — его и ваши потребности могут расходиться. Более того, тот, кто вам их предоставил, со временем может изменить способ их получения или вообще прекратить их обновление. Чтобы обнаружить подобные ситуации, необходимо контролировать каналы поступления данных. Иметь надежные, гибкие и безопасные конвейеры обработки данных более критично для достижения успеха, чем использовать точно выверенные детали алгоритма машинного обучения. Происхождение данных также важно по юридическим причинам, таким как соблюдение закона о конфиденциальности.

В любой задаче будут возникать вопросы о данных. Это подходящие данные для данной задачи? Достаточно ли в них правильных входных примеров, чтобы обеспечить изучение модели? Содержат ли они те результаты, которые требуется предсказывать? Если нет, то существует ли возможность построить модель методом обучения без учителя? Или есть ли возможность пометить часть данных, а затем обратиться к обучению с учителем? Являются ли эти данные релевантными? Иметь в руках 14 млн фотографий — это просто замечательно, но если все будущие пользователи являются специалистами в определенной области, то общая база данных едва ли окажется полезной: потребуется собирать фотографии по конкретной тематике. Обучающего набора какого размера будет достаточно? (Требуется ли собрать больше данных? Можно ли отбросить некоторые данные, чтобы ускорить вычисления?) Лучший способ ответить на этот вопрос — провести аналогию с подобным проектом, размер обучающего набора которого известен.

Когда работа будет начата, появится возможность нарисовать кривую обучения (см. рис. 19,7), чтобы увидеть, поможет ли увеличение объема данных, или же понять, что обучение уже стабилизировалось. Существует бесконечное число особых, никак не обоснованных эмпирических правил для определения "на глаз" необходимого количества обучающих примеров: миллионы для сложных задач; тысячи для задач среднего масштаба; сотни или тысячи для каждого класса в задачах классификации; в 10 раз больше примеров, чем параметров модели; в 10 раз больше примеров, чем входных функций; $O(d \log d)$ примеров для d входных функций; больше примеров для нелинейных моделей, чем для линейных; больше примеров, если требуется бо́льшая точность; меньше примеров, если используется регуляризация; достаточно примеров для достижения статистической силы, необходимой для отклонения нулевой гипотезы в классификации. Все эти правила

приводятся с оговорками, так же как разумное правило, предлагающее попробовать применить то, что успешно работало для подобных задач в прошлом.

Все имеющиеся данные следует рассмотреть с точки зрения защиты. Возможны ли в них ошибки ввода данных? Что можно сделать с пропущенными полями данных? Если данные поступают от будущих клиентов (или других людей), могут ли некоторые из них быть противниками, играющими против системы? Есть ли в текстовых данных орфографические ошибки, используется ли в них противоречивая терминология? (Например, следует ли рассматривать значения "Apple", "AAPL" и "Apple, Inc." как ссылки на одну и ту же компанию?) Потребуются определенные процедуры обнаружения и исправления всех таких потенциальных источников ошибок в данных.

Когда количество данных ограничено, может помочь метод ▶**аугментации** (или **дополнения) данных** (*data augmentation*). Например, в случае набора данных изображений можно создать несколько версий каждого изображения за счет их вращения, преобразования, кадрировании либо масштабирования или же путем изменения яркости, цветового баланса либо добавления шума. Пока это будут относительно небольшие изменения, метка изображения должна оставаться неизменной, при этом модель, обученная на таких дополненных данных, будет более устойчивой.

Иногда данных много, но они делятся на ▶**несбалансированные классы**. Например, обучающий набор транзакций по кредитным картам может состоять из 10 000 000 действительных транзакций и 1000 мошеннических. Классификатор, всегда выдающий прогноз "действительная" независимо от входных данных, для этого набора достигнет точности 99,99%. Чтобы пойти дальше, классификатор должен уделять больше внимания мошенническим примерам. Помочь ему в этом можно, используя ▶**понижающую выборку** (*undersample*) для класса большинства (т.е. игнорировать некоторые из примеров класса "действительных" транзакций) или ▶**повышающую выборку** (*over-sample*) для класса меньшинства (т.е. продублировать некоторые из примеров класса "мошеннических" транзакций). Также можно использовать взвешенную функцию потерь, которая налагает большой штраф за пропуск мошеннического примера.

Применение методов бустинга также способно помочь сфокусироваться на классе меньшинства. Используя ансамблевые методы, можно применять правила, по которым ансамблевое голосование дает ответ "мошенническая" даже в том случае, когда лишь небольшое меньшинство голосов в ансамбле было отдано за этот ответ. Также можно попробовать сбалансировать несбалансированные классы, генерируя синтетические данные с помощью таких методов, как SMOTE (Чаула и др. [404], 2002) или ADASYN (Хэ и др. [994], 2008).

Имеющиеся в данных ▶**выбросы** следует внимательно анализировать. Выброс — это точка данных, расположенная далеко от других точек. Например, в задаче ожидания в ресторане, если бы атрибут *Price* имел цифровое значение, а не категориальное и если бы в одном примере значение атрибута *Price* было равно 316 долл., а во всех остальных — 30 долл. или меньше, то этот пример был бы

выбросом. Такие методы, как линейная регрессия, подвержены искажениям из-за выбросов, поскольку они формируют единую глобальную линейную модель, которая учитывает *все* входные данные — они не способны обрабатывать выбросы не так, как другие точки примеров, а следовательно, единственный выброс способен оказать очень большое влияние на все параметры модели.

В случае атрибутов, таких как цены, которые представлены положительными числами, можно уменьшить эффект от выбросов путем преобразования данных с использованием логарифма каждого значения, так что 20, 25 и 316 долл. превратятся в 1,3, 1,4 и 2,5. Это имеет смысл как с практической точки зрения, поскольку в результате высокое значение окажет гораздо меньшее влияния на модель, так и с теоретической точки зрения, — как было показано в разделе 16.3.2, полезность денег носит логарифмический характер.

Такие методы, как деревья решений, которые строятся из нескольких локальных моделей, могут рассматривать выбросы по отдельности: не имеет никакого значения, чему точно равно большее значение атрибута *Price*, 300 долл. или 31 долл., важно, что оно большее. Другой способ — обработка выброса в собственном локальном узле после проверки типа $Price \leq 30$. Все это делает метод деревьев решений (и следовательно, случайного леса и градиентного бустинга) более устойчивыми к резко отклоняющимся значениям.

Конструирование новых атрибутов

После исправления явных ошибок можно также предварительно обработать данные таким образом, чтобы с ними было легче работать. Ранее уже приводился пример процедуры квантования: распределения входных данных с непрерывной областью определения — например, времени ожидания — по фиксированным диапазонам (0–10 минут, 10–30, 30–60 или > 60). В этом случае сама предметная область может подсказать, какие пороговые значения являются важными, например сравнение возраста по условию больше 18 лет при изучении моделей голосования. Также было показано (раздел 19.7.1), что алгоритмы ближайших соседей работают лучше, когда данные нормализованы так, чтобы получить стандартное отклонение 1. С категориальными атрибутами, такими как *солнечно/облачно/дождливо*, часто бывает полезно преобразовать исходные данные в три отдельных булевых атрибута, только один из которых может принимать истинное значение; такой метод называют ▶**быстрым** (или **однократным**) **кодированием** (*one-hot encoding*). Этот подход будет особенно полезен, когда модель машинного обучения представляет собой нейронную сеть.

Также можно вводить новые атрибуты, исходя из особенностей предметной области. Например, при работе с набором данных о заказах клиентов, в котором каждая запись включает атрибут даты, может потребоваться дополнить данные новыми атрибутами, определяющими, является ли конкретная дата выходным днем или праздником.

Как другой пример рассмотрим задачу оценки истинной стоимости домов, выставленных на продажу. На рис. 19.13 была представлена игрушечная версия этой задачи с применением линейной регрессии для размера дома относительно запрашиваемой цены. Но в действительности требуется оценить продажную стоимость дома, а не запрашиваемую за него цену. Для решения этой задачи необходимы данные о фактических продажах. Но это не означает, что можно отбросить данные о запрашиваемой цене, — ее можно использовать в качестве одного из входных параметров. Помимо информации о размере дома, нужно иметь больше другой информации: число комнат, спален и ванных; были ли кухня и ванные недавно отремонтированы; возраст дома и, возможно, оценка необходимости его ремонта; имеется ли центральное отопление и кондиционер, размер двора и состояние его озеленения.

Также может понадобиться информация не только об участке, но и о районе. Но как определить, что представляют собой окрестности дома? По почтовому индексу? Но что если почтовый индекс перекрывает как желательное, так и нежелательное соседство? А как насчет школьного округа? Будет ли *название* школьного округа достаточной характеристикой или потребуются *средние результаты тестирования*? Способность хорошо выполнить работу по разработке необходимых атрибутов имеет решающее значение для успеха. Как сказал Педро Домингос ([632], 2012): "В конечном счете одни проекты машинного обучения оказываются успешными, а другие — нет. В чем же разница? Безусловно, самый важный фактор — это используемые атрибуты".

Разведочный анализ данных и визуализация

Джон Тьюки в работе [2230] (1977) ввел термин **разведочный анализ данных** (*exploratory data analysis* — EDA) для процесса изучения данных с целью получения представления о них до того, как начинать делать прогнозы или проверять гипотезы. Этот подход реализуется в основном средствами визуализации, но также с привлечением сводных статистических данных. Рассмотрение нескольких гистограмм или диаграмм рассеяния часто помогает установить, что некоторые данные отсутствуют или являются ошибочными, понять, являются ли исходные данные нормально распределенными или они имеют "тяжелые хвосты", и определить, какая модель обучения в данном случае будет наиболее подходящей.

Может оказаться полезным кластеризовать имеющиеся данные, а затем визуализировать точки данных прототипа в центре каждого кластера. Например, можно определить, что в наборе данных изображений присутствует кластер кошачьих мордочек, близкий к нему кластер спящих кошек, а все другие кластеры представляют иные объекты. Следует ожидать, что переход от визуализации к моделированию придется повторить несколько раз, — для создания кластеров необходима функция расстояния, определяющая, какие входы близки друг другу, но чтобы выбрать хорошую функцию расстояния, требуется некоторое представление о данных.

Также полезно будет выявить имеющиеся выбросы, расположенные далеко от прототипов, — их можно рассматривать как **критиков** модели прототипов, помогающих понять, какие типы ошибок может совершать система. Хорошим примером будет кошка в костюме льва.

Компьютерные средства отображения (экраны или бумага) двухмерны, а это означает, что проще всего визуализировать двухмерные данные. Глаза человека также позволяют понимать трехмерные данные, спроецированные в двух измерениях. Но многие наборы данных имеют десятки или даже миллионы измерений. Чтобы визуализировать их, следует выполнить сокращение размерности, проецируя данные вниз на **карту** в двух измерениях (или иногда до трех измерений, которые затем могут исследоваться в интерактивном режиме).[17]

Карта не позволяет сохранить все взаимосвязи между точками данных, но должна обладать таким свойством, что схожие точки исходного набора данных на ней также будут находиться близко друг к другу. Метод, называемый ▶**стохастическим вложением соседей с *t*-распределением** (*t-distributed stochastic neighbor embedding* — ▶**t-SNE**), обеспечивает именно это. На рис. 19.27 приведена карта t-SNE для набора данных цифрового распознавания MNIST. Программные пакеты анализа данных и визуализации, такие как Pandas, Bokeh и Tableau, помогут упростить работу с любыми имеющимися данными.

19.9.3. Выбор модели и обучение

Имея очищенные данные и получив о них достаточно полное представление, можно приступать к построению модели. А это означает, что необходимо выбрать ее класс (случайный лес, глубокая нейронная сеть или ансамбль), провести обучение построенной модели, используя обучающий набор данных, настроить любые гиперпараметры, существующие в выбранном классе (количество деревьев и количество слоев), на основании проверочных данных отладить все процессы и наконец оценить модель на тестовых данных.

Не существует гарантированно хорошего способа выбора лучшего класса модели, но имеются некоторые общие рекомендации. Случайные леса хороши, когда есть множество категориальных признаков и можно предположить, что многие из них могут оказаться неактуальными. Непараметрические методы хороши, когда есть много данных и отсутствуют какие-либо предварительные знания, а также нет желания слишком сильно беспокоиться о выборе только правильных признаков (до тех пор, пока их меньше 20 или около того). Однако непараметрические методы обычно приводят к функции h, более дорогой в смысле вычислительных затрат.

---

[17] Жоффрей Хинтон дал по этому поводу полезный совет: "Чтобы работать с 14-мерным пространством, визуализируйте трехмерное пространство и очень громко скажите себе "четырнадцать".

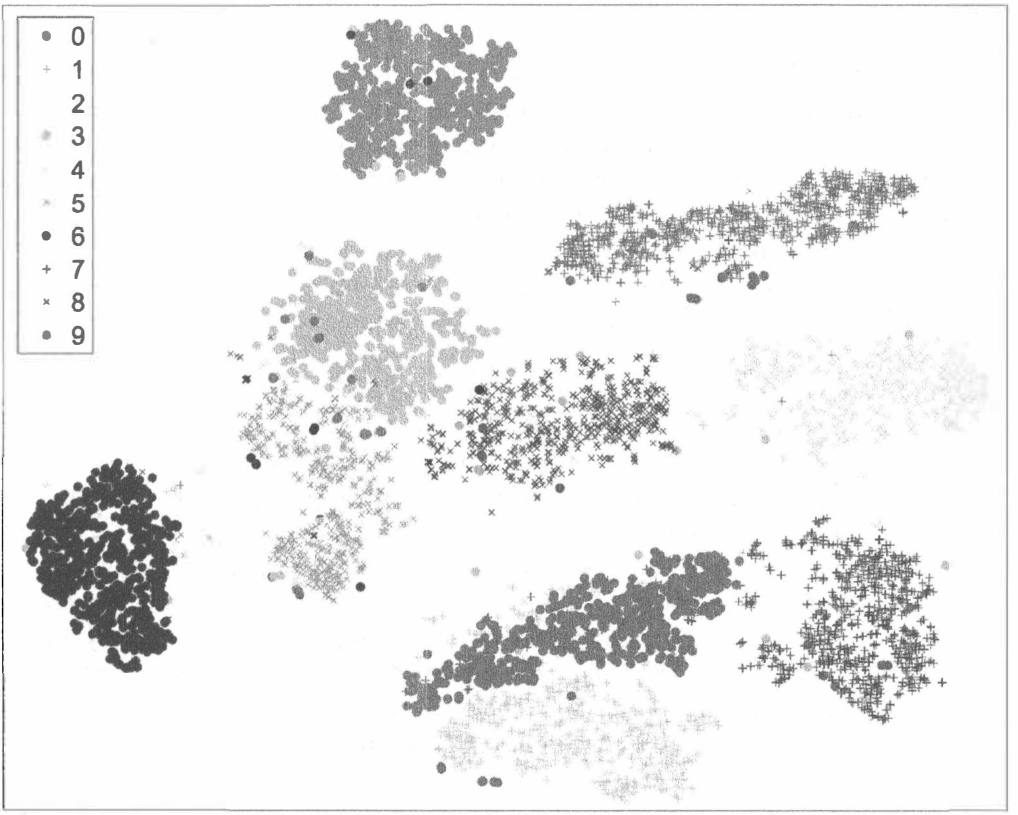


Рис. 19.27. Двухмерная t-SNE-карта для набора данных MNIST, представляющего собой подборку из 60 тыс. изображений рукописных цифр, каждое размером 28 × 28 пикселей, и, следовательно, имеющим 784 измерения. На рисунке четко видны кластеры для десяти цифр, но с несколькими наложениями для каждого кластера. Например, верхний кластер представляет цифру 0, но в пределах этого кластера есть несколько точек данных, представляющих цифры 3 и 6. Алгоритм t-SNE находит такое представление, которое в максимальной степени подчеркивает различия между кластерами

Логистическая регрессия прекрасно работает, когда данные линейно разделимы или могут быть преобразованы таким образом, чтобы стать таковыми при продуманном проектировании признаков. Машины опорных векторов — хороший вариант для опробования, когда набор данных не слишком велик; они работают подобно логистической регрессии на разделяемых данных и могут оказаться более эффективными для многомерных данных. Задачи, связанные с распознаванием определенных шаблонов, такие как классификация изображений или обработка речи, чаще всего решаются с использованием глубоких нейронных сетей (см. главу 21).

Выбор значений гиперпараметров может быть осуществлен посредством комбинирования уже имеющегося опыта — принять то, что хорошо работает в уже реализованных схожих проектах — и поиска: проводить эксперименты с множеством возможных значений гиперпараметров. По мере проведения все большего количества экспериментов будут возникать идеи проверки иных моделей. Однако, если измерять производительность модели на проверочных данных, находить новые идеи, а затем проводить дополнительные эксперименты на тех же данных, возникает риск переобучить модель на проверочных данных. Если имеется достаточное количество примеров, лучше подготовить несколько отдельных проверочных наборов, что позволит избежать этой серьезной проблемы. Данное замечание будет особенно верным, если инспектировать проверочные данные на глаз, вместо того чтобы просто запускать оценки на них.

Предположим, что разрабатывается система классификации, например система обнаружения спама в поступающих сообщениях электронной почты. Присвоение легитимному сообщению метки "спам" называется ▶**ложно-положительной** (*false positive*) **оценкой**. Необходимо найти компромисс между ложно-положительными и ложно-отрицательными (маркировка сообщения со спамом как легитимного) оценками. Если желательно, чтобы меньше легитимной почты попадало в папку для спама, это неизбежно приведет к тому, что в папке входящих окажется больше спама. Но какой способ достижения компромисса будет наилучшим? Можно попробовать различные значения гиперпараметров и получить различные соотношения для этих двух типов ошибок — разные точки на кривой компромисса. Диаграмма, называемая ▶**рабочей характеристикой приемника** (*receiver operating characteristic* — ▶**ROC**) отображает соотношение между ложно-позитивными и истинно-позитивными оценками для каждого значения гиперпараметра, позволяя визуально оценить значения, которые были бы хорошим выбором для достижения компромисса. Метрика, называемая "область под ROC-кривой" или ▶**AUC** (от *area under the ROC curve*), обеспечивает обобщение ROC-кривой одним числовым значением и будет полезна, если требуется развернуть систему и позволить каждому пользователю выбрать собственную точку компромисса.

Другим полезным инструментом визуализации для задач классификации является ▶**матрица ошибок** (*confusion matrix*) — двухмерная таблица результатов подсчета того, как часто каждая категория классифицируется правильно или классифицируется ошибочно, как каждая из других категорий.

Компромиссы возможны для любых других факторов, кроме функции потерь. Если получится обучить модель прогнозирования состояния фондового рынка так, что она будет приносить по 10 долл. за каждую сделку, то это будет замечательно, но не в том случае, если ее эксплуатация будет обходиться по 20 долл. в расчете на каждый прогноз. Программа машинного перевода, работающая на телефоне и позволяющая читать вывески в чужом городе, безусловно, полезна, но не в том случае, если она полностью разряжает аккумулятор всего за один час ее использования. Внимательно анализируйте любые факторы, способные влиять на решение

пользователей принять или отклонить создаваемую систему, а также разработайте процесс, позволяющий быстро проверить новую идею, провести эксперимент и оценить его результаты, чтобы понять, достигнут ли какой-либо прогресс. Сделать подобный процесс как можно более быстрым — одно из важнейших условий достижения успеха в машинном обучении.

19.9.4. Проблема доверия, интерпретируемости и объяснимости

Выше была описана методология машинного обучения, охватывающая разработку модели и подготовку обучающего набора данных, выбор гиперпараметров с использованием проверочного набора и получение окончательных значений показателей системы посредством ее испытаний на тестовом наборе. Успешное достижение требуемых значений показателей является необходимым, но не достаточным условием для безусловного доверия к созданной модели, причем не только со стороны разработчиков. Все заинтересованные стороны, включая контролирующие органы, законодателей, прессу и самих пользователей системы, также интересует возможный уровень доверия к ней (а также связанные с ней атрибуты, такие как надежность, ответственность и безопасность).

Система машинного обучения все так же является элементом программного обеспечения, и доверия к ней можно добиться с помощью все тех же классических инструментов проверки и валидации любой системы программного обеспечения.

- **Контроль версий**. Системы контроля версий, сборок и отслеживания ошибок или проблем.
- **Тестирование**. Модульные тесты для всех компонентов, охватывающие как простые канонические случаи, так и сложные конфронтационные случаи, нечеткие тесты (когда генерируются случайные входные данные), регрессионные тесты, нагрузочные тесты и тесты интеграции системы: все это важно для любой программной системы. Для случая машинного обучения также выполняются дополнительные тесты на обучающих, проверочных и тестовых наборах данных.
- **Анализ.** Анализ программного кода и его оценка, оценка конфиденциальности, оценка справедливости (см. раздел 27.3.3), а также другие оценки юридического соответствия.
- **Мониторинг**. Сводки и сообщения, подтверждающие, что система запущена, работает и продолжает функционировать с высоким уровнем точности.
- **Ответственность**. Что произойдет, если система совершит ошибку? Каков процесс подачи жалобы или аппеляции на решение системы? Как можно отследить, кто ответственен за ошибку? Общество ожидает (но не всегда получает) подотчетности в отношении важных решений, принимаемых банками, политиками и законом, и оно также должно ожидать подотчетности от программных систем, включая системы машинного обучения.

Помимо этого, есть еще некоторые факторы, которые особенно важны для системы машинного обучения, — они подробно обсуждаются ниже.

► **Интерпретируемость**. Говорят, что модель машинного обучения является **интерпретируемой**, если можно проанализировать фактически существующую модель и понять, почему она выбрала конкретное выходное значение для определенного входа и как это выходное значение изменится при изменении входа.[18] Модели деревьев решений считаются в высшей степени интерпретируемыми, — совершенно очевидно, что следование по пути *Patrons* = *Full* и *WaitEstimate* = 0–10 в дереве решений приведет к решению *wait*. Деревья решений хорошо интерпретируемы по двум причинам. Во-первых, у нас, людей, есть опыт понимания правила *Если/То*. (В противоположность этому людям очень трудно достичь интуитивного понимания результата матричного умножения с последующим применением функции активации, как это делается в некоторых моделях нейронных сетей.) Во-вторых, деревья решений, по сути, строятся именно так, чтобы их можно было интерпретировать, — в качестве корня дерева выбирается атрибут, дающий наибольший информационный прирост.

Модели линейной регрессии также считаются интерпретируемыми: можно изучить модель задачи прогнозирования арендной платы за жилье и понять, что в соответствии с этой моделью при добавлении каждой спальни размер арендной платы увеличивается на 500 долл. Собственно, сама идея "Если изменить x, как изменится значение на выходе?" представляет собой суть понятия интерпретируемости. Безусловно, корреляция — это не причинно-следственная связь, поэтому интерпретируемые модели отвечают на вопрос "*Что* происходит?", но не обязательно "*Почему* это так?"

Объясняемость. Объясняемая модель — это модель, помогающая понять, *почему* этот результат был выбран для данного входа. В нашей терминологии интерпретируемость строится на проверке фактической модели, тогда как объясняемость может обеспечиваться отдельным процессом. То есть модель сама по себе может являться трудным для понимания черным ящиком, но объясняющий модуль способен проанализировать то, что делает модель. Для случая нейронной сети системы распознавания изображений, которая классифицировала некоторый рисунок как *собака*, если попытаться интерпретировать модель напрямую, лучшим возможным результатом может быть что-то вроде "После обработки на сверточных слоях для отклика *собака* активация в слое softmax была выше, чем для любого другого класса". Это не очень убедительный аргумент. Однако специальный модуль объяснения может исследовать модель нейронной сети и предложить такое объяснение: "Оно имеет четыре ноги, мех, хвост, висячие уши и длинную морду; оно меньше волка и лежит на собачьей подстилке, поэтому можно полагать, что

[18] Этот термин не является повсеместно принятым: некоторые авторы используют термины "интерпретируемый" и "объяснимый" как синонимы, — в обоих случаях речь идет о достижении некоторого понимания модели.

это собака". Объяснения являются одним из способов укрепления доверия, а некоторые нормативные акты, такие как европейское Общее положение о защите данных (*General Data Protection Regulation* — GDPR) требует, чтобы системы предоставляли объяснения.

В качестве примера отдельного объясняющего модуля можно привести систему локально интерпретируемого независимого от модели объяснения (*Local Interpretable Model-agnostic Explanations* — LIME), которая работает следующим образом. Независимо от того, модель какого класса используется, система LIME строит свою интерпретирующую модель — обычно дерево решений или линейную модель, — аппроксимирующую действительную модель, а затем использует интерпретирующую модель для построения объяснений, сообщающих, насколько важным является каждый из признаков. Система LIME выполняет это путем обращения с моделью обучающей системы как с черным ящиком и зондирования ее различными случайными входными значениями для создания набора данных, на котором может быть построена интерпретирующая модель. Такой подход будет целесообразным для структурированных данных, но не для таких, как изображения, на которых каждый пиксель является признаком и ни один из пикселей не является "важным" сам по себе.

Иногда класс модели выбирается по причине его объяснимости — можно предпочесть деревья решений нейронным сетям не потому, что они имеют более высокую точность, а потому, что объясняемость дает нам больше доверия к ним.

Однако простое объяснение может привести к ложному ощущению безопасности. В конце концов, как правило, предпочтение отдается модели машинного обучения (а не написанию традиционной программы) в тех случаях, когда задача, которую предстоит решить, по своей сути настолько сложна, что просто неизвестно, как написать традиционную программу ее решения. В таких случаях не следует ожидать, что простое объяснение обязательно будет найдено для каждого прогноза.

Если модель машинного обучения создается в первую очередь с целью достижения понимания предметной области, то интерпретируемость и объяснимость будут этому способствовать. Но если цель лишь в том, чтобы получить эффективно работающее программное обеспечение, то обычное тестирование может дать больше уверенности и доверия, чем любые объяснения. К чему вы проявите больше доверия: к экспериментальному воздушному судну, которое еще никогда не поднималось в воздух, но имеется подробное объяснение того, почему это будет безопасно, или к самолету, благополучно совершившему 100 полетов и тщательно поддерживаемому в отличном состоянии, но совершающему полеты без гарантированного предоставления подобного объяснения?

19.9.5. Эксплуатация, мониторинг и сопровождение

Как только желаемая производительность модели будет достигнута, можно предоставить ее для доступа пользователям. И здесь возникают дополнительные проблемы. Во-первых, существует проблема ▶ "**длинного хвоста**" для входов от пользователей. Можно протестировать систему на сколь угодно большом тестовом наборе, но если она станет популярной, скоро непременно поступят входы, которые никогда ранее не встречались и не тестировались. Необходимо выяснить, будет ли модель хорошо их обобщать, а значит, необходим ▶ **мониторинг** производительности системы на реальных данных — отслеживание статистики, отображение ее на графической итоговой панели и отправка извещений, когда ключевые показатели опускаются ниже установленного порога. Помимо автоматического обновления статистики по результатам взаимодействия с пользователями, может потребоваться нанять и обучить людей-оценщиков для слежения за системой и оценки, насколько хорошо она функционирует.

Во-вторых, существует проблема ▶ **нестационарности** — мир неизбежно меняется со временем. Предположим, что система классифицирует электронную почту на спам и не спам. Как только большая партия сообщений будет успешно классифицирована как спам, спамеры заметят этот факт и изменят свою тактику, рассылая новый тип сообщений, с которыми система еще не встречалась. Сообщения, не являющиеся спамом, также изменяются, когда пользователи меняют соотношение в использовании электронной почты и обмена сообщениями или в использовании ПК и мобильных устройств.

Придется постоянно искать ответ на вопрос что лучше: модель, которая уже хорошо проверена, но была построена на старых данных, или же модель, построенная на новейших данных, но еще не прошедшая проверки в режиме реального использования. Различные системы имеют разные требования к актуальности: для одних задач можно получать преимущества от замены прежней модели новой каждый день или даже каждый час, тогда как для других задач одну и ту же модель можно использовать в течение нескольких месяцев. Если новая модель развертывается каждый час, будет нецелесообразно прогонять ее через мощный набор тестов и подвергать ручной проверке при каждом обновлении. Разумнее будет автоматизировать процесс тестирования и сдачи релиза так, чтобы небольшие изменения можно было принять автоматически, но при значительных изменениях выполнялись все соответствующие проверки. Можно рассмотреть компромисс между динамической моделью, когда новые данные автоматически постепенно изменяют существующую модель, и стационарной моделью, для которой каждый новый релиз требует построения новой модели с нуля.

И дело не только в том, что данные со временем будут меняться, например в почтовых сообщениях, относящихся к спаму, будут использоваться новые слова. Помимо этого, вся схема данных также может измениться; возможно, систему, создававшуюся для отсеивания спама в виде сообщений электронной почты, придется

адаптировать для отсеивания спама в виде СМС-сообщений, голосовых сообщений, видеороликов и т.д. На рис. 19.28 приведен набор критериев, который будет полезен при выборе соответствующего уровня тестирования и мониторинга.

Тесты для признаков и данных

1. Все ожидаемые признаки представлены в схеме. 2. Все признаки являются полезными.
3. Нет признаков со слишком большой стоимостью. 4. Признаки соответствуют требованиям метауровня. 5. Линия передачи данных обеспечивает требуемый уровень приватности.
6. Новые признаки можно добавить быстро. 7. Коды всех входных признаков проверены.

Тесты для разрабатываемых моделей

1. Каждая спецификация модели подвергалась проверке кода. 2. Каждая модель зарегистрирована в хранилище. 3. Показатели прокси-сервера в автономном режиме коррелируют с фактическими показателями. 4. Все гиперпараметры настроены.
5. Влияние устаревания модели известно. 6. Более простая модель не лучше.
7. Качество модели достаточное для всех важных срезов данных.
8. Модель тестировалась по соображениям включения.

Тесты для инфраструктуры системы машинного обучения

1. Обучение воспроизводимо. 2. Код спецификации модели проходит юнит-тестирование.
3. Полный конвейер машинного обучения проходит интеграционные испытания.
4. Качество модели проверяется перед попыткой ее обслуживания.
5. Модель позволяет выполнять отладку, наблюдая пошаговые вычисления обучения или вывода на одном примере. 6. Модели тестируются на платформе Canary Deployment, прежде чем передаются в производственную обслуживающую среду.
7. Для моделей можно быстро и безопасно выполнить откат к предыдущей рабочей версии.

Тесты для процессов мониторинга системы машинного обучения

1. Изменения зависимостей вызывают отправку уведомления. 2. В обучающих и обслуживающих наборах хранятся одинаковые данные. 3. Функции обучения и обслуживания вычисляют одинаковые значения. 4. Модели не слишком устаревшие.
5. Модели численно стабильны. 6. Модель не проявляла снижения скорости обучения, задержек в обслуживании, изменений в пропускной способности или использовании RAM.
7. Модель не проявляла снижения качества прогнозов на обслуживаемых данных.

Рис. 19.28. Набор критериев, позволяющий оценить, насколько хорошо решается задача развертывания модели машинного обучения при достаточном количестве проведенных тестов. Это сокращенный вариант метрик аналогичной оценки, предоставленных Бреком и соавт. в работе [295] (2016)

Резюме

В этой главе дано общее введение в машинное обучение с последующим подробным анализом методов обучения с учителем на примерах. Основные вопросы, рассматриваемые в этой главе, перечислены ниже.

- Машинное обучение может принимать много различных форм в зависимости от характера агента, подлежащего усовершенствованию компонента и доступной обратной связи.
- Если доступна обратная связь, предоставляющая правильные ответы на примеры на входе, то задачу обучения называют **обучением с учителем**. Цель состоит в обучении функции $y = h(x)$. Обучение функции, выход которой представляет собой непрерывную величину или ряд упорядоченных значений (например, вес), называется **регрессией**, а обучение функции с небольшим количеством возможных выходных категорий называют **классификацией**.
- Необходимо так обучить функцию, чтобы она была согласована не только с имеющимися данными, но и в будущем с большой вероятностью будет согласована с пока неизвестными ей данными. Также следует помнить о необходимости баланса между согласованностью с данными и простотой гипотезы $y = h(x)$.
- **Деревья решений** позволяют представить любую булеву функцию. Эвристика **приращения информации** обеспечивает эффективный метод нахождения простого, совместимого дерева решений.
- Производительность обучающего алгоритма можно визуализировать с помощью **кривой обучения**, представляющей точность его прогнозов при применении к **тестовому набору** как функцию от размера **обучающего набора**.
- Если существует много моделей, из которых можно выбирать, то процедура **выбора модели** поможет подобрать хорошие значения гиперпараметров, подтвержденные результатами **перекрестной проверки** на проверочных данных. После того как значение гиперпараметра будет выбрано, появится возможность построить наилучшую модель на основе всех обучающих данных.
- Иногда не все ошибки будут равнозначными. **Функция потерь** показывает, насколько плохой является каждая из ошибок, и цель в этом случае состоит в том, чтобы минимизировать потери по всему проверочному набору.
- **Теория вычислительного обучения** анализирует выборочную сложность и вычислительную сложность индуктивного обучения. Ее цель — поиск компромисса между выразительностью пространства гипотез и легкостью обучения.

- **Линейная регрессия** — широко используемый класс моделей. Оптимальные значения всех параметров модели линейной регрессии можно точно вычислить или найти методом градиентного спуска — методом поиска, который можно применять и к таким моделям, которые не имеют решения в замкнутой форме.
- Линейный классификатор с жестким порогом — также известный как **перцептрон** — может быть обучен с помощью простого правила обновления веса вплоть до полного соответствия данным, которые **линейно разделимы**. В противном случае это правило не обеспечивает сходимости.
- **Логистическая регрессия** заменяет жесткий порог перцептрона мягким порогом, определяемым логистической функцией. В этом случае метод градиентного спуска хорошо работает даже на зашумленных данных, которые не являются линейно разделимыми.
- **Непараметрические модели**, чтобы сделать каждый прогноз, используют все имеющиеся данные вместо того, чтобы попытаться обобщить эти данные на основании нескольких параметров. Примеры такого подхода включают методы **ближайших соседей** и **локально-взвешенной регрессии**.
- В методе **машин опорных векторов** отыскиваются линейные разделители с **максимальным зазором**, что позволяет улучшить эффективность обобщения классификатора. **Ядерные методы** предполагают неявное преобразование входных данных в пространство высокой размерности, где линейный разделитель может существовать и в том случае, когда исходные данные не являются линейно разделимыми.
- Ансамблевые методы, такие как **бэггинг** и **бустинг**, часто показывают более высокую производительность по сравнению с методами обучения отдельных деревьев решений. При **динамическом обучении** можно комбинировать мнения различных экспертов, что позволяет произвольно близко подойти к лучшей экспертной производительности, даже если распределение данных постоянно меняется.
- Построение хорошей модели машинного обучения требует опыта в процессе ее разработки в полном масштабе, от управления данными до выбора и оптимизации модели, от процедур ее тестирования до решений по мониторингу и дальнейшему ее сопровождению в процессе эксплуатации.

Библиографические и исторические заметки

В главе 1 описывается история философских исследований в области индуктивного обучения. Уильям из Оккема (1280–1349), наиболее влиятельный философ своего времени, внесший наибольший вклад в развитие эпистемологии, логики и метафизики в эпоху средневековья, заслужил признание потомков тем, что выдвинул важный принцип, называемый "бритва Оккама". На латыни он формулируется так: "Entia non sunt multiplicanda praeter necessitatem", что в переводе звучит как "Не следует множить сущности без необходимости". К сожалению, это замечательное изречение не удалось найти в его трудах точно в такой формулировке (хотя он действительно сказал "Pluralitas non est ponenda sine necessitate", или "Множественность не должно ложить в основу без необходимости"). Подобное мнение было выражено Аристотелем в 350 году до н.э. в книге "Физика", т. I, глава VI: "Более ограниченное, если оно адекватно, всегда предпочтительнее".

Дэвид Юм (1711–1776) сформулировал *проблему индукции*, признавая, что обобщение на основе примеров допускает возможность ошибок таким путем, который в логическом выводе невозможен. Он понимал, что здесь нет никакого способа получить гарантированно правильное решение задачи, но предложил принцип *единообразия природы*, который теперь называют *стационарностью*. То, к чему пришли Оккем и Юм, фактически состоит в том, что, когда применяется индукция, из множества совместимых моделей выбирается та, которая более вероятна, — просто потому, что она проще и больше соответствует нашим ожиданиям. В настоящее время теорема "бесплатных обедов не бывает" (Уолперт и Макреди [2373], 1997; Уолперт [2372], 2013) говорит о том, что если обучающий алгоритм хорошо работает на определенном множестве задач, то это только потому, что он будет плохо работать на другом множестве: если дерево решений правильно предсказывает поведение Стюарта Рассела в отношении принятия решения об ожидании в ресторане, то оно должно плохо работать для некоторого другого гипотетического человека, предпочитающего противоположный вариант поведения в аналогичных ситуациях, определяемых еще не наблюдавшимися входами.

Машинное обучение являлось одной из ключевых идей при рождении компьютерных наук. Алан Тьюринг ([2237], 1947) предварил его, сказав: "Давайте предположим, что уже создана машина с определенными начальными таблицами команд, построенными таким образом, что одни таблицы могли бы в некоторых случаях, если имеется хорошая причина, изменять другие таблицы". Артур Самуэль ([1969], 1959) определил машинное обучение как "поле исследований в отношении того, как дать компьютерам способность к обучению, не будучи явно запрограммированными", когда работал над созданием своей самообучающейся программы игры в шашки.

Первым известным случаем использования **деревьев решений** была система EPAM (*Elementary Perceiver And Memorizer*) Фейгенбаума ([715], 1961), представлявшая собой имитатор концепции обучения человека. В системе ID3 (Куинлан

[1834], 1979) была добавлена ключевая идея выбора атрибута с максимальной энтропией. Концепция энтропии и теория информации были разработаны Клодом Шенноном в качестве инструмента исследования процессов связи (Шеннон и Уивер [2039], 1949). (Шеннон также способствовал разработке одного из ранних примеров применения принципа машинного обучения — механической мыши, получившей название "Theseus" (*Тесей*), которая обучалась прохождению через лабиринт по методу проб и ошибок.) Метод χ^2-обрезки деревьев был описан Квинланом ([1835], 1986). Описание программного пакета промышленного уровня C4.5 для работы с деревьями решений можно найти в работе Квинлана [1834] (1993). Альтернативный программный пакет промышленного уровня CART (*Classification and Regression Trees*) был создан статистиком Лео Брейманом и его коллегами (Брейман и др. [298], 1984).

Яфиль и Ривест ([1111], 1976) доказали, что нахождение *оптимального* дерева решений (вместо нахождения хорошего дерева посредством локального жадного отбора) является NP-полной задачей. Однако Берцимас и Дунн ([205], 2017) отметили, что за последние 25 лет прогресс в области аппаратного обеспечения и в алгоритмах смешанного целочисленного программирования привел к ускорению требуемых расчетов в 800 млрд раз, и это означает, что в настоящее время представляется возможным решить эту NP-сложную задачу, — по крайней мере для проблем не более чем с несколькими тысячами примеров и несколькими десятками признаков.

Перекрестная проверка впервые была введена Ларсоном ([1355], 1931) в форме, близкой к той, которая была предложена Стоуном ([2174], 1974), а также Голубом и соавт. ([885], 1979). Разработка процедуры регуляризации принадлежит Тихонову ([2216], 1963).

В отношении вопроса о переобучении Джон фон Нейман был процитирован Дайсоном ([669], 2004) как похвастaвшийся: "При четырех параметрах я могу подогнать слона, а при пяти я могу заставить его шевелить хоботом" в том смысле, что многочлен высокой степени можно подогнать до полного соответствия практически любым данным, но за счет потенциального переобучения. Майер и соавт. ([1519], 2010) доказал его правоту, продемонстрировав слона на четырех параметрах и "шевеление" — на пяти, а Буйе ([263], 2019) пошел еще дальше, демонстрируя слона и других животных с помощью хаотической функции с одним параметром.

Чжан и соавт. ([2429], 2016) проанализировали, при каких условиях модель может запомнить обучающие данные. Они проводили эксперименты с использованием случайных данных — нет сомнения, что алгоритм, достигающий нулевой ошибки на обучающем наборе со случайными метками, должен просто запоминать этот набор. Тем не менее они пришли к выводу, что в этой области еще предстоит найти точную меру того, что означает для модели быть "простой" в смысле, определяемом бритвой Оккама. Арпит и соавт. ([77], 2017) показали, что условия, при которых может происходить запоминание, зависят от деталей как модели, так и набора данных.

В работе Белкина и соавт. ([157], 2019) рассматривается компромисс *смещение–дисперсия* в машинном обучении и обсуждается, почему модели одних классов продолжают улучшаться после достижения точки интерполяции, в то время как модели других классов демонстрируют U-образную форму кривой обучения. Беррада и соавт. ([193], 2019) разработали новый обучающий алгоритм, основанный на градиентном спуске, который использует способность моделей запоминать для установки хороших значений гиперпараметра скорости обучения.

Теоретический анализ обучающих алгоритмов начался с работы Голда [878] (1967) по проблеме **идентификации в пределе**. Стимулом к развитию этого подхода отчасти явились модели научного открытия, разработанные в рамках философии науки (Поппер [1813], 1962), но этот подход в основном применялся к задаче изучения грамматик на примерах предложений (Ошерсон и др. [1719], 1986).

Подход, основанный на изучении "идентификации в пределе", главным образом посвящен достигаемой в конечном итоге сходимости, тогда как в исследованиях ▶**колмогоровской сложности**, или **алгоритмической сложности**, проведенных независимо Соломоновым ([2107], 1964; [2108], 2009) и Колмогоровым ([1273], 1965) предпринималась попытка дать формальное определение понятию простоты, используемому в принципе бритвы Оккама. Чтобы исключить проблему зависимости простоты от способа представления информации, было предложено измерять простоту как длину кратчайшей программы для универсальной машины Тьюринга, которая правильно воспроизводит наблюдаемые данные. Хотя существует множество возможных универсальных машин Тьюринга и, следовательно, множество возможных "кратчайших" программ, эти программы различаются по длине не больше чем на некоторую константу, которая не зависит объема данных. Это великолепное открытие, которое, по сути, показывает, что *любое* исходное смещение в представлении данных в конечном итоге может быть преодолено также с помощью данных. Его широкому применению препятствует лишь то, что задача вычисления длины кратчайшей программы является неразрешимой. Однако вместо точного значения длины могут применяться такие приближенные критерии, как **минимальная длина описания**, или MDL (*Minimum Description Length*) (Риссанен [1890], 1984; [1891], 2007), позволяющие добиться превосходных результатов на практике. Наилучшим источником по проблеме колмогоровской сложности является учебник Ли и Витаньи [1403] (2008).

Теория вычислительного обучения была выдвинута Лесли Валиантом ([2244], 1984). В ней подчеркивается важность вычислительной и выборочной сложности. Совместно с Майклом Кернсом ([1205], 1990) Валиант показал, что задача ВПК-обучения некоторых классов концепций является трудноразрешимой, даже если в примерах присутствует достаточный объем информации. Некоторые положительные результаты были получены для таких классов, как списки решений (Ривест [1892], 1987).

Независимое направление анализа выборочной сложности существует в статистике, его начало было положено работой по **теории равномерной сходимости**

(Вапник и Червоненкис [2262], 1971). Понятие ▶**VC-размерности** предоставляет критерий, приблизительно аналогичный, но более общий, чем критерий $\ln|\mathcal{H}|$, определяемый на основе анализа ВПК-обучения. Критерий VC-размерности может применяться к непрерывным классам функций, на которые не распространяется стандартный анализ ВПК-обучения. Теория ВПК-обучения и теория VC-размерности были впервые связаны в единое целое группой ученых, называемых "четырьмя немцами" (хотя ни один из них в действительности не является немцем) — Блумером, Эренфойхтом, Хаусслером и Вармутом ([236], 1989).

Линейная регрессия с квадратичной оценкой потерь восходит к Лежандру ([1378], 1805) и Гауссу ([821], 1809), — оба работали над предсказанием орбит вокруг Солнца. (Гаусс утверждал, что использовал этот метод с 1795 года, но отложил его публикацию.) Современное использование множественной регрессии в машинном обучении освещается, например, в работе Бишопа ([222], 2007). Различия между регуляризацией L_1 и L_2 анализируются в работах Энга ([1672], 2004), а также Мура и ДеНеро ([1618], 2011).

Термин **логистическая функция** был предложен Пьером-Франсуа Верхюльстом (1804–1849), статистиком, использовавшим эту кривую для моделирования роста населения при ограниченных ресурсах, — более реалистичной модели в сравнении с неограниченным геометрическим ростом, предложенным Томасом Мальтусом. Верхюльст назвал ее *courbe logistique* (логистической кривой) из-за связи с логарифмической кривой. Термин **проклятие размерности** принадлежит Ричарду Беллману ([165], 1961).

Задачи **логистической регрессии** могут быть решены с помощью метода градиентного спуска или с помощью метода Ньютона-Рафсона (Ньютон [1671], 1671; Рафсон [1852], 1690). Вариант метода Ньютона, называемый L-BFGS, часто используется для задач большой размерности. В этой аббревиатуре "L" соответствует *limited memory* (ограниченная память) и означает, что в этом методе исключается одновременное создание полных матриц, — вместо этого создаются их части "на лету", а BFGS — это инициалы авторов (Берд и др. [355], 1995). Идея градиентного спуска восходит к Коши ([381], 1847), а стохастический градиентный спуск (SGD) был введен в сообществе статистической оптимизации Роббинсом и Монро ([1894], 1951), вновь открыт, но уже для нейронных сетей Розенблаттом ([1911], 1960), а затем популяризирован для крупномасштабных задач машинного обучения Ботту и Буске ([261], 2008). Позднее Ботту и соавт. ([262], 2018) вновь проанализировали проблему крупномасштабного машинного обучения, но уже с учетом десятилетнего опыта.

Модель **ближайших соседей** восходит по крайней мере к работе Фикса и Ходжеса ([746], 1951) и с тех пор уже стала стандартным инструментом в статистике и распознавании шаблонов. В области ИИ ее популяризировали Стэнфилл и Вальс ([2123], 1986), исследовавшие методы адаптации метрики расстояния к данным. Хэсти и Тибширани ([980], 1996) разработали способ локализации метрики для каждой точки в пространстве в зависимости от распределения данных вокруг

этой точки. Гионис и соавт. ([863], 1999) ввели метод локально-чувствительного хеширования (LSH), который произвел революцию в процессах получения доступа к схожим объектам в многомерных пространствах. Андони и Индик ([50], 2006) предоставили обзор работ по LSH и родственным методам, а Самет ([1967], 2006) описал свойства многомерных пространств. Этот метод является особенно полезным для данных о геномах, в которых каждая запись включает миллионы атрибутов (Берлин и др. [184], 2015).

Идеи, лежащие в основе метода **ядерных машин**, исходят от Айзермана и соавт. ([25], 1964) (они также предложили ядерный трюк), но полная разработка этой теории была выполнена Вапником и его коллегами (Бозер и др. [257], 1992). Метод SVMs был поставлен на практические рельсы с введением понятия мягкого разделяющего классификатора для обработки зашумленных данных в статье, которая в 2008 году была удостоена премии ACM по теории и практике (Кортес и Вапник [480], 1995), и разработки алгоритма последовательной минимальной оптимизации (*Sequential Minimal Optimization* — SMO) для эффективного решения задач SVM с помощью квадратичного программирования (Платт [1797], 1999). В отношении метода SVM было доказано, что он является очень эффективным для таких классов задач, как категоризации текста (Иоахим [1139], 2001), вычислительная геномика (Кристианини и Хан [498], 2007) и распознавание рукописных цифр (Декост и Шекопф [589], 2002).

В рамках этого процесса было разработано много новых ядер, работающих со строками, деревьями и другими нечисловыми типами данных. Связанный метод, в котором также используется ядерный трюк для неявного представления экспоненциального пространства признаков, — это **voted perceptron** (*голосующий перцептрон*) (Фройнд и Шапир [781], 1999; Коллинз и Даффи [464], 2002). Учебники по методам SVM были предоставлены Кристианини и Шоу-Тейлором ([500], 2000), а также Шелкопфом и Смола ([1998], 2002). Более доходчивые разъяснения представлены в статье Кристианини и Шелкопфа в журнале *AI Magazine* ([499], 2002). Бенжио и ЛеКун в работе [174] (2007) представили некоторые ограничения для метода SVM и других локальных, непараметрических методов при изучении функций, имеющих глобальную структуру, но не имеющих локальной гладкости.

Первым математическим доказательством ценности ансамблей была теорема Кондорсе о жюри ([549], 1785), в которой он доказал, что если присяжные являются независимыми и отдельные присяжные имеют по крайней мере 50%-ный шанс решить дело правильно, то чем больше присяжных будет привлечено, тем больше шансов на правильное решение дела. В последнее время **ансамблевое обучение** становится все более популярным методом повышения производительности обучающих алгоритмов.

Первый алгоритм **случайного леса**, в котором использовался случайный выбор атрибута, был предложен Хо ([1035], 1995); позднее независимая версия была представлена Амитом и Геманом ([43], 1997). Брейман ([299], 2001) добавил идеи **бэггинга** и "ошибки out-of-bag". Фридман ([788], 2001) ввел термин *машина*

градиентного бустинга (GBM), расширив этот подход возможностью решать задачи мультиклассной классификации, регрессии и ранжирования.

Мишель Кернс ([1209], 1988) определил проблему гипотезы о бустинге: имея слабую гипотезу, которая предсказывает лишь немного лучше, чем случайное гадание, можно ли вывести гипотезу, производительность которой будет сколь угодно хорошей? Положительный ответ на эту проблему был дан в теоретической работе Шапире ([1992], 1990), что привело к разработке алгоритма AdaBoost Фройндом и Шапире ([780], 1996) и к последующей теоретической работе Шапире ([1993], 2003). Фридман и соавт. в работе [787] (2000) объясняют бустинг с точки зрения статистики. Чен и Гестрин ([414], 2016) описывают систему XGBoost, которая с большим успехом используется во многих крупномасштабных приложениях.

Динамическое обучение рассматривается в обзоре Блума ([233], 1996) и книге Чеза-Бьянки и Лугоси ([382], 2006). Дредце и соавт. ([649], 2008) представляют идею доверительно-взвешенного динамического обучения для задач классификации: в дополнение к сохранению веса для каждого параметра сохраняется и степень доверия к нему, так что новый пример может оказать большое влияние на признаки, которые редко наблюдались до этого (и следовательно, имели низкую степень доверия), и лишь небольшое влияние — на распространенные признаки, которые уже были хорошо оценены. Ю и соавт. ([2414], 2011) описывают, как команда студентов вместе работала над созданием ансамблевого классификатора в соревнованиях на платформе KDD. Одной из замечательных возможностей является создание "невероятно большого" смешанного ансамбля экспертов, в котором разреженное подмножество экспертов используется для каждого входящего примера (Шазир и др. [2045], 2017). Сени и Элдер в работе [2026] (2010) приводят обзор ансамблевых методов.

В области практических советов по построению систем машинного обучения Педро Домингос описывает несколько вещей, которые необходимо знать ([632], 2012), а Эндрю Энг дает советы по разработке и отладке продуктов с использованием машинного обучения ([1673], 2019). О'Нил и Шутт ([1715], 2013) описывают весь процесс правильной подготовки данных, Тьюки ([2230], 1977) предлагает метод **разведочного анализа данных**, а Гельман ([831], 2004) приводит обновленное представление об этом процессе. Бьен и соавт. ([211], 2011) описывают процесс выбора прототипов для интерпретируемости, а Ким и соавт. ([1223], 2017) показывают, как найти критика, который будет максимально удален от прототипов, используя метрику под названием "максимальное среднее расхождение". Ваттенберг и соавт. ([2301], 2016) описывает, как использовать метод t-SNE. Чтобы получить полное представление о том, насколько хорошо работает развернутая система машинного обучения, Брек и соавт. ([295], 2016) предлагают контрольный список из 28 тестов, которые можно применить, чтобы получить общий тестовый балл по результатам машинного обучения. Райли ([1882], 2019) описывает три распространенных подводных камня в процессе разработки систем машинного обучения.

Банко и Брилл ([122], 2001), Халеви и соавт. ([950], 2009) и Гандоми и Хайдер ([810], 2015) обсуждают преимущества использования больших объемов данных,

которые теперь доступны. По оценкам Лимана и Вариана ([1469], 2003), в 2002 году было получено около 5 экзабайт (5×10^{18} байт) данных, при этом скорость их производства удваивается каждые 3 года. Хилберт и Лопес ([1021], 2011) в 2007 году оценили этот объем в 2×10^{21} байт, что указывает на ускорение темпов. Гийон и Элиссефф ([940], 2003) обсуждают проблему выбора признаков при использовании больших наборов данных.

Доши-Велес и Ким ([639], 2017) предложили структуру для **интерпретируемого машинного обучения** или **объяснимого AI** (*explainable AI* — **XAI**). Миллер и соавт. ([1578], 2017) указывают на то, что существует два вида объяснений: одно для разработчиков системы ИИ, а другое — для ее пользователей, и нужно четко понимать, к чему стремиться. Система LIME (Рибейро и др. [1877], 2016) строит интерпретируемые линейные модели, которые аппроксимируют любую имеющуюся систему машинного обучения. Похожая система SHAP (Лундберг и Ли [1465], 2018) — что является сокращением от *Shapley Additive exPlanations* — использует понятие значения Шепли (см. раздел 18.3.2) для определения вклада каждого признака.

Идея, что можно применить машинное обучение к решению задач машинного обучения, является довольно заманчивой. Тран и Пратт ([2213], 2012) дают ранний обзор этой области в редакторском сборнике под названием "Learning to Learn" (*Обучение учить*). Недавно для этого поля исследований было принято название ►**автоматизированное машинное обучение** (*automated machine learning* — ►**AutoML**); Хаттер и соавт. в работе [1105] (2019) предоставили его обзор.

Кантер и Вирамаханени ([1183], 2015) описывают систему автоматического выбора признаков. Бергстра и Бенгио ([180], 2012) описывают систему поиска пространства гиперпараметров, так же как и Торнтон и соавт. ([2209], 2013) и Бермудес-Чакон и соавт. ([188], 2015). Вонг и соавт. ([2374], 2019) показали, как трансферное обучение может ускорить работу системы AutoML для моделей глубокого обучения. Были организованы соревнования с целью определить, какие системы лучше всего подходят для задач AutoML (Гийон и др. [939], 2015). В работе Штейнрукена и соавт. [2127] (2019) описывается система под названием "Automatic Statistician" (*Автоматический статистик*): ей передаются некоторые данные, и она составляет отчет, объединяя текст, диаграммы и вычисления. Основные поставщики облачных вычислений включили системы AutoML как часть своих предложений. Некоторые исследователи предпочитают термин **метаобучение** (*metalearning*); например, система MAML (*Model-Agnostic Meta-Learning*) (Финн и др. [741], 2017) работает с любой моделью, которую можно обучить с помощью градиентного спуска; она обучает базовую модель, так что в дальнейшем будет несложно настроить модель по новым данным из новых задач.

Несмотря на всю эту работу, пока еще не существует полной системы автоматического решения задач машинного обучения. Чтобы реализовать ее с помощью метода машинного обучения с учителем, следовало бы начать с набора данных из примеров $(\mathbf{x}_j, y_j)$. Здесь вход $\mathbf{x}_j$ представляет собой спецификацию задачи в форме, в которой задачи поступают изначально: расплывчатое описание целей

и некоторые данные, с которыми предстоит работать (возможно, с расплывчатым указанием на то, как получить больше таких данных). Выход y_i должен представлять собой законченную, работоспособную программу машинного обучения вместе с методологией ее полного обслуживания и сопровождения: способов сбора дополнительных данных и их очистки, методов тестирования и контроля за системой и т.д. Можно ожидать, что потребуется набор данных из тысяч подобных примеров. Но таких наборов данных не существует, поэтому существующие системы AutoML имеют ограничения в том, чего они позволяют достичь.

Существует огромное количество книг, знакомящих читателя с наукой о данных и машинном обучении в сочетании с такими программными пакетами, как ***Python*** (Сегаран [2020], 2007; Рашка [1853], 2015; Нильсен [1680], 2015), ***Scikit-Learn*** (Педрегоза и др. [1763], 2011), ***R*** (Конвей и Уайт [471], 2012), ***Pandas*** (Маккинни [1550], 2012), ***NumPy*** (Марсланд [1501], 2014), ***PyTorch*** (Говард и Гуггер [1071], 2020), ***TensorFlow*** (Рамсундар и Заде [1849], 2018) и ***Keras*** (Шолле [420], 2017; Герон [840], 2019).

Издан ряд ценных учебников по машинному обучению (Бишоп [222], 2007; Мерфи [1638], 2012), а также по тесно связанным и пересекающимся с ним областям *распознавания образов* (Рипли [1889], 1996; Дуда и др. [660], 2001), *статистики* (Вассерман [2298], 2004; Асти и др. [981], 2009; Джеймс и др. [1127], 2013), *науки о данных* (Блум и др. [234], 2020), *интеллектуального анализа данных* (Хан и др. [955], 2011; Уиттен и Франк [2366], 2016; Тан и др. [2172], 2019), *теории вычислительного обучения* (Кернс и Вазирани [1208], 1994; Вапник [2261], 1998) и *теории информации* (Шеннон и Уивер [2039], 1949; Ковер и Томас [482], 2006). Кроме того, Бурков ([346], 2019) попытался дать кратчайшее введение в машинное обучение, а Домингос ([633], 2015) предложил нетехнический обзор этой области. Результаты современных исследований в области машинного обучения публикуются в ежегодных выпусках материалов Международной конференции по машинному обучению (*International Conference on Machine Learning* — ICML), Международной конференции по обучающим представлениям (*International Conference on Learning Representations* — ICLR) и Конференции по нейронным системам обработки информации (*Neural Information Processing Systems* — NeurIPS), а также в журналах *Machine Learning* и *Journal of Machine Learning Research*.

Упражнения

19.1. Рассмотрим проблему, стоящую перед младенцем, который учится говорить и понимать язык. Объясните, как этот процесс вписывается в общую модель обучения. Опишите восприятие и действия младенца, а также типы обучения, которые младенец должен использовать. Опишите подфункции, которые младенец пытается освоить, с точки зрения входных и выходных данных, а также имеющихся примеров данных.

19.2. Повторите упражнение 19.1 для случая обучения игре в теннис (или обучения какому-то другому виду спорта, с которым вы знакомы). К какому типу оно относится: обучению с учителем или обучению с подкреплением?

19.3. Нарисуйте дерево решений для задачи принятия решения, следует ли двигаться вперед на уличном перекрестке, если на светофоре только что загорелся зеленый свет.

19.4. При движении вдоль одного пути в дереве решений один и тот же атрибут никогда не проверяется дважды. Почему так происходит?

19.5. Предположим, что обучающий набор формируется на основе некоторого дерева решений, а затем этот обучающий набор используется для обучения нового дерева решений. Можно ли рассчитывать, что в этих условиях обучающий алгоритм в конечном итоге возвратит правильное дерево, по мере того как размер обучающего набора будет стремиться к бесконечности? Почему да или почему нет?

19.6. При рекурсивном формировании деревьев решений иногда случается так, что в листовом узле остается смешанное множество положительных и отрицательных примеров, несмотря даже на то, что были использованы все атрибуты. Предположим, что количество положительных примеров равно p, а количество отрицательных примеров — n.

а) Покажите, что решение, используемое в алгоритме DECISION-TREE-LEARNING, в котором выбирается мажоритарная классификация, минимизирует абсолютную ошибку на множестве примеров в листовом узле.

б) Покажите, что **вероятность класса** $p/(p+n)$ минимизирует сумму квадратичных ошибок.

19.7. Предположим, что некоторый атрибут разбивает множество примеров E на подмножества E_k и что каждое подмножество включает p_k положительных примеров и n_k отрицательных примеров. Покажите, что этот атрибут обеспечивает строго положительное приращение информации, если отношение $p_k/(p_k+n_k)$ не является одинаковым для всех k.

19.8. Рассмотрим следующий набор данных, состоящий из трех булевых входных атрибутов (A_1, A_2, A_3) и одного булева выходного атрибута y.

Пример	A_1	A_2	A_3	Выход y
$\mathbf{x}_1$	1	0	0	0
$\mathbf{x}_2$	1	0	1	0
$\mathbf{x}_3$	0	1	0	0
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	1	1	0	1

Используйте представленный на рис. 19.5 алгоритм для обучения дерева решений на этих данных. Приведите вычисления, сделанные для определения атрибута, подлежащего разделению в каждом узле.

19.9. Создайте набор данных (множество примеров с атрибутами и классификациями), который заставит алгоритм обучения дерева решений найти дерево не минимального размера. Нарисуйте дерево, построенное с помощью алгоритма, и дерево минимального размера, которое можно сгенерировать вручную.

19.10. *Граф* решений — это обобщение дерева решений, в котором узлам (т.е. атрибутам, используемым для разделения) разрешается иметь несколько родительских узлов, а не один, как в случае дерева решений. Полученный граф все равно должен быть ациклическим. Рассмотрим функцию XOR для *трех* двоичных входных атрибутов, которая имеет на выходе значение 1 тогда и только тогда, когда нечетное число из трех входных атрибутов имеет значение 1.

а) Нарисуйте *дерево* решений минимального размера для функции XOR с тремя входами.

б) Нарисуйте *граф* решений минимального размера для функции XOR с тремя входами.

19.11. В этом упражнении рассматривается χ^2-обрезка деревьев решений (раздел 19.3.4).

а) Создайте набор данных с двумя входными атрибутами — такой, что приращение информации в корне дерева для обоих атрибутов будет равно нулю, но существует дерево решений глубиной 2, совместимое со всеми данными. Что сделает с этим набором данных χ^2-обрезка при применении в направлении снизу вверх? А в направлении сверху вниз?

б) Измените алгоритм DECISION-TREE-LEARNING так, чтобы включить в него χ^2-обрезку. Необходимые подробности можно найти в статье Квинлиана [1835] (1986).

19.12. В стандартном алгоритме DECISION-TREE-LEARNING, описанном в данной главе, не учитываются такие случаи, когда отдельные примеры имеют атрибуты с отсутствующими значениями.

а) Прежде всего необходимо найти способ классификации таких примеров, если дано дерево решений, включающее проверки по атрибутам, для которых могут отсутствовать значения. Предположим, что в некотором примере $\mathbf{x}$ отсутствует значение для атрибута A и что в дереве решений выполняется проверка атрибута A в узле, достигнутом примером $\mathbf{x}$. Один из способов справиться с такой ситуацией состоит в том, чтобы предположить, что в этом примере заданы все возможные значения для данного атрибута, но взвесить каждое значение с учетом частоты его появления среди всех примеров, достигших этого узла в дереве решений. Алгоритм классификации должен проследовать по всем ветвям в каждом узле, для которого отсутствует некоторое значение, и умножить полученные значения на веса вдоль каждого пути. Напишите модифицированный алгоритм классификации для деревьев решений, действующий по такому принципу.

б) Теперь следует внести изменения в расчеты приращения информации таким образом, чтобы в любой конкретной коллекции примеров C в любом конкретном узле дерева в процессе его формирования в качестве значений любого из оставшихся атрибутов присваивались "предполагаемые" величины, соответствующие частотам этих значений в множестве C.

19.13. В разделе 19.3.5 было отмечено, что при использовании атрибутов со многими различными возможными значениями могут возникать проблемы, связанные с применением критерия приращения информации. Такие атрибуты, как правило, разбивают примеры на многочисленные небольшие или даже одноэлементные классы, из-за чего они начинают казаться в высшей степени релевантными в соответствии с критерием приращения информации. Критерий **коэффициента приращения** позволяет выбирать атрибуты с учетом отношения между соответствующим им приростом и свойственным им информационным содержанием, т.е. количеством информации, содержащемся в ответе на вопрос "Каково значение этого атрибута?" Поэтому при использовании критерия коэффициента приращения предпринимается попытка измерить, насколько полно этот атрибут предоставляет информацию о правильной классификации некоторого примера. Запишите математическое выражение для информационного содержания атрибута и реализуйте критерий коэффициента приращения в алгоритме DECISION-TREE-LEARNING.

19.14. Предположим, что проводится эксперимент по обучению с помощью нового алгоритма булевой классификации. Имеется набор данных, состоящий из 100 положительных и 100 отрицательных примеров. Планируется использовать перекрестную проверку по отдельным объектам и сравнить алгоритм с базовой функцией — простым мажоритарным классификатором. (Мажоритарный классификатор получает на входе обучающий набор данных, а затем всегда выводит тот класс, который имел большинство в обучающем наборе, независимо от входных данных.) Предполагается, что мажоритарный классификатор должен обеспечить получение оценки около 50% при перекрестной проверке по отдельным объектам, но, к вашему удивлению, он выдает нулевую оценку. Можете ли вы объяснить причины, по которым получен такой результат?

19.15. Предположим, что в обучающем алгоритме предпринимается попытка найти совместимую гипотезу, когда классификации примеров фактически являются случайными. Имеется n булевых атрибутов, и примеры с равномерным распределением выбираются из набора в 2^n возможных примеров. Рассчитайте количество примеров, которые требуется выбрать, прежде чем вероятность обнаружения противоречия в данных достигнет 0,5.

19.16. Создайте *список решений* для классификации приведенных ниже данных. Выбирайте тесты так, чтобы они были как можно меньше (в терминах атрибутов), разрывая связи между тестами с одинаковым количеством атрибутов посредством выбора того, который правильно классифицирует наибольшее количество примеров. Если несколько тестов имеют одинаковое количество атрибутов и классифицируют одинаковое количество примеров, разрывая связь, используйте

атрибуты с более низкими значениями индексов (например, выберите A_1 вместо A_2).

Пример	A_1	A_2	A_3	A_4	Выход y
$\mathbf{x}_1$	1	0	0	0	1
$\mathbf{x}_2$	1	0	1	1	1
$\mathbf{x}_3$	0	1	0	0	1
$\mathbf{x}_4$	0	1	1	0	0
$\mathbf{x}_5$	1	1	0	1	1
$\mathbf{x}_6$	0	1	0	1	0
$\mathbf{x}_7$	0	0	1	1	1
$\mathbf{x}_8$	0	0	1	0	0

19.17. Докажите, что список решений может представлять ту же функцию, что и дерево решений, с использованием количества правил, не превышающего количества листьев в дереве решений для этой функции. Приведите пример функции, представленной списком решений, использующим строго меньшее количество правил, чем количество листьев в дереве решений минимального размера для той же самой функции.

19.18. Это упражнение связано с выразительностью списков решений (см. раздел 19.5.1).

а) Покажите, что списки решений позволяют представить любую булеву функцию, если размер проверок не ограничен.

б) Покажите, что если проверки могут включать самое большее k литералов каждая, то списки решений позволяют представить любую функцию, которая может быть представлена с помощью дерева решений с глубиной k.

19.19. Предположим, что при регрессионном поиске методом 7 ближайших соседей было получено множество $\{7, 6, 8, 4, 7, 11, 100\}$ как 7 ближайших значений y для данного значения x. Какое значение $\hat{y}$ минимизирует функцию потерь L_1 для этих данных? В статистике для этого значения есть общее название как функции от значений y. Какое именно? Дайте ответ на те же два вопроса для функции потерь L_2.

19.20. Предположим, что при регрессионном поиске методом 7 ближайших соседей было получено множество $\{4, 2, 8, 4, 9, 11, 100\}$ как 7 ближайших значений y для данного значения x. Какое значение $\hat{y}$ минимизирует функцию потерь L_1 для этих данных? В статистике для этого значения есть общее название как функции от значений y. Какое именно? Дайте ответ на те же два вопроса для функции потерь L_2.

19.21. На рис. 19.22 показано, как граница решения в виде окружности в плоскости координат может быть представлена линейным разделением путем отображения характеристик (x_1, x_2) на два измерения (x_1^2, x_2^2). Но что если эта окружность не находится в плоскости координат? Что если это будет эллипс, а не окружность? Общее уравнение для окружности (и следовательно, данной границы решения) — это $(x_1 - a)^2 + (x_2 - b)^2 - r^2 = 0$, а общее уравнение для эллипса — $c(x_1 - a)^2 + d(x_2 - b)^2 - 1 = 0$.

а) Разверните уравнение окружности и покажите, какими будут веса w_i для границы решения в четырехмерном пространстве признаков (x_1, x_2, x_1^2, x_2^2). Объясните, почему это означает, что любая окружность линейно разделима в этом пространстве.

б) Сделайте то же самое для эллипсов в пятимерном пространстве признаков $(x_1, x_2, x_1^2, x_2^2, x_1x_2)$.

19.22. Постройте машину опорных векторов, вычисляющую функцию XOR. Используйте значения +1 и −1 (вместо 1 и 0) как для входов, так и для выходов, чтобы пример выглядел как ([−1, 1], 1) или ([−1, −1], −1). Отобразите вход $[x_1, x_2]$ в пространство, состоящее из x_1 и x_1x_2. Нарисуйте четыре точки входов в этом пространстве и оптимально разделяющий классификатор. Каким будет зазор? Теперь нарисуйте эту разделяющую линию в исходном евклидовом пространстве ввода.

19.23. Рассмотрим алгоритм ансамблевого обучения, в котором используется простое большинство голосов среди K изученных гипотез. Предположим, что каждая гипотеза имеет ошибку ϵ и что ошибка каждой из гипотез не зависит от других гипотез. Вычислите формулу ошибки ансамблевого алгоритма в терминах K и ϵ и оцените ее для случаев, когда $K = 5$, 10 и 20, а $\epsilon = 0{,}1$, 0,2 и 0,4. Если исключить предположение о независимости, может ли ошибка ансамбля оказаться *хуже*, чем ϵ?

19.24. Рассмотрим проблему разделения N точек данных на положительные и отрицательные примеры с помощью линейного разделителя. Ясно, что это всегда можно сделать для $N = 2$ точек на линии (размерность пространства $d = 1$) независимо от того, как точки помечены или где они расположены (если только точки не находятся в одном и том же месте).

а) Покажите, что это всегда можно сделать для $N = 3$ точек на плоскости (размерность пространства $d = 2$), если они не лежат на одной прямой.

б) Покажите, что это не всегда можно сделать для $N = 4$ точек на плоскости (размерность пространства $d = 2$).

в) Покажите, что это всегда можно сделать для $N = 4$ точек в пространстве размерности $d = 3$, если они не компланарны.

г) Покажите, что это не всегда можно сделать для $N = 5$ точек в пространстве размерности $d = 3$.

д) Амбициозный студент при желании может попробовать доказать, что N точек в обычном расположении (но не $N + 1$) линейно разделимы в пространстве размерности $N - 1$.

ГЛАВА 20

Статистические методы обучения

В этой главе обучение рассматривается как форма рассуждений в условиях неопределенности на основании результатов наблюдений и разрабатываются модели для представления неопределенного мира.

В главе 12 было указано, что в реальных средах всегда господствует неопределенность. Агенты могут справиться с неопределенностью, используя методы теории вероятностей и теории принятия решений, но сначала они должны изучить свои вероятностные теории на основании полученного опыта. В этой главе объясняется, как можно достичь такой цели, сформулировав саму задачу обучения как процесс вероятностного вывода (раздел 20.1). Будет показано, что байесовский подход к обучению является чрезвычайно мощным — он предоставляет общее решение для проблем шума, переобучения и оптимального прогнозирования. В нем также учитывается тот факт, что агент, не обладающий абсолютным всезнанием, может никогда не достигнуть полной уверенности в том, какая именно теория о мире является правильной, но все равно обязан принимать решения, используя некоторую теорию о мире.

В разделах 20.2 и 20.3 описаны методы обучения вероятностных моделей, в основном — байесовских сетей. Часть материала этой главы имеет богатое математическое содержание, хотя основные приведенные здесь научные результаты можно понять, не углубляясь в детали. Читателю может оказаться полезным еще раз просмотреть главы 12 и 13, а при необходимости заглянуть и в приложение А.

20.1. Статистическое обучение

Основными понятиями в данной главе, как и в главе 19, являются **данные** и **гипотезы**. Но в этой главе данные рассматриваются как **свидетельства**, т.е. конкретизация некоторых или всех случайных переменных, описывающих проблемную область. В свою очередь, гипотезы в этой главе представляют собой

вероятностные теории того, как функционирует проблемная область, включающие логические теории в качестве частного случая.

Рассмотрим простой пример. Любимые многими леденцы "Сюрприз" выпускаются двух вкусов: вишни (*cherry*) — сладкие (ням-ням) и лайма (*lime*) — кислые (фу-фу). Изготовитель леденцов имеет особое чувство юмора, поэтому обертка у всех конфет одинаковая, из непрозрачной бумаги и не зависит от вкуса. Леденцы продаются в очень больших пакетах (также не имеющих внешних отличий), содержимое которых может быть представлено в пяти следующих вариантах:

h_1: 100% *cherry*,
h_2: 75% *cherry* + 25% *lime*,
h_3: 50% *cherry* + 50% *lime*,
h_4: 25% *cherry* + 75% *lime*,
h_5: 100% *lime*.

Рассматривая новый пакет леденцов, обозначим его тип случайной переменной H (от слова *hypothesis*) с пятью возможными значениями от h_1 до h_5. Понятно, что переменная H не является непосредственно наблюдаемой. По мере развертывания и проверки вкуса леденцов регистрируются данные о них — $D_1, D_2, \ldots, D_N$, где D_i представляет собой случайную переменную с двумя возможными значениями: *cherry* или *lime*. Поставленная перед агентом задача заключается в том, что он должен предсказать, какой вкус будет у очередного леденца.[1] Несмотря на кажущуюся простоту, этот сценарий позволяет ознакомиться со многими важными вопросами. В действительности агент здесь должен логическим путем вывести теорию о мире, в котором он существует, пусть даже очень простую.

В ▶**байесовском обучении**, исходя из полученных данных, просто вычисляется вероятность каждой гипотезы и на этой основе делаются предсказания. Это означает, что предсказания составляются с использованием *всех* гипотез, взвешенных по их вероятностям, а не с применением только одной "наилучшей" гипотезы. Следовательно, обучение здесь сводится к вероятностному выводу.

Пусть **D** представляет все данные с наблюдаемым значением **d**. Ключевыми величинами в байесовском подходе являются ▶**распределение априорных вероятностей гипотез** (*hypothesis prior*), $P(h_i)$, и ▶**правдоподобие** (*likelihood*) данных согласно каждой гипотезе, $P(\mathbf{d} \mid h_i)$. Вероятность каждой гипотезы может быть определена по правилу Байеса:

$$P(h_i \mid \mathbf{d}) = \alpha P(\mathbf{d} \mid h_i) P(h_i). \tag{20.1}$$

Теперь предположим, что необходимо сделать прогноз в отношении неизвестного количества X. В таком случае применяется следующее уравнение:

[1] Читатели, искушенные в статистике, узнают в этом сценарии один из вариантов постановки задачи с урнами и шарами. Но авторы считают урны и шары менее привлекательными, чем леденцы.

$$\mathbf{P}(X \mid \mathbf{d}) = \sum_i \mathbf{P}(X \mid h_i) P(h_i \mid \mathbf{d}), \tag{20.2}$$

где каждая гипотеза определяет распределение вероятностей по X. Это уравнение показывает, что предсказания представляют собой взвешенные средние по предсказаниям отдельных гипотез, где вес $P(h_i \mid \mathbf{d})$ пропорционален априорной вероятности гипотезы h_i и степени ее соответствия согласно уравнению (20.1). Сами гипотезы, по сути, являются "посредниками" между фактическими данными и предсказаниями.

Для нашего примера с леденцами предположим, что на данный момент априорное распределение вероятностей по $h_1,\ldots, h_5$ определяется вектором $\langle 0,1;\ 0,2;\ 0,4;\ 0,2;\ 0,1\rangle$, как следует из рекламы производителя. Правдоподобие данных рассчитывается в соответствии с предположением, что наблюдения характеризуются свойством **i.i.d.**, т.е. являются *независимыми* и *одинаково распределенными* (см. раздел 19.4), поэтому выполняется следующее соотношение:

$$P(\mathbf{d} \mid h_i) = \prod_j P(d_j \mid h_i). \tag{20.3}$$

Например, предположим, что полученный пакет в действительности относится к типу, когда все леденцы — лайм (h_5) и все первые 10 проверенных конфет, конечно же, оказались этого вкуса. Тогда значение $P(\mathbf{d} \mid h_3)$ будет равно $0{,}5^{10}$, поскольку в пакете типа h_3 только половина леденцов имеет вкус лайма.[2] На рис. 20.1, *а* показано, как изменяются апостериорные вероятности для каждой из пяти гипотез по мере наблюдения последовательности из 10 леденцов со вкусом лайма. Обратите внимание, что кривые вероятностей начинаются с их априорных значений, поэтому первоначально наиболее вероятным вариантом является гипотеза h_3 и остается таковой после проверки первого леденца, оказавшегося со вкусом лайма. После проверки двух конфет с одинаковым вкусом лайма наиболее вероятной становится гипотеза h_4, а после проверки трех и более леденцов с одинаковым кислым вкусом наиболее вероятной становится гипотеза h_5 (ужасный пакет, содержащий только кислые леденцы со вкусом лайма). После обнаружения 10 подряд таких леденцов можно почти с полной уверенностью сожалеть о злосчастной судьбе. На рис. 20.1, *б* приведена прогнозируемая вероятность того, что следующий леденец будет иметь вкус лайма согласно уравнению (20.2). Как и следовало ожидать, она монотонно увеличивается к пределу 1.

[2] Выше было указано, что пакеты с леденцами — очень большие, так как в противном случае предположение i.i.d. не соблюдалось бы. Формально было бы более правильно (но менее гигиенично) снова заворачивать каждую конфету в бумагу после ее осмотра и возвращать в пакет.

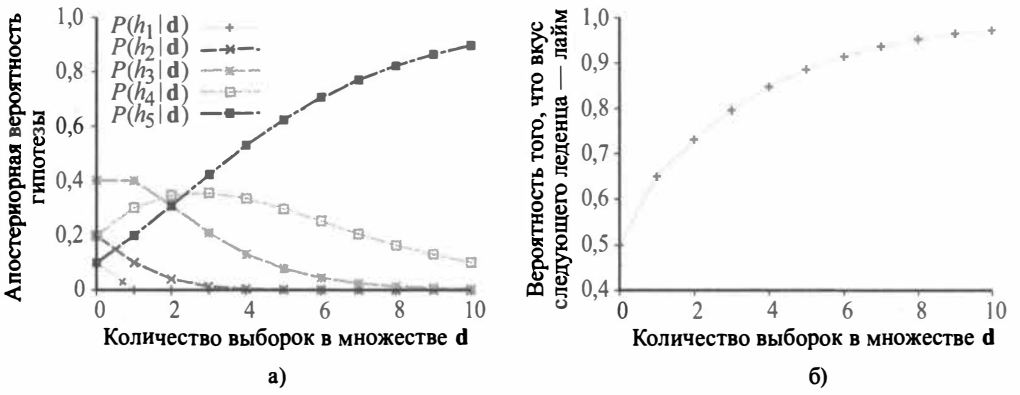


Рис. 20.1. а) Апостериорные вероятности $P(hi \mid d_1,\ldots, d_N)$, полученные с помощью уравнения (20.1). Количество наблюдений N возрастает от 1 до 10, и в каждом из них обнаруживается леденец со вкусом лайма. **б)** Байесовские предсказания $P(D_{N+1} = lime \mid d_1,\ldots, d_N)$, полученные из уравнения (20.2)

Этот пример показывает, что ➤ *байесовское предсказание в конечном счете согласуется с истинной гипотезой.* Это характерная особенность байесовского обучения. При любом заданном распределении априорных вероятностей, которое не исключает с самого начала истинную гипотезу, апостериорная вероятность любой ложной гипотезы при определенных формальных условиях в конечном итоге полностью исчезает. Так происходит просто потому, что вероятность неопределенно долгого формирования "нехарактерных" данных исчезающе мала. (Это замечание аналогично тому, которое было сделано при обсуждении PAC-обучения в главе 19.) Еще более важно то, что байесовское предсказание является *оптимальным* независимо от того, какой набор данных применяется — большой или маленький. При наличии распределения априорных вероятностей гипотез все другие предсказания реже будут правильными.

Но, безусловно, за оптимальность байесовского обучения приходится платить. В реальных задачах обучения пространство гипотез обычно является очень большим или бесконечным, как было показано в главе 19. В некоторых случаях операция вычисления суммы в уравнении (20.2) (или операция интегрирования в непрерывном случае) может быть успешно выполнена, но в большинстве случаев приходится прибегать к приближенным или упрощенным методам.

Один из широко распространенных приближенных подходов — из числа тех, которые обычно применяются в научных исследованиях — состоит в том, чтобы делать предсказания на основе единственной *наиболее вероятной* гипотезы, т.е. той гипотезы h_i, которая максимизирует значение $P(h_i \mid \mathbf{d})$. Такую гипотезу часто называют ➤ **максимальной апостериорной гипотезой** (*maximum*

a posteriori), или сокращенно гипотезой MAP (произносится "эм-эй-пи"). Предсказания h_{MAP}, сделанные на основе MAP-гипотезы, являются приближенно байесовскими в пределах, пока $\mathbf{P}(X|\,\mathbf{d}) \approx \mathbf{P}(X|\,h_{MAP})$. В рассматриваемом примере с леденцами $h_{MAP} = h_5$ после обнаружения трех леденцов со вкусом лайма подряд, поэтому агент, обучающийся с помощью MAP-гипотезы, после этого предсказывает, что четвертая конфета представляет собой такой же леденец с вероятностью 1,0, а это гораздо более рискованное предсказание, чем байесовское предсказание вероятности 0,8, приведенное на рис. 20.1. *б*. По мере поступления дополнительных данных предсказания с помощью MAP-гипотезы и байесовские предсказания сближаются, поскольку появление гипотез, конкурирующих с MAP-гипотезой, становится все менее и менее вероятным.

Хотя в рассматриваемом примере это не представлено, поиск MAP-гипотез часто намного проще по сравнению с байесовским обучением, поскольку требует решения задачи оптимизации, а не задачи вычисления большой суммы (или интегрирования).

И в байесовском обучении, и в обучении с помощью MAP-гипотез важную роль играет распределение априорных вероятностей гипотез $P(h_i)$. Как было показано в главе 18, если пространство гипотез является слишком выразительным в том смысле, что содержит много гипотез, хорошо согласующихся с набором данных, то может иметь место **переобучение**. Методы байесовского и MAP-обучения используют распределение априорных вероятностей для *наложения штрафа за сложность*. Как правило, более сложные гипотезы имеют более низкую априорную вероятность, — отчасти потому, что сложных гипотез обычно намного больше, чем простых. С другой стороны, более сложные гипотезы обладают бо́льшей способностью согласования с данными. (В пределе поисковая таблица может точно воспроизводить данные.) Следовательно, в распределении априорных вероятностей гипотез воплощен компромисс между сложностью гипотезы и степенью ее согласования с данными.

Эффект от этого компромисса наиболее наглядно проявляется в случае логических гипотез, когда переменная H содержит только *детерминированные* гипотезы (такие, как гипотеза h_1, утверждающая, что каждый леденец будет иметь вкус вишни). Для этого случая значение $P(\mathbf{d} \mid h_i)$ равно 1, если гипотеза h_i согласуется с данными, и 0 — в противном случае. Проанализировав уравнение (20.1), можно прийти к заключению, что гипотеза h_{MAP} в таком случае представляет собой ➔ *простейшую логическую теорию, согласованную с данными*. Следовательно, обучение с помощью максимальной апостериорной гипотезы представляет собой естественное воплощение принципа бритвы Оккама.

Еще один способ анализа компромисса между сложностью и степенью согласованности состоит во взятии логарифма обеих частей уравнения (20.1). Тогда выбор значения h_{MAP} для максимизации выражения $P(\mathbf{d} \mid h_i)P(h_i)$ будет эквивалентен минимизации выражения

$$-\log_2 P(\mathbf{d} \mid h_i) - \log_2 P(h_i).$$

Используя связь между информационным содержанием и вероятностью, которая была введена в разделе 19.3.3, можно понять, что терм $-\log_2 P(h_i)$ определяет количество битов, требуемых для задания гипотезы h_i. Далее, терм $-\log_2 P(\mathbf{d} \mid h_i)$ представляет собой дополнительное количество битов, требуемых для задания данных, если дана рассматриваемая гипотеза. (Чтобы убедиться в этом, достаточно отметить, что если гипотеза точно предсказывает данные, как в случае гипотезы h_5 и непрерывного ряда леденцов со вкусом лайма, $\log_2 1 = 0$.) Таким образом, MAP-обучение равносильно выбору гипотезы, которая обеспечивает максимальное сжатие данных. Такую же задачу можно решить более прямо с помощью метода обучения на основе **минимальной длины описания**, или сокращенно — MDL (*minimum description length*). В то время как MAP-обучение выражает простоту посредством присвоения более высокой вероятности более простым гипотезам, в методе MDL она выражается прямым подсчетом битов в двоичной кодировке гипотезы и данных.

Окончательное упрощение может быть достигнуто путем принятия предположения о **равномерном** распределении априорных вероятностей по пространству гипотез. В этом случае MAP-обучение сводится к выбору гипотезы h_i, которая максимизирует значение $P(\mathbf{d} \mid h_i)$. Такую гипотезу называют гипотезой с ▶**максимальным правдоподобием** (*maximum likelihood* — ML) и обозначают как h_{ML}. Обучение на основе максимального правдоподобия очень широко применяется в статистике — научной области, в которой многие исследователи не доверяют субъективной природе распределения априорных вероятностей гипотез. Это разумный подход в тех случаях, когда нет оснований *априорно* отдавать предпочтение одной гипотезе перед другой, например, когда все гипотезы являются одинаково сложными.

Когда набор данных велик, априорное распределение вероятностей гипотез становится менее важным — свидетельство самих данных является достаточно сильным, чтобы отодвинуть на второй план предварительное распределение по гипотезам. А это означает, что обучение по максимальному правдоподобию является хорошим приближением к байесовскому и MAP-обучению в случае больших наборов данных, но подвержено определенным проблемам (как будет показано ниже) при использовании небольших наборов данных.

20.2. Обучение при полных данных

Общая задача обучения вероятностной модели, исходя из данных, которые, как предполагается, были сгенерированы по этой модели, называется ▶**оценкой плотности**. (Первоначально этот термин применяли к функциям плотности вероятности для непрерывных переменных, но в настоящее время он используется и для дискретных распределений.) Оценка плотности является формой обучения без учителя. В этом разделе рассматривается простейший случай, когда имеются

► **полные данные**. Данные являются полными, когда каждая точка данных содержит значения для каждой переменной в обучаемой вероятностной модели. В этом разделе мы сосредоточимся на ► **параметрическом обучении** — нахождении числовых параметров вероятностной модели, имеющей фиксированную структуру. Например, может представлять интерес определение значений условных вероятностей в байесовской сети с заданной структурой. Помимо этого, кратко рассматривается задача обучения структуры и оценка непараметрической плотности.

20.2.1. Параметрическое обучение методом максимального правдоподобия для дискретных моделей

Предположим, что был куплен пакет леденцов двух вкусов, вишни и лайма, но уже от нового производителя, для которого пропорция леденцов разных вкусов в пакете совершенно неизвестна: доля вишневых леденцов может быть равна любому значению между 0 и 1. В этом случае придется рассматривать континуум гипотез. Далее, пусть **параметром**, который мы обозначим как θ, будет доля леденцов со вкусом вишни, а гипотезой будет h_θ. (Долю леденцов со вкусом лайма можно выразить как $1-\theta$.) Если принять предположение, что все возможные пропорции между двумя видами конфет в пакете *априорно* равновероятны, то выбор подхода на основе максимального правдоподобия будет обоснованным. Чтобы смоделировать эту ситуацию с помощью байесовской сети, нам потребуется только одна случайная переменная, *Flavor* (вкус леденца, случайным образом выбранного из пакета), и эта переменная может принимать только два значения, *cherry* и *lime*, где вероятность значения *cherry* будет равна θ (рис. 20.2, *а*). Теперь предположим, что было выбрано и развернуто N леденцов, из которых c леденцов имели вкус вишни, а $\ell = N - c$ — вкус лайма. Согласно уравнению (20.3), правдоподобие этого конкретного набора данных выражается следующей формулой:

$$P(\mathbf{d} \mid h_\theta) = \prod_{j=1}^{N} P(d_j \mid h_\theta) = \theta^c \cdot (1-\theta)^\ell.$$

Гипотеза с максимальным правдоподобием задается значением θ, которое максимизирует это выражение. Поскольку логарифмическая функция является монотонной, такое же значение может быть найдено путем максимизации ► **логарифмического правдоподобия** (*log likelihood*):

$$L(\mathbf{d} \mid h_\theta) = \log P(\mathbf{d} \mid h_\theta) = \sum_{j=1}^{N} \log P(d_j \mid h_\theta) = c \log\theta + \ell \log(1-\theta).$$

(После взятия логарифмов произведение было преобразовано в сумму по данным, поскольку ее обычно проще максимизировать.) Чтобы найти значение максимального правдоподобия, т.е. параметр θ, продифференцируем L по θ и приравняем полученное выражение к нулю:

$$\frac{dL(\mathbf{d}\,|\,h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0, \qquad \Rightarrow \qquad \theta = \frac{c}{c+\ell} = \frac{c}{N}.$$

Таким образом, если описать это выражение обычным языком, то гипотеза с максимальным правдоподобием h_{ML} утверждает, что фактическая доля леденцов со вкусом вишни в пакете равна наблюдаемой доле этих леденцов среди конфет, развернутых до сих пор!

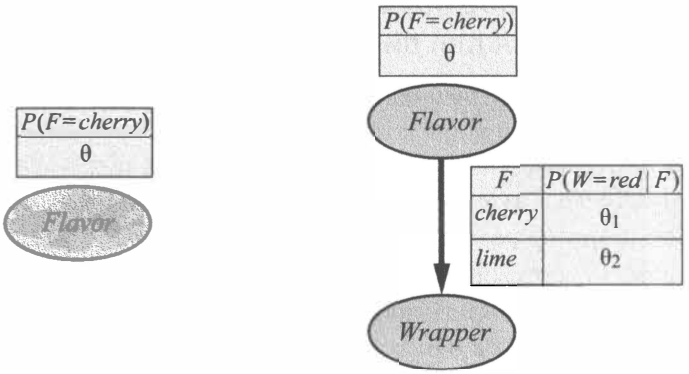


Рис. 20.2. а) Модель в виде байесовской сети для случая, когда пропорция леденцов со вкусом вишни и со вкусом лайма в пакете неизвестна. **б)** Модель для случая, когда цвет обертки связан (вероятностной) зависимостью со вкусом леденца

На первый взгляд создается впечатление, что было проделано немало работы лишь для того, чтобы открыть этот очевидный факт. Но в действительности описанным выше путем был создан один из стандартных методов параметрического обучения с максимальным правдоподобием, который широко применяется на практике.

1. Записать выражение для правдоподобия данных как функцию от параметра (параметров).
2. Определить производную логарифмического правдоподобия по отношению к каждому параметру.
3. Найти такие значения параметров, при которых производные станут равны нулю.

Как правило, самым сложным является последний из этих этапов. В рассматриваемом примере он был тривиален, но ниже будет показано, что во многих случаях на этом этапе приходится прибегать к использованию итерационных алгоритмов поиска решений или других числовых методов оптимизации, которые

были описаны в разделе 4.2. (Потребуется убедиться, что матрица Гессе отрицательно определена.) Этот пример также иллюстрирует важную проблему, которая в общем случае характерна для обучения методом максимального правдоподобия: ➨ *если набор данных достаточно мал и поэтому некоторые события еще не наблюдались — например, не было обнаружено ни одного леденца с вишневым вкусом, — то гипотеза с максимальным правдоподобием присваивает этим событиям нулевую вероятность.* Для предотвращения возникновения этой проблемы использовались различные приемы, такие как инициализация счетчиков для каждого события значением 1, а не 0.

Рассмотрим еще один пример. Предположим, что этот новый изготовитель конфет решил дать потребителю небольшую подсказку, заворачивая леденцы в обертки красного (*red*) и зеленого (*green*) цветов. Значение переменной *Wrapper*, соответствующей цвету обертки каждой конфеты, выбирается по *вероятностным* законам в соответствии с некоторым неизвестным условным распределением, но в зависимости от вкуса леденца. Соответствующая вероятностная модель показана на рис. 20.2, *б*. Обратите внимание, что она имеет три параметра: θ, θ_1 и θ_2. При их использовании правдоподобие события, связанного, скажем, с обнаружением леденца со вкусом вишни в зеленой обертке, можно определить на основе стандартной семантики для байесовских сетей (раздел 13.2):

$$P(Flavor = cherry, Wrapper = green \mid h_{\theta,\theta_1,\theta_2}) =$$
$$= P(Flavor = cherry \mid h_{\theta,\theta_1,\theta_2})\, P(Wrapper = green \mid Flavor = cherry, h_{\theta,\theta_1,\theta_2}) =$$
$$= \theta \cdot |\, (1-\theta_1).$$

Теперь допустим, что было развернуто N конфет, из которых c оказались со вкусом вишни, а ℓ — со вкусом лайма. При этом количество оберток оказалось таковым: r_c леденцов с вишневым вкусом имели красные обертки, а g_c — зеленые, в свою очередь, r_ℓ леденцов со вкусом лайма имели красные обертки, а g_ℓ — зеленые. Правдоподобие этих данных выражается следующим образом:

$$P(\mathbf{d} \mid h_{\theta,\theta_1,\theta_2}) = \theta^c (1-\theta)^\ell \cdot \theta_1^{r_c} (1-\theta_1)^{g_c} \cdot \theta_2^{r_\ell} (1-\theta_2)^{g_\ell}.$$

На первый взгляд это выглядит весьма сложно, но выражение можно упростить, взяв логарифмы:

$$L = [c \log\theta + \ell \log(1-\theta)] + [r_c \log\theta_1 + g_c \log(1-\theta_1)] +$$
$$+ [r_\ell \log\theta_2 + g_\ell \log(1-\theta_2)].$$

Преимущество от взятия логарифмов очевидно — логарифмическое правдоподобие представляет собой сумму трех термов, каждый из которых содержит только один параметр. После взятия производных по каждому параметру и приравнивания их к нулю будет получено три независимых уравнения, каждое из которых содержит только один параметр:

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0, \quad \Rightarrow \quad \theta = \frac{c}{c+\ell};$$
$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0, \quad \Rightarrow \quad \theta_1 = \frac{r_c}{r_c + g_c};$$
$$\frac{\partial L}{\partial \theta_2} = \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0, \quad \Rightarrow \quad \theta_2 = \frac{r_\ell}{r_c + g_\ell}.$$

Решение для θ осталось таким же, как и прежде. Решение для θ_1, вероятности того, что леденец со вкусом вишни имеет красную обертку, представляет собой наблюдаемую долю таких леденцов в красных обертках, и аналогичным образом определяется решение для θ_2.

Полученные результаты очень удобны, и легко показать, что их можно распространить на любую байесовскую сеть, условные вероятности в которой представлены в виде таблиц. Наиболее важный вывод состоит в том, что ➔ *при наличии полных данных задача параметрического обучения методом максимального правдоподобия для байесовской сети декомпонуется на отдельные задачи обучения, по одной для каждого параметра.* (См. упражнение 20.7 для случая, когда на каждый параметр одновременно влияет несколько условных вероятностей, которые нельзя представить таблицами.) Еще один важный вывод состоит в том, что значения параметра для любой переменной при наличии ее родительских значений представляют собой наблюдаемые частоты значений переменных для каждого набора родительских значений. Как и прежде, если набор данных является небольшим, необходимо внимательно следить за предотвращением появления нулевых значений.

20.2.2. Наивные байесовские модели

Вероятно, наиболее распространенной моделью байесовской сети, используемой в машинном обучении, является ►**наивная байесовская модель**, впервые представленная в разделе 12.6. В этой модели переменная "класса" C (значение которой должно быть предсказано) представлена корневым узлом, а переменные "атрибутов" X_i являются листовыми узлами. Такие модели называют "наивными", поскольку в них предполагается, что атрибуты являются условно независимыми друг от друга, если определен рассматриваемый класс. (Модель на рис. 20.2, *б* является наивной байесовской моделью класса *Flavor* только с одним атрибутом *Wrapper*.) При условии, что эти переменные являются булевыми, рассматриваемые параметры принимают такой вид:

$$\theta = P(C = true), \quad \theta_{i1} = P(X_i = true \mid C = true), \quad \theta_{i2} = P(X_i = true \mid C = false).$$

Значения параметров с максимальным правдоподобием можно найти точно таким же способом, как на рис. 20.2, *б*. Когда данная модель будет подобным образом обучена, ее можно будет использовать для классификации новых примеров, в которых переменная класса C является ненаблюдаемой. При известных

наблюдаемых значениях атрибутов $x_1,\ldots,x_n$, вероятность каждого класса определяется следующим соотношением:

$$\mathbf{P}(C \mid x_1,\ldots,x_n) = \alpha \mathbf{P}(C)\prod_i \mathbf{P}(x_i \mid C).$$

Детерминированный прогноз можно получить, выбрав наиболее вероятный класс. На рис. 20.3 показана кривая обучения для этого метода в случае его применения к задаче об ожидании в ресторане из главы 19. Обучение с помощью этого метода проходит довольно успешно, но не так хорошо, как при обучении деревьев решений, — следует полагать, это связано с тем, что истинная гипотеза (представляющая собой дерево решений) не является точно представимой с помощью наивной байесовской модели. Как оказалось, метод наивного байесовского обучения удивительно успешно действует в самых разнообразных приложениях, а его версия с использованием бустинга (упражнение 20.5) является одним из наиболее эффективных алгоритмов обучения общего назначения. Метод наивного байесовского обучения хорошо масштабируется на очень большие задачи: при наличии n булевых атрибутов имеется только $2n+1$ параметров, и ➡ *для нахождения h_{ML}, наивной байесовской гипотезы с максимальным правдоподобием, поиск не требуется.* И наконец, системы с использованием наивного байесовского обучения хорошо работают с зашумленными или неполными данными и при необходимости могут предоставить вероятностные прогнозы даже в таких случаях. Их основным недостатком является тот факт, что предположение об условной независимости параметров редко соблюдается в точности. Как отмечено в разделе 12.6, это предположение ведет к переоценке вероятностей, которые в результате оказываются очень близкими к 0 или 1, особенно при большом количестве атрибутов.

Рис. 20.3. Кривая обучения наивной байесовской модели в применении к задаче об ожидании в ресторане из главы 19. Для сравнения показана кривая обучения дерева решений для той же задачи

20.2.3. Генеративные и дискриминативные модели

Модели машинного обучения, используемые в классификаторах, можно разделить на два вида: генеративные (или *порождающие*) и дискриминативные (или *разделяющие*). ►**Генеративная модель** моделирует распределение вероятностей каждого класса. Например, наивный байесовский классификатор текста из раздела 12.6.1 создает отдельную модель для каждой возможной категории текстов: одну — для спорта, одну — для погоды и т.д. Каждая модель включает априорную вероятность данной категории — например, $P(Category = weather)$, — а также условные вероятности $\mathbf{P}(Inputs \mid Category = weather)$. На основании этих вероятностей можно вычислить совместные вероятности $\mathbf{P}(Inputs, Category = weather)$ и генерировать случайные выборки слов, характеризующих тексты в категории "Погода".

►**Дискриминативная модель** изучает непосредственно границы решения между классами, т.е. она изучает $\mathbf{P}(Category \mid Inputs)$. На основании примеров входных данных дискриминативная модель построит выходную категорию, но дискриминативную модель нельзя использовать, скажем, для генерации случайным образом слов, представляющих категорию. Логистическая регрессия, деревья решений и машины опорных векторов являются дискриминативными моделями.

Поскольку дискриминативные модели делают все возможное для определения границы решения, т.е. фактически выполняют задачу классификации, для которой они и предназначены, они, как правило, лучше работают в пределе при произвольном количестве обучающих данных. Однако при ограниченном количестве данных генеративная модель в некоторых случаях работает лучше. Эндрю Энг и Джордан в статье [1676] (2002) сравнивают генеративный наивный байесовский классификатор с дискриминативным классификатором на основе логистической регрессии на 15 (небольших) наборах данных и приходят к заключению, что при максимальном количестве данных дискриминативная модель работает лучше на 9 из 15 наборов данных, тогда как при небольшом количестве данных генеративная модель работает лучше на 14 из 15 наборов данных.

20.2.4. Параметрическое обучение методом максимального правдоподобия: непрерывные модели

Непрерывные вероятностные модели, такие как **линейная гауссова** модель, описывались ранее, в разделе 13.2.3. Поскольку непрерывные переменные очень широко используются в реальных приложениях, важно знать, как следует осуществлять обучение параметров непрерывных моделей на основе данных. Принципы обучения методом максимального правдоподобия в случае непрерывных и дискретных переменных идентичны.

Начнем с очень простого случая: обучение параметров гауссовой функции плотности от одной переменной. Это означает, что данные генерируются следующим образом:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Параметрами этой модели являются среднее значение μ и стандартное отклонение σ. (Обратите внимание, что нормализующая "константа" зависит от σ, поэтому мы не можем ее игнорировать.) Пусть наблюдаемыми значениями являются $x_1,\ldots,$ x_N. Тогда логарифмическое правдоподобие определяется следующим образом:

$$L = \sum_{j=1}^{N} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}.$$

Приравняв, как обычно, производные к нулю, получим уравнения

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2}\sum_{j=1}^{N}(x_j-\mu) = 0, & \Rightarrow \quad \mu &= \frac{\sum_j x_j}{N}; \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(x_j-\mu)^2 = 0, & \Rightarrow \quad \sigma &= \sqrt{\frac{\sum_j (x_j-\mu)^2}{N}}. \end{aligned} \tag{20.4}$$

Таким образом, значение максимального правдоподобия среднего значения μ представляет собой среднее по выборкам, а значение максимального правдоподобия стандартного отклонения σ выражается квадратным корнем от дисперсии выборки. И вновь получены удобные результаты, подтверждающие обоснованность практических методов, созданных на основе "здравого смысла".

Теперь рассмотрим линейную гауссову модель с одним непрерывным родительским значением X и непрерывным дочерним значением Y. Как было описано в разделе 13.2.3, значение Y имеет гауссово распределение, среднее значение которого линейно зависит от значения X, а стандартное отклонение является постоянным. Чтобы определить в результате обучения распределение условных вероятностей $P(Y|X)$, можно максимизировать условное правдоподобие:

$$P(y \mid x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}}. \tag{20.5}$$

Здесь параметрами являются θ_1, θ_2 и σ. Данные представляют собой множество пар (x_j, y_j), как показано на рис. 20.4. Используя обычные методы (упражнение 20.6), можно найти значения параметров с максимальным правдоподобием. Но здесь нужно сделать еще одно замечание. Если рассматриваются только параметры θ_1 и θ_2, определяющие линейную зависимость между x и y, то становится очевидным, что максимизация логарифмического правдоподобия по отношению к этим параметрам равносильна *минимизации* числителя $(y-(\theta_1 x+\theta_2))^2$ в экспоненте уравнения (20.5). Это L_2-потери, квадратичная ошибка между фактическим значением y и прогнозом $\theta_1 x+\theta_2$.

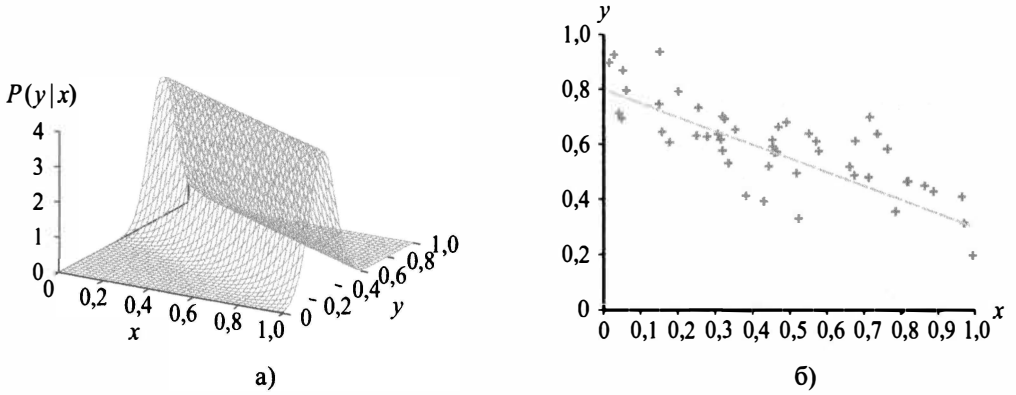


Рис. 20.4. а) Линейная гауссова модель, описываемая как $y = \theta_1 x + \theta_2$, к которой добавлен гауссов шум с постоянной дисперсией. **б)** Множество из 50 точек данных, сгенерированных с помощью этой модели, и линия наибольшего соответствия

Эту величину можно минимизировать с помощью стандартной процедуры **линейной регрессии**, как описывается в разделе 19.6. Теперь можно понять, с чем это связано: минимизация суммы квадратичных ошибок позволяет получить линейную модель с максимальным правдоподобием *при условии, что данные генерировались с гауссовым шумом, имеющим постоянную дисперсию.*

20.2.5. Байесовское параметрическое обучение

Метод обучения с максимальным правдоподобием может стать основой простых процедур, но обнаруживает серьезные недостатки при работе с небольшими наборами данных. Например, после обнаружения одного леденца с вишневым вкусом в этом методе вырабатывается гипотеза с максимальным правдоподобием, согласно которой данный пакет на 100% состоит только из таких леденцов (т.е. $\theta = 1{,}0$). Но если принятое распределение априорных вероятностей гипотез не сводится к тому, что в пакетах должны находиться лишь леденцы с вишневым вкусом либо исключительно леденцы со вкусом лайма, то подобное заключение является необоснованным: скорее всего, в пакете находится смесь леденцов обоих вкусов. Байесовский подход к параметрическому обучению начинается с априорных вероятностей, и распределение обновляется по мере поступления данных.

В примере с леденцами, приведенном на рис. 20.2, *а*, имеется только один параметр θ — вероятность того, что случайно выбранная конфета относится к леденцам с вишневым вкусом. С точки зрения байесовского подхода θ представляет (неизвестное) значение случайной переменной Θ, определяющей пространство гипотез, но тогда распределение априорных вероятностей гипотез представляет

собой распределение априорных вероятностей $\mathbf{P}(\Theta)$. Следовательно, $P(\Theta = \theta)$ — это априорная вероятность того, что в пакете доля леденцов с вишневым вкусом равна θ.

Если параметр θ может иметь любое значение от 0 до 1, то априорное распределение $\mathbf{P}(\Theta)$ должно представлять собой непрерывную функцию плотности вероятности (см. раздел А.3). Если о возможных значениях θ ничего неизвестно, можно использовать функцию плотности вероятности для равномерного распределения $P(\theta) = \mathit{Uniform}(\theta; 0, 1)$, в котором все значения одинаково вероятны.

Более гибкое семейство функций плотности вероятности известно как ▶**бета-распределение**. Каждое бета-распределение определяется двумя ▶**гиперпараметрами**,[3] a и b, такими, что в диапазоне значений [0, 1] справедливо следующее соотношение:

$$\mathit{Beta}(\theta; a, b) = \alpha\, \theta^{a-1}(1-\theta)^{b-1}. \tag{20.6}$$

Константа нормализации α, позволяющая интегрировать распределение к 1, зависит от a и b. На рис. 20.5 показано, как выглядит это распределение при различных значениях a и b. Среднее значение бета-распределения равно $a/(a+b)$, поэтому бо́льшие значения a предполагают нашу уверенность, что Θ ближе к 1, чем к 0. При бо́льших значениях $a+b$ распределение становится более заостренным, что выражает бо́льшую уверенность в правильности значения Θ. Было установлено, что функция равномерной плотности — это то же самое, что и $\mathit{Beta}(1, 1)$: среднее значение равно 1/2, а распределение является плоским.

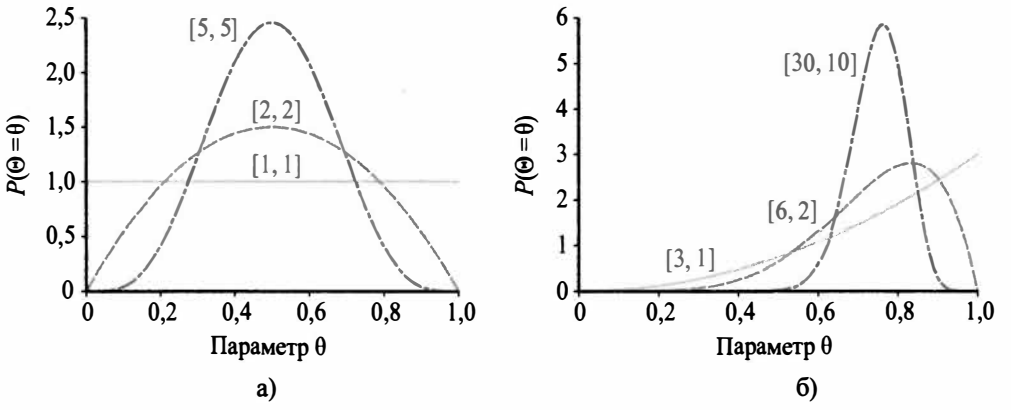


Рис. 20.5. Примеры распределения $\mathit{Beta}(a, b)$ для различных значений (a, b)

[3] Они называются гиперпараметрами, поскольку параметризуют распределение по величине θ.

Помимо такой гибкости, семейство бета-распределений обладает еще одним замечательным свойством: если переменная Θ имеет распределение априорных вероятностей $Beta(a, b)$, то после наблюдения некоторой точки данных распределение апостериорных вероятностей для Θ также становится бета-распределением. Другими словами, бета-распределение закрыто на обновление. Семейство бета-распределений называется ►**сопряженным априорным распределением** (*conjugate prior*) для семейства распределений, относящихся к некоторой булевой переменной.[4] Рассмотрим, как применяется это свойство. Предположим, что наблюдается леденец со вкусом вишни; в таком случае имеет место следующее соотношение:

$$\begin{aligned} P(\theta \mid D_1 = cherry) &= \alpha P(D_1 = cherry \mid \theta) P(\theta) = \\ &= \alpha' \theta \cdot Beta(\theta; a, b) = \alpha' \theta \cdot \theta^{a-1}(1-\theta)^{b-1} = \\ &= \alpha' \theta^a (1-\theta)^{b-1} = \alpha' \, Beta(\theta; a+1, b). \end{aligned}$$

Таким образом, после обнаружения леденца со вкусом вишни параметр a просто увеличивается для получения нового апостериорного распределения. Аналогичным образом после обнаружения леденца со вкусом лайма увеличивается параметр b. Следовательно, гиперпараметры a и b можно рассматривать как ►**виртуальные счетчики**, в том смысле, что распределение априорных вероятностей $Beta(a, b)$ ведет себя точно так же, как если бы обучение начиналось с равномерного распределения априорных вероятностей $Beta(1, 1)$, после чего было фактически обнаружено $a - 1$ леденцов со вкусом вишни и $b - 1$ леденцов со вкусом лайма.

Изучая последовательность бета-распределений, соответствующих возрастающим значениям параметров a и b, и поддерживая постоянные пропорции, можно наглядно продемонстрировать, как изменяется распределение апостериорных вероятностей по параметру Θ по мере поступления новых данных. Например, предположим, что пакет в действительности содержит 75% леденцов со вкусом вишни. На рис. 20.5, *б* показана последовательность распределений $Beta(3, 1)$, $Beta(6, 2)$, $Beta(30, 10)$. Очевидно, что эта последовательность сходится к узкому пику вокруг истинного значения Θ. Поэтому при достаточно больших наборах данных процесс байесовского обучения (по меньшей мере в данном случае) сходится и позволяет получить такие же результаты, как и обучение с учетом максимального правдоподобия.

Теперь давайте рассмотрим более сложный случай. Сеть, показанная на рис. 20.2, *б*, имеет три параметра: θ, θ_1 и θ_2, где θ_1 — вероятность наличия красной обертки на леденце с вишневым вкусом, а θ_2 — вероятность наличия красной

[4] К другим сопряженным априорным распределениям относятся семейство распределений **Дирихле** для параметров дискретного многомерного распределения и семейство распределений **нормальных–Висхарта** для параметров гауссова распределения. (См. Бернардо и Смит [189] (1994).)

обертки на леденце со вкусом лайма. Распределение априорных вероятностей байесовской гипотезы должно охватывать все три параметра, — это значит, что необходимо задать распределение $\mathbf{P}(\Theta, \Theta_1, \Theta_2)$. Обычно предполагается, что соблюдается свойство ▶**независимости параметров**, как показано ниже:

$$\mathbf{P}(\Theta, \Theta_1, \Theta_2) = \mathbf{P}(\Theta)\mathbf{P}(\Theta_1)\mathbf{P}(\Theta_2).$$

Согласно этому предположению, каждый параметр может иметь собственное бета-распределение, которое обновляется отдельно по мере поступления данных. На рис. 20.6 показано, как распределение априорных вероятностей гипотез и любые данные можно включить в байесовскую сеть, в которой есть узел для каждой переменной параметра.

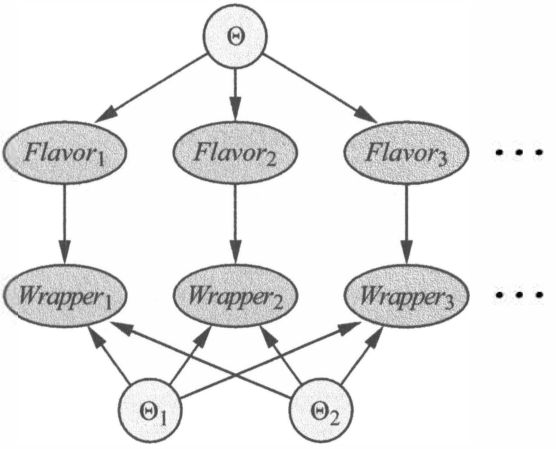


Рис. 20.6. Байесовская сеть, соответствующая байесовскому процессу обучения. Распределения апостериорных вероятностей для параметрических переменных Θ, Θ_1 и Θ_2 можно вывести из их априорных вероятностей и свидетельств, касающихся переменных $Flavor_i$ и $Wrapper_i$

Узлы Θ, Θ_1 и Θ_2 не имеют родителей. Для представления i-го наблюдения цвета обертки и действительного вкуса леденца в сеть добавляются узлы $Wrapper_i$ и $Flavor_i$. Параметр $Flavor_i$ зависит от параметра вкуса Θ:

$$P(Flavor_i = cherry \mid \Theta = \theta) = \theta.$$

Параметр $Wrapper_i$ зависит от параметров Θ_1 и Θ_2:

$$P(Wrapper_i = red \mid Flavor_i = cherry, \Theta_1 = \theta_1) = \theta_1,$$
$$P(Wrapper_i = red \mid Flavor_i = lime, \Theta_2 = \theta_2) = \theta_2.$$

Теперь весь процесс байесовского обучения для исходной байесовской сети, представленной на рис. 20.2, *б*, можно сформулировать как задачу *вероятностного вывода* в производной байесовской сети, приведенной на рис. 20.6, где данные и параметры становятся узлами. После того как будут добавлены узлы всех новых свидетельств, можно выполнить запрос для получения значений переменных параметров (в данном случае это Θ, Θ_1 и Θ_2). Согласно этой формулировке ➡ *существует только один алгоритм обучения* — алгоритм вывода для байесовских сетей.

Конечно, природа этих сетей несколько отличается от представленных в главе 13 по причине потенциально огромного числа переменных свидетельств, представляющих обучающий набор, и преобладания переменных параметров с непрерывными значениями. Точный вывод может оказаться невозможным за исключением очень простых случаев, таких как наивная байесовская модель. На практике обычно используют методы приближенного вывода, такие как MCMC (раздел 13.4.2), — многие статистические программные пакеты включают в себя эффективные реализации метода MCMC для этой цели.

20.2.6. Байесовская линейная регрессия

В этом разделе иллюстрируется, как байесовский подход можно применить к стандартной статистической задаче: линейной регрессии. В разделе 19.6 был описан обычный подход — как минимизация суммы квадратов ошибок, а затем в разделе 20.2.4 он был переосмыслен и представлен как максимизация правдоподобия в предположении гауссовой модели ошибки. В обоих случаях результат — единственная лучшая гипотеза: прямая линия с конкретными значениями наклона и точки пересечения с осью и с фиксированной дисперсией для ошибки прогноза в любой заданной точке. В этом случае нет способа оценить нашу уверенность в значениях, определяющих точку пересечения и наклон прямой.

Более того, если нужен прогноз значения еще не наблюдавшейся точки данных, расположенной далеко от наблюдавшихся точек данных, то, кажется, нет смысла предполагать, что ошибка прогноза будет такой же, как ошибка прогноза для точек данных по соседству с наблюдавшимися точками. Кажется более разумным, что ошибка прогноза будет тем больше, чем дальше точка данных от уже наблюдавшихся данных, поскольку даже небольшое изменение в наклоне прямой вызовет большое изменение в значении прогноза для удаленных точек.

В байесовском подходе обе эти проблемы устраняются. Общая идея та же, что и в предыдущем разделе: поместить в сеть априорное распределение параметров модели — в данном варианте это коэффициенты линейной модели и дисперсия шума, — а затем вычислить параметры апостериорного распределения по предоставленным данным. В случае многомерных данных и неизвестной модели шума это приводит к сложным выкладкам из линейной алгебры, поэтому здесь мы сосредоточим внимание на более простом случае: одномерные данные (линейная модель, ограниченная прохождением через начало координат) и известный характер

шума (нормальное распределение с дисперсией σ^2). В этом случае имеем только один параметр θ и модель, определяемую как

$$P(y \mid x, \theta) = \mathcal{N}(y; \theta x, \sigma_y^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(y-\theta x)^2}{\sigma^2}\right)}. \quad (20.7)$$

Поскольку логарифмическое правдоподобие квадратично по отношению к θ, соответствующей формой для сопряженного априорного распределения для θ также будет гауссиан. Это гарантирует, что апостериорное распределение для θ также будет гауссианом. Будем считать, для априорного распределения среднее — θ_0, а дисперсия — σ_0^2, поэтому

$$P(\theta) = \mathcal{N}(\theta; \theta_0, \sigma_0^2) = \frac{1}{\sigma_0\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(\theta-\theta_0)^2}{\sigma_0^2}\right)}. \quad (20.8)$$

В зависимости от конкретных моделируемых данных могут иметь место определенные идеи в отношении ожидаемого наклона θ либо он может быть совершенно неизвестен. В последнем случае имеет смысл выбрать значение θ_0 равным 0, а дисперсию σ_0^2 большой, — это так называемое ▶**неинформативное априорное распределение**. Наконец, можно сделать предположение об априорном распределении $P(x)$ для значений x каждой точки данных, но это совершенно бесполезно для анализа, поскольку это распределение не зависит от θ.

Теперь вся настройка выполнена, поэтому можно вычислить апостериорное распределение для θ, воспользовавшись уравнением (20.1): $P(\theta \mid \mathbf{d}) \propto P(\mathbf{d} \mid \theta)P(\theta)$. Наблюдаемыми точками данных являются $\mathbf{d} = (x_1, y_1), \ldots, (x_N, y_N)$, поэтому правдоподобие для данных можно получить из уравнения (20.7) следующим образом:

$$P(\mathbf{d} \mid \theta) = \left(\prod_i P(x_i)\right) \prod_i P(y_i \mid x_i, \theta) = \alpha \prod_i e^{-\frac{1}{2}\left(\frac{(y_i-\theta x_i)^2}{\sigma^2}\right)} =$$
$$= \alpha e^{-\frac{1}{2}\sum_i \left(\frac{(y_i-\theta x_i)^2}{\sigma^2}\right)},$$

где априорное распределение значений x и константы нормализации для N гауссианов были включены в константу α, не зависящую от θ. Далее, чтобы получить апостериорное распределение, объединим это выражение и параметр априорного распределения из уравнения (20.8):

$$P(\theta \mid \mathbf{d}) = \alpha'' e^{-\frac{1}{2}\left(\frac{(\theta-\theta_0)^2}{\sigma_0^2}\right)} e^{-\frac{1}{2}\sum_i \left(\frac{(y_i-\theta x_i)^2}{\sigma^2}\right)}$$

Хотя выражение выглядит довольно сложным, каждый показатель в нем является квадратичной функцией от θ, так что сумма двух показателей также является ею.

Следовательно, все выражение представляет собой гауссово распределение для θ. После некоторых алгебраических манипуляций, очень похожих на те, которые были описаны в разделе 14.4, находим

$$P(\theta \mid \mathbf{d}) = \alpha''' e^{-\frac{1}{2}\left(\frac{(\theta-\theta_N)^2}{\sigma_N^2}\right)}$$

с "обновленным" средним и дисперсией

$$\theta_N = \frac{\sigma^2\theta_0 + \sigma_0^2 \sum_i x_i y_i}{\sigma^2 + \sigma_0^2 \sum_i x_i^2} \quad \text{и} \quad \sigma_N^2 = \frac{\sigma^2\sigma_0^2}{\sigma^2 + \sigma_0^2 \sum_i x_i^2}.$$

Давайте внимательнее посмотрим на эти формулы, чтобы понять, что они означают. Когда данные узко сконцентрированы на небольшой области оси x вблизи начала координат, сумма $\sum_i x_i^2$ невелика, а апостериорная дисперсия σ_N^2 является большой, примерно равной априорной дисперсии σ_0^2. Этого следовало ожидать: имеющихся данных слишком мало, чтобы ограничить вращение прямой относительно начала координат. С другой стороны, когда точки данных широко разбросаны вдоль оси x, сумма $\sum_i x_i^2$ большая, а апостериорная дисперсия σ_N^2 — нет, примерно $\sigma^2/(\sum_i x_i^2)$, поэтому наклон прямой будет очень жестко ограничен.

Чтобы сделать прогноз для конкретной точки данных, необходимо взять интеграл по всем возможным значениям θ, как было предложено в уравнении (20.2):

$$P(y \mid x, \mathbf{d}) = \int_{-\infty}^{\infty} P(y \mid x, \mathbf{d}, \theta) P(\theta \mid x, \mathbf{d}) d\theta = \int_{-\infty}^{\infty} P(y \mid x, \theta) P(\theta \mid \mathbf{d}) d\theta =$$

$$= \alpha \int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{(y-\theta x)^2}{\sigma^2}\right)} e^{-\frac{1}{2}\left(\frac{(\theta-\theta_N)^2}{\sigma_N^2}\right)} d\theta.$$

И вновь сумма двух показателей является квадратичной функцией от θ, поэтому для θ имеет место гауссово распределение, интеграл которого равен 1. Остальные члены в y образуют другой гауссиан:

$$P(y \mid x, \mathbf{d}) \propto e^{-\frac{1}{2}\left(\frac{(y-\theta_N x)^2}{\sigma^2 + \sigma_N^2 x^2}\right)}.$$

Внимательно посмотрев на это выражение, можно увидеть, что среднее значение прогноза для y равно $\theta_N x$, т.е. основано на среднем апостериорного распределения для θ. Дисперсия для прогноза задается моделью шума σ^2 плюс член, пропорциональный x^2, а это означает, что стандартное отклонение прогноза асимптотически линейно увеличивается с увеличением расстояния от начала координат. На рис. 20.7 проиллюстрировано это явление. Как отмечалось в начале данного раздела, возрастание неопределенности для прогнозов, которые находятся дальше от наблюдаемых точек данных, действительно имеет смысл.

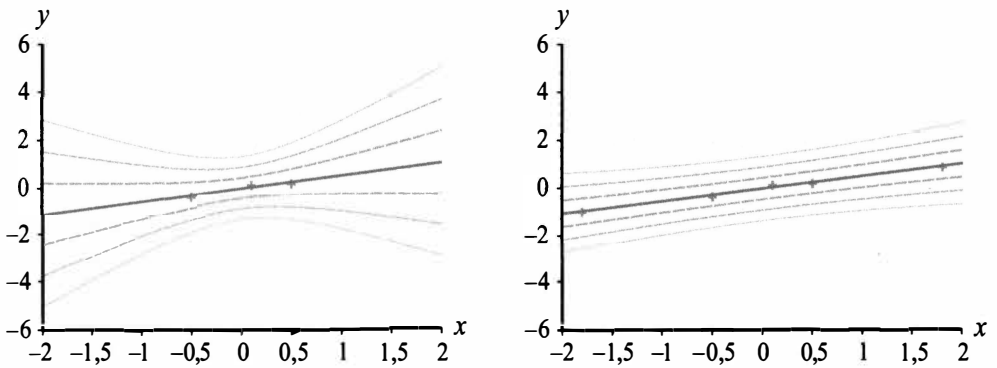


Рис. 20.7. Байесовская линейная регрессия с моделью, ограниченной проходом через начало координат и фиксированным шумом с дисперсией $\sigma^2 = 0{,}2$. Контуры плотности прогнозов показаны при стандартном отклонении ± 1, ± 2 и ± 3. **а)** Когда даны лишь три точки данных вблизи начала координат, наклон прямой регрессии является довольно неопределенным, с $\sigma_N^2 \approx 0{,}3861$. Обратите внимание, как неопределенность возрастает с увеличением расстояния от наблюдаемых точек данных. **б)** При добавлении двух дополнительных точек данных, расположенных дальше от начала координат, наклон прямой регрессии θ становится жестко ограниченным, с $\sigma_N^2 \approx 0{,}0286$. Оставшаяся дисперсия в плотности прогнозов полностью определяется фиксированным шумом σ^2

20.2.7. Определение структуры байесовских сетей посредством обучения

До сих пор предполагалось, что структура байесовской сети задана и в процессе обучения просто нужно определить ее параметры. Структура самой сети представляет основные причинные знания о проблемной области, которые часто легко может сформулировать не только специалист, но даже неопытный пользователь. Однако в некоторых случаях причинная модель может оказаться недоступной или стать предметом дискуссий (например, одни корпорации долгое время утверждали, что курение не может стать причиной рака, а другие — что рост концентрации CO_2 в атмосфере не оказывает никакого влияния на климат), поэтому важно понять, как структура байесовской сети может быть определена путем ее обучения на основе данных. В этом разделе приведен лишь краткий обзор основных идей.

Наиболее очевидным подходом к решению данной задачи является *поиск* хорошей модели. Эту работу можно начать с модели, не содержащей связей, и приступить к введению родительских узлов для каждого узла, согласуя параметры

с помощью только что описанных методов и измеряя точность результирующей модели. Альтернативный вариант состоит в том, чтобы начать с какого-то исходного предположения о структуре сети и использовать поиск восхождением к вершине или методом эмуляции отжига для внесения модификаций, возвращая параметры после каждого изменения в структуре. Модификации могут включать обращение, добавление или удаление связей. В этом процессе следует избегать появления циклов, поскольку во многих алгоритмах изначально принято предположение, что для переменных задано некоторое упорядочение и что узел может иметь родительские узлы только среди тех узлов, которые находятся перед ним в этом упорядочении (точно так же, как и в процессе создания сети, описанном в главе 13). Для достижения полной общности необходимо также обеспечить поиск среди возможных упорядочений.

Существует два альтернативных метода принятия решения о том, что хорошая структура уже найдена. Первый предусматривает проверку, действительно ли в данных удовлетворяются предположения об условной независимости, неявно заданные в этой структуре. Например, сам факт использования наивной байесовской модели для задачи об ожидании в ресторане равносилен предположению о том, что справедливо соотношение

$$\mathbf{P}(Hungry, Bar \mid WillWait) = \mathbf{P}(Hungry \mid WillWait)\mathbf{P}(Bar \mid WillWait)$$

и можно проверить по самим данным, соблюдается ли оно применительно к соответствующим условным частотам. Но даже если полученная структура описывает истинный причинный характер проблемной области, наличие статистических флуктуаций в наборе данных означает, что это уравнение никогда не будет выполняться *точно*, поэтому необходимо провести подходящую статистическую проверку для определения, есть ли достаточно весомые свидетельства нарушения гипотезы о независимости. Сложность результирующей сети будет зависеть от пороговых значений, используемых для этой проверки, — чем строже проверка на независимость, тем больше связей будет введено в сеть и тем выше опасность ее переобучения.

Подход, более совместимый с идеями, изложенными в этой главе, состоит в определении того, в какой степени предложенная модель объясняет данные (в вероятностном смысле). Однако здесь необходимо соблюдать осторожность в отношении выбора способа измерения этой степени. Если просто предпринять попытку найти гипотезу с максимальным правдоподобием, то в конечном итоге будет получена полносвязная сеть, поскольку введение дополнительных родительских узлов для некоторого узла не может уменьшить его правдоподобие (см. упражнение 20.9). Поэтому приходится каким-то образом вводить штраф за сложность модели. В подходе MAP (или MDL) штраф просто вычитается из значений правдоподобия каждой структуры (после настройки параметров) до сравнения различных структур. В байесовском подходе налагается совместное распределение априорных вероятностей на структуры и параметры. Но обычно количество структур, по которым должно быть выполнено суммирование, слишком велико (оно

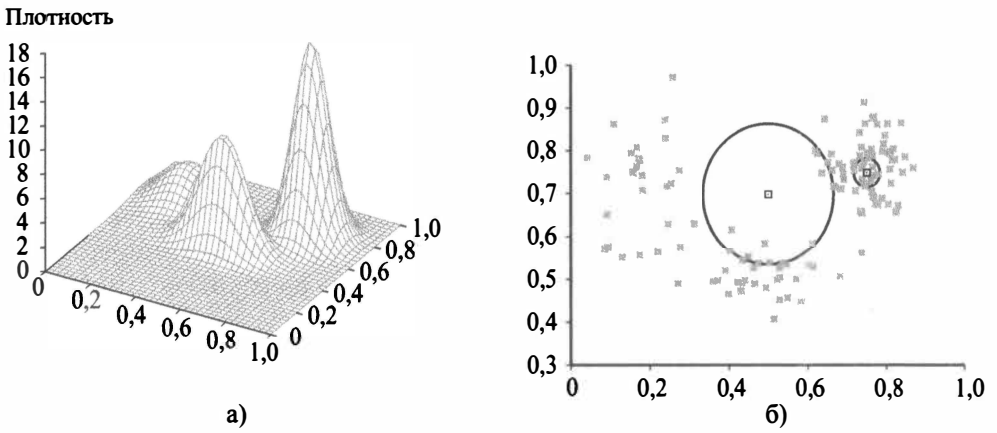
определяется суперэкспоненциальной зависимостью от количества переменных), поэтому большинство практиков используют алгоритм MCMC для формирования выборок по структурам.

Применение подхода с штрафованием за сложность (с помощью методов MAP или байесовских методов) влечет за собой появление важной связи между оптимальной структурой и характером представления для распределений условных вероятностей в сети. При использовании табличных распределений значения штрафов за сложность для распределения вероятностей узла растут экспоненциально в зависимости от количества родительских узлов, а при использовании, скажем, распределений зашумленного ИЛИ они растут только линейно. Это означает, что обучение с помощью моделей зашумленного ИЛИ (или других компактно параметризованных моделей), как правило, приводит к получению в результате обучения таких структур, в которых имеется больше родительских узлов, чем при обучении с помощью табличных распределений.

20.2.8. Оценка плотности с помощью непараметрических моделей

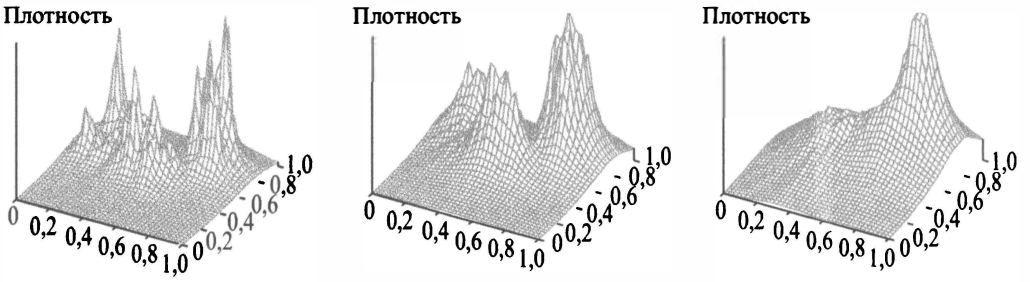
Обучать вероятностную модель волне возможно и без принятия каких-либо допущений в отношении ее структуры и параметризации — применяя непараметрические методы, предложенные в разделе 19.7. Задачи ►**оценки непараметрической плотности**, как правило, реализуются в непрерывных проблемных областях, например в таких, как на рис. 20.8, *а*, где приведена функция плотности вероятности в пространстве, определяемом двумя непрерывными переменными. На рис. 20.8, *б* представлена некоторая выборка точек данных из этой функции плотности. Вопрос заключается в том, можно ли восстановить модель на основе имеющихся примеров?

Сначала рассмотрим модели ***k*-ближайших соседей**. (В главе 19 обсуждалось применение моделей ближайших соседей к задачам классификации и регрессии, а здесь рассматривается их применение к задаче оценки плотности.) При заданной выборке точек данных, чтобы оценить неизвестную плотность вероятности в точке запроса $\mathbf{x}$, можно просто измерить плотность в точках данных в окрестности точки $\mathbf{x}$. На рис. 20.8, *б* показаны две точки запроса (маленькие черные квадраты) и для каждой из них приведена окружность, охватывающая 10 их ближайших соседей. Как видим, окружность в центре рисунка велика и это означает, что в этой области плотность вероятности низкая, тогда как окружность справа намного меньше, а значит, плотность вероятности здесь высокая. На рис. 20.9 представлены три графика оценки плотности вероятности методом k-ближайших соседей для различных значений k. Кажется очевидным, что вариант *б* наиболее удачен, тогда как для варианта *а* график поверхности очень "колючий" — имеется много острых пиков (значение k слишком мало), а в варианте *в* график чрезмерно гладкий (значение k слишком велико).



а) б)

Рис. 20.8. а) Трехмерный график смешанных гауссовых распределений, приведенных на рис. 20.12, *а*. **б)** Выборка из 128 точек данных для смешанных гауссовых распределений, вместе с двумя точками запроса (небольшие черные квадраты) и областью 10 их ближайших соседей (большой круг в центре и меньший круг справа)



а) б) в)

Рис. 20.9. Оценка плотности вероятности методом k-ближайших соседей в применении к данным на рис. 20.8, *б* для значений $k = 3$, 10 и 40 соответственно. При $k = 3$ график имеет слишком много острых пиков, при $k = 40$ он слишком гладкий, а при $k = 10$ его можно считать почти правильным. Наилучшее значение для k может быть выбрано с использованием перекрестной проверки

Другая возможность заключается в использовании **ядерных функций**, как это делается в методе локально взвешенной регрессии. Чтобы применить ядерную модель к задаче оценке плотности, предположим, что каждая точка данных генерирует собственную небольшую функцию плотности. Например, можно использовать

сферические гауссианы со стандартным отклонением w вдоль каждой оси. Тогда оценочная плотность в точке запроса $\mathbf{x}$ будет средним для ядер данных:

$$P(\mathbf{x}) = \frac{1}{N}\sum_{j=1}^{N} \mathcal{K}(\mathbf{x}, \mathbf{x}_j), \text{ где } \mathcal{K}(\mathbf{x}, \mathbf{x}_j) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{\frac{D(\mathbf{x}, \mathbf{x}_j)^2}{2w^2}},$$

где d — число измерений в $\mathbf{x}$, а D — евклидова функция расстояния. Однако пока еще остается нерешенной проблема выбора подходящего значения для ширины ядра w. На рис. 20.10 приведены графики, построенные с таким значением w, которое оказалось слишком малым, достаточно хорошим и слишком большим. И вновь хорошее значение w можно выбрать с помощью перекрестной проверки.

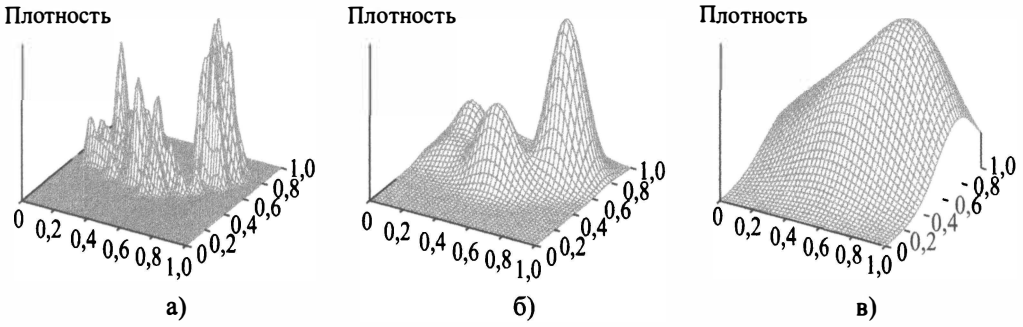


Рис. 20.10. Оценка плотности вероятности ядерным методом в применении к данным на рис. 20.8, *б с* ядрами в виде сферических гауссианов шириной $w = 0{,}02$; 0,07 и 0,20 соответственно. При ширине ядра $w = 0{,}07$ получен наилучший результат

20.3. Обучение с использованием скрытых переменных: алгоритм EM

В предыдущем разделе рассматривался полностью наблюдаемый случай. Но многие реальные задачи характеризуются наличием ►**скрытых переменных** (иногда называемых ►**латентными переменными**), которые не наблюдаются в данных. Например, медицинские карточки часто включают описания наблюдаемых симптомов, диагнозы врачей, назначенное лечение и, возможно, результаты лечения, но редко содержат записи наблюдений за ходом самой болезни! (Обратите внимание, что *диагноз* — это не сама *болезнь*; это причинное следствие наблюдаемых симптомов, которые, в свою очередь, вызваны болезнью.) Напрашивается вопрос: "Если ход болезни не наблюдается, можно ли построить модель на основании лишь наблюдаемых переменных?" Ответ можно найти на рис. 20.11, на котором приведена небольшая фиктивная диагностическая модель для сердечного

заболевания (*HeartDisease*). Имеется три наблюдаемых предрасполагающих (или наоборот) фактора — курение (*Smoking*), диета (*Diet*) и упражнения (*Exercise*) — и три наблюдаемых симптома (которым здесь намеренно присвоены сугубо условные обозначения $Symptom_x$ с индексом от 1 до 3). Предположим, что каждая переменная имеет три возможных значения (например, *none* (отсутствует), *moderate* (умеренно) и *severe* (серьезно)). Удаление скрытой переменной из сети, приведенной на рис. 20.7, *а*, влечет за собой создание сети, показанной на рис. 20.7, *б*; при этом общее количество параметров увеличивается с 78 до 708. Следовательно, ➡ *скрытые переменные позволяют резко уменьшить количество параметров, требуемых для определения байесовской сети.* А это, в свою очередь, позволяет резко уменьшить объем данных, необходимых для параметрического обучения.

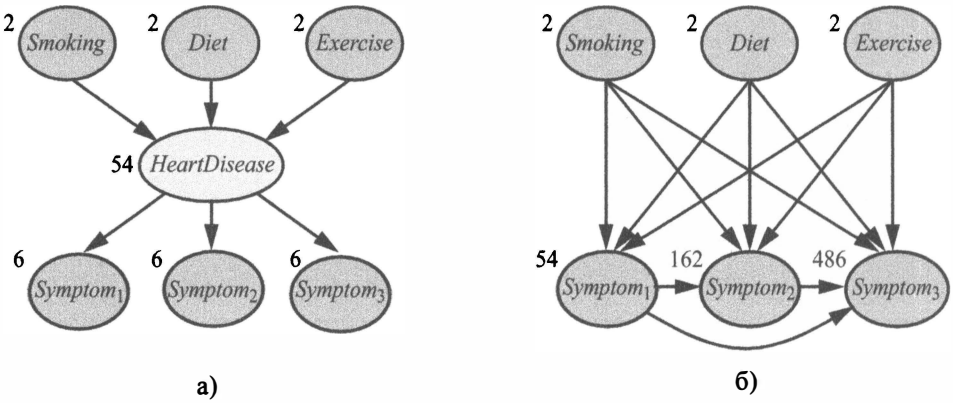


Рис. 20.11. а) Простая диагностическая сеть для сердечного заболевания, в которой предполагается наличие скрытой переменной. Каждая переменная имеет три возможных значения, и соответствующий ей узел обозначен количеством независимых параметров в распределении ее условной вероятности, — общее их количество равно 78. **б)** Эквивалентная сеть с удаленным узлом *HeartDisease*. Обратите внимание, что переменные симптомов больше не являются условно независимыми, если даны значения их родительских переменных. Для этой сети необходимо 708 параметров

Скрытые переменные важны, но их введение приводит к усложнению задачи обучения. Например, глядя на рис. 20.11, *а*, вовсе неочевидно, как в процессе обучения найти распределение условных вероятностей для переменной *HeartDisease*, когда заданы ее родительские переменные, поскольку значение переменной *HeartDisease* в каждом случае неизвестно. Такая же проблема возникает и при поиске в процессе обучения распределений вероятностей для симптомов. В этом разделе рассматривается алгоритм, называемый ▶**ожидание–максимизация** или сокращенно — EM (*Expectation–Maximization*), позволяющий решить эту проблему очень общим образом. Сначала будут рассмотрены три конкретных примера,

а затем приведено общее описание. На первых порах складывается впечатление, что этот алгоритм действует как по волшебству, но как только будет достигнуто его интуитивное понимание, станет очевидным, что алгоритму EM можно найти применение в самом широком спектре задач обучения.

20.3.1. Кластеризация без учителя: обучение смешанных гауссовых распределений

►**Кластеризацией без учителя** называется задача выделения многочисленных категорий в коллекции объектов. Эта задача относится к классу "обучение без учителя" потому, что категориям не назначены метки. Например, предположим, что были зарегистрированы спектры сотен тысяч звезд и что существуют различные *типы* звезд, которые можно определить по их спектру. Если это действительно так, то сколько таких типов существует и каковы их характеристики? Все знакомы с такими астрономическими терминами, как "красный гигант" и "белый карлик", но звезды не носят эти надписи на своих шляпах, поэтому астрономам приходится решать задачу кластеризации без учителя для определения их категорий. К другим примерам относится выявление видов, родов, отрядов и других категорий в таксономии живых организмов, установленной Линнеем, а также создание естественных разновидностей для распределения по категориям обычных объектов (см. главу 10).

Кластеризация без учителя начинается с данных. На рис. 20.12, *б* приведено 500 точек данных, каждая из которых определяет значения двух непрерывных атрибутов. Точки данных могут соответствовать звездам, а их атрибуты — интенсивностям спектра на двух конкретных частотах.

Далее необходимо понять, какого рода распределение вероятностей может быть сформировано имеющимися данными. Кластеризация равносильна предположению, что данные формировались из ►**смешанного распределения** P. Подобное распределение имеет k ►**компонентов**, каждый из которых представляет собой отдельное распределение. При формировании точки данных сначала выбирается компонент, а затем из этого компонента формируется выборка. Пусть случайная переменная C определяет компонент со значениями $1,\ldots, k$; тогда смешанное распределение задается следующим выражением:

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(C = i)\, P(\mathbf{x} \mid C = i),$$

где $\mathbf{x}$ определяет значения атрибутов для точки данных. В случае непрерывных данных естественным вариантом выбора для распределений компонентов является многомерное гауссово распределение, что приводит к созданию семейства распределений, называемых ►**смешанными гауссовыми распределениями**. Параметрами смешанного гауссова распределения являются $w_i = \mathbf{P}(C = i)$ (вес каждого компонента), μ_i (среднее значение для каждого компонента) и Σ_i (ковариация

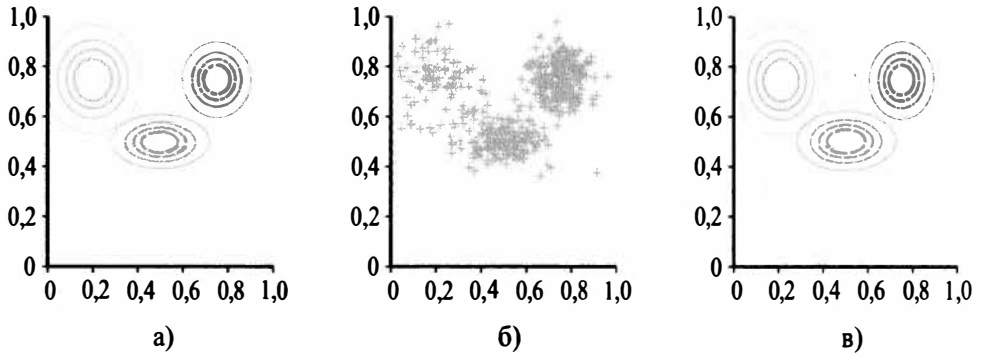


Рис. 20.12. а) Модель смешанного гауссова распределения с тремя компонентами: веса компонентов (слева направо) равны 0,2, 0,3 и 0,5. **б)** 500 двухмерных точек данных, выбранных из модели, представленной на рис. 20.12, *а*. **в)** Модель, реконструированная с помощью алгоритма EM из данных, приведенных на рис. 20.12, *б*

каждого компонента). На рис. 20.12, *а* показано смешанное гауссово распределение из трех распределений, — именно это смешанное распределение было источником данных, приведенных на рис. 20.12, *б*, и является моделью, приведенной на рис. 20.8, *а* (раздел 20.2.8).

Таким образом, задача кластеризации без учителя заключается в восстановлении модели смешанного гауссова распределения, подобной приведенной на рис. 20.12, *а*, на основании исходных данных, подобных представленным на рис. 20.12, *б*. Очевидно, что если *знать*, на основе какого компонента формировалась каждая точка данных, то будет легко восстановить гауссово распределение для этого компонента, поскольку появляется возможность просто выбрать все точки данных, соответствующие данному компоненту, а затем применить уравнение (20.4) (вернее, его многомерную версию) для согласования параметров гауссова распределения с набором данных. С другой стороны, если *знать* параметры каждого компонента, то можно, по меньшей мере в вероятностном смысле, присвоить каждую точку данных соответствующему компоненту.

К сожалению, проблема в том, что неизвестны ни присваивания, ни параметры. Основная идея алгоритма EM в данном контексте состоит в том, что следует *принять допущение*, будто нам известны параметры этой модели, а затем вычислить вероятность того, что каждая точка данных принадлежит к тому или иному компоненту. После этого потребуется вновь согласовать компоненты с данными так, чтобы каждый компонент был согласован со всем набором данных, каждой точке которого назначается вес, соответствующий вероятности того, что она принадлежит к данному компоненту. Этот процесс итерационно продолжается до достижения сходимости. По сути, при этом происходит "дополнение"

данных путем вычисления распределений вероятностей по скрытым переменным (определяющим, какому компоненту принадлежит каждая точка данных) на основании текущей модели. При использовании смешанного гауссова распределения модель смешанного распределения инициализируется произвольными значениями параметров, а затем последовательно выполняются итерации, включающие следующие два этапа.

1. **Е-этап.** Вычисляются вероятности $p_{ij} = P(C = i \mid \mathbf{x}_j)$ того, что данные $\mathbf{x}_j$ были сформированы компонентом i. Согласно правилу Байеса, справедливо соотношение $p_{ij} = \alpha P(\mathbf{x}_j \mid C = i) P(C = i)$. Здесь терм $P(\mathbf{x}_j \mid C = i)$ представляет собой вероятность значений данных $\mathbf{x}_j$ в i-м гауссовом распределении, а терм $P(C = i)$ — это весовой параметр i-го гауссова распределения. Далее определяется $n_i = \Sigma_j p_{ij}$, эффективное количество точек данных, назначенных в настоящее время компоненту i.
2. **М-этап.** Вычисляются новые средние значения, ковариации веса компонентов в такой последовательности:

$$\mu_i \leftarrow \sum_j p_{ij}\mathbf{x}_j / n_i$$
$$\Sigma_i \leftarrow \sum_j p_{ij}(\mathbf{x}_j - \mu_i)(\mathbf{x}_j - \mu_i)^\top / n_i \ ,$$
$$w_i \leftarrow n_i / N$$

где N — общее количество точек данных. Е-этап, или этап ожидания (*expectation*), может рассматриваться как вычисление ожидаемых значений p_{ij} скрытых ►**индикаторных переменных** Z_{ij}, где значение Z_{ij} равно 1, если данные $\mathbf{x}_j$ были сформированы i-м компонентом, и равно 0 — в противном случае. На М-этапе, или этапе *максимизации* (maximization), осуществляется поиск новых значений параметров, которые максимизируют логарифмическое правдоподобие данных с учетом ожидаемых значений скрытых индикаторных переменных.

Окончательная модель, обученная с помощью алгоритма EM при его применении к данным, приведенным на рис. 20.12, *а*, показана на рис. 20.12, *в*; она практически не отличается от первоначальной модели, на основе которой были сформированы данные. На рис. 20.13, *а* показан график изменения логарифмического правдоподобия данных, соответствующих текущей модели, в процессе выполнения алгоритма EM.

На этом графике заслуживают внимания две особенности. Во-первых, логарифмическое правдоподобие модели, окончательно полученной в процессе обучения, немного *превышает* соответствующее значение для первоначальной модели, на основании которой были сформированы исходные данные. Это явление на первый взгляд может показаться удивительным, но оно просто отражает тот

факт, что данные были сформированы случайным образом, поэтому существует вероятность того, что они не являются точным отражением самой базовой модели. Во-вторых, любопытно то, что в алгоритме EM ➔ *логарифмическое правдоподобие данных повышается после каждой итерации*. Можно доказать, что такова общая особенность данного алгоритма. Кроме того, можно доказать, что при определенных условиях алгоритм EM достигает локального максимума правдоподобия. (В редких случаях он может достичь также точки перегиба или даже локального минимума.) В этом смысле алгоритм EM напоминает алгоритм восхождения к вершине с учетом градиента, но следует отметить то, что в нем уже не применяется параметр "размер шага"!

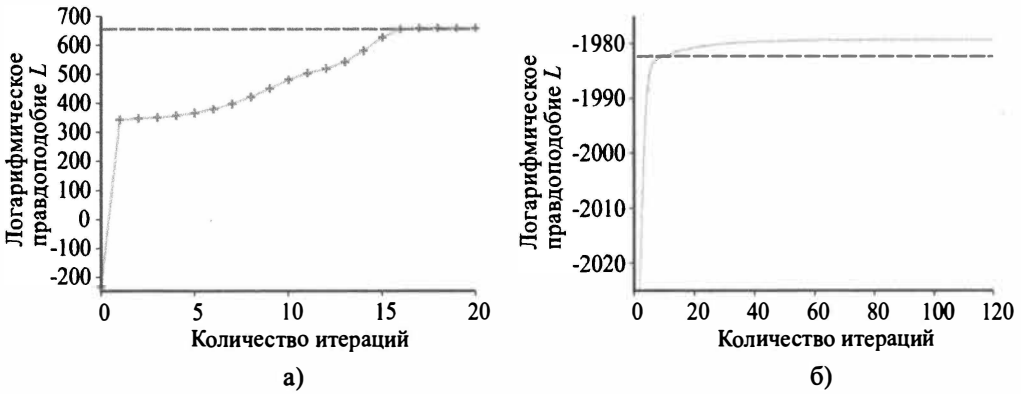


Рис. 20.13. Графики, показывающие изменение логарифмического правдоподобия данных L, как функции от количества итераций алгоритма EM. Горизонтальная линия соответствует логарифмическому правдоподобию истинной модели. **а)** График для модели смешанного гауссова распределения, показанной на рис. 20.12. **б)** График для байесовской сети, приведенной на рис. 20.14, *а*

Но события не всегда развиваются так удачно, как можно было бы судить на основании рис. 20.13, *а*. Например, может случиться так, что один компонент гауссова распределения сузится настолько, что будет охватывать лишь единственную точку данных. В таком случае его дисперсия достигнет нуля, а правдоподобие увеличится до бесконечности! Если неизвестно, сколько компонентов входит в смешанное распределение, потребуется попробовать различные значения k и определить, какое из них является наилучшим, что может стать источником ошибки. Еще одна проблема состоит в том, что может произойти "слияние" двух компонентов, в результате чего они примут одинаковые значения средних и дисперсий, а точки данных станут для них общими. Вырожденные локальные максимумы такого рода представляют собой серьезную проблему, особенно в случае большого количества измерений. Одно из решений состоит в наложении

распределений априорных вероятностей на параметры модели и применении версии MAP алгоритма EM. Еще одно решение состоит в перезапуске вычислений для некоторого компонента с новыми случайно выбранными параметрами, если он становится слишком малым или слишком близким к другому компоненту. Иногда также имеет смысл выбирать для инициализации параметров более обоснованные значения.

20.3.2. Параметрическое обучение байесовских сетей со скрытыми переменными

Чтобы определить посредством обучения параметры байесовской сети со скрытыми переменными, можно применить те же подходы, которые позволили добиться успеха в случае смешанных гауссовых распределений. На рис. 20.14, *а* представлена ситуация, в которой два пакета леденцов уже были смешаны друг с другом. Для описания леденцов здесь применяются три характеристики: кроме вкуса *Flavor* и цвета обертки *Wrapper*, некоторые из леденцов имеют отверстие (*Hole*) в середине, а остальные леденцы отверстий не имеют. Распределение леденцов в каждом пакете описано с помощью **наивной байесовской** модели: в каждом отдельном пакете характеристики леденцов являются независимыми, но распределение условных вероятностей каждой характеристики зависит от пакета. Используются следующие параметры: θ — априорная вероятность того, что леденец взят из первого пакета (*Bag* 1); θ_{F1} и θ_{F2} — вероятности того, что леденец имеет вкус вишни и взят из пакета *Bag* 1 или *Bag* 2 соответственно; θ_{W1} и θ_{W2} задают вероятности того, что обертка имеет красный цвет (для пакетов *Bag* 1 и *Bag* 2); а θ_{H1} и θ_{H2} — вероятности того, что леденец имеет отверстие.

Общая модель представляет собой модель смешанного распределения: взвешенная сумма двух различных распределений, каждое из которых является результатом независимых одномерных распределений. (В действительности это смешанное гауссово распределение можно также смоделировать в виде байесовской сети, как показано на рис. 20.14, *б*.) На этом рисунке скрытая переменная соответствует пакету, поскольку после смешивания леденцов мы больше не имеем возможности определить, из какого пакета взята каждая конфета. Можно ли в таком случае восстановить описание этих двух пакетов, наблюдая за характеристиками леденцов, взятых из этой смеси? Выполним одну итерацию алгоритма EM для этой задачи. Сначала давайте посмотрим на данные. Сформирована 1000 выборок из модели, истинными параметрами которой являются следующие:

$$\theta = 0{,}5; \quad \theta_{F1} = \theta_{W1} = \theta_{H1} = 0{,}8; \quad \theta_{F2} = \theta_{W2} = \theta_{H2} = 0{,}3. \tag{20.9}$$

Это означает, что равновероятно получение леденцов из обоих пакетов. В первом пакете в основном находятся леденцы со вкусом вишни в красных обертках и с отверстиями, а во втором — в основном леденцы со вкусом лайма в зеленых

обертках и без отверстий. Количество восьми возможных разновидностей леденцов является следующим.

	W = *red*		*W* = *green*	
	H = 1	*H* = 0	*H* = 1	*H* = 0
F = *cherry*	273	93	104	90
F = *lime*	79	100	94	167

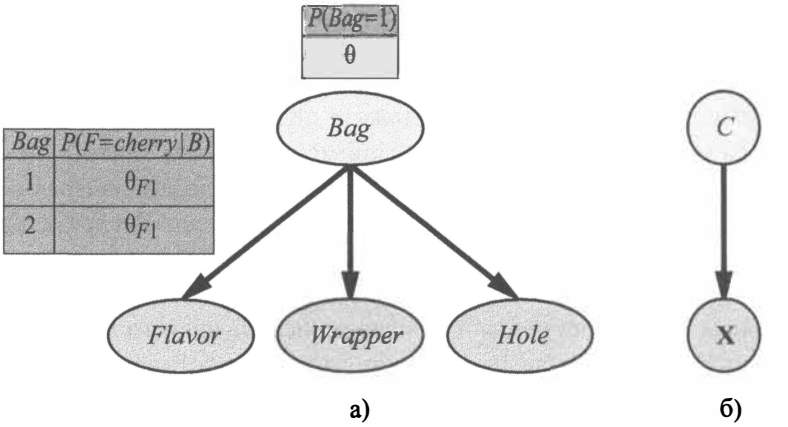


Рис. 20.14. а) Модель смешанного распределения для задачи с леденцами. Относительные пропорции леденцов с различным вкусом, оберткой и наличием/отсутствием отверстий зависят от пакета, определяемого ненаблюдаемой переменной. **б)** Байесовская сеть, соответствующая смешанному гауссову распределению. Среднее и ковариация наблюдаемой переменной **X** зависят от компонента *C*

Начнем с инициализации параметров. Для упрощения числовых расчетов произвольно выбираем следующие их значения:[5]

$$\theta^{(0)} = 0,6; \quad \theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0,6; \quad \theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0,4. \tag{20.10}$$

Первым проведем расчет для параметра θ. В полностью наблюдаемом случае можно получить оценку этого параметра непосредственно из *наблюдаемых* значений количества леденцов, взятых из пакетов 1 и 2. Но поскольку номер пакета — это скрытая переменная, вместо этого рассчитаем *ожидаемые* значения количества. Ожидаемое количество $\hat{N}(Bag = 1)$ представляет собой сумму вероятностей того, что леденец взят из пакета 1:

[5] На практике лучше выбирать значения параметров случайным образом — это предотвращает появление локальных максимумов из-за симметрии.

$$\theta^{(1)} = \hat{N}(Bag = 1)/N = \sum_{j=1}^{N} P(Bag = 1 \mid flavor_j, wrapper_j, holes_j)/N.$$

Эти вероятности можно вычислить с помощью любого алгоритма вероятностного вывода для байесовских сетей. А применительно к наивной байесовской модели, подобной той, которая рассматривается в данном примере, этот вероятностный вывод можно выполнить "вручную", используя правило Байеса и применяя выражение для условной независимости:

$$\theta^{(1)} = \frac{1}{N}\sum_{j=1}^{N} \frac{P(flavor_j \mid Bag = 1)P(wrapper_j \mid Bag = 1)P(holes_j \mid Bag = 1)P(Bag = 1)}{\sum_i P(flavor_j \mid Bag = i)P(wrapper_j \mid Bag = i)P(holes_j \mid Bag = i)P(Bag = i)}.$$

Применяя эту формулу, например, к данным о 273 конфетах в красной обертке, среди которых находятся вишневые леденцы с отверстиями, определим, какой вклад они вносят в распределение вероятностей:

$$\frac{273}{1000} \cdot \frac{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)}}{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)} + \theta_{F2}^{(0)}\theta_{W2}^{(0)}\theta_{H2}^{(0)}(1-\theta^{(0)})} \approx 0,22797.$$

Продолжив эти расчеты для семи других видов леденцов, количество которых указано в таблице выше, получим, что $\theta^{(1)} = 0,6124$.

Теперь рассмотрим другие параметры, такие как θ_{F1}. В полностью наблюдаемом случае это значение можно было бы оценить непосредственно на основе *наблюдаемых* значений количества леденцов с вишневым вкусом и вкусом лайма из пакета 1. Ожидаемое количество леденцов с вишневым вкусом из пакета 1 задается с помощью следующего выражения:

$$\sum_{j:Flavor_j = cherry} P(Bag = 1 \mid Flavor_j = cherry, wrapper_j, holes_j).$$

Эти вероятности также можно вычислить с помощью любого алгоритма для байесовской сети. Продолжая этот процесс, получим новые значения для всех параметров:

$$\begin{aligned} &\theta^{(1)} = 0,6124; \quad \theta_{F1}^{(1)} = 0,6684; \quad \theta_{W1}^{(1)} = 0,6483; \quad \theta_{H1}^{(1)} = 0,6558; \\ &\theta_{F2}^{(1)} = 0,3887; \quad \theta_{W2}^{(1)} = 0,3817; \quad \theta_{H1}^{(1)} = 0,3827. \end{aligned} \tag{20.11}$$

Логарифмическое правдоподобие данных возрастает от первоначального значения, примерно равного –2044, приблизительно до –2021 после первой итерации, как показано на рис. 20.13, *б*. Это означает, что в данном обновлении само значение правдоподобия улучшается примерно на коэффициент $e^{23} \approx 10^{10}$. К десятой итерации модель, полученная в процессе обучения, лучше согласуется с данными, чем первоначальная модель ($L = -1982,214$), но дальнейший прогресс очень сильно

замедляется. Такая ситуация в приложениях алгоритма EM встречается весьма часто, поэтому во многих практически применяемых системах алгоритм EM на последнем этапе обучения используется в сочетании с таким алгоритмом на основе градиента, как алгоритм Ньютона–Рафсона (см. главу 4).

Общий вывод, который можно сделать на основании данного примера, состоит в том, что ► *обновления параметров при обучении байесовской сети со скрытыми переменными являются непосредственно доступными из результатов вероятностного вывода по каждому примеру. Более того, для каждого параметра требуются только локальные апостериорные вероятности.* Здесь "локальный" означает, что таблица условных вероятностей (CPT) для каждой переменной X_i может быть изучена на основании апостериорных вероятностей, включающих только X_i и ее родителей $\mathbf{U}_i$. После определения θ_{ijk} как параметра $P(X_i = x_{ij} \mid \mathbf{U}_i = \mathbf{u}_{ik})$ в таблице CPT обновление задается по нормализованным ожидаемым значениям следующим образом:

$$\theta_{ijk} \leftarrow \hat{N}(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik}) / \hat{N}(\mathbf{U}_i = \mathbf{u}_{ik}).$$

Эти ожидаемые количества можно получить путем суммирования по всем примерам и вычисления вероятностей $P(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik})$ для каждого из них с использованием любого алгоритма вероятностного вывода в байесовской сети. Для использования в точных алгоритмах (включая алгоритм устранения переменных) все эти вероятности могут быть получены непосредственно как побочный продукт стандартного вероятностного вывода, без необходимости применения дополнительных вычислений, характерных для обучения. Более того, информация, необходимая для обучения, доступна *локально* применительно к каждому параметру.

Вернувшись немного назад, то, что алгоритм EM делает в этом примере, можно понимать как восстановление семи параметров (θ, θ_{F1}, θ_{W1}, θ_{H1}, θ_{F2}, θ_{W2}, θ_{H2}) из семи ($2^3 - 1$) наблюдаемых результатов подсчета данных. (Восьмое значение определяется тем фактом, что общая сумма равна 1000.) Если бы каждый леденец описывался двумя, а не тремя атрибутами (скажем, без учета отверстий), то было бы всего пять параметров (θ, θ_{F1}, θ_{W1}, θ_{F2}, θ_{W2}), но только три ($2^2 - 1$) наблюдаемых результата подсчета. В таком случае будет невозможно восстановить весовой коэффициент смеси θ или характеристики двух пакетов, содержимое которых было перемешано. Говорят, что модель с двумя атрибутами не ►**идентифицируема**.

Идентифицируемость в байесовской сети является сложным вопросом. Обратите внимание, что даже с тремя атрибутами и семью результатами подсчетов невозможно однозначно восстановить модель, поскольку допустимы две эквивалентные для наблюдения модели с обратными значениями переменной *Bag*. В зависимости от того, как параметры инициализируются, алгоритм EM будет сходиться либо к модели, в которой в пакете 1 были в основном леденцы со вкусом вишни, а в пакете 2 — со вкусом лайма, либо наоборот. Этот вариант неидентифицируемости является неизбежным при наличии переменных, которые никогда не наблюдаются.

20.3.3. Обучение скрытых марковских моделей

Последнее рассматриваемое в данной главе приложение алгоритма EM касается изучения вероятностей перехода в скрытых марковских моделях (*Hidden Markov Model* — HMM). Напомним (см. раздел 14.3), что скрытая марковская модель может быть представлена с помощью динамической байесовской сети с одной дискретной переменной состояния, как показано на рис. 20.15. Каждая точка данных состоит из *последовательности* наблюдений конечной длины, поэтому задача заключается в том, чтобы в процессе обучения изучить вероятности перехода на основании множества последовательностей наблюдений (или, возможно, на основании только одной длинной последовательности).

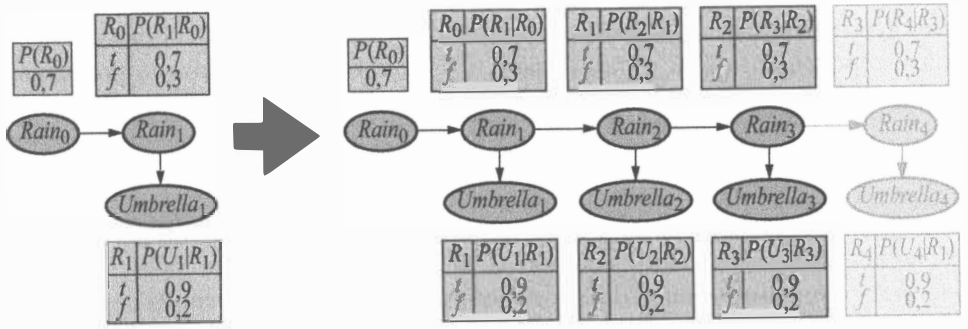


Рис. 20.15. Развернутая динамическая байесовская сеть, представляющая скрытую марковскую модель (повторение рис. 14.16)

Выше в этой главе уже было показано, как проводить обучение байесовских сетей, но в данном случае возникает одна сложность: в байесовских сетях каждый параметр является различным, тогда как в скрытой марковской модели отдельные вероятности перехода из состояния i в состояния j во время t, $\theta_{ijt} = P(X_{t+1} = j \mid X_t = i)$, *повторяются* во времени, т.е. $\theta_{ijt} = \theta_{ij}$ для всех t. Чтобы оценить вероятность перехода из состояния i в состояние j, достаточно вычислить ожидаемую долю случаев, в которых система подвергается переходу в состояние j, находясь в состоянии i:

$$\theta_{ij} \leftarrow \sum_t \hat{N}(X_{t+1} = j, X_t = i) / \sum_i \hat{N}(X_t = i).$$

Опять-таки, эти ожидаемые количества могут быть вычислены с помощью любого алгоритма вероятностного вывода для скрытой марковской модели. Так, для вычисления необходимых вероятностей можно очень легко модифицировать **прямой–обратный** алгоритм, приведенный на рис. 14.4. Но есть одно важное замечание: требуются вероятности, полученные путем **сглаживания**, а не **фильтрации**. Фильтрация дает вероятности распределения в текущем состоянии с учетом

известного прошлого, тогда как сглаживание дает распределение с учетом всех свидетельств, включая и те, которые имели место после того, как произошел определенный переход. Свидетельства в случае убийства обычно могут быть получены только *после* совершения этого преступления (т.е. после перехода из состояния i в состояние j).

20.3.4. Общая форма алгоритма EM

Выше было описано несколько вариантов применения алгоритма EM. Каждый из них сводился к вычислению ожидаемых значений скрытых переменных для каждого примера, а затем повторно вычислялись параметры с использованием ожидаемых значений так, как если бы они были наблюдаемыми значениями. Допустим, что $\mathbf{x}$ — все наблюдаемые значения во всех примерах, и пусть $\mathbf{Z}$ обозначает все скрытые переменные для всех примеров, а θ представляет собой все параметры для вероятностной модели. В таком случае алгоритм EM можно представить с помощью следующего уравнения:

$$\theta^{(i+1)} = \underset{\theta}{\operatorname{argmax}} \sum_{\mathbf{z}} P(\mathbf{Z} = \mathbf{z} \mid \mathbf{x}, \theta^{(i)}) L(\mathbf{x}, \mathbf{Z} = \mathbf{z} \mid \theta).$$

Это уравнение представляет саму суть алгоритма EM. При этом E-этап сводится к вычислению выражения для суммы, которая представляет собой ожидаемое значение логарифмического правдоподобия "дополненных" данных по отношению к распределению $P(\mathbf{Z} = \mathbf{z} \mid \mathbf{x}, \theta^{(i)})$, т.е. распределение апостериорных вероятностей по скрытым переменным при наличии полученных данных. Далее, M-этап представляет собой максимизацию этого ожидаемого логарифмического правдоподобия по отношению к параметрам. При использовании смешанного гауссова распределения скрытыми переменными являются Z_{ij}, где Z_{ij} равна 1, если пример j был сформирован компонентом i. Для байесовских сетей скрытые переменные Z_{ij} представляют собой значения ненаблюдаемых переменных X_i для каждого примера j. При использовании скрытых марковских моделей скрытые переменные Z_{ij} являются состояниями последовательности в примере j во время t. Начиная с этой общей формы, можно вывести алгоритм EM для конкретного приложения, как только будут определены соответствующие скрытые переменные.

Поняв основную идею алгоритма EM, можно легко вывести все возможные его варианты и усовершенствования. Так, во многих случаях E-этап (вычисление распределений апостериорных вероятностей по скрытым переменным) является трудновыполнимым, — например, при работе с большими байесовскими сетями. Оказалось, что в этом случае можно применить *приближенный* E-этап и все еще получить эффективный алгоритм обучения. А если используется такой алгоритм формирования выборки, как MCMC (см. раздел 13.4), то процесс обучения становится полностью интуитивно понятным: каждое состояние (конфигурация скрытых и наблюдаемых переменных), посещенное алгоритмом MCMC,

рассматривается точно так же, как если бы оно было полным наблюдением. Следовательно, в алгоритме MCMC параметры могут обновляться непосредственно после каждого перехода из состояния в состояние. Другие формы приближенного вероятностного вывода, такие как вариационные методы и циклическое распространение доверия, также уже доказали свою эффективность при обучении очень больших сетей.

20.3.5. Обучение байесовских сетевых структур со скрытыми переменными

В разделе 20.2.7 обсуждалась задача обучения байесовских сетевых структур на основе полных данных. Но когда на наблюдаемые данные оказывают влияние ненаблюдаемые переменные, задача осложняется. В простейшем случае человек-эксперт может указать обучающему алгоритму, что существуют некоторые скрытые переменные, предоставив алгоритму самому найти для них место в сетевой структуре. Например, при заданной информации, что в модель должна быть включена переменная *HeartDisease* (трехзначная переменная), с помощью алгоритма может быть предпринята попытка определить в процессе обучения структуру, показанную на рис. 20.11, *а*. Как и в случае полных данных, общий алгоритм имеет внешний цикл, в котором осуществляется поиск других структур, и внутренний цикл, в котором параметры сети согласуются с ее заданной структурой.

Если алгоритму обучения не предоставлена информация о том, что существуют скрытые переменные, то возможны два варианта: либо исходить из того, что данные действительно являются полными (что вынудит алгоритм определить в процессе обучения модель с гораздо большим количеством параметров, как показано на рис. 20.11, *б*), либо *изобрести* новые скрытые переменные для упрощения модели. Последний подход можно реализовать путем включения новых вариантов операций модификации в процедуру поиска структуры — предусмотреть в алгоритме возможность не только модифицировать связи, но и добавлять или удалять скрытые переменные либо менять их арность. Безусловно, такой алгоритм не будет знать о том, что добавленная им новая переменная имеет название *HeartDisease*, а также не сможет дать осмысленные имена ее значениям. Но, к счастью, вновь изобретенные скрытые переменные обычно связываются с ранее существовавшими переменными, поэтому часто человек-эксперт, проверив локальные распределения условных вероятностей, включающие новую переменную, сможет установить ее смысл.

Как и в случае полных данных, процесс определения структуры с помощью обучения с учетом максимального правдоподобия в чистом виде приводит к созданию полносвязной сети (более того, сети без скрытых переменных), поэтому требуется ввести какую-то форму штрафования за сложность. Также для аппроксимации байесовского обучения можно применить алгоритм MCMC для выборки многих возможных сетевых структур. Например, можно предусмотреть

определение в процессе обучения параметров смешанного гауссова распределения с неизвестным количеством компонентов, осуществляя выборки по некоторому количеству; приближенное распределение апостериорных вероятностей для количества заданных гауссовых распределений определяется частотами выборки в процессе применения алгоритма MCMC.

В случае полных данных внутренний цикл выполняется очень быстро, поскольку при этом достаточно лишь извлечь информацию об условных частотах из набора данных. Если же имеются скрытые переменные, то для выполнения внутреннего цикла может потребоваться применить много итераций алгоритма EM или алгоритма на основе градиента, а каждая итерация потребует вычисления распределений апостериорных вероятностей в байесовской сети, что само по себе представляет NP-трудную задачу. К настоящему времени доказано, что такой подход является практически не применимым для обучения сложных моделей.

Одно из возможных усовершенствований состоит в использовании так называемого алгоритма ►**структурного EM**, который действует во многом таким же образом, как и обычный (параметрический) алгоритм EM, за исключением того, что этот алгоритм может обновлять не только параметры, но и структуру. Так же, как в обычном алгоритме EM используются текущие параметры для вычисления ожидаемых количеств на E-этапе, а затем эти количества применяются на M-этапе для выбора новых параметров, так и в алгоритме структурного EM используется структура для вычисления ожидаемых количеств, после чего эти количества применяются на M-этапе для оценки правдоподобия потенциальных новых структур (в этом состоит отличие данного метода от метода внешнего цикла/внутреннего цикла, в котором вычисляются новые ожидаемые количества для каждой потенциальной структуры). Таким образом, структурный алгоритм EM позволяет вносить в сеть несколько структурных изменений без каких-либо повторных вычислений ожидаемых количеств и обладает способностью определять в процессе обучения нетривиальные структуры байесовских сетей. Структурный алгоритм EM поддерживает пространство поиска по пространству структур, а не пространству структур и параметров. Тем не менее многое еще предстоит сделать, прежде чем можно будет сказать, что проблема структурного обучения решена.

Резюме

Статистические методы обучения охватывают широкий диапазон, начиная от простого вычисления средних и заканчивая построением сложных моделей, таких как байесовские сети. Эти методы находят широкое применение в компьютерных науках, техническом проектировании, вычислительной биологии, нейробиологии, психологии и физике. В данной главе представлены некоторые основные идеи из этой области и приведена часть математических выкладок. Основные темы, рассматриваемые в этой главе, следующие.

- В методах **байесовского обучения** задача обучения формулируется как один из видов вероятностного вывода, в котором наблюдения используются для обновления распределений априорных вероятностей по гипотезам. Такой подход представляет собой хороший способ реализации принципа бритвы Оккама, но очень быстро становится трудноосуществимым при возрастании сложности пространства гипотез.
- В методе обучения на основе **максимальной апостериорной вероятности** (*maximum a posteriori* — MAP) выбирается единственная, наиболее вероятная согласно имеющимся данным гипотеза. При этом все еще используется распределение априорных вероятностей гипотезы, но данный метод часто в большей степени осуществим, чем полное байесовское обучение.
- В обучении с учетом **максимального правдоподобия** просто выбирается гипотеза, которая максимизирует правдоподобие данных, — этот метод эквивалентен методу обучения MAP с равномерным распределением априорных вероятностей. В простейших случаях, таких как линейная регрессия и полностью наблюдаемые байесовские сети, решения с учетом максимального правдоподобия можно легко найти в замкнутой форме. Особенно эффективным методом, который хорошо масштабируется, является **наивное байесовское** обучение.
- Если некоторые переменные скрыты, то можно найти решения с локальным максимальным правдоподобием, воспользовавшись алгоритмом **ожидание–максимизация** (*expectation maximization* — EM). В число приложений соответствующего метода входит кластеризация с помощью смешанных гауссовых распределений, обучение байесовских сетей и скрытых марковских моделей.
- Определение на основе обучения структур байесовских сетей представляет собой пример метода **выбора модели**. В этом методе обычно предусматривается дискретный поиск в пространстве структур. При этом необходимо каким-то образом обеспечить поиск компромисса между сложностью модели и степенью ее соответствия данным.
- **Непараметрические модели** представляют распределение с использованием набора точек данных. В результате количество параметров модели растет с

увеличением обучающего набора. В методах ближайших соседей рассматриваются примеры, ближайшие к поступившему запросу, тогда как в **ядерных** методах образуется взвешенная по расстоянию комбинация всех примеров.

Статистическое обучение продолжает оставаться очень активной областью исследований. Были достигнуты колоссальные успехи как в теории, так и в практике, и на данный момент есть возможность определить в процессе обучения параметры почти любой модели, для которой осуществим точный или приближенный вероятностный вывод.

Библиографические и исторические заметки

В первые годы развития искусственного интеллекта приложения статистических методов обучения были активной областью исследований (см. Дуда и Харт [659], 1973), но отделились от основного направления искусственного интеллекта после того, как работы в этом направлении сосредоточились на символических методах. Возрождение всеобщего интереса к этой теме началось вскоре после появления моделей байесовских сетей в конце 1980-х годов; приблизительно в то же время начала формироваться статистическая научная трактовка процесса обучения нейронных сетей. В конце 1990-х годов произошло заметное пробуждение интереса к машинному обучению, статистике и нейронным сетям и значительные усилия были сосредоточены на разработке методов создания больших вероятностных моделей на основе данных.

Наивная байесовская модель представляет собой одну из самых старых и наиболее простых форм байесовской сети, которая впервые была описана в 1950-х годах. Об ее истоках упоминалось в аналогичном разделе главы 12, а ее удивительный успех был частично объяснен Домингосом и Паццани в работе [631] (1997). Программа на основе наивного байесовского обучения с использованием бустинга стала победителем первого соревнования по интеллектуальному анализу данных на кубок KDD Cup (Элкан [681], 1997). Эккерман в работе [1000] (1998) дал превосходное введение в общую проблему обучения байесовской сети. Определение параметров байесовской сети с помощью распределений априорных вероятностей Дирихле для байесовских сетей рассматривалось Шпигельхалтером и соавт. в статье [2114] (1993). Бета-распределение как сопряженное априорное распределение для переменной Бернулли было впервые предложено Томасом Байесом ([148], 1763), а затем вновь введено Карлом Пирсоном ([1759], 1895) в качестве модели искаженных данных, — в течение многих лет она была известна как "распределение Пирсона 1-го типа". Байесовская линейная регрессия обсуждается в учебнике Бокса и Тяо ([277], 1973), а в работе Минка [1579] (2010) содержится краткое изложение выводов для общего многомерного случая.

Несколько пакетов программного обеспечения включают механизмы для статистического обучения с использованием моделей байесовских сетей. К ним

относятся BUGS (*Bayesian inference Using Gibbs Sampling*) (Гилкс и др. [855], 1994; Лунн и др. [1467], 2000, [1466], 2013), JAGS (*Just Another Gibbs Sampler*) (Пламмер [1799], 2003) и STAN (Карпентер и др. [375], 2017).

В первых алгоритмах обучения структур байесовских сетей использовались проверки условной независимости (Перл [1755], 1988; Перл и Верма [1758], 1991). Спиртес и соавт. ([2115], 1993) разработали и описали исчерпывающий подход в пакете TETRAD для обучения байесовских сетей. Достигнутые с тех пор усовершенствования алгоритмов привели к убедительной победе метода обучения байесовской сети в соревновании по интеллектуальному анализу данных на кубок KDD Cup 2001 года (Ченг и др. [416], 2002). (На этих соревнованиях рассматривалась конкретная задача из биоинформатики с 139 351 характеристикой!) Подход к определению структуры сети в процессе обучения, основанный на учете максимального правдоподобия, был разработан Купером и Херсковицем ([475], 1992), а затем усовершенствован Хекерманом и соавт. ([1001], 1994).

Более поздние алгоритмы достигли весьма приличной производительности в случае полных данных (Мур и Вонг [1612], 2003; Тейсье и Коллер [2196], 2005). Одним из важных компонентов является эффективная структура данных, AD-дерево, предназначенное для кеширования результатов расчетов по всем возможным комбинациям переменных и значений (Мур и Ли [1614], 1997). Фридман и Голдшмидт в работе [790] (1996) указали на влияние представления локальных условных распределений на структуру, полученную в результате обучения.

Общая задача обучения вероятностных моделей со скрытыми переменными и неполными данными была рассмотрена Хартли ([974], 1958), который описал общую идею того, что позже было названо алгоритмом EM, и дал несколько примеров. Следующим импульсом был алгоритм Баума–Уэлша для обучения скрытых марковских моделей (HMM) (Баум и Питри [143], 1966), представлявший собой частный случай алгоритма EM. В статье Демпстера, Лейрда и Рубина [600] (1977) был представлен алгоритм EM в общей форме и дан анализ его сходимости, — эта работа является одной из самых цитируемых как в компьютерных науках, так и в статистике. (Сам Демпстер рассматривал алгоритм EM скорее как схему, а не как алгоритм в чистом виде, поскольку, прежде чем получить возможность применить подход на основе EM к новому семейству распределений, может потребоваться выполнить большой объем математической работы.) Маклахлан и Кришнан посвятили этому алгоритму и его свойствам целую книгу ([1551], 1997). Конкретная задача обучения моделей на основе смешанных распределений, включая смешанные гауссовы распределения, рассматривается Титтерингтоном и соавт. в работе [2218] (1985).

В рамках искусственного интеллекта первой успешной системой, в которой алгоритм EM использовался для моделирования смешанных распределений, была система AUTOCLASS (Чиземан и др. [408], 1988; Чиземан и Штутц [409], 1996). Система AUTOCLASS применялась для решения многих реальных задач научной классификации, включая открытие новых типов звезд на основе спектральных данных

(Гёбель и др. [875], 1989) и новых классов белков и интронов в базах данных последовательностей "ДНК/белок" (Хантер и Стейтс [1099], 1992).

Для обучения параметров максимального правдоподобия в байесовских сетях со скрытыми переменными, помимо алгоритма EM, использовались методы на основе градиента, предложенные примерно в то же время Лауритценом ([1359], 1995) и Расселом и др. ([1943], 1995). Структурный алгоритм EM был разработан Фридманом ([789], 1998) и применяется для обучения методом максимального правдоподобия структур байесовских сетей со скрытыми переменными. Фридман и Коллер в работе [791] (2003) описывают обучение байесовских структур. Дали и соавт. ([520], 2011) дают обзор области обучения байесовских сетей, предоставляя множество ссылок на литературу.

Способность к обучению структуры байесовских сетей тесно связана с вопросом извлечения *причинной* информации из данных. Существует ли возможность обучать байесовские сети таким образом, чтобы полученная структура сети демонстрировала реальные причинные связи? В течение многих лет статистики избегали анализа этого вопроса, считая, что данные самих наблюдений (в отличие от данных, выработанных в результате экспериментальных попыток) могут предоставить только информацию о корреляции; в конце концов, любые две переменные, которые кажутся взаимосвязанными, могут в действительности испытывать влияние третьего, неизвестного причинного фактора, а не влиять друг на друга непосредственно. Перл в работе [1756] (2000) представил убедительные доводы, опровергающие это мнение, и показал, что фактически возникает много ситуаций, в которых причинно-следственные связи можно подтвердить и выявить с помощью формальных средств **причинной сети** для выражения причин и результатов вмешательства, а также обычных условных вероятностей.

Оценка непараметрической плотности, также называемая оценкой плотности методом **парзеновского окна**, изначально исследовалась Розенблаттом ([1913], 1956) и Парзеном ([1737], 1962). С того времени был опубликован огромный объем материалов, полученных в результате исследований свойств различных методов оценки. Девруа ([616], 1987) дал подробное введение в эту тему. Также быстро увеличивается объем литературы о непараметрических байесовских методах, изначально предложенных в плодотворной работе Фергюсона ([731], 1973) о ►**процессах Дирихле**, которые можно рассматривать как распределение по распределениям Дирихле. Эти методы являются особенно полезными для смешанных распределений с неизвестным количеством компонентов. Гхахрамани ([845], 2005) и Джордан ([1149], 2005) предоставили полезные учебные пособия по многим приложениям этих идей для статистического обучения. Учебник Расмуссена и Вильямса ([1856], 2006) охватывает ►**гауссов процесс**, предоставляющий способ определения априорных распределений по пространству непрерывных функций.

В материалах этой главы собраны вместе работы из области статистики и распознавания образов, поэтому излагаемые в ней сведения освещались в литературе много раз и многими способами. К хорошим учебникам по байесовской статистике

относятся книги ДеГрота ([592], 1970), Бергера ([179], 1985), Гельмана и соавт. ([832], 1995). Книги Бишопа ([222], 2007), Хасти и соавт. ([981], 2009), Барбера ([131], 2012) и Мерфи ([1638], 2012) представляют собой превосходное введение в методы статистического машинного обучения. Классическим учебником по классификации образов в течение многих лет была книга Дуды и Харта ([659], 1973), которая позднее была переиздана в обновленном виде (Дуда и др. [660], 2001). Ежегодно проводится конференция NeurIPS (*Neural Information Processing Systems*, ранее *NIPS*), — ее труды публикуются в виде серии *Advances in Neural Information Processing Systems*, включающей много статей по обучению байесовских сетей, которые появляются также в трудах ежегодной конференции *Artificial Intelligence and Statistics*. Непосредственно байесовской тематике посвящены также симпозиум *Valencia International Meetings on Bayesian Statistics* и журнал *Bayesian Analysis*.

Упражнения

20.1. Данные, которые использовались для графика, приведенного на рис. 20.1, можно рассматривать как сформированные с помощью гипотезы h_5. Для каждой из остальных четырех гипотез сформируйте набор данных длиной 100 и вычертите соответствующие графики для $P(h_i \mid d_1,\ldots,d_N)$ и $P(D_{N+1} = lime \mid d_1,\ldots,d_N)$. Прокомментируйте полученные результаты.

20.2. Повторите упражнение 20.1, но на этот раз нанесите на графики значения $P(D_{N+1} = lime \mid h_{MAP})$ и $P(D_{N+1} = lime \mid h_{ML})$.

20.3. Предположим, что для Анны полезности леденцов со вкусом вишни и лайма равны c_A и ℓ_A, а для Боба эти полезности равны c_B и ℓ_B. (Также известно, что, если Анна снимает с леденца обертку, Боб эту конфету уже не покупает.) Можно предположить, что если Боб любит леденцы со вкусом лайма гораздо больше, чем Анна, то со стороны Анны будет разумно решить продать Бобу свой пакет с леденцами после того, как она в достаточной степени убедится, что в этом пакете находятся леденцы со вкусом лайма. С другой стороны, если при оценке содержимого пакета Анна развернет слишком много конфет, стоимость пакета уменьшится, поскольку Боб не станет платить за развернутые конфеты. Рассмотрите задачу определения оптимальной точки, в которой Анна должна принимать решение о продаже своего пакета. Найдите ожидаемую полезность этой оптимальной процедуры с учетом распределения априорных вероятностей, приведенного в разделе 20.1.

20.4. Два статистика побывали на приеме у врача, который поставил им одинаковый диагноз: с вероятностью 40% проблемы с их здоровьем связаны со смертельным заболеванием A, а с вероятностью 60% оно вызвано тяжелым заболеванием B. К счастью, имеются лекарства и от заболевания A, и от заболевания B, которые недороги, эффективны на 100% и не вызывают побочных эффектов. Таким образом, у обоих статистиков есть возможность выбрать, как поступить: принимать одно из этих лекарств, оба эти лекарства или ни одного из них. Как

поступит первый статистик, являющийся убежденным сторонником байесовского подхода? А как поступит второй статистик, который обычно предпочитает использовать гипотезу с максимальным правдоподобием?

Врач провел некоторые исследования и обнаружил, что заболевание B в действительности протекает в двух вариантах (правостороннее заболевание B и левостороннее заболевание B), которые являются равновероятными и одинаково хорошо излечиваются с помощью лекарства против заболевания B. Теперь, когда количество гипотез стало равным трем, как поступят эти два статистика?

20.5. Объясните, как метод бустинга, описанный в разделе 19.8.4, можно применить к задаче наивного байесовского обучения. Проверьте производительность результирующего алгоритма на обучении модели для задачи с ожиданием в ресторане.

20.6. Рассмотрим N точек данных (x_j, y_j), где координаты y_j определяются на основании координат x_j в соответствии с моделью линейного гауссова распределения, приведенной в уравнении (20.5). Найдите значения θ_1, θ_2 и σ, максимизирующие условное логарифмическое правдоподобие этих данных.

20.7. Рассмотрим модель зашумленного OR для задачи о повышенной температуре, описанной в разделе 13.2.2. Объясните, как применить обучение с учетом максимального правдоподобия для согласования параметров такой модели с множеством полных данных. (*Подсказка.* Используйте цепное правило для частичных производных.)

20.8. В этом упражнении исследуются свойства бета-распределения, которое определено в уравнении (20.6).

а) Выполнив интегрирование по отрезку $[0, 1]$, покажите, что константа нормализации для распределения $[a, b]$ задается выражением $\alpha = \Gamma(a+b)/\Gamma(a)\Gamma(b)$, где $\Gamma(x)$ — **гамма-функция**, определяемая выражением $\Gamma(x+1) = x \cdot \Gamma(x)$, а $\Gamma(1) = 1$ (для целого числа x, $\Gamma(x+1) = x!$).

б) Покажите, что ожидаемое среднее значение равно $a/(a+b)$.

в) Найдите моду (моды) — наиболее вероятное значение (значения) θ.

г) Опишите вид распределения $[\epsilon, \epsilon]$ для очень малого значения ϵ. Что происходит при обновлении такого распределения?

20.9. Рассмотрим произвольную байесовскую сеть, полный набор данных для этой сети и правдоподобие этого набора данных согласно этой сети. Дайте простое доказательство того, что правдоподобие данных не может уменьшиться после добавления к сети новой связи и повторного вычисления значений параметров максимального правдоподобия.

20.10. Рассмотрим единственную булеву случайную величину Y ("классификация"). Пусть априорная вероятность $P(Y = true)$ равна π. Попробуйте найти π при заданном обучающем множестве $D = (y_1, \ldots, y_N)$ с N независимых выборок из Y. Кроме того, предположим, что p значений из N положительны, а n значений из N отрицательны.

а) Запишите выражение для правдоподобия последовательности D (т.е. вероятность увидеть эту конкретную последовательность примеров при фиксированном значении π) в терминах π, p и n.

б) Продифференцировав логарифмическое правдоподобие L, найдите значение π, максимизирующее это правдоподобие.

в) Теперь предположим, что в сеть добавляется k булевых случайных переменных $X_1, X_2, \ldots, X_k$ ("атрибуты"), описывающих каждый пример, и допустим, что все эти атрибуты условно независимы друг от друга при заданной цели Y. Нарисуйте байесовскую сеть, соответствующую этому предположению.

г) Запишите правдоподобие для следующих данных, включая атрибуты, воспользовавшись следующими дополнительными обозначениями.
- α_i — это $P(X_i = true \mid Y = true)$.
- β_i — это $P(X_i = true \mid Y = false)$.
- p_i^+ — это количество примеров, для которых $X_i = true$ и $Y = true$.
- n_i^+ — это количество примеров, для которых $X_i = false$ и $Y = true$.
- p_i^- — это количество примеров, для которых $X_i = true$ и $Y = false$.
- n_i^- — это количество примеров, для которых $X_i = false$ и $Y = false$.

(*Подсказка.* Сначала рассмотрите вероятность увидеть один пример с указанными значениями для $X_1, X_2, \ldots, X_k$ и Y.)

д) Продифференцировав логарифмическое правдоподобие L, найдите значения α_i и β_i (в терминах различных количеств), максимизирующих вероятность, и на словах поясните, что представляют эти значения.

е) Пусть $k = 2$, рассмотрим набор данных со всеми четырьмя возможными примерами функции XOR. Рассчитайте оценку максимального правдоподобия для π, α_1, α_2, β_1 и β_2.

ж) При известных значениях оценок для π, α_1, α_2, β_1 и β_2 каковы будут апостериорные вероятности $P(Y = true \mid x_1, x_2)$ для каждого примера?

20.11. Рассмотрим применение алгоритма EM для обучения параметров сети, приведенной на рис. 20.14, *а*, если даны истинные параметры в уравнении (20.7).

а) Объясните, почему алгоритм EM не будет работать, если в модели имеются только два атрибута, а не три.

б) Приведите расчеты для первой итерации алгоритма EM, начиная с уравнения (20.8).

в) Что произойдет, если применение алгоритма начинается с присваивания всем параметрам одинакового значения p? (*Подсказка.* Рекомендуется вначале провести эмпирическое исследование этого вопроса и только после этого выводить общий результат.)

г) Запишите выражение для логарифмического правдоподобия табличных данных о леденцах, приведенных в разделе 20.3.2, в терминах параметров, рассчитайте частичные производные по отношению к каждому параметру и исследуйте характер фиксированной точки, достигнутой при выполнении п. *в* этого упражнения.

ГЛАВА 21

Глубокое обучение

В этой главе рассматривается, как метод градиентного спуска можно применить при обучении в многослойных структурах, реализованных программным путем, и насколько важное влияние этот подход оказал на результаты, достигнутые в важнейших областях искусственного интеллекта.

Понятие ▶ **глубокое обучение** — охватывает широкое семейство методов машинного обучения, в которых гипотезы принимают форму сложных алгебраических схем с настраиваемой силой связей. Слово "глубокий" связано с тем фактом, что эти схемы, как правило, организованы в виде нескольких ▶ **слоев**, а это означает, что пути вычислений от входов до выходов включают много этапов. В настоящее время глубокое обучение является наиболее широко используемым подходом для приложений визуального распознавания объектов, машинного перевода, распознавания речи, синтеза речи и синтеза изображений; а также оно играет значительную роль в приложениях обучения с подкреплением (см. главу 22).

Глубокое обучение берет свое начало в ранней работе, описывающей попытку смоделировать сети нейронов в мозгу человека с помощью вычислительных схем (Мак-Калок и Питс [1537], 1943). Именно по этой причине сети, обученные методами глубокого обучения, часто называют ▶ **нейронными сетями**, хотя их сходство с реальными клетками-нейронами и их структурами весьма поверхностно.

Не смотря на то, что истинные причины успеха глубокого обучения еще предстоит выяснить, оно имеет очевидные преимущества по сравнению с некоторыми методами, описанными в главе 19, особенно для многомерных данных, таких как изображения. Например, хотя такие методы, как линейная и логистическая регрессия, позволяют работать с большим количеством входных переменных, вычислений, выполняемых на пути от каждого входа до выхода, очень мало: умножение на единственный весовой коэффициент, а затем добавление в совокупный выход. Более того, различные входные переменные вносят независимый вклад в выходное значение, не взаимодействуя друг с другом (рис. 21.1, *а*). Это существенно ограничивает выразительную силу таких моделей. Они могут представлять только линейные функции и границы во входном пространстве, тогда как большинство концепций реального мира гораздо сложнее.

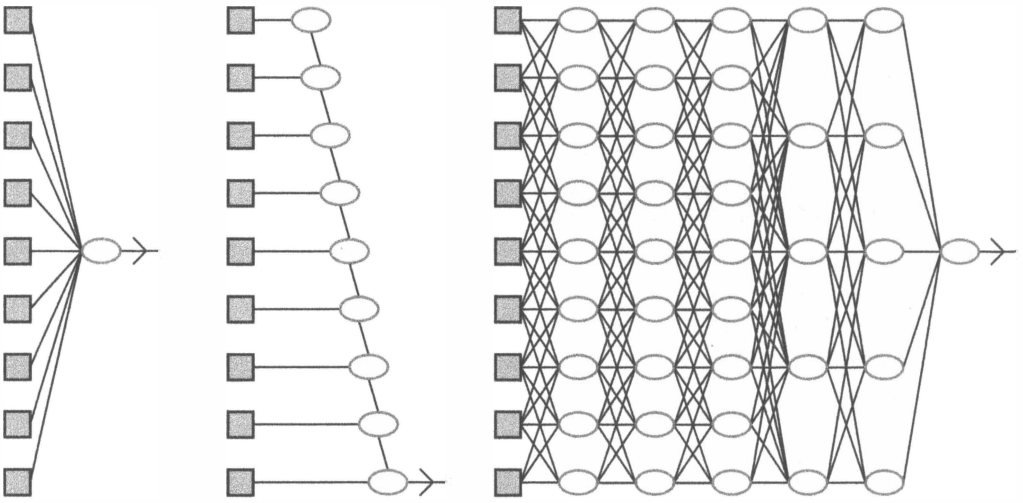


Рис. 21.1. а) Неглубокой модели, такой как линейная регрессия, свойственны короткие пути вычислений между входами и выходами. **б)** Сеть списка решений (раздел 19.5.1) имеет несколько длинных путей для некоторых возможных входных значений, но большинство путей вычислений короткие. **в)** Сеть глубокого обучения обладает длинными путями вычислений, позволяющими каждой переменной взаимодействовать со всеми остальными

С другой стороны, списки решений и деревья решений допускают длинные пути вычислений, которые могут зависеть от многих входных переменных, — но только для относительно небольшого диапазона возможных векторов входных значений (рис. 21.1, *б*). Если дерево решений имеет длинные пути вычислений для значительного диапазона возможных входных данных, оно непременно будет экспоненциально большим по количеству входных переменных. Основная идея глубокого обучения состоит в обучении схем таким образом, чтобы пути вычислений были длинными, позволяя всем входным переменным сложным образом взаимодействовать между собой (рис. 21.1, *в*). Подобные схемные модели оказываются достаточно выразительными, чтобы охватить сложность реальных данных для многих важных типов задач обучения.

В разделе 21.1 описываются простые сети с прямой связью, их компоненты и основы обучения в таких сетях. В разделе 21.2 более подробно рассказывается о том, как собираются глубокие сети, а в разделе 21.3 обсуждается класс сетей, называемых сверточными нейронными сетями, которые особенно важны для приложений, связанных с визуальным представлением. В разделах 21.4 и 21.5 более подробно рассматриваются алгоритмы обучения сетей на основе данных и методы

улучшения обобщения. В разделе 21.6 анализируются сети с рекуррентной структурой, которые лучше всего подходят для последовательных данных, а в разделе 21.7 описываются способы использования глубокого обучения для задач, отличающихся от обучения с учителем. Наконец в разделе 21.8 рассматривается весь диапазон приложений на основе глубокого обучения.

21.1. Простые сети с прямой связью

▶**Сеть с прямой связью**, как следует из названия, имеет соединения, действующие только в одном направлении, — иначе говоря, эта сеть образует ориентированный ациклический граф с указанными входными и выходными узлами. В каждом узле вычисляется некоторая функция от его входных значений и результат передается преемникам узла в сети. Информация проходит через сеть от входных узлов к выходным узлам, при этом петли или циклы отсутствуют. С другой стороны, ▶**рекуррентная сеть** передает свои промежуточные или конечные результаты в обратном направлении, к ее собственным входам. Это означает, что значения сигнала в сети образуют динамическую систему, имеющую внутреннее состояние или память. Рекуррентные сети будут рассматриваться в разделе 21.6.

Логические схемы, реализующие булевы функции, являются примером сетей с прямой связью. В логической схеме значения входных сигналов ограничены 0 и 1, и каждый узел реализует простую логическую функцию от значений своих входов, выдавая на выход 0 или 1. В нейронных сетях входные значения, как правило, являются непрерывными, и узлы принимают непрерывные входные значения и выдают непрерывные выходные значения. Некоторые из входных значений узлов являются **параметрами** сети, — сеть учится, настраивая значения этих параметров таким образом, чтобы вся сеть как целое соответствовала данным обучающего набора.

21.1.1. Сети как сложные функции

Каждый узел в сети называется **элементом** (*unit*). Традиционно, в соответствии с моделью, предложенной Мак-Каллоком и Питтсом, элемент сначала вычисляет взвешенную сумму входных данных от предшествующих узлов, а затем применяет к ней некоторую нелинейную функцию для получения выходного результата. Пусть a_j обозначает выход элемента j и пусть $w_{i,j}$ будет весовым коэффициентом, присвоенным связи от элемента i к элементу j. Тогда мы имеем

$$a_j = g_j(\Sigma_i w_{i,j} a_i) \equiv g_j(in_j),$$

где g_j — нелинейная ▶**функция активации**, связанная с элементом j, а in_j — взвешенная сумма входов элемента j.

Как и в разделе 19.6.3, поставим условием, что каждый элемент имеет дополнительный вход от фиктивного элемента 0, выход которого равен +1 с весом $w_{0,j}$ для этого входа. Это позволит суммарному взвешенному входному значению in_j

элемента j быть отличным от нуля даже в том случае, когда все выходы предыдущего слоя будут нулевыми. С учетом этого соглашения предыдущее уравнение можно переписать в векторной форме:

$$a_j = g_j(\mathbf{w}^\top \mathbf{x}), \tag{21.1}$$

где $\mathbf{w}$ является вектором весов, ведущих в элемент j (включая w_{0j}), а $\mathbf{x}$ — вектор входов в элемент j (включая +1).

Тот факт, что функция активации является нелинейной, важен потому, что, если бы это было не так, любой состав элементов по-прежнему представлял бы линейную функцию. Нелинейность — это как раз то, что позволяет достаточно большим сетям элементов представлять произвольные функции. ►**Универсальная теорема аппроксимации** утверждает, что сеть с двумя слоями вычислительных элементов, нелинейных в первом слое и линейных во втором, способна аппроксимировать любую непрерывную функцию с любой степенью точности. Доказательство строится на том, что экспоненциально большая сеть может представить экспоненциально много "выпуклостей" разной высоты в разных местах входного пространства, тем самым аппроксимируя желаемую функцию. Другими словами, достаточно большие сети могут реализовать таблицу поиска для непрерывных функций точно так, как достаточно большие деревья решений позволяют реализовать таблицу поиска для булевых функций.

Широко используется множество различных функций активации. Наиболее распространенными из них являются следующие.

- Логистическая функция, или ►**сигмоида**, которая также применяется в логистической регрессии (см. раздел 19.6.5):

$$\sigma(x) = 1/(1 + e^{-x}).$$

- Функция ►**ReLU**, название которой является аббревиатурой от **rectified linear unit** (*выпрямленный линейный элемент*):

$$\mathrm{ReLU}(x) = \max(0, x).$$

- Функция ►**softplus** — гладкая версия функции ReLU:

$$\mathrm{softplus}(x) = \log(1 + e^{x}).$$

  Производной от функции softplus является сигмоида (логистическая функция).

- Функция ►**tanh** — гиперболический тангенс:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

  Обратите внимание, что диапазон функции гиперболического тангенса равен $(-1, +1)$. Функция tanh — это уменьшенная и смещенная версия сигмоиды, так как $\tanh(x) = 2\sigma(2x) - 1$.

Графики упомянутых выше функций приведены на рис. 21.2. Обратите внимание, что все они монотонно неубывающие, а это означает, что их производные g' неотрицательны. Выбор функции активации будет подробно обсуждаться в следующих разделах.

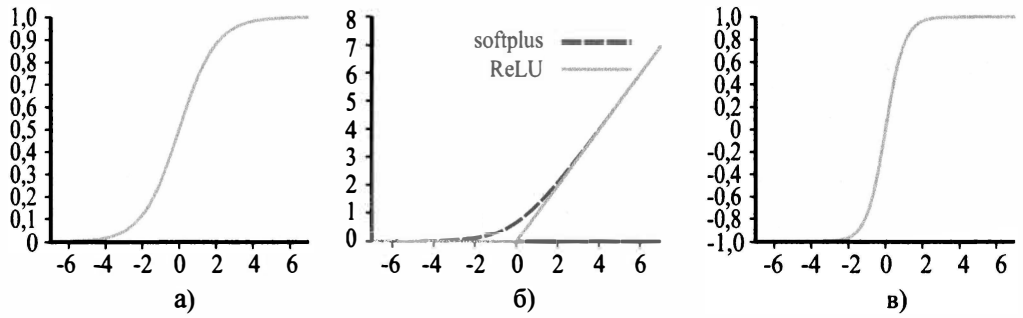


Рис. 21.2. Функции активации, обычно используемые в системах глубокого обучения. **а)** Логистическая функция, или сигмоида. **б)** Функции ReLU и softplus. **в)** Функция гиперболического тангенса tanh

Соединение нескольких элементов в сеть создает сложную функцию, которую можно рассматривать как композицию алгебраических выражений, представляемых отдельными элементами. Например, приведенная на рис. 21.3, *а* сеть представляет функцию $h_{\mathbf{w}}(\mathbf{x})$, параметризованную весами $\mathbf{w}$, которая отображает входной вектор $\mathbf{x}$ с двумя элементами в скалярное выходное значение $\hat{y}$. Внутренняя структура этой функции отражает структуру сети. Например, выражение для выходной функции $\hat{y}$ в данном случае можно записать следующим образом:

$$\begin{aligned}\hat{y} &= g_5(in_5) = g_5(w_{0,5} + w_{3,5}\,a_3 + w_{4,5}\,a_4) = \\ &= g_5(w_{0,5} + w_{3,5}\,g_3(in_3) + w_{4,5}\,g_4(in_4)) = \\ &= g_5(w_{0,5} + w_{3,5}\,g_3(w_{0,3} + w_{1,3}\,x_1 + w_{2,3}\,x_2) + \\ &\qquad + w_{4,5}\,g_4(w_{0,4} + w_{1,4}\,x_1 + w_{2,4}\,x_2)).\end{aligned} \tag{21.2}$$

Таким образом, мы получили выход $\hat{y}$, выраженный как функция $h_{\mathbf{w}}(\mathbf{x})$ от входов и весов.

На рис. 21.3, *а* приведен традиционный способ, которым сеть может быть представлена в книге о нейронных сетях. Более общее средство представления сети — это ►**граф вычислений** (или ►**граф потока данных**), по существу являющийся схемой, в которой каждый узел представляет элементарные вычисления. На рис. 1.3, *б* приведен граф вычислений, соответствующий сети на рис. 21.3, *а*: на нем явно представлен каждый элемент общих вычислений. Здесь также отмечено различие между входами (темно-серая заливка) и весами (светло-серая заливка): веса можно настроить так, чтобы выходной сигнал $\hat{y}$ более точно соответствовал

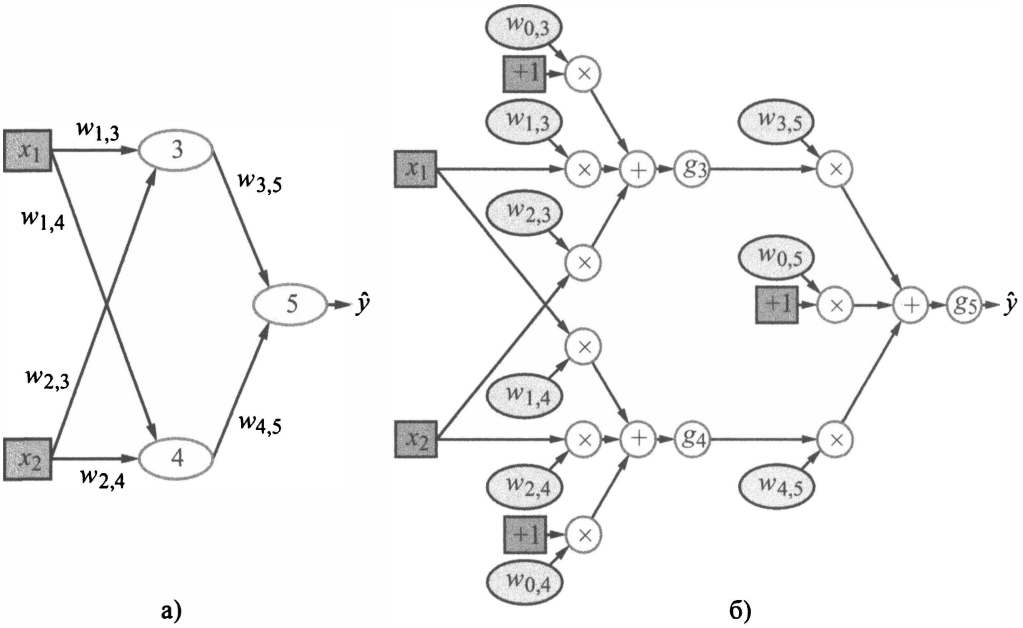


Рис. 21.3. а) Нейронная сеть с двумя входами, одним скрытым слоем из двух элементов и одним выходным элементом (фиктивные входные данные и связанные с ними веса не показаны). **б)** Та же сеть, распакованная в полный граф вычислений

истинному значению y в обучающих данных. Каждый вес напоминает ручку регулятора громкости, определяющего, в какой степени следующий узел в графе будет "слышать" этого конкретного предшественника в графе.

Подобно тому, как уравнение (21.1) в векторной форме описывает работу одного элемента, можно сделать нечто похожее и для сети в целом. Обычно для обозначения матрицы весов используется нотация $\mathbf{W}$, — для приведенной на рис. 21.3 сети $\mathbf{W}^{(1)}$ будет обозначать веса первого слоя ($w_{1,3}$, $w_{1,4}$ и т.д.), а $\mathbf{W}^{(2)}$ — веса второго слоя ($w_{3,5}$ и т.д.). Наконец, пусть $\mathbf{g}^{(1)}$и $\mathbf{g}^{(2)}$ обозначают функции активации в первом и втором слоях. Тогда вся сеть может быть описана следующим образом:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{g}^{(2)}(\mathbf{W}^{(2)}\mathbf{g}^{(1)}(\mathbf{W}^{(1)}\mathbf{x})). \tag{21.3}$$

Как и уравнение (21.2), это выражение соответствует графу вычислений, хотя оно намного проще, чем граф, приведенный на рис. 21.3, *б*: здесь граф — это просто цепочка матриц весов, подаваемых на каждый слой.

Граф вычислений на рис. 21.3, *б* относительно невелик и неглубок, но подробная идея может применяться ко всем формам глубокого обучения: строится граф вычислений и веса в нем корректируются так, чтобы соответствовать данным.

Следует отметить, что граф на рис. 21.3, *б* является ▶**полносвязным**, — это означает, что каждый узел в каждом слое связан с каждым узлом в следующем слое. В каком-то смысле этот тип принимается по умолчанию, но в разделе 21.3 будет показано, что выбор связности сети также важен для достижении эффективного обучения.

21.1.2. Градиенты и обучение

В разделе 19.6.2 был представлен подход к обучению с учителем, основанный на **градиентном спуске**: рассчитать градиент функции потерь по отношению к весам и так настроить веса вдоль направления градиента, чтобы уменьшить потери. (Если вы еще не прочитали раздел 19.6.2, настоятельно рекомендуется сделать это, прежде чем продолжить чтение данного раздела.) Тот же самый подход можно применить и к обучению весов в графах вычислений. Для весов, ведущих к элементам ▶**выходного слоя**, т.е. к тем, которые формируют выход сети, вычисление градиента, по существу, идентично процессу, описанному в разделе 19.6.2. Для весов, ведущих к элементам ▶**скрытых слоев**, не связанных напрямую с выходами, процесс будет лишь немного сложнее.

В данном случае будем использовать квадратичную функцию потерь L_2 и рассчитаем градиент для приведенной на рис. 21.3 сети относительно одного примера обучения $(\mathbf{x}, y)$. (Градиент для нескольких примеров — это просто сумма градиентов для отдельных примеров.) Выходом сети является прогноз $\hat{y} = h_{\mathbf{w}}(\mathbf{x})$, а истинным значением является y, поэтому имеем

$$Loss(h_{\mathbf{w}}) = L_2(y, h_{\mathbf{w}}(\mathbf{x})) = \| y - h_{\mathbf{w}}(\mathbf{x}) \|^2 = (y - \hat{y})^2.$$

Чтобы вычислить градиент потерь относительно весов, необходимы те же инструменты расчета, которые использовались в главе 19, — главным образом **цепное правило**, $\partial g(f(x))/\partial x = g'(f(x))\partial f(x) / \partial x$. Начнем с простого случая: вес, например, $w_{3,5}$, связанный с выходным элементом. Будем работать непосредственно с определяющими сеть выражениями из уравнения (21.2):

$$\begin{aligned}
\frac{\partial}{\partial w_{3,5}} Loss(h_{\mathbf{w}}) &= \frac{\partial}{\partial w_{3,5}} (y - \hat{y})^2 = -2(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_{3,5}} = \\
&= -2(y - \hat{y}) \frac{\partial}{\partial w_{3,5}} g_5(in_5) = -2(y - \hat{y}) g_5'(in_5) \frac{\partial}{\partial w_{3,5}} in_5 = \\
&= -2(y - \hat{y}) g_5'(in_5) \frac{\partial}{\partial w_{3,5}} (w_{0,5} + w_{3,5} a_3 + w_{4,5} a_4) = \\
&= -2(y - \hat{y}) g_5'(in_5) a_3.
\end{aligned} \tag{21.4}$$

Упрощение в последней строке следует из-за того, что $w_{0,5}$ и $w_{4,5}a_4$ не зависят ни от $w_{3,5}$, ни от коэффициента при $w_{3,5}$, — a_3.

Несколько более сложный случай связан с таким весом, как $w_{1,3}$, который прямо не подключен к выходному элементу. Здесь потребуется применить цепное правило еще раз. Первые несколько этапов идентичны, поэтому здесь их опускаем:

$$\begin{aligned}\frac{\partial}{\partial w_{1,3}} Loss(h_{\mathbf{w}}) &= -2(y-\hat{y})g_5'(in_5)\frac{\partial}{\partial w_{1,3}}(w_{0,5}+w_{3,5}a_3+w_{4,5}a_4)= \\ &= -2(y-\hat{y})g_5'(in_5)w_{3,5}\frac{\partial}{\partial w_{1,3}}a_3= \\ &= -2(y-\hat{y})g_5'(in_5)w_{3,5}\frac{\partial}{\partial w_{1,3}}g_3(in_3)= \\ &= -2(y-\hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)\frac{\partial}{\partial w_{1,3}}in_3= \\ &= -2(y-\hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)\frac{\partial}{\partial w_{1,3}}(w_{0,3}+w_{1,3}x_1+w_{2,3}x_2)= \\ &= -2(y-\hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)x_1.\end{aligned} \tag{21.5}$$

Таким образом, у нас есть относительно простые выражения для градиента потерь по отношению к весам $w_{3,5}$ и $w_{1,3}$.

Если определить $\Delta_5 = 2(\hat{y}-y)\,g_5'(in_5)$ как своего рода "воспринимаемую ошибку" в точке, где элемент 5 получает свой вход, то градиентом по отношению к $w_{3,5}$ будет просто $\Delta_5 a_3$. Это имеет простой смысл: если значение Δ_5 положительное, то значение прогноза $\hat{y}$ слишком велико (напомним, что g' всегда неотрицательно). Если при этом значение a_3 также положительно, то увеличение веса $w_{3,5}$ только ухудшит ситуацию, тогда как если значение a_3 отрицательно, то увеличение веса $w_{3,5}$ приведет к уменьшению ошибки. Абсолютная величина a_3 также имеет значение: если для данного обучающего примера величина a_3 невелика, вес $w_{3,5}$ не играет главной роли в возникновении ошибки и не нуждается в значительном изменении.

Если также определить $\Delta_3 = \Delta_5 w_{3,5}\,g_3'(in_3)$, то градиент для $w_{1,3}$ будет просто $\Delta_3 x_1$. Поэтому воспринимаемая ошибка на входе элемента 3 — это воспринимаемая ошибка на входе элемента 5, умноженная согласно информации вдоль пути от элемента 5 до элемента 3. Это явление носит общий характер и в свое время послужило причиной появления термина ►**обратное распространение**, определяющего способ, которым ошибка на выходе передается по сети в обратном направлении.

Другой важной характеристикой этих градиентных выражений является то, что в качестве факторов они включают локальные производные $g_j'(in_j)$. Как отмечалось ранее, эти производные всегда неотрицательны, но они могут быть очень близки к нулю (в случае функций активации типа сигмоида, softplus и tanh) или точно равны нулю (в случае функции ReLU), если для входных данных

рассматриваемого обучающего примера элемент j случайно оказывается в плоской части рабочей области. Если производная g'_j мала или равна нулю, то это означает, что изменение весов, ведущих к элементу j, окажет совершенно незначительное влияние на его выход. В результате глубокие сети со многими слоями могут страдать от проблемы ▶**исчезновения градиента** (*vanishing gradient*), когда сигналы об ошибках полностью исчезают вследствие их обратного распространения по сети. В разделе 21.3.3 рассматривается одно из возможных решений этой проблемы.

Выше было показано, что градиенты в рассматриваемой в качестве примера крошечной сети — это простые выражения, которые могут быть вычислены путем передачи информации от выходных элементов по сети в обратном направлении. Как оказалось, это свойство имеет место и в общем случае. На самом деле, как будет показано в разделе 21.4.1, вычисления градиента для *любого* графа вычислений с прямой связью имеют ту же структуру, что и сам лежащий в их основе граф вычислений. Это свойство прямо следует из правил дифференцирования.

Выше были приведены довольно обременительные подробности вычисления градиента, однако не стоит беспокоиться: нет никакой необходимости повторять все приведенные в уравнениях (21.4) и (21.5) выводы для каждой новой сетевой структуры! Все эти градиенты могут быть вычислены методом ▶**автоматического дифференцирования**, когда правила исчисления систематически применяются для расчета градиентов любой числовой программы.[1] На самом деле метод обратного распространения в глубоком обучении — это просто применение ▶**обратного режима** дифференцирования, в котором цепное правило применяется "извне внутрь" и используются преимущества в эффективности динамического программирования, когда рассматриваемая сеть имеет много входов и относительно мало выходов.

Все ведущие программные пакеты глубокого обучения предоставляют средства автоматического дифференцирования, так что пользователь имеет возможность свободно экспериментировать с различной структурой сети, функциями активации, функциями потерь и формами композиции без необходимости выполнять множество расчетов, чтобы вывести новый алгоритм обучения для каждого эксперимента. Все это поощряет подход, называемый ▶**сквозным обучением** (*end-to-end learning*), в котором сложная вычислительная система для такой задачи, как машинный перевод, может быть собрана из нескольких обучаемых подсистем. Затем вся система обучается сквозным образом на парах "вход-выход". При таком подходе разработчику нужна лишь расплывчатая идея о том, как система в целом должна быть структурирована, и нет необходимости заранее точно знать, что именно должна делать каждая подсистема или как маркировать ее входы и выходы.

---

[1] Автоматические методы дифференцирования были разработаны в 1960-х и 1970-х годах и предназначались для оптимизации параметров систем, реализованных в виде больших и сложных программ на Фортране.

21.2. Графы вычислений для глубокого обучения

Выше были определены основные идеи глубокого обучения: представить гипотезы как графы вычислений с настраиваемыми весами и вычислить градиент функции потерь по отношению к этим весам с целью достижения соответствия обучающим данным. Теперь рассмотрим, как собирать воедино графы вычислений. Начнем с входного слоя, в котором обучающий набор или тестовый пример **x** кодируется в виде значений входных узлов. Затем будет рассмотрен выходной слой, в котором выходы $\hat{y}$ сравниваются с истинными значениями **y** с целью получения обучающего сигнала для настройки весов. В завершение будут рассмотрены скрытые слои сети.

21.2.1. Входное кодирование

Входными и выходными узлами графа вычислений являются такие узлы, которые прямо связаны с входными данными **x** и выходными данными **y**. Кодировка входных данных, как правило, проста: по крайней мере для случая развернутого представления данных, когда каждый обучающий пример содержит значения для n атрибутов входных данных. Если атрибуты логические, то будем иметь n входных узлов и в этом случае значению *false* обычно соответствует входное значение 0, а значению *true* — входное значение 1, хотя иногда используются и варианты −1 и +1. Числовые атрибуты, целочисленные или действительные, обычно используются в том виде, как они есть, хотя иногда могут масштабироваться, чтобы их можно было вписать в фиксированный диапазон. Если абсолютные величины числовых параметров в разных примерах сильно различаются, эти значения могут быть представлены в логарифмическом масштабе.

Изображения не вполне вписываются в категорию данных в развернутом представлении: хотя RGB-изображение размером $X \times Y$ пикселей вполне можно рассматривать как обладающее $3XY$ целочисленными атрибутами (обычно со значениями в диапазоне $\{0,\ldots,255\}$), при этом, однако, игнорируется тот факт, что триплеты RGB принадлежат одному пикселю в изображении, и еще более важный факт, что на самом деле соседство (или *смежность*) пикселей также имеет значение. Конечно, можно отобразить смежность пикселей через соседство представляющих их входных узлов в сети, но такое понятие смежности полностью утрачивается, если внутренние слои сети являются полносвязными. На практике сети, работающие с входными данными в виде изображений, имеют массивоподобную внутреннюю структуру, назначение которой — отразить семантику смежности пикселей. Подробнее речь об этом пойдет в разделе 21.3.

Категориальные атрибуты более чем с двумя значениями (например, атрибут *Type* в задаче об ожидании в ресторане из главы 19, имеющий четыре допустимых значения: *French*, *Italian*, *Thai* и *burger*) обычно кодируются с использованием так называемого **быстрого** (или **однократного**) **кодирования**. Атрибут с d возможными значениями представляется с помощью d отдельных входных битов.

Для любого заданного значения соответствующий входной бит устанавливается в 1, а все остальные $d-1$ бит устанавливаются в 0. Обычно этот подход работает лучше, чем отображение категориальных значений на целые числа. Так, если для кодирования атрибута *Type* использовать последовательные целые числа, то значение *Thai* будет представлено как 3, а значение *burger* — как 4. Поскольку сеть представляет собой совокупность непрерывных функций, она неизбежно будет учитывать смежность числовых значений, но в данном случае числовое соседство между значениями *Thai* и *burger* является семантически бессмысленным.

21.2.2. Выходные слои и функции потерь

На выходной стороне сети проблема кодирования значений необработанных данных в реальные значения y для выходных узлов графа во многом совпадает с проблемой входного кодирования. Например, если сеть должна прогнозировать значение переменной *Weather* из главы 12, имеющей четыре допустимых значения $\{sun, rain, cloud, snow\}$, можно было бы воспользоваться быстрым кодированием с четырьмя битами.

Но это не подходит в случае вектора значений выходных данных $\mathbf{y}$. Что можно сказать в отношении прогноза $\hat{\boldsymbol{y}}$? В идеале, если он точно соответствует желаемому значению $\mathbf{y}$, потери будут равны нулю и цель будет достигнута. Но на практике так случается редко — особенно в том случае, когда процесс корректировки весов еще не был начат! Следовательно, нужно подумать о том, что, собственно, означает это неправильное выходное значение и как можно измерить потери. В уравнениях (21.4) и (21.5) вывод градиентов начинается с вычисления квадратичной ошибки функции потерь. Подобный подход упрощает вычисления, но это не единственная возможность. В действительности в большинстве приложений глубокого обучения выходные значения $\hat{\boldsymbol{y}}$ чаще интерпретируются как вероятности и в качестве функции потерь используется **отрицательное логарифмическое правдоподобие** — точно так, как это делалось в главе 20 при обучении на основе **максимального правдоподобия**.

При обучении по методу максимального правдоподобия ищется значение $\mathbf{w}$, максимизирующее вероятность наблюдаемых данных. И поскольку функция логарифма монотонна, это эквивалентно максимизации логарифмического правдоподобия данных, что, в свою очередь, эквивалентно минимизации функции потерь, определенной как отрицательное логарифмическое правдоподобие. (Вспомните из раздела 20.2.1, что логарифмирование превращает произведение вероятностей в суммы, которые более удобны для вычисления производных.) Другими словами, мы ищем вектор весов $\mathbf{w}^*$, минимизирующий сумму отрицательных логарифмических правдоподобий N примеров

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} - \sum_{j=1}^{N} \log P_{\mathbf{w}}(\mathbf{y}_j \mid \mathbf{x}_j).$$

В литературе по глубокому обучению принято говорить о минимизации ▶**перекрестной энтропии** потерь. Перекрестная энтропия, обозначаемая как $H(P, Q)$, является своего рода мерой различия между распределениями P и Q.[2] Общее определение перекрестной энтропии следующее:

$$H(P,Q) = \mathbf{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log Q(\mathbf{z})] = \int P(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z}. \quad (21.7)$$

В машинном обучении это определение обычно используется при P, представляющем истинное распределение по обучающим примерам $P^*(\mathbf{x}, \mathbf{y})$, и Q, представляющем гипотезу прогноза $P_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x})$. Минимизация перекрестной энтропии $H(P^*(\mathbf{x},\mathbf{y}), P_{\mathbf{w}}(\mathbf{y} \mid \mathbf{x}))$ путем корректировки $\mathbf{w}$ делает гипотезу, насколько это возможно, согласованной с истинным распределением $P^*(\mathbf{x}, \mathbf{y})$. В действительности эту перекрестную энтропию минимизировать невозможно, поскольку нам неизвестно истинное распределение данных $P^*(\mathbf{x}, \mathbf{y})$, однако имеется доступ к образцам из $P^*(\mathbf{x}, \mathbf{y})$, поэтому сумма по фактическим данным в уравнении (21.6) аппроксимирует ожидание в уравнении (21.7).

Чтобы минимизировать отрицательную логарифмическую вероятность (или перекрестную энтропию), нужно иметь возможность интерпретировать выходные значения сети как вероятность. Например, если сеть имеет один выходной элемент с функцией активации типа сигмоида и она обучается булевой классификации, можно интерпретировать выходное значение непосредственно как вероятность того, что пример принадлежит положительному классу. (В действительности именно таким образом и работает логистическая регрессия, — см. раздел 19.6.5.) Поэтому для задач булевой классификации обычно используется сигмоидальный выходной слой.

Задачи многоклассовой классификации также очень распространены в машинном обучении. Например, программы-классификаторы, используемые для распознавания объектов, часто должны распознавать тысячи различных категорий объектов. Модели обработки естественного языка, пытающиеся предсказать следующее слово в предложении, могут оказаться в ситуации выбора среди десятков тысяч возможных слов. Для таких прогнозов необходима сеть с выводом в виде категориального распределения, т.е. при наличии d возможных ответов необходимо иметь d выходных узлов, представляющих вероятности, суммируемые к 1.

Чтобы достичь этого, обычно используется слой ▶**softmax**, выводящий вектор из d значений при заданном векторе его входных значений $\mathbf{in} = \langle in_1, \ldots, in_d \rangle$. Элемент k выходного вектора определяется как

---

[2] Перекрестная энтропия не является расстоянием в обычном смысле этого слова, поскольку $H(P, P)$ не равно нулю, — скорее это равно энтропии $H(P)$. Легко показать, что $H(P, Q) = H(P) + D_{KL}(P \parallel Q)$, где D_{KL} — **расстояние Кульбака–Лейблера**, удовлетворяющее равенству $D_{KL}(P \parallel P) = 0$. Таким образом, для фиксированного P варьирование Q с целью минимизации перекрестной энтропии также минимизирует расстояние Кульбака–Лейблера.

$$\text{softmax}(\mathbf{in})_k = \frac{e^{in_k}}{\sum_{k'=1}^{d} e^{in_{k'}}}.$$

По построению функция softmax выводит вектор неотрицательных чисел, сумма которых равна 1. Как обычно, входное значение in_k для каждого из выходных узлов будет взвешенной линейной комбинацией выходов предыдущего слоя. Благодаря экспоненциальности функции softmax, такой слой подчеркивает различия во входных данных; например, если вектор входных данных задан как $\mathbf{in} = \langle 5, 2, 0, -2 \rangle$, то выходными значениями будут $\langle 0{,}946; 0{,}047; 0{,}006; 0{,}001 \rangle$. Тем не менее, в отличие от функции max, функция softmax является гладкой и дифференцируемой. Так же легко показать, что сигмоида является функцией softmax при $d = 2$. Другими словами, так же, как элементы с сигмоидой передают через сеть информацию двоичного класса, элементы с функцией softmax передают через сеть многоклассовую информацию.

Для задачи регрессии, в которой целевое значение y непрерывно, обычно используют линейный выходной слой (другими словами, $\hat{y}_j = in_j$, без какой-либо функции активации g), интерпретируя его выход как среднее значение гауссова прогноза с фиксированной дисперсией. Как отмечалось в разделе 20.2.4, максимизация правдоподобия (т.е. минимизация отрицательного логарифмического правдоподобия) для гауссиана с фиксированной дисперсией — это то же самое, что минимизация квадратичной ошибки. Следовательно, линейный выходной слой в этом случае можно представлять как выполняющий классическую линейную регрессию. Входными признаками для этой линейной регрессии являются выходные данные предыдущего слоя, которые обычно являются результатом нескольких нелинейных преобразований исходных входных данных сети.

Возможно и множество других типов выходных слоев. Например, слой ▶ **смешанной плотности** представляет выходные значения, используя смешанные гауссовы распределения. (Подробнее о смешанных гауссовых распределениях читайте в разделе 20.3.1.) Такие слои прогнозируют относительную частоту каждого компонента смеси, а также его среднее значение и дисперсию. Если эти выходные значения соответствующим образом интерпретируются функцией потерь как определяющие вероятность истинного выходного значения $\mathbf{y}$, то после обучения сеть обеспечит соответствие модели гауссовой смеси в пространстве признаков, определяемых предыдущими слоями.

21.2.3. Скрытые слои

В процессе обучения нейронной сети передается много входных значений $\mathbf{x}$ и соответствующих выходных значений y. При обработке входного вектора $\mathbf{x}$ нейронная сеть выполняет несколько промежуточных вычислений, прежде чем получить выходное значение y. Вычисляемые на каждом слое сети значения можно рассматривать как различное *представление* для входного вектора $\mathbf{x}$. Каждый

слой преобразует созданное на предыдущем слое представление с целью создания нового представления. Соединение всех этих преобразований позволяет успешно (если все идет хорошо) преобразовать входные значения в желаемые выходные. В действительности одна из гипотез в отношении того, почему глубокое обучение хорошо работает, состоит в том, что сложное сквозное преобразование, отображающее входные данные на выходные — скажем, от некоторого входного изображения до выходной категории "жираф", — декомпонуется множеством слоев в структуру из многих относительно простых преобразований, каждое из которых довольно легко обучается посредством процедуры локального обновления.

В процессе формирования всех этих внутренних преобразований глубокие сети часто обнаруживают значимые промежуточные представления для данных. Например, сеть, обучаемая распознаванию сложных объектов на изображениях, может сформировать внутренние слои, обнаруживающие полезные элементы изображений: края, углы, эллипсы, глаза, лица — даже для кошек. А могут и не создавать — глубокие сети способны формировать внутренние слои, значение которых совершенно непрозрачно для человека, хотя результат все так же остается корректным.

Скрытые слои нейронных сетей обычно менее разнообразны, чем выходные слои. В течение первых 25 лет исследований в области многослойных сетей (примерно 1985–2010 годы) для внутренних узлов в качестве функции активации почти исключительно использовали сигмоиду и функцию tanh. Позднее, примерно с 2010 года, все более популярными становятся функции ReLU и softplus, — отчасти потому, что они, как предполагалось, позволяют избежать возникновения проблемы исчезающих градиентов, упомянутой в разделе 21.1.2. Экспериментирование со все более глубокими сетями проводилось в предположении, что во многих случаях лучшее обучение демонстрируют глубокие и относительно узкие сети, а не мелкие широкие сети, — при заданном общем количестве весов. Типичный пример подобных результатов приведен на рис. 21.7 в разделе 21.5.1.

Конечно, имеется много других структур, которые следовало бы рассмотреть применительно к графам вычислений, помимо простых манипуляций шириной и глубиной. На момент написания этих строк еще не было достигнуто достаточно понимания, почему одни структуры работают лучше, чем другие, при решении какой-то конкретной проблемы. С накоплением опыта практики вырабатывают некоторое интуитивное представление о том, как разрабатывать глубокие сети и как модернизировать их, если они не работают так, как ожидалось, — подобно тому, как повара получают интуитивное представление о том, как создавать новые рецепты и как их корректировать, если вкус их блюд отличается от желаемого. По этой причине для достижения успеха при решении реальных задач разработчикам необходимы инструменты, обеспечивающие быстрое выполнение всесторонних исследований и оценки различных структур.

21.3. Сверточные сети

В разделе 21.2.1 уже упоминалось, что изображение не может рассматриваться как простой вектор входных значений пикселей, — прежде всего потому, что информация о смежности пикселей действительно имеет большое значение. Если принять решение построить сеть с полносвязными слоями и изображениями в качестве входных данных, один и тот же результат будет получен при ее обучении на неискаженных изображениях и при обучении на тех же изображениях, но со случайным образом перемешанными пикселями. Более того, предположим, что имеется n пикселей, а в первом скрытом слое имеется n элементов, для которых эти пиксели являются входными данными. Если входной слой и первый скрытый слой являются полносвязными, то это означает, что необходимо n^2 весов, — для типичного мегапиксельного RGB-изображения это означает 9 трлн весов. Столь обширное пространство параметров потребует соответствующего огромного количества обучающих изображений и колоссального бюджета вычислительных затрат для запуска алгоритма обучения.

Эти соображения приводят к мысли, что первый скрытый слой следует построить таким образом, чтобы ➡ *каждый скрытый элемент получал входные данные только из небольшой локальной области изображения.* Такой подход позволяет одним выстрелом убить двух зайцев. Во-первых, он учитывает сведения о смежности, по крайней мере локально. (Позже будет показано, что если последующие слои будут обладать таким же свойством локальности, то вся сеть будет учитывать смежность пикселей в глобальном смысле.) Во-вторых, он сокращает количество весов: если в каждой локальной области есть $l \ll n$ пикселей, то потребуется всего $ln \ll n^2$ весов.

Пока все хорошо. Но было упущено еще одно важное свойство изображений: грубо говоря, все, что можно обнаружить в одной небольшой локальной области изображения — возможно, глаз или травинка, — выглядело бы так же, если бы оно появилось в другой небольшой, локальной области изображения. Другими словами, ожидается, что данные изображений будут проявлять приблизительную ▶ **пространственную инвариантность** (*spatial invariance*), — по крайней мере в малых и средних масштабах.[3] Однако не следует ожидать, что верхние половины фотографий будут выглядеть, как нижние половины, поэтому всегда существует некоторый масштаб, за пределами которого пространственная инвариантность уже не сохраняется.

Локальная пространственная инвариантность может быть достигнута путем ограничения весов l, соединяющих локальный регион с элементом в скрытом слое, таким образом, чтобы они были одинаковы для каждого скрытого элемента.

[3] Аналогичные идеи могут быть применены к источникам данных временных рядов, таким как звуковые сигналы. Последние обычно проявляют **временную неизменность** — слово звучит одинаково независимо от того, в какое время дня оно произнесено. Рекуррентные нейронные сети (раздел 21.6) автоматически проявляют временную инвариантность.

(То есть для скрытых элементов i и j, веса $w_{1,i},\ldots, w_{l,i}$ будут такими же, как $w_{1,j},\ldots, w_{l,j}$.) Это превращает скрытые элементы в детекторы признаков, обнаруживающие одну и ту же функцию, где бы она ни появлялась на изображении. Как правило, требуется, чтобы первый скрытый слой обнаруживал многие виды признаков, а не только один; поэтому для каждой локальной области изображения можно иметь d скрытых элементов с d различными наборами весов. А это означает, что всего потребуется dl весов — количество, которое не только намного меньше, чем n^2, но в действительности еще и не зависит от n, т.е. от размера изображения. Таким образом, вводя некоторые предварительные знания — а именно, знание о смежности и пространственной инвариантности, — можно разработать модели, имеющие гораздо меньше параметров и способные обучаться гораздо быстрее.

► **Сверточной нейронной сетью** (*convolutional neural network* — ► **CNN**) называется сеть, содержащая пространственно локальные соединения, по крайней мере в первых слоях, и имеющая структуры весов, повторяющиеся для всех элементов в каждом слое. Структура весов, повторяющаяся в нескольких локальных регионах, называется ► **ядром**, а процесс применения ядра к пикселям изображения (или пространственно организованным элементам в последующем слое) называется ► **сверткой**.[4]

Понятие ядра и свертки проще всего проиллюстрировать в одном измерении, а не в двух или больше, поэтому далее рассмотрим входной вектор $\mathbf{x}$ размером n, соответствующий n пикселям в одномерном изображении, и векторное ядро $\mathbf{k}$ размером l. (Для простоты будем считать, что l является нечетном числом.) Все изложенные ниже идеи можно непосредственно перенести на многомерные случаи.

На письме операцию свертки будем представлять с помощью символа $*$, например $\mathbf{z} = \mathbf{x} * \mathbf{k}$. Эта операция определяется следующим образом:

$$z_i = \sum_{j=1}^{l} k_j x_{j+i-(l+1)/2}. \tag{21.8}$$

Другими словами, для каждой выходной позиции i берется точечное произведение ядра $\mathbf{k}$ и фрагмента $\mathbf{x}$ с центром в x_i с шириной l.

Процесс свертки проиллюстрирован на рис. 21.4 для вектора ядра $[+1, -1, +1]$, позволяющего обнаружить более темную точку на одномерном (1D) изображении размером 7 пикселей (нижний ряд квадратиков). (Аналогично 2D-версия позволяет обнаружить более темную линию.) Обратите внимание, что в этом примере пиксели, на которых центрируются ядра, разделены расстоянием в 2 пикселя, — говорят, что ядро применено с ► **шагом** (*stride*) $s = 2$. Также обратите внимание, что выходной слой имеет меньше пикселей: из-за применения шага количество пикселей уменьшилось с n до примерно n/s. (В двух измерениях количество пикселей

[4] В терминологии области обработки сигналов эту операцию назвали бы взаимной корреляцией, а не сверткой. Однако в области нейронных сетей для нее используется термин "свертка".

уменьшится примерно до $n/s_x s_y$, где s_x и s_y являются размером шага в направлении осей x и y на изображении.) "Примерно" сказано из-за того, что происходит на краях изображения: на рис. 21.4 свертка останавливается на краях изображения, но можно также пополнить ввод дополнительными пикселями (нулями или копиями внешних пикселей), чтобы ядро можно было применить ровно $\lfloor n/s \rfloor$ раз. Для небольших ядер обычно используется $s = 1$, поэтому вывод имеет те же размеры, что и изображение (пример приведен ниже, на рис. 21.5).

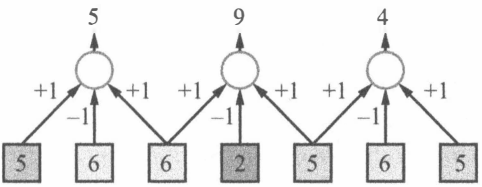


Рис. 21.4. Пример одномерной операции свертки с размером ядра $l = 3$ и шагом $s = 2$. Пиковый отклик сосредоточен на более темном (с меньшей интенсивностью) входном пикселе. Результаты обычно пропускаются через нелинейную функцию активации (не показана), прежде чем будут переданы следующему скрытому слою

Операция применения ядра к изображению очевидным образом может быть реализована с помощью программы с подходящими вложенными циклами. Но ее также можно сформулировать как единственную матричную операцию, — аналогично применению матрицы весов в уравнении (21.1). Например, процедуру свертки, приведенную на рис. 21.4, можно рассматривать как приведенную здесь операцию умножения матриц.

$$\begin{pmatrix} +1 & -1 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & -1 & +1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 2 \\ 5 \\ 6 \\ 5 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \\ 4 \end{pmatrix} \tag{21.9}$$

В этой матрице весов ядро появляется в каждой строке, смещенное в соответствии с шагом относительно предыдущей строки. Однако вовсе необязательно строить матрицу весов явно — в конечном счете это в основном нули, — но сам тот факт, что свертка является линейной матричной операцией, служит напоминанием о том, что градиентный спуск можно легко и эффективно применять к CNN, как и к прочим обычным нейронным сетям.

Как упоминалось ранее, может существовать d ядер, а не только одно; поэтому, при шаге 1 выходов будет в d раз больше. А это означает, что двухмерный входной массив превращается в трехмерный массив скрытых элементов, где третье измерение имеет размер d. Важно организовать скрытый слой таким образом, чтобы выходы от всех ядер для определенного местоположения в изображении оставались связанными с этим местоположением. Однако следует отметить, что в отличие от пространственных измерений изображения, это дополнительное "ядерное измерение" *не имеет* каких-либо свойств смежности, поэтому не имеет смысла применять к нему операцию свертки.

Исходно идея построения сверточных нейронных сетей была подсказана моделями зрительной коры, предложенными в нейробиологии. В этих моделях ▶**рецептивное поле** нейрона является той частью общего поля входных сигналов от сенсоров ввода, которая может повлиять на активацию этого нейрона. В CNN рецептивное поле элементов в первом скрытом слое маленькое — это просто размер ядра, т.е. l пикселей, однако в более глубоких слоях сети оно может быть намного больше. На рис. 21.5 это положение проиллюстрировано для элемента второго скрытого слоя, рецептивное поле которого содержит пять пикселей. Когда шаг равен 1, как на рисунке, узел m скрытого слоя будет иметь рецептивное поле размером $(l-1)m+1$, так что увеличение размера рецептивного поля происходит линейно относительно m. (В двухмерном изображении с увеличением m линейно возрастает каждая размерность рецептивного поля, поэтому размер всего поля увеличивается квадратично.) Когда шаг больше 1, каждый пиксель слоя m представляет s пикселей в слое $m-1$; следовательно, рецептивное поле растет с глубиной как $O(ls^m)$, т.е. экспоненциально. Тот же эффект возникает и в объединяющих слоях, которые обсуждаются в следующем разделе.

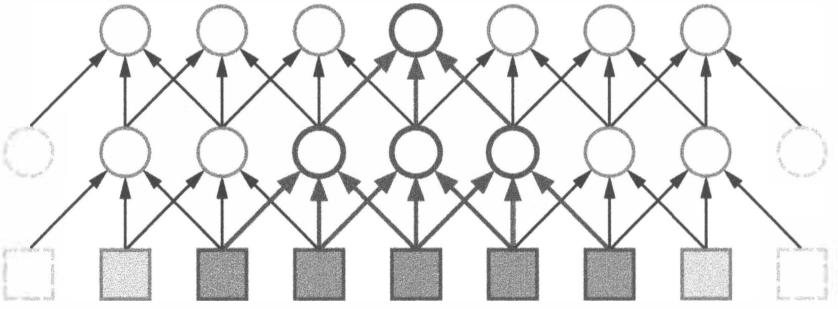


Рис. 21.5. Первые два слоя сверточной нейронной сети для одномерного изображения с размером ядра $l=3$ и шагом $s=1$. Чтобы скрытые слои имели тот же размер, что и входной слой, слева и справа добавляются заполняющие пиксели. Темной заливкой выделено рецептивное поле элемента во втором скрытом слое. В общем случае, чем глубже расположен элемент, тем больше будет его рецептивное поле

21.3.1. Объединение и субдискретизация

В нейронной сети ▶**объединяющий слой** (*pooling layer*) обобщает некоторое множество смежных элементов из предыдущего слоя, заменяя его единственным значением. Объединяющий слой работает как сверточный слой с размером ядра l и шагом s, но применяемая операция является фиксированной, а не обучаемой. Как правило, с объединяющим слоем не связывается какая-либо функция активации. Существуют две распространенные формы ▶**объединения**, или ▶**пулинга** (*pooling*).

- При объединении по среднему (*average-pooling*) вычисляется среднее значение для l входов. Эта операция идентична свертке с единообразным вектором ядра $\mathbf{k} = [1/l, \ldots, 1/l]$. Если задать $l = s$, эффектом будет снижение разрешение изображения — его ▶**субдискретизация** (*downsample*), уплотнение, т.е. уменьшение изображения с коэффициентом s. В результате объект, который в исходном изображении занимал, скажем, $10s$ пикселей, после уплотнения будет занимать только 10 пикселей. Тот же самый обученный классификатор, который способен распознавать в исходном изображении объект размером 10 пикселей, теперь сможет распознать подобный объект в уплотненном изображении, даже если он был слишком большим, чтобы распознать его в оригинальном изображении. Другими словами, объединение по среднему упрощает многоуровневое распознавание. Оно также позволяет уменьшить количество весов, необходимых в последующих слоях, что ведет к снижению вычислительных затрат и, возможно, к ускорению обучения.
- При объединении по максимуму из l входных значений выбирается максимальное значение. Этот метод также может использоваться просто для субдискретизации, но у него несколько иная семантика. Предположим, объединение по максимуму было применено к скрытому слою $[5, 9, 4]$, приведенному на рис. 21.4, — результатом будет 9, и это указывает, что где-то во входном изображении есть более темная точка, обнаруженная ядром. Другими словами, объединение по максимуму действует как своего рода логическое разделение, сообщающее, что где-то в рецептивном поле элемента присутствует некая особенность.

Если цель состоит в классификации изображения в одну из c категорий, то последним слоем сети будет слой softmax с c выходными элементами. Размер первых слоев CNN соответствует размеру изображения, поэтому где-то в средней части сети размеры слоев должны существенно уменьшаться. Для уменьшения размера слоя можно использовать сверточные или объединяющие слои с шагом больше 1. Уменьшить размер слоя также возможно за счет полносвязного слоя с меньшим количеством элементов, чем в предыдущем слое. В сверточные нейронные сети часто включают один или два таких слоя, предшествующих конечному слою softmax.

21.3.2. Тензорные операции в сверточных нейронных сетях

В уравнениях (21.1) и (21.3) было показано, что использование векторной и матричной нотаций может оказаться полезным с точки зрения упрощения и придания изящности математическим выкладкам, а также получения кратких описаний графов вычислений. Векторы и матрицы являются соответственно одномерным и двухмерным специальными случаями ▶**тензоров**, которые (в терминологии глубокого обучения) представляют собой просто многомерные массивы любой размерности.[5]

Для сверточных нейронных сетей тензоры являются способом отслеживания "формы" данных по мере их продвижения через слои сети. Это важно, потому что само понятие свертки зависит от идеи смежности: смежные элементы данных предполагаются семантически связанными, поэтому есть смысл применять тензорные операторы к локальным областям данных. Более того, при наличии подходящих языковых примитивов для построения тензоров и применения операторов, слои сами по себе могу быть кратко описаны как отображение тензорных входов на тензорные выходы.

Последней причиной описания сверточных нейронных сетей в терминах тензорных операций является вычислительная эффективность: при наличии описания сети как последовательности тензорных операций любой программный пакет глубокого обучения сможет сгенерировать откомпилированный код, который будет в высшей степени оптимизирован для базовой вычислительной основы. В глубоком обучении параллельные вычислительные потоки часто выполняются на графических процессорах (*graphic sprocessing units* — GPU) или тензорных процессорах (*tensor processing units* — TPU), обеспечивающих высокую степень параллелизма. Например, один блок TPU третьего поколения от Google обеспечивает пропускную способность, эквивалентную примерно десяти миллионам ноутбуков. Эффективное использование подобных возможностей — важное условие при обучении большой сверточной нейронной сети на большой базе изображений. В результате появляется возможность обрабатывать изображения не по одному, а сразу по многу параллельно. Как будет показано в разделе 21.4, этот подход также хорошо сочетается с тем способом, которым алгоритм стохастического градиентного спуска вычисляет градиенты по отношению к мини-пакетам обучающих примеров.

Давайте соединим все сказанное выше в виде примера. Предположим, что обучение проводится на RGB-изображениях размером 256×256 пикселей мини-пакетами по 64 штуки. Входом в этом случае будет четырехмерный тензор размером $256 \times 256 \times 3 \times 64$. Затем в сети к изображениям применяется 96 ядер размером $5 \times 5 \times 3$ с шагом 2 в обоих направлениях, x и y. Это дает выходной тензор размером $128 \times 128 \times 96 \times 64$. Такой тензор часто называют ▶**картой признаков** (*feature map*), поскольку он показывает, как каждый извлеченный ядром признак

[5] Правильное математическое определение тензоров требует, чтобы определенные инварианты сохранялись при изменении базиса.

появляется по всему изображению. В данном случае он состоит из 96 ▶**каналов**, и каждый канал несет информацию одного из признаков. Обратите внимание, что, в отличие от входного тензора, карта признаков уже не имеет выделенных цветовых каналов; тем не менее информация о цвете все еще может присутствовать в различных каналах признаков, если алгоритм обучения обнаружит цвет, который будет полезен для выбора прогнозов на выходе сети.

21.3.3. Остаточные сети

Метод ▶**остаточных сетей** (*residual networks*) — это весьма популярный и успешный подход к построению очень глубоких сетей, позволяющий избежать проблемы исчезающих градиентов.

Типичные модели глубокого обучения используют слои, которые изучают новое представление в слое i, полностью замещая им представление в слое $i-1$. Используя матрично-векторную нотацию, введенную в уравнении (21.3), где $\mathbf{z}^{(i)}$ представляет значения элементов в слое i, получим

$$\mathbf{z}^{(i)} = f(\mathbf{z}^{(i-1)}) = \mathbf{g}^{(i)}(\mathbf{W}^{(i)}\mathbf{z}^{(i-1)}).$$

Поскольку каждый слой полностью заменяет представление предыдущего слоя, все слои в сети должны учиться делать что-то полезное. Каждый слой должен по крайней мере сохранять релевантную для задания информацию, содержащуюся в предыдущем слое. Если для любого слоя i установить $\mathbf{W}^{(i)} = 0$, вся сеть перестает функционировать. Если дополнительно установить $\mathbf{W}^{(i-1)} = 0$, то сеть будет уже не в состоянии даже учиться: слой i не будет обучаться, поскольку не будет наблюдать никаких изменений в его входах из слоя $i-1$, а слой $i-1$ не сможет обучаться, поскольку градиент обратного распространения из слоя i всегда будет нулевым. Конечно, эти примеры — крайний случай, но они хорошо иллюстрируют необходимость того, чтобы слои служили каналами для сигналов, проходящих через сеть.

Основная идея остаточных сетей состоит в том, что слой должен *нарушать* представление, полученное из предыдущего слоя, а не *заменять* его полностью. Если выученное возмущение невелико, следующий слой будет близок к тому, чтобы оказаться копией предыдущего слоя. Этот эффект достигается с помощью следующего уравнения для слоя i с точки зрения слоя $i-1$:

$$\mathbf{z}^{(i)} = \mathbf{g}_r^{(i)}(\mathbf{z}^{(i-1)} + f(\mathbf{z}^{(i-1)})), \tag{21.10}$$

где вектор $\mathbf{g}_r$ представляет функции активации для остаточного слоя. Здесь мы понимаем f как ▶**остаточный блок** (*residual*), нарушающий принятый по умолчанию ход передачи информации от слоя $i-1$ через слой i. Функция, используемая для вычислений в остаточном блоке, обычно представляет собой нейронную сеть из одного нелинейного слоя, скомбинированного с одним линейным слоем:

$$f(\mathbf{z}) = \mathbf{V}\mathbf{g}(\mathbf{W}\mathbf{z}),$$

где $\mathbf{W}$ и $\mathbf{V}$ — матрицы изученных весов с добавленными обычными весами смещения.

Остаточные сети позволяют надежно обучать существенно более глубокие сети. Посмотрим, что произойдет, если для определенного слоя установить $\mathbf{V} = \mathbf{0}$ с целью отключить этот слой. Тогда остаточный блок f исчезнет и уравнение (21.10) упростится до

$$\mathbf{z}^{(i)} = \mathbf{g}_r(\mathbf{z}^{(i-1)}).$$

Теперь предположим, что $\mathbf{g}_r$ состоит из функций активации ReLU и что в $\mathbf{z}^{(i-1)}$ также применяется функция ReLU к ее входам: $\mathbf{z}^{(i-1)} = \text{ReLU}(\mathbf{in}^{(i-1)})$. В этом случае имеем

$$\mathbf{z}^{(i)} = \mathbf{g}_r(\mathbf{z}^{(i-1)}) = \text{ReLU}(\mathbf{z}^{(i-1)}) = \text{ReLU}(\text{ReLU}(\mathbf{in}^{(i-1)})) = \text{ReLU}(\mathbf{in}^{(i-1)}) = \mathbf{z}^{(i-1)},$$

где предпоследний этап следует из того, что $\text{ReLU}(\text{ReLU}(x)) = \text{ReLU}(x)$. Другими словами, в остаточных сетях с функцией активации ReLU слой с нулевым весом просто пропускает свои входные данные без изменений. Оставшаяся часть сети функционирует так, словно этот слой никогда не существовал. В то время как традиционные сети должны *научиться* распространять информацию и подвержены катастрофическим отказам в ее распространении при неправильном выборе параметров, остаточные сети обеспечивают распространение информации по умолчанию.

Остаточные сети часто используются со сверточными слоями в визуальных приложениях, но на самом деле они являются универсальным инструментом, делающим глубокие сети более надежными и позволяющим исследователям более свободно экспериментировать со сложными и разнородными сетевыми конструкциями. На момент написания этой главы достаточно часто встречались остаточные сети с сотнями слоев. Конструкции таких сетей быстро развиваются, поэтому любые дополнительные особенности, которые мы могли бы здесь предоставить, вероятно, устареют еще до того, как эта книга будет напечатана. Читателям, желающим больше узнать об архитектуре сетей для конкретных приложений, следует обратиться к последним публикациям о результатах соответствующих исследований.

21.4. Алгоритмы обучения

Обучение нейронной сети состоит в изменении ее параметров с целью минимизации функции потерь на обучающем наборе данных. В принципе, использоваться может любой тип алгоритма оптимизации. На практике современные нейронные сети почти всегда обучаются с использованием того или иного варианта алгоритма стохастического градиентного спуска (SGD).

Стандартный алгоритм градиентного спуска и его стохастическая версия рассматривались в разделе 19.6.2. В данном случае цель состоит в минимизации потерь $L(\mathbf{w})$, где $\mathbf{w}$ представляет все параметры сети. Каждый этап обновления в процессе градиентного спуска выглядит так:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}),$$

где α — скорость обучения. Для алгоритма стандартного градиентного спуска потери L определяются в отношении всего обучающего набора. Для алгоритма SGD они определяются в отношении мини-пакета из m примеров, случайным образом выбранных на каждом этапе.

Как было отмечено в разделе 4.2, литература по методам оптимизации для многомерных непрерывных пространств содержит описание бесчисленного множества улучшений базового алгоритма градиентного спуска. Здесь не предполагается охватить их все, но будет полезно упомянуть несколько важных соображений, особенно актуальных для обучения нейронных сетей.

- Для большинства сетей, решающих реальные проблемы, как размерность $\mathbf{w}$, так и размер обучающего набора обычно очень велики. Эти обстоятельства решительно склоняют нас к использованию алгоритма SGD с относительно небольшим размером мини-пакета m: стохастичность помогает алгоритму избежать малых локальных минимумов в многомерном пространстве весов (как в алгоритме имитации отжига; см. раздел 4.1.2), а небольшой размер мини-пакета гарантирует, что вычислительная стоимость каждого этапа обновления весов будет небольшой константой, не зависящей от размера обучающего набора.
- Поскольку вклад градиента каждого обучающего примера в мини-пакете алгоритма SGD может рассчитываться независимо, размер мини-пакета часто выбирается так, чтобы максимально использовать преимущества возможного аппаратного распараллеливания вычислений в графических процессорах или TPU.
- Для улучшения сходимости обычно рекомендуется использовать скорость обучения, значение которой уменьшается со временем. Выбор правильного графика такого изменения обычно осуществляется методом проб и ошибок.
- Вблизи локального или глобального минимума функции потерь по отношению ко всему обучающему набору градиенты, оцененные на небольших мини-пакетах, часто будут иметь высокую дисперсию и могут указывать в совершенно неверном направлении, затрудняя тем самым схождение. Одно из возможных решений состоит в увеличении размера мини-пакета в процессе обучения; другое — в подключении идеи ▶**импульса**, сохраняющего скользящее среднее значение градиентов прошлых мини-пакетов с целью компенсации отрицательного эффекта, оказываемого небольшими размерами мини-пакетов.

- Необходимо соблюдать осторожность, чтобы уменьшить числовую нестабильность, которая может возникнуть из-за ошибок переполнения, потери значимости или округления. Этот момент вызывает особенно много проблем при использовании в качестве функций активации функций с экспоненциальным поведением softmax, tanh и сигмоиды, а также при рекуррентных вычислениях в очень глубоких сетях и рекуррентных сетях (раздел 21.6), что часто приводит к исчезающим или взрывным активациям и градиентам.

В целом процесс обучения весов в сети, как правило, демонстрирует убывающую отдачу. Обучение ведется до тех пор, пока уменьшение ошибки на тестовом наборе за счет продолжения обучения не потеряет смысл. Обычно это не означает, что был достигнут глобальный или даже локальный минимум функции потерь. На самом деле это означает, что для продолжения снижения стоимости потребуется выполнить непрактично большое количество очень маленьких этапов или что дополнительные этапы приведут лишь к переобучению, или что оценки градиента слишком неточны, чтобы можно было добиться дальнейшего прогресса.

21.4.1. Вычисление градиентов в графах вычислений

В разделе 21.1.2 был выведен градиент функции потерь по отношению к весам в конкретной (и очень простой) сети. Было отмечено, что градиент может быть рассчитан обратным распространением информации об ошибках от выходного слоя сети к ее скрытым слоям. Также было сказано, что в общем случае этот результат справедлив для любого графа вычислений с прямой связью. Ниже объясняется, почему это так.

На рис. 21.6 показан типичный узел в графе вычислений. (Узел h имеет входную и выходную степень 2, но в приведенном ниже анализе от этого ничего не зависит.) В процессе прямого прохода в узле вычисляется некоторая произвольная функция h на основании его входных данных, поступающих из узлов f и g. В свою очередь, узел h передает вычисленное в нем значение узлам j и k.

Процесс обратного распространения передает сообщения в обратном направлении по каждой связи в сети. В каждом узле входящие сообщения собираются и рассчитываются новые сообщения для передачи слою, следующему в обратном направлении. Как показано на рисунке, все сообщения являются частными производными функции потерь L. Например, обратное сообщение $\partial L/\partial h_j$ является частной производной функции L по отношению к первому входу узла j, являющегося прямым сообщением от узла h к узлу j. Далее узел h воздействует на потери L через оба узла, j и k, так что имеем

$$\partial L/\partial h = \partial L/\partial h_j + \partial L/\partial h_k. \tag{21.11}$$

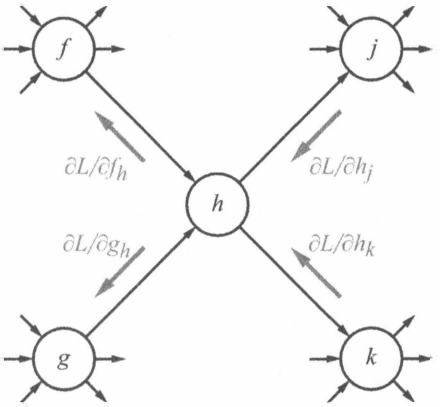


Рис. 21.6. Иллюстрация обратного распространения информации о градиенте в произвольном графе вычислений. Вычисление выхода сети в прямом направлении происходит слева направо, а обратное распространение градиентов выполняется справа налево

С помощью этого уравнения узел h может вычислить производную от функции потерь L по h, суммируя входящие сообщения от узлов j и k. Теперь для вычисления исходящих сообщений $\partial L/\partial f_h$ и $\partial L/\partial g_h$ мы используем следующие уравнения:

$$\frac{\partial L}{\partial f_h} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial f_h} \quad \text{и} \quad \frac{\partial L}{\partial g_h} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial g_h}. \tag{21.12}$$

В уравнении (21.12) $\partial L/\partial h$ уже вычислено по уравнению (21.11), а $\partial h/\partial f_h$ и $\partial h/\partial_{gh}$ — это просто производные от h по первым и вторым аргументам соответственно. Например, если h является узлом умножения, т.е. $h(f, g) = f \cdot g$, то $\partial h/\partial f_h = g$ и $\partial h/\partial g_h = f$. Программные пакеты для глубокого обучения обычно поставляются с библиотекой узлов разных типов (сложение, умножение, сигмоида и т.д.), в каждом из которых обеспечивается вычисление его собственных производных, необходимых для уравнения (21.12).

Процесс обратного распространения начинается с выходных узлов, в которых каждое начальное сообщение $\partial L/\partial \hat{y}_j$ вычисляется непосредственно из выражения для функции потерь L через предсказанное значение $\hat{\mathbf{y}}$ и истинное значение $\mathbf{y}$ из данных обучения. На каждом внутреннем узле входящие обратные сообщения суммируются в соответствии с уравнением (21.11), а исходящие сообщения генерируются по уравнению (21.12). Процесс заканчивается на каждом узле в графе вычислений, который представляет вес w (например, светло-серые овалы на рис. 21.3, *б*). На этом этапе сумма входящих сообщений к узлу w равна $\partial L/\partial w$ — именно тому градиенту, который нужен для обновления значения w.

Распределение веса, используемое в сверточных сетях (раздел 21.3) и рекуррентных сетях (раздел 21.6), обрабатывается просто посредством обработки каждого общего веса как одного узла с несколькими исходящими ребрами в графе вычислений. В процессе обратного распространения это приводит к появлению нескольких входящих сообщений градиента. Согласно уравнению (21.11) данный факт означает, что градиент для общего веса представляет собой сумму значений градиентов от каждого места, где этот вес используется в сети.

Из приведенного выше описания процесса обратного распространения следует, что его вычислительная стоимость линейно зависит от количества узлов в графе вычислений, так же как и стоимость расчетов в прямом направлении. Более того, когда разработка сети будет завершена, типы ее узлов будут точно определены и зафиксированы, а это означает, что для каждого узла в графе вычислений все вычисления градиента могут быть заранее подготовлены в символической форме, а затем скомпилированы в весьма эффективный код. Отметим также, что сообщения на рис. 21.6 необязательно должны быть скалярными — с равным успехом они могут представлять собой векторы, матрицы или тензоры высокой размерности и в таких случаях вычисления градиента могут быть эффективно отображены на GPU или TPU, обеспечивая дополнительный выигрыш от распараллеливания вычислений.

Единственный недостаток метода обратного распространения состоит в том, что вычисления градиентов при обратном проходе требуют сохранения большей части промежуточных значений, вычисленных в процессе прямого распространения. А это означает, что общая стоимость памяти, требуемой для обучения сети, пропорциональна количеству элементов во всей сети. В результате, даже если сама сеть представлена неявно в виде кода распространения с большим количеством циклов, а не явно в виде соответствующей структуры данных, все промежуточные результаты этого кода распространения должны храниться в явном виде.

21.4.2. Пакетная нормализация

Метод ▶**пакетной нормализации** — это широко используемый способ повышения скорости сходимости алгоритма стохастического градиентного спуска (SGD) путем пересчета значений, генерируемых на внутренних слоях сети для примеров в каждом мини-пакете. Хотя причины эффективности этого метода на момент написания этой главы были еще не совсем понятны, он упоминается здесь, поскольку обеспечивает значительные преимущества на практике. До некоторой степени пакетная нормализация, по-видимому, оказывает эффект, подобный результатам применения остаточной сети.

Рассмотрим узел z, расположенный где-то в сети: значениями z для m примеров в мини-пакете будут $z_1,\ldots,z_m$. Пакетная нормализация заменяет каждое значение z_i новым значением $\hat{z}_i$:

$$\hat{z}_i = \gamma \frac{z_i - \mu}{\sqrt{\epsilon + \sigma^2}} + \beta,$$

где μ — среднее значение z в мини-пакете, σ — стандартное отклонение $z_1, \ldots, z_m$, константа ϵ — небольшая постоянная, добавленная для предотвращения деления на нуль, а γ и β — обучаемые параметры.

Пакетная нормализация стандартизирует среднее значение и дисперсию значений в той степени, которая определяется значениями параметров β и γ. Это значительно упрощает обучение глубокой сети. Без пакетной нормализация информация может быть потеряна, если вес слоя слишком мал, а стандартное отклонение в нем затухает почти до нуля. Пакетная нормализация предотвращает это явление. Она также уменьшает необходимость тщательной инициализации всех весов в сети с целью получения гарантии, что узлы в каждом слое находятся в правильной рабочей области, чтобы разрешить распространение информации.

Использование пакетной нормализации обычно требует включения в число параметров сети дополнительных параметров β и γ, которые могут быть специфическими для узла или для слоя и, следовательно, должны быть включены в процесс обучения. После обучения значения параметров β и γ фиксируются на тех значениях, которые были достигнуты в результате обучения.

21.5. Обобщение

До сих пор речь шла о том, как согласовать нейронную сеть с обучающим набором, выбранным для ее обучения, однако основная цель машинного обучения состоит в обобщении новых данных, которые ранее не рассматривались, что характеризуется производительностью сети на тестовом наборе. В этом разделе рассматриваются три подхода к повышению производительности при обобщении: выбор правильной сетевой архитектуры, штрафование больших весов и случайное возмущение значений, проходящих через сеть в процессе обучения.

21.5.1. Выбор сетевой архитектуры

Множество усилий в исследованиях глубокого обучения было направлено на поиск сетевых архитектур, обеспечивающих хорошее обобщение. В действительности для каждого конкретного типа данных — изображения, речь, текст, видео и т.д. — значительный прогресс в повышении производительности был достигнут благодаря изучению различных видов сетевых архитектур и варьированию количества слоев, их связности и типов узлов в каждом слое.[6]

[6] Отметим, что большая часть связанной с этим дополнительной исследовательской работы выполняется аспирантами, некоторые из которых называют это процессом "спуска аспирантуры" — **graduate student descent** (GSD).

Некоторые архитектуры нейронных сетей специально разрабатывались с целью хорошего обобщения определенных типов данных: в сверточных сетях закодирована идея о том, что один и тот же шаблон извлечения признака будет полезен во всех местоположениях в пространственной сетке, тогда как в рекуррентных сетях закодирована идея, что одно и то же правило обновления будет полезно во всех точках потока последовательных данных. В той степени, в какой эти предположения полагаются верными, можно ожидать, что сверточные архитектуры будут хорошо обобщать изображения, а рекуррентные сети способны хорошо обобщать тексты и аудиосигналы.

Одним из наиболее важных эмпирических выводов в области глубокого обучения является то, что при сравнении двух сетей с одинаковым количеством весов более глубокая сеть обычно демонстрирует лучшую производительность обобщения. На рис. 21.7 этот эффект представлен как минимум для одного приложения из реального мира — системы распознавания номеров домов. Полученные результаты показывают, что для любого фиксированного числа параметров сеть с одиннадцатью слоями демонстрирует гораздо меньшую погрешность на тестовом наборе данных, чем трехслойная сеть.

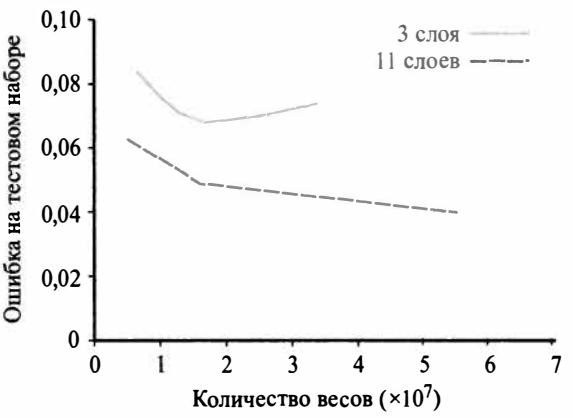


Рис. 21.7. Ошибка на тестовом наборе данных как функция ширины слоя (измеренная по общему числу весов) для сверточных сетей с тремя и одиннадцатью слоями. Данные получены от ранней версии системы Google для получения транскрипции адресов на фотографиях, сделанных автомобилями службы Street View

Системы глубокого обучения хорошо справляются с некоторыми, но не со всеми задачами. Для задач с входными данными высокой размерности — изображения, видео, речевые сигналы и т.д. — они работают лучше, чем любые другие подходы чистого машинного обучения. Большинство алгоритмов, описанных в главе 19, позволяют обрабатывать входные данные с высокой размерностью

только в том случае, если они предварительно обрабатываются с целью уменьшения размерности с использованием разработанных вручную функций. Такой подход с предварительной обработкой, преобладавший до 2010 года, не обеспечивает производительности, сопоставимой с той, которая достигается системами глубокого обучения.

Очевидно, что модели глубокого обучения захватывают некоторые важные аспекты таких задач. В частности, их успех подразумевает, что подобные задачи могут быть решены параллельными программами с относительно небольшим количеством этапов (от 10 до 10^3, а не, скажем, 10^7). И это, возможно, не удивительно, поскольку подобные задачи решаются мозгом, как правило, менее чем за секунду, чего достаточно лишь для нескольких десятков последовательных включений нейронов. Кроме того, изучая представления внутренних слоев обученных глубоких сверточных сетей для задач компьютерного зрения, можно найти доказательства того, что этапы обработки, по-видимому, включают в себя извлечение последовательности все более абстрактных представлений сцены, начиная с крошечных фрагментов границ, точек и угловых элементов и заканчивая объектами в целом и взаимным расположением нескольких объектов.

С другой стороны, моделям глубокого обучения, представляющим собой простые сетевые схемы, недостает позиционной и квантифицирующей выразительной силы, которая свойственна логике первого порядка (глава 8) и контекстно-свободным грамматикам (глава 23).

Хотя модели глубокого обучения во многих случаях демонстрируют хорошее обобщение, они также могут выдавать интуитивно непонятные ошибки. Им свойственна тенденция так выполнять отображение ввода на вывод, что его нельзя воспринимать как непрерывное: очень небольшое изменение на входе способно вызвать большие изменения на выходе. Например, может оказаться достаточным изменить всего несколько пикселей в изображении собаки — и сеть, прежде выдававшая правильную классификацию, вдруг сочтет его изображением страуса или школьного автобуса, хотя измененное изображение все еще будет выглядеть, как собака. Измененное изображение такого рода называется ▶**состязательным примером** (*adversarial example*).

В низкоразмерных пространствах трудно найти состязательные примеры. Но для изображения с миллионами значений, представляющих пиксели, часто бывает так, что, хотя вклад большинства пикселей определяет размещение изображения в середине области пространства признаков, соответствующей категории “собака”, имеется несколько размерностей, в которых значение пикселя находится вблизи границы с другой категорией. Противник, обладающий способностью реконструировать сеть, может найти наименьшую разность векторов, смещающую изображение за эту границу.

Когда состязательные примеры были впервые обнаружены, вспыхнула ожесточенная схватка мирового масштаба между двумя направлениями исследований: одни стремились найти алгоритмы обучения и сетевые архитектуры,

невосприимчивые к атакам типа состязательных примеров, а другие прилагали максимум усилий для поиска все более эффективных методов таких атак, направленных против всех видов систем обучения. Пока что нападающие, похоже, впереди. В действительности, хотя изначально предполагалось, что необходимо иметь доступ к внутренним компонентам обученной сети, чтобы создать состязательный пример специально для нее, на практике было установлено, что можно построить *надежные* состязательные примеры, способные обмануть множество сетей с различной архитектурой, гиперпараметрами и обучающими наборами. Эти результаты показывают, что модели глубокого обучения распознают объекты способами, совершенно отличными от зрительной системы человека.

21.5.2. Поиск нейронной архитектуры

К сожалению, пока не существует четкого набора рекомендаций в отношении выбора лучшей архитектуры сети для конкретной задачи. Достижение успеха в развертывании системы глубокого обучения требует опыта и здравого смысла.

С самых первых дней исследований в области нейронных сетей были предприняты попытки автоматизировать процесс выбора архитектуры. Можно рассматривать эту задачу как процедуру настройки гиперпараметров (раздел 19.4.4), определяющих глубину, ширину, связность и другие атрибуты сети. Однако здесь есть так много вариантов, из которых можно выбирать, что такие простые подходы, как сеточный поиск, не позволяют охватить все доступные возможности в разумные сроки.

Поэтому для исследования пространства состояний возможных сетевых архитектур обычно используется подход, называемый ▶**поиском нейронной архитектуры** (*neural architecture search*). Для решения задачи поиска нейронной архитектуры применялись многие из методов поиска и обучения, рассмотренные ранее в этой книге.

Эволюционные алгоритмы были популярны потому, что разумно одновременно выполнять и рекомбинацию (объединение частей двух сетей), и мутацию (добавление или удаление слоя или изменение значения параметра). Совместно с этими мутационными операциями также мог использоваться поиск восхождением к вершине. Одни исследователи трактовали эту задачу как обучение с подкреплением, а другие — как байесовскую оптимизацию. Еще один возможный подход состоит в том, чтобы рассматривать возможные архитектурные решения как непрерывное дифференцируемое пространство и использовать метод градиентного спуска для нахождения локально оптимального решения.

Для всех этих методов поиска основной проблемой является оценка ценности сети-кандидата. Простой способ оценить архитектуру — обучить ее на тестовом наборе из нескольких пакетов, а затем оценить ее точность на проверочном наборе. Но в случае больших сетей это может потребовать многих дней расчетов на GPU.

Поэтому было предпринято много попыток ускорить этот процесс оценки путем исключения или по крайней мере сокращения дорогостоящего процесса обучения. Можно проводить обучение на меньших наборах данных. Можно провести обучение на небольшом количестве пакетов, а затем сделать прогноз, как сеть будет улучшать свои показатели при дальнейшем увеличении количества пакетов. Можно использовать сокращенную версию сетевой архитектуры, которая, как предполагается, сохраняет свойства полной версии. Можно обучить одну большую сеть, а затем искать подграфы этой сети, которые работают лучше, — этот поиск может быть быстрым, потому что подграфы разделяют параметры и их не нужно переучивать.

Другой подход заключается в изучении эвристической функции оценки (как это было сделано для поиска A*). В этом случае следует начать с выбора нескольких сотен сетевых архитектур, обучить и оценить их. В результате будет получен набор данных из пар (*сеть*, *оценка*). Затем изучается отображение признаков сети на прогнозируемую оценку. С этого момента появляется возможность генерировать большое количество сетей-кандидатов и быстро оценивать их ценность. После поиска в пространстве сетей лучшие из кандидатов могут быть полностью оценены при полной процедуре обучения.

21.5.3. Сокращение веса

В разделе 19.4.3 было показано, что **регуляризация**, ограничивающая сложность модели, может способствовать улучшению обобщения. Это правило справедливо и для моделей глубокого обучения. В контексте нейронных сетей такой подход обычно называют ▶ **сокращением** веса (*weight decay*).

Сокращение веса заключается в добавлении штрафа $\lambda \Sigma_{ij} W_{i,j}^2$ к функции потерь, используемой при обучении нейронной сети, где λ — гиперпараметр, определяющий величину штрафа, а сумма обычно вычисляется по всем весам в сети. Использование значения $\lambda = 0$ эквивалентно отказу от применения метода сокращения веса, тогда как выбор больших значений гиперпараметра λ способствует быстрому уменьшению весов. Обычно метод сокращения веса используется при значении λ, близком к 10^{-4}.

Выбор конкретной сетевой архитектуры может рассматриваться как абсолютное ограничение пространства гипотез: функция либо представима в этой архитектуре, либо нет. Наложение штрафа на функцию потерь, как это делается в методе сокращения веса, накладывает более мягкое ограничение: функции, представленные большими весами, также входят в семейство доступных функций, но обучающий набор должен предоставить больше доказательств в пользу этих функций, чем это требуется для выбора функции с небольшими весами.

Интерпретировать эффект от применения метода сокращения веса в нейронной сети — задача непростая. Предполагается, что в сетях с функциями активации в виде сигмоиды сокращение веса помогает удерживать активацию нейрона

вблизи линейной части сигмоиды, избегая плоской рабочей области, выход в которую приводит к исчезновению градиентов. С функциями активации типа ReLU применение метода сокращения веса также кажется выгодным, но объяснение, которое имело смысл в случае сигмоиды, здесь уже не применимо, поскольку выход функции ReLU является либо линейным, либо нулевым. Более того, при наличии остаточных связей сокращение веса склоняет сеть к поддержанию небольших различий между последовательными слоями вместо того, чтобы использовать веса с малыми абсолютными значениями. Несмотря на эти различия в последствиях применения метода сокращения веса в различных архитектурах, этот метод все еще широко используется.

Одним из объяснений положительного эффекта от применения метода сокращения веса является то, что он реализует некоторую форму максимального апостериорного обучения (MAP) (см. раздел 20.1). Если обозначить через $\mathbf{X}$ и $\mathbf{y}$ входы и выходы по всему обучающему набору, то максимальная апостериорная гипотеза h_{MAP} будет удовлетворять выражению

$$\begin{aligned} h_{\text{MAP}} &= \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{y} \mid \mathbf{X}, \mathbf{W}) P(\mathbf{W}) = \\ &= \operatorname*{argmin}_{\mathbf{w}} [-\log P(\mathbf{y} \mid \mathbf{X}, \mathbf{W}) - \log P(\mathbf{W})]. \end{aligned}$$

Здесь первый член — это обычные потери перекрестной энтропии, а второй член представляет веса, которые, вероятно, подчиняются априорному распределению. Это будет точно совпадать с регуляризованной функцией потерь, если установить

$$\log P(\mathbf{W}) = -\lambda \sum_{i,j} W_{i,j}^2,$$

а это означает, что $P(\mathbf{W})$ является априорным гауссовым распределением с нулевым средним.

21.5.4. Метод исключения

Другой способ, который можно применить для уменьшения ошибки на тестовом наборе — за счет усложнения достижения соответствия обучающему набору, — это применить метод ▶**исключения**. В методе исключения на каждой итерации обучения сети применяется один этап обучения обратным распространением в новой версии сети, создаваемой путем деактивации выбранного случайным образом подмножества элементов. Фактически это грубое и очень дешевое приближение к обучению большого ансамбля различных сетей (см. раздел 19.8).

Говоря более конкретно, давайте предположим, что в сети используется стохастический градиентный спуск с мини-пакетом размером m. Для каждого очередного мини-пакета алгоритм исключения применяет к каждому узлу в сети следующий процесс: с вероятностью p выход элемента умножается на коэффициент $1/p$; в противном случае выход элемента устанавливается равным нулю. В методе исключения

к элементам скрытого слоя обычно применяется вероятность $p = 0,5$; а для входных элементов наиболее эффективным оказалось значение $p = 0,8$. В результате применения этой процедуры получается прореженная сеть, содержащая примерно в два раза меньше элементов, чем исходная, к которой и применяется обратное распространение для мини-пакета из m обучающих примеров. Процесс повторяется обычным способом до тех пор, пока обучение не будет завершено. Во время тестирования модель запускается без исключения, на полном количестве узлов.

Метод исключения можно рассматривать с нескольких точек зрения.

- За счет введении шума во время обучения модель по необходимости должна стать более устойчивой к шуму.
- Как отмечалось выше, метод исключения аппроксимирует процесс создания большого ансамбля прореженных сетей. Это утверждение может быть проверено аналитически для линейных моделей и, по-видимому, подтверждается экспериментально для моделей глубокого обучения.
- Скрытые элементы, обученные по методу исключения, должны научиться не только быть полезными скрытыми элементами, — они также вынуждены научиться быть совместимыми со многими другими возможными наборами других скрытых элементов, которые могут быть, а могут не быть включены в полную модель. Это напоминает процесс отбора, который управляет эволюцией генов: каждый ген должен быть не только эффективен в отношении собственной функции, но и хорошо работать с другими генами, идентичность которых в будущих организмах может значительно отличаться.
- Исключение, применяемое к более поздним слоям глубокой сети, вынуждает принимать окончательное решение надежно, обращая внимание на все абстрактные особенности примера, а не сосредоточиваясь только на одном и игнорируя все остальные. Например, классификатор изображений животных может достичь высокой производительности на обучающем наборе, просто сосредоточив анализ на носу животного, но, вероятно, потерпит неудачу на тестовом примере, в котором нос животного закрыт или поврежден. При применении метода исключения будут иметь место этапы обучения, когда внутренний "носовой элемент" будет обнулен, в результате чего учебному процессу придется найти дополнительные идентифицирующие функции. Обратите внимание, что пытаться достичь той же степени надежности, просто добавляя к входным данным шум, будет сложно: нет простого способа заранее узнать, что сеть собирается сосредоточиться на носах, и нет простого способа автоматически удалить носы из каждого изображения.

В целом метод исключения заставляет модель выучить несколько надежных объяснений для каждого входа. Это приводит к тому, что модель хорошо обобщает, но также затрудняет достижения соответствия обучающему набору, — обычно приходится использовать более крупную модель и обучать ее на дополнительных итерациях.

21.6. Рекуррентные нейронные сети

Рекуррентные нейронные сети (*Recurrent Neural Networks* — RNN) отличаются от сетей с прямой связью тем, что допускают наличие циклов в графе вычислений. Во всех рассмотренных здесь случаях каждый цикл имеет задержку, так что элементы могут принимать в качестве входных данных значение, вычисленное на основании их собственного выхода, выданного на более раннем этапе расчетов. (Без задержки циклический контур может достичь несовместимого состояния.) Такой подход позволяет рекуррентным сетям иметь внутреннее состояние или ▶**память**: входные данные, полученные на более ранних временны*х* этапах, оказывают влияние на ответ рекуррентной сети на текущие входные данные.

Сети RNN также могут быть использованы для выполнения более общих вычислений — в конце концов, обычные компьютеры являются просто логическими схемами с памятью, — как и для моделирования реальных нейронных систем, многие из которых содержат циклические соединения. В этом разделе будет рассмотрено использование сетей RNN для последовательного анализа данных в предположении, что новый входной вектор $\mathbf{x}_t$ поступает на каждом временно*м* этапе.

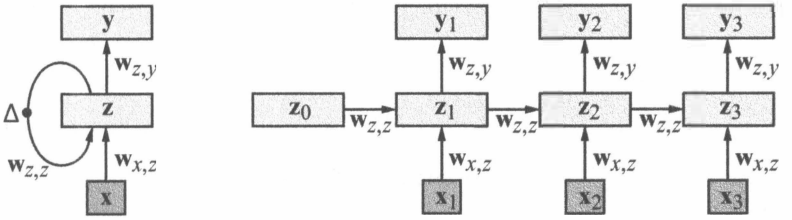
Рассматривая рекуррентные сети как инструмент анализа последовательных данных, их можно сравнить со скрытыми марковскими моделями, динамическими байесовскими сетями и фильтром Калмана, описанными в главе 14. (Читателю может быть полезно еще раз обратиться к этой главе, прежде чем продолжить чтение данного раздела.) Как и в случае этих моделей, для сетей RNN принимается **марковское предположение** (см. раздел 14.1.2): скрытого состояния сети $\mathbf{z}_t$ достаточно для фиксации информации из всех предыдущих входов. Кроме того, предположим, что процесс обновления скрытого состояния сети RNN описывается уравнением $\mathbf{z}_t = f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t)$, где $f_{\mathbf{w}}$ — некоторая параметризированная функция. После обучения эта функция представляет собой некоторый **стационарный** процесс (см. раздел 14.1.2) — по сути, это универсально определенное утверждение о том, что динамика, представленная функцией $f_{\mathbf{w}}$, остается неизменной для всех временны*х* этапов. В результате выразительная сила рекуррентных нейронных сетей в сравнении с сетями с прямой связью значительно возрастает (как и в случае сверточных сетей), подобно тому, как динамические байесовские сети обладают большей выразительной мощностью в сравнении с обычными байесовскими сетями. И действительно, если попытаться использовать сеть с прямой связью для анализа последовательных данных, фиксированный размер входного слоя вынудит сеть обрабатывать эти данные отдельными порциями конечного размера, а значит, сеть не сможет обнаружить зависимости на достаточно большом расстоянии.

21.6.1. Обучение базовой рекуррентной нейронной сети

Базовая модель, рассматриваемая в этом разделе, имеет входной слой $\mathbf{x}$, скрытый слой $\mathbf{z}$ с рекуррентными соединениями и выходной слой $\mathbf{y}$, как показано на рис. 21.8, *а*. Предполагается, что оба значения, $\mathbf{x}$ и $\mathbf{y}$, наблюдаются как обучающие данные на каждом временном этапе. Уравнения, определяющие модель, включают значения переменных, проиндексированных по временному этапу t:

$$\begin{aligned} \mathbf{z}_t &= f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t) = \mathbf{g}_z(\mathbf{W}_{z,z}\mathbf{z}_{t-1} + \mathbf{W}_{x,z}\mathbf{x}_t) \equiv \mathbf{g}_z(\mathbf{in}_{z,t}), \\ \hat{\mathbf{y}}_t &= \mathbf{g}_y(\mathbf{W}_{z,y}\mathbf{z}_t) \equiv \mathbf{g}_y(\mathbf{in}_{y,t}). \end{aligned} \tag{21.13}$$

Здесь $\mathbf{g}_z$ и $\mathbf{g}_y$ обозначают функции активации для скрытого и выходного слоев соответственно. Как обычно, предполагается наличие дополнительного фиктивного входа со значением +1 для каждого элемента, равно как и весов смещения, связанных с этими входами.



а) б)

Рис. 21.8. а) Принципиальная схема базовой рекуррентной нейронной сети, в которой скрытый слой $\mathbf{z}$ включает рекуррентные соединения, — символ Δ здесь указывает на задержку. **б)** Та же самая сеть, развернутая по трем временным этапам для создания сети с прямой связью. Обратите внимание, что одни и те же веса используются на всех временных этапах

При заданной последовательности входных векторов $\mathbf{x}_1, \ldots, \mathbf{x}_T$ и наблюдаемых выходных значениях $\mathbf{y}_1, \ldots, \mathbf{y}_T$ можно превратить эту модель в сеть с прямой связью, "развернув" ее для T временных этапов, как показано на рис. 21.8, *б*. Обратите внимание, что весовые матрицы $\mathbf{W}_{x,z}$, $\mathbf{W}_{z,z}$ и $\mathbf{W}_{z,y}$ являются общими для всех временных этапов. В развернутой сети легко увидеть, что рассчитать градиенты для обучения весов можно обычным способом, — единственное отличие состоит в том, что использование одних и тех же матриц весов во всех слоях делает вычисление градиента немного сложнее.

Чтобы сохранить уравнения простыми, рассмотрим расчет градиента для сети RNN с только одним входным элементом, одним скрытым элементом и одним выходным элементом. Для этого случая, сделав веса смещения явными, получим

$z_t = g_z(w_{z,z}z_{t-1} + w_{x,z}x_t + w_{0,z})$ и $\hat{y}_t = g_y(w_{z,y}z_t + w_{0,y})$. Как и в уравнениях (21.4) и (21.5), примем квадратичную функцию потерь L — в данном случае суммируемую с течением времени. Производные для весов входного слоя и выходного слоя $w_{x,z}$ и $w_{z,y}$, по существу, идентичны уравнению (21.4), поэтому оставим их определение читателю в качестве упражнения. Для весов скрытого слоя $w_{z,z}$ первые несколько этапов следуют той же схеме, что и в уравнении (21.4):

$$\begin{aligned}
\frac{\partial L}{\partial w_{z,z}} &= \frac{\partial}{\partial w_{z,z}}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2 = \sum_{t=1}^{T}-2(y_t - \hat{y}_t)\frac{\partial \hat{y}_t}{\partial w_{z,z}} = \\
&= \sum_{t=1}^{T}-2(y_t - \hat{y}_t)\frac{\partial}{\partial w_{z,z}}g_y(in_{y,t}) = \\
&= \sum_{t=1}^{T}-2(y_t - \hat{y}_t)g_y'(in_{y,t})\frac{\partial}{\partial w_{z,z}}in_{y,t} = \\
&= \sum_{t=1}^{T}-2(y_t - \hat{y}_t)g_y'(in_{y,t})\frac{\partial}{\partial w_{z,z}}(w_{z,y}z_t + w_{0,y}) = \\
&= \sum_{t=1}^{T}-2(y_t - \hat{y}_t)g_y'(in_{y,t})w_{z,y}\frac{\partial z_t}{\partial w_{z,z}}.
\end{aligned} \tag{21.14}$$

Теперь градиент для скрытого элемента z_t можно получить из предыдущего временного этапа следующим образом:

$$\begin{aligned}
\frac{\partial z_t}{\partial w_{z,z}} &= \frac{\partial}{\partial w_{z,z}}g_z(in_{z,t}) = g_z'(in_{z,t})\frac{\partial}{\partial w_{z,z}}in_{z,t} = \\
&= g_z'(in_{z,t})\frac{\partial}{\partial w_{z,z}}(w_{z,z}z_{t-1} + w_{x,z}x_t + w_{0,z}) = \\
&= g_z'(in_{z,t})(z_{t-1} + w_{z,z}\frac{\partial z_{t-1}}{\partial w_{z,z}}),
\end{aligned} \tag{21.15}$$

где в последней строке используется правило для производных произведения: $\partial(uv)/\partial x = v\partial u/\partial x + u\partial v/\partial x$.

Глядя на уравнение (21.15), можно отметить два момента. Во-первых, выражение для градиента является рекурсивным: вклад в градиент от временного этапа t рассчитывается с использованием вклада от временного этапа $t - 1$. Если правильно упорядочить вычисления, общее время выполнения расчетов по вычислению градиента будет линейно пропорционально размеру сети. Данный алгоритм носит название ►**обратное распространение ошибки во времени** (*back-propagation through time* — BPTT) и, как правило, автоматически реализуется программным обеспечением для систем глубокого обучения. Во-вторых, если повторить рекурсивный расчет, то можно увидеть, что градиенты в точке T будут включать термы,

пропорциональные $w_{z,z} \prod_{t=1}^{T} g_z'(in_{z,t})$. Для сигмоиды, функций tanh и ReLU $g' \leq 1$, поэтому наша простая сеть RNN, несомненно, будет страдать от проблемы исчезновения градиента (см. раздел 21.1.2), если $w_{z,z} < 1$. С другой стороны, если $w_{z,z} > 1$, можно столкнуться с проблемой ▶**взрывного градиента**. (В общем случае данные результаты зависят от первого собственного значения матрицы весов $\mathbf{W}_{z,z}$.) В следующем разделе описывается более сложный вариант структуры сети RNN, предназначенный для смягчения этой проблемы.

21.6.2. Рекуррентные нейронные сети с долгой краткосрочной памятью

Несколько специализированных архитектур сетей RNN были разработаны с целью обеспечения сохранения информации в течение многих временных этапов. Одним из самых популярных типов является ▶**долгая краткосрочная память** (*long short-term memory* — LSTM). В архитектуре LSTM компонент долгосрочной памяти называется ▶**ячейкой памяти** и обозначается как **c**; по существу, его содержимое **копируется** от одного временного этапа к другому. (В противовес этому в базовой рекуррентной сети ее память умножается на матрицу весов на каждом временном этапе, как показано в уравнении (21.13).) Новая информация поступает в память путем *добавления* обновлений, благодаря чему выражения градиента не накапливаются с течением времени мультипликативно. Архитектура LSTM также включает ▶**вентили** (*gating units*), представляющие собой векторы, управляющие потоком информации в LSTM посредством поэлементного умножения соответствующего информационного вектора.

- ▶**Вентиль забывания f** определяет, запоминается каждый элемент ячейки памяти (копируется на следующем временном этапе) или забывается (сбрасывается в нуль).
- **Входной вентиль i** определяет, будет ли каждый элемент ячейки памяти аддитивно обновляться новой информацией из входного вектора на текущем временном этапе.
- **Выходной вентиль o** определяет, передается ли каждый элемент ячейки памяти в долгую краткосрочную память **z**, которая здесь играет роль, схожую с ролью скрытого состояния в базовой архитектуре RNN.

В то время как в схемотехнике слово "вентиль" обычно подразумевает реализацию булевой функции, в архитектуре LSTM вентили "мягкие" — например, элементы вектора ячейки памяти будут частично забыты, если соответствующие им элементы вектора вентиля забывания малы, но не равны нулю. Значения вентильных элементов всегда находятся в диапазоне [0, 1] и поступают как выходные сигналы сигмоидной функции, применяемой к текущему входу и предыдущему скрытому состоянию. Если говорить подробнее, то обновляющие уравнения в архитектуре LSTM выглядят следующим образом.

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_{x,f}\mathbf{x}_t + \mathbf{W}_{z,f}\mathbf{z}_{t-1}) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_{x,i}\mathbf{x}_t + \mathbf{W}_{z,i}\mathbf{z}_{t-1}) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{x,o}\mathbf{x}_t + \mathbf{W}_{z,o}\mathbf{z}_{t-1}) \\
\mathbf{c}_t &= \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \tanh(\mathbf{W}_{x,c}\mathbf{x}_t + \mathbf{W}_{z,c}\mathbf{z}_{t-1}) \\
\mathbf{z}_t &= \tanh(\mathbf{c}_t) \odot \mathbf{o}_t
\end{aligned}
$$

Здесь нижние индексы различных весовых матриц $\mathbf{W}$ указывают на источник и пункт назначения соответствующих связей. Символ $\odot$ обозначает поэлементное умножение.

Архитектура LSTM была одной из первых практически используемых форм рекуррентных нейронных сетей. Системы на ее основе продемонстрировали отличную производительность в широком круге задач, включая распознавание речи и распознавание рукописных текстов. Использование сетей данного типа при обработке естественного языка обсуждается в главе 24.

21.7. Обучение без учителя и трансферное обучение

Системы глубокого обучения, обсуждавшиеся до сих пор, строились на принципах обучения с учителем, согласно которым требуется, чтобы каждый обучающий пример был связан с меткой, предоставляющей соответствующее значение целевой функции. Хотя подобные системы могут достигать высокого уровня точности на тестовых наборах — о чем, например, свидетельствуют результаты соревнований ImageNet, — часто они требуют гораздо больше снабженных метками данных, чем требуется человеку для решения той же задачи. Например, ребенку достаточно увидеть только одну фотографию жирафа, а не тысячу, чтобы он смог уверенно распознавать жирафов в широком диапазоне внешнего окружения и ракурсов. Очевидно, что в нашем подходе к глубокому обучению чего-то не хватает. И действительно, могут иметь место случаи, когда нынешний подход к глубокому обучению с учителем сделает решение некоторых задач абсолютно невозможным, поскольку требование использования помеченных данных превысит возможности человеческой расы (или даже всей вселенной) в отношении их предоставления. Более того, даже в тех случаях, когда эта задача выполнима, снабжение метками больших наборов данных обычно требует дефицитного и дорогостоящего человеческого труда.

По этим причинам существует большой интерес к нескольким парадигмам обучения, которые уменьшают зависимость процесса обучения от помеченных данных. Как было показано в главе 19, эти парадигмы включают **обучение без учителя**, **трансферное обучение** и **обучение с частичным привлечением учителя** (*semisupervised learning*), иначе называемое *полуавтоматическим* или *частичным* обучением. Алгоритмы обучения без учителя проводят обучение исключительно на непомеченных входах $\mathbf{x}$, которые часто более доступны, чем помеченные примеры. Алгоритмы обучения без учителя обычно создают порождающие модели,

способные генерировать реалистичный текст, изображения, аудио- и видеоматериалы, а не просто предсказывать метки для данных этих типов. Алгоритмы трансферного обучения нуждаются в некотором количестве помеченных примеров, но способны дополнительно повышать свою эффективность посредством изучения помеченных примеров для других задач, таким образом делая возможным использование дополнительных существующих источников данных. Алгоритмы обучения с частичным привлечением учителя также требуют некоторых помеченных примеров, но способны в дальнейшем улучшать свою производительность за счет изучения непомеченных примеров. В этом разделе рассматриваются подходы к глубокому обучению с использованием методов обучения без учителя и трансферного обучения. Хотя обучение с частичным привлечением учителя в настоящее время также является активной областью исследований в сообществе глубокого обучения, методы, разработанные на момент написания этой главы, в целом еще не подтвердили своей эффективности на практике и поэтому здесь не рассматриваются.

21.7.1. Обучение без учителя

Все алгоритмы обучения с учителем имеют "в сущности" одну и ту же цель: при заданном обучающем наборе входов $\mathbf{x}$ и соответствующих выходов $y = f(\mathbf{x})$ найти путем обучения функцию h, хорошо аппроксимирующую функцию f. Алгоритмы обучения без учителя, с другой стороны, получают на вход лишь обучающий набор из непомеченных примеров $\mathbf{x}$. Здесь описываются две цели, которые такой алгоритм может попытаться достичь. Во-первых, это изучение новых представлений — например, новых признаков изображений, облегчающих идентификацию объектов на них. Во-вторых, это изучение порождающей модели — обычно в форме распределения вероятностей, на основании которого могут быть сгенерированы новые выборки. (Алгоритмы для обучения байесовских сетей, обсуждавшиеся в главе 20, попадают именно в эту категорию.) Многие алгоритмы способны как к изучению представлений, так и к порождающему моделированию.

Предположим, что обучается совместная модель $P_W(\mathbf{x}, \mathbf{z})$, где $\mathbf{z}$ — множество скрытых ненаблюдаемых переменных, некоторым образом представляющих содержание данных $\mathbf{x}$. В соответствии с общим подходом, принятым в этой главе, смысл значений переменных $\mathbf{z}$ заранее не определяется: модели предоставлена свобода изучать любые ассоциации между $\mathbf{z}$ и $\mathbf{x}$, которые она выберет. Например, модель, обучаемая на изображениях рукописных цифр, может выбрать одно направление в пространстве $\mathbf{z}$ для представления толщины штрихов, другое — для представления цвета чернил, еще одно — для представления цвета фона и т.д. В случае обучения на изображениях лиц алгоритм обучения может выбрать одно направление для представления пола, а другое — для фиксации наличия или отсутствия очков, как показано на рис. 21.9.

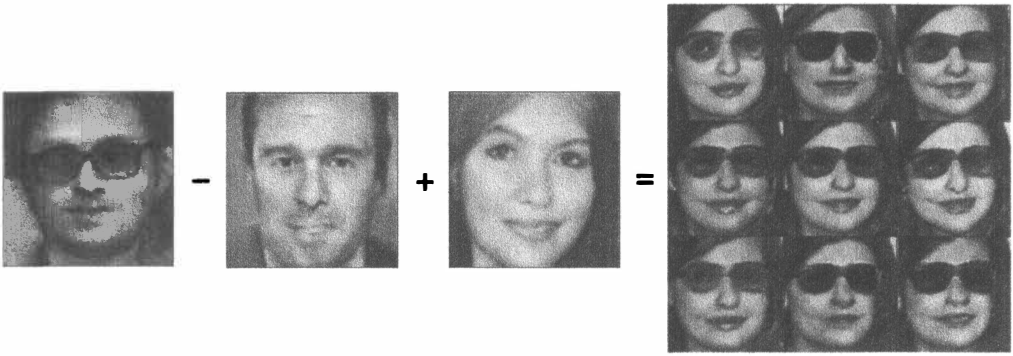


Рис. 21.9. Демонстрация того, как порождающая модель научилась использовать разные направления в пространстве **z** для представления различных аспектов лиц. На практике в пространстве **z** можно выполнять даже арифметические операции. Все изображения здесь сгенерированы обученной моделью и показывают, что происходит, когда декодируются разные точки в пространстве **z**. Начнем с координат для концепции "Мужчина в очках", затем вычтем координаты для концепции "Мужчина", добавим координаты для концепции "Женщина" в результате получим координаты для концепции "женщина в очках"

Обученная вероятностная модель $P_W(\mathbf{x}, \mathbf{z})$ достигает успеха как в изучении представлений (она построила значимые векторы **z** из исходных необработанных векторов **x**), так и в порождающем моделировании: если интегрировать **z** из $P_W(\mathbf{x}, \mathbf{z})$, получим $P_W(\mathbf{x})$.

Вероятностная PCA: простая порождающая модель

Было выдвинуто много предложений относительно формы, которую может принимать модель $P_W(\mathbf{x}, \mathbf{z})$. Одной из самых простых является модель ▶**вероятностного анализа главных компонент** (*probabilistic principal components analysis* — ▶**PPCA**).[7] В модели PPCA вектор **z** выбирается из сферического гауссиана с нулевым средним, а затем **x** генерируется из **z** путем применения матрицы весов **W** и добавления сферического гауссова шума:

$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$
$$P_W(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{Wz}, \sigma^2\mathbf{I}).$$

Матрицу весов **W** (и, необязательно, параметр шума σ^2) можно изучить путем максимизации правдоподобия данных, заданного как

[7] Стандартный вариант PCA предусматривает подгонку многомерного гауссиана к необработанным входным данным, а затем выбор самых длинных осей — главных компонент — этого эллипсоидального распределения.

$$P_W(\mathbf{x}) = \int P_W(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{W}\mathbf{W}^\top + \sigma^2 \mathbf{I}). \quad (21.16)$$

Максимизация относительно $\mathbf{W}$ может быть выполнена градиентными методами или с использованием эффективного итерационного алгоритма EM (см. раздел 20.3). Как только матрица весов $\mathbf{W}$ будет изучена, новые образцы данных могут быть сгенерированы непосредственно из $P_W(\mathbf{x})$ с помощью уравнения (21.16). Более того, новые наблюдения x, имеющие очень низкую вероятность согласно уравнению (21.16), могут быть помечены как потенциальные аномалии.

В случае метода PPCA обычно предполагается, что размерность $\mathbf{z}$ намного меньше размерности $\mathbf{x}$, так что модель учится объяснять данные как можно лучше в терминах небольшого количества признаков. Эти признаки могут быть извлечены для использования в стандартных классификаторах посредством вычисления $\hat{\mathbf{z}}$, ожидания для $P_W(\mathbf{z} \mid \mathbf{x})$.

Генерация данных из вероятностной модели PCA проста: сначала делается выборка $\mathbf{z}$ из ее фиксированного априорного гауссиана, а затем — выборка $\mathbf{x}$ из гауссиана со средним $\mathbf{W}\mathbf{z}$. Как скоро будет показано, многие другие порождающие модели напоминают этот процесс, но используют сложные отображения, определяемые глубокими моделями, а не линейными отображениями из $\mathbf{z}$-пространства в $\mathbf{x}$-пространство.

Автокодировщики

Многие алгоритмы глубокого обучения без учителя основаны на идее ▶**автокодировщика** (*autoencoder*) или *автоассоциатора*. Автокодировщик — это модель, состоящая из двух частей: энкодера, отображающего данные из вектора $\mathbf{x}$ в представление $\hat{\mathbf{z}}$, и декодера, отображающего данные из представления $\hat{\mathbf{z}}$ в наблюдаемые данные $\mathbf{x}$. В общем случае энкодер — это просто параметризованная функция f, а декодер — параметризованная функция g. Модель обучается таким образом, что $\mathbf{x} \approx g(f(\mathbf{x}))$, так что процесс кодирования является приблизительно обратным процессу декодирования. Функции f и g могут быть простыми линейными моделями, параметризованными одиночной матрицей, или они могут быть представлены глубокой нейронной сетью.

Очень простой вариант автокодировщика — линейный автокодировщик, когда функции f и g являются линейными с общей матрицей весов $\mathbf{W}$:

$$\hat{\mathbf{z}} = f(\mathbf{x}) = \mathbf{W}\mathbf{x},$$
$$\mathbf{x} = g(\hat{\mathbf{z}}) = \mathbf{W}^\top \hat{\mathbf{z}}.$$

Одним из способов обучения этой модели является минимизация квадратичной ошибки $\Sigma_j \|\mathbf{x}_j - g(f(\mathbf{x}_j))\|^2$, чтобы $\mathbf{x} \approx g(f(\mathbf{x}))$. Идея заключается в обучении матрицы весов $\mathbf{W}$ таким образом, чтобы низкоразмерный вектор $\hat{\mathbf{z}}$ сохранял как можно больше информации, обеспечивая возможность восстановления многомерных данных $\mathbf{x}$. Такой линейный автокодировщик оказывается тесно связанным с

классическим методом анализа главных компонент (PCA). Если вектор $\mathbf{z}$ m-мерный, то матрица весов $\mathbf{W}$ должна быть обучена охватывать m основных компонент данных, — другими словами, множество из m ортогональных направлений, в которых данные имеют наибольшую дисперсию, что эквивалентно m собственным векторам ковариационной матрицы данных, имеющих наибольшие собственные значения, — в точности так, как в методе PCA.

Модель PCA — это простая порождающая модель, соответствующая простому линейному автокодировщику. Такое соответствие предполагает, что может существовать способ создать более сложные виды порождающих моделей, использующих более сложные виды автокодировщиков. ►**Вариационный автокодировщик** (*variational autoencoder* — VAE) является одним из средств достижения этого.

Вариационные методы были кратко представлены в разделе "Библиографические и исторические заметки" главы 13 как средства аппроксимации апостериорного распределения в сложных вероятностных моделях, где суммирование или интегрирование большого количества скрытых переменных невыполнимо. Идея состоит в том, чтобы использовать ►**вариационное апостериорное распределение** (*variational posterior*) $Q(\mathbf{z})$, взятое из вычислимого семейства распределений, в качестве приближения к истинному апостериорному распределению. Например, распределение Q можно выбрать из семейства гауссовых распределений с диагональной ковариационной матрицей. В пределах выбранного семейства вычислимых распределений распределение Q должно быть оптимизировано так, чтобы быть как можно ближе к истинному апостериорному распределению $P(\mathbf{z} \mid \mathbf{x})$.

Для наших целей условие "как можно ближе" определяется дивергенцией KL, которая упоминалась в разделе 21.2.2. Она определяется как

$$D_{KL}(Q(\mathbf{z}) \parallel P(\mathbf{z} \mid \mathbf{x})) = \int Q(\mathbf{z}) \log \frac{Q(\mathbf{z})}{P(\mathbf{z} \mid \mathbf{x})} d\mathbf{z}$$

и является средним (относительно Q) логарифмическим отношением между Q и P. Легко увидеть, что $D_{KL}(Q(\mathbf{z}) \parallel P(\mathbf{z} \mid \mathbf{x})) \geq 0$ и равно нулю, когда Q и P совпадают. Теперь можно определить ►**нижнюю вариационную границу** $\mathcal{L}$ (иногда называемую ►**доказательной нижней границей** (*evidence lower bound* — ►**ELBO**) для логарифмического правдоподобия данных:

$$\mathcal{L}(\mathbf{x}, Q) = \log P(\mathbf{x}) - D_{KL}(Q(\mathbf{z}) \parallel P(\mathbf{z} \mid \mathbf{x})). \tag{21.17}$$

Можно видеть, что $\mathcal{L}$ является нижней границей для $\log P$, поскольку дивергенция KL является неотрицательной. Вариационное обучение максимизирует значение $\mathcal{L}$ относительно параметров $\mathbf{w}$ вместо максимизации $\log P(\mathbf{x})$ в предположении, что найденное решение $\mathbf{w}^*$ будет близко и к максимизированному значению $\log P(\mathbf{x})$.

В том виде, в каком оно записано, кажется, что максимизировать значение $\mathcal{L}$ ничуть не легче, чем значение $\log P$. К счастью, уравнение (21.17) можно переписать

таким образом, чтобы сделать очевидной возможность улучшений в отношении его вычислимости:

$$\begin{aligned}\mathcal{L} &= \log P(\mathbf{x}) - \int Q(\mathbf{z}) \log \frac{Q(\mathbf{z})}{P(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} = \\ &= -\int Q(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z} + \int Q(\mathbf{z}) \log P(\mathbf{x}) P(\mathbf{z} \mid \mathbf{x}) d\mathbf{z} = \\ &= H(Q) + \mathbf{E}_{\mathbf{z} \sim Q} \log P(\mathbf{z}, \mathbf{x}),\end{aligned}$$

где $H(Q)$ — энтропия распределения Q. Для некоторых вариационных семейств Q (таких, как распределение Гаусса) энтропия $H(Q)$ может быть оценена аналитически. Более того, ожидание $\mathbf{E}_{\mathbf{z} \sim Q} \log P(\mathbf{z}, \mathbf{x})$ допускает эффективную несмещенную оценку через выборки $\mathbf{z}$ из Q. Для каждой выборки оценить значение $P(\mathbf{z}, \mathbf{x})$ обычно можно достаточно эффективно; например, если P является байесовской сетью, то $P(\mathbf{z}, \mathbf{x})$ будет просто произведением условных вероятностей, поскольку $\mathbf{z}$ и $\mathbf{x}$ включают все переменные.

Вариационные автокодировщики предоставляют средства для проведения вариационного обучения в системах глубокого обучения. Вариационное обучение предполагает максимизацию $\mathcal{L}$ по отношению к обоим параметрам — и P, и Q. Для вариационного автокодировщика декодер $g(\mathbf{z})$ интерпретируется как определяющий $\log P(\mathbf{x} \mid \mathbf{z})$. Например, выходной сигнал декодера может определять среднее значение условного гауссиана. Аналогично выход энкодера $f(\mathbf{x})$ здесь может интерпретироваться как определяющий параметры Q, — например, Q может быть гауссианом со средним значением $f(\mathbf{x})$. В этом случае обучение вариационного автокодировщика заключается в максимизации значения $\mathcal{L}$ относительно параметров как энкодера f, так и декодера g, которые сами могут быть представлены сколь угодно сложным глубокими сетями.

Глубокие модели авторегрессии

►**Модель авторегрессии** (*autoregressive model* — AR) — это такая модель, в которой каждый элемент x_i вектора данных $\mathbf{x}$ предсказывается на основе других элементов этого вектора. Такая модель не имеет скрытых переменных. Если вектор $\mathbf{x}$ имеет фиксированный размер, то модель авторегрессии можно рассматривать как полностью наблюдаемую и, возможно, полносвязную байесовскую сеть. А это означает, что вычисление правдоподобия заданного вектора данных в соответствии с моделью AR является тривиальным, и это верно также для прогнозирования значения одной отсутствующей переменной при заданных всех остальных, а также для выборки вектора данных из модели.

Наиболее распространенным применением авторегрессионных моделей является анализ временных рядов, в которых модель авторегрессии порядка k предсказывает значение x_t при заданных значениях $x_{t-k}, \ldots, x_{t-1}$. В терминологии главы 14 модель авторегрессии является не скрытой марковской моделью. В терминологии

главы 23 N-граммная модель последовательности букв или слов является моделью авторегрессии порядка $n-1$.

В классических моделях авторегрессии, где значения переменных являются действительными числами, условное распределение $P(x_t \mid x_{t-k},\ldots,x_{t-1})$ является линейной гауссовой моделью с фиксированной дисперсией, среднее значение которой является взвешенной линейной комбинацией $x_{t-k},\ldots,x_{t-1}$, — другими словами, это стандартная модель линейной регрессии. Максимально правдоподобное решение в этом случае определяется ►**уравнениями Юла–Уокера**, которые тесно связаны с **нормальными уравнениями**, обсуждавшимися в разделе 19.6.3.

►**Модель глубокой авторегрессии** — это модель, в которой линейная гауссова модель заменяется глубокой сетью произвольной глубины с подходящим выходным слоем в зависимости от того, какие значения имеет x_t, дискретные или непрерывные. Недавние применения подобного авторегрессионного подхода включают модель WaveNet от компании DeepMind, предназначенную для генерации речи (Ван ден Оорд и др. [2251], 2016). Модель WaveNet была обучена на необработанных акустических сигналах, дискретизированных с частотой 16 000 Гц и реализована как нелинейная модель авторегрессии порядка 4800 с многослойной сверточной структурой. При тестировании она продемонстрировала значительно большую реалистичность своих результатов в сравнении с ранее разработанными современными системами генерации речи.

Генеративно-состязательные сети

►**Генеративно-состязательная сеть** (*generative adversarial network* — ►GAN) — это фактически две сети, объединенные с целью образования порождающей системы. Одна из сетей, ►**генератор**, отображает значения вектора $\mathbf{z}$ на вектор $\mathbf{x}$ для получения образцов из распределения $P_{\mathbf{w}}(\mathbf{x})$. В типичной схеме генератор строит образец $\mathbf{z}$ из единичного гауссиана умеренной размерности, а затем пропускает его через глубокую сеть $h_{\mathbf{w}}$ для получения вектора $\mathbf{x}$. Другая сеть, ►**дискриминатор**, представляет собой классификатор, обученный классифицировать входы $\mathbf{x}$ как реальные (взятые из обучающего набора) или подделки (созданные генератором). Модель GAN — это своего рода ►**неявная модель** в том смысле, что образцы могут быть сгенерированы, но их вероятности не всегда доступны, тогда как в байесовской сети вероятность образца является просто произведением условных вероятностей вдоль пути его генерации.

Генератор "по сути" очень близок к декодеру в системе вариационного автокодировщика. Задача неявного моделирования состоит в разработке такой функции потерь, которая делает возможным обучение модели с использованием образцов из распределения, а не посредством максимизации правдоподобия, присвоенного обучающим примерам из набора данных.

Как генератор, так и дискриминатор обучаются одновременно, при этом генератор учится обманывать дискриминатор, а дискриминатор учится точно различать

данные реальные и поддельные. Соревнование между генератором и дискриминатором может быть описано языком теории игр (см. главу 18). Основная идея состоит в том, что в состоянии игрового равновесия генератор должен воспроизводить обучающее распределение идеально, чтобы дискриминатор не мог действовать лучше, чем выдвигать случайные догадки. Системы GAN особенно хорошо работают в приложениях генерации изображений. Например, генеративно-состязательные сети могут создавать фотореалистичные изображения с высоким разрешением людей, которых никогда не существовало (Каррас и др. [1192], 2017).

Перевод при обучении без учителя

Задачи перевода, в широком смысле, состоят из преобразования входных данных $\mathbf{x}$, имеющих богатую структуру, в выходные данные $\mathbf{y}$, которые также обладают богатой структурой. В этом контексте выражение "богатая структура" означает, что данные являются многомерными и имеют интересные статистические зависимости между различными размерностями. Богатую структуру имеют изображения и предложения на естественном языке, тогда как одно число, такое как идентификатор класса, ее не имеет. Преобразование предложения с английского языка на французский или преобразование фотографии ночной сцены в эквивалентную фотографию, сделанную в дневное время, — обе эти задачи являются примерами задач перевода.

Обучение системы перевода методом обучения с учителем состоит в подготовке множества пар $(\mathbf{x}, \mathbf{y})$ и обучении модели сопоставлению каждого вектора $\mathbf{x}$ соответствующему вектору $\mathbf{y}$. Например, системы машинного перевода часто обучаются на парах предложений, переведенных профессиональными переводчиками-людьми. Для других типов перевода данные для обучающих наборов с использованием метода обучения с учителем могут быть просто недоступны. Например, обратимся к фотографии ночной сцены со множеством движущихся автомобилей и пешеходов. Совершенно очевидно, что невозможно отыскать все автомобили и пешеходов, а затем вернуть их в такое же положение, как на ночной фотографии, чтобы сделать аналогичную фотографию, но в дневное время. Чтобы преодолеть эту трудность, можно воспользоваться методами ▶ **перевода без учителя**, позволяющими проводить обучение на многих примерах $\mathbf{x}$ и на многих отдельных примерах $\mathbf{y}$, но без использования соответствующих пар $(\mathbf{x}, \mathbf{y})$.

Такие подходы обычно строятся на использовании генеративно-состязательных сетей. Например, можно обучить один генератор системы GAN созданию реалистичного примера $\mathbf{y}$ при условии $\mathbf{x}$, а другой генератор системы GAN — выполнению обратного отображения. Система обучения с использованием генеративно-состязательных сетей позволяет научить генератор генерировать любой из множества возможных образцов, которые дискриминатор принимает как *реалистичный* пример $\mathbf{y}$ при заданном $\mathbf{x}$, без какой-либо потребности в определенном парном $\mathbf{y}$, что обычно требуется при обучении с учителем. Подробнее о методах перевода без учителя применительно к изображениям речь пойдет в разделе 25.7.5.

21.7.2. Трансферное обучение и многозадачное обучение

При ▶**трансферном обучении** опыт, полученный в одной задаче обучения, помогает агенту лучше учиться на другой задаче. Например, человеку, который уже научился играть в теннис, обычно будет проще научиться играть в родственные виды спорта, такие как бадминтон или сквош. Пилот, который научился управлять одним типом коммерческого пассажирского самолета, быстро научится управлять самолетом другого типа, а студенту, который хорошо усвоил алгебру, будет легче осваивать более сложные математические дисциплины.

Механизмы трансферного обучения человека нам еще не известны. Для нейронных сетей их обучение заключается в корректировке весов, поэтому наиболее вероятным подходом к трансферному обучению будет копирование весов, полученных при обучении для задачи A, в сеть, которая будет обучаться решению задачи B. Далее эти веса будут обновляться обычным способом методом градиентного спуска на основании обучающих данных для задачи B. Может оказаться хорошей идеей использовать в задаче B меньшую скорость обучения, в зависимости от того, насколько схожи задачи A и B и сколько обучающих данных было использовано в задаче A.

Обратите внимание, что этот подход при выборе задач нуждается в экспертных знаниях людей; например, веса, полученные во время обучения алгебре, могут оказаться не очень полезными для обучения сети, предназначенной для игры в бадминтон. Кроме того, понятие копирования весов требует простого отображения между входными пространствами двух задач и, по сути, идентичной сетевой архитектуры.

Одной из причин популярности трансферного обучения является наличие высококачественных обученных моделей. Например, можно загрузить предварительно обученную модель визуального распознавания объектов, такую как модель ResNet-50, обученную на наборе данных COCO, и тем самым сэкономить недели работы. Затем можно изменить параметры модели, предоставив ей дополнительные изображения и метки объектов, характерных для конкретной решаемой задачи.

Предположим, что требуется классифицировать типы одноколесных велосипедов. Имеется всего несколько сотен картинок различных одноколесных велосипедов, но набор данных COCO содержит более 3 тыс. изображений в каждой из категорий велосипедов, мотоциклов и скейтбордов. Это означает, что модель, предварительно обученная на наборе COCO, уже имеет опыт работы с колесами, дорогами и другими соответствующими признаками, которые могут быть полезны в интерпретации изображений одноколесных велосипедов.

Часто может потребоваться заморозить первые несколько слоев предварительно обученной модели — эти слои служат детекторами признаков, которые будут полезны для новой создаваемой модели. Примерам из нового обучающего набора данных будет разрешено изменять параметры только более высоких слоев, — это слои, определяющие проблемно специфические признаки и выполняющие

классификацию. Однако наличие различий между датчиками, используемыми в двух моделях, иногда может означать, что потребуется переподготовка даже слоев самого низкого уровня.

Вот другой пример: для тех, кто строит системы работы с естественным языком, в настоящее время общепринятый подход — начать с предварительно обученной модели, такой как модель RoBERTa (см. раздел 24.6), которая уже многое "знает" о словарном запасе и синтаксисе повседневного языка. Следующим этапом обычно является точная настройка новой модели, выполняемая двумя способами. Во-первых, системе предоставляются специализированные примеры словарного запаса, используемого в нужной предметной области, — возможно, это медицина (где система узнает об "инфаркте миокарда") или же международные финансы (где система познакомится с понятием "фидуциарная ответственность"). Во-вторых, обучая модель выполнению задания, для которого она предназначена. Если задание заключается в предоставлении ответов на вопросы, то система обучается на известных парах "вопрос-ответ".

Один очень важный вид трансферного обучения включает в себя переход между симуляцией и реальным миром. Например, система управления беспилотного автомобиля может обучаться на миллиардах миль симуляции вождения, что просто невозможно в реальном мире. В конечном счете, когда система управления будет перенесена на реальный автомобиль, она быстро адаптируется к новой среде.

► **Многозадачное обучение** является такой формой трансферного обучения, в которой модель обучается одновременно нескольким целям. Например, вместо обучения системы работе с естественным языком на помеченных примерах частей речи, а затем переносу изученных весов в новую задачу — такую, как классификация документов, — одна система обучается одновременно выделению частей речи, классификации документов, определению языка, предсказанию слов, моделированию сложности предложений, обнаружению плагиата, логическому следствию предложений и предоставлению ответов на вопросы. Идея состоит в том, что для решения любой из этих задач модель может оказаться способной воспользоваться внешними признаками данных. Но решая все восемь задач одновременно с общим уровнем представления, модель, скорее всего, создаст общее представление, отражающее реальное использование естественного языка и содержание.

21.8. Применение глубокого обучения на практике

Глубокое обучение успешно применяется во многих важных проблемных областях ИИ. Более глубокое рассмотрение этой темы можно найти в соответствующих главах этой книги: в главе 22 — использование глубокого обучения в системах обучения с подкреплением, в главе 24 — в системах обработки естественного языка, в главе 25 (в частности, в разделе 25.4) — в системах компьютерного зрения, а в главе 26 — в робототехнике.

21.8.1. Компьютерное зрение

Начнем с компьютерного зрения, которое можно считать областью применения, оказавшей, вероятно, наибольшее влияние на глубокое обучение, и наоборот. Хотя глубокие сверточные сети использовались для таких задач, как распознавание рукописного ввода, еще с начала 1990-х годов, а нейронные сети смогли превзойти порождающие вероятностные модели распознавания речи уже к началу 2010 года, область глубокого обучения смогла, наконец, оказаться в центре внимания сообщества ИИ только благодаря успеху системы глубокого обучения AlexNet в конкурсе ImageNet 2012 года.

Конкурс ImageNet представлял собой задачу машинного обучения с учителем на базе набора из 1 200 000 изображений, относящихся к 1000 различных категорий. Системы оценивались по баллам "топ-5" — как часто правильная категория оказывалась первой в числе пяти первых прогнозов, выдаваемых системой в порядке убывания достоверности для каждого изображения. Система AlexNet достигла уровня ошибок 15,3%, тогда как следующая лучшая система имела уровень ошибок более 25%. Система AlexNet включала пять сверточных слоев, перемежающихся слоями с максимальным объединением, за которыми следовали три полносвязных слоя. В ней использовались функция активации ReLU и графические процессоры для ускорения процесса обучения 60 млн весов.

С 2012 года, благодаря улучшениям в конструкции сети, методах обучения и вычислительных ресурсах, показатель ошибок "топ-5" уменьшился до уровня менее 2%, что намного ниже уровня обученного человека (около 5%). Сверточные нейронные сети нашли применение при решении широкого круга задач компьютерного зрения, от беспилотных автомобилей до сортировки огурцов.[8] Беспилотное вождение, которое рассматривается в разделе 25.7.6 и в нескольких разделах главы 26, является, пожалуй, одной из самых сложных задач компьютерного зрения: алгоритм должен не просто обнаруживать, локализовать, отслеживать и распознавать голубей, бумажные пакеты и пешеходов, а делать все это в реальном времени и почти с идеальной точностью.

21.8.2. Обработка естественного языка

Глубокое обучение также оказало огромное влияние на приложения обработки естественного языка (*natural language processing* — NLP), такие как машинный перевод и распознавание речи. Некоторые преимущества глубокого обучения для этих приложений включают в себя возможность сквозного обучения,

[8] Широко известная история о выращивавшем огурцы японском фермере, который на базе пакета TensorFlow построил собственного робота для сортировки огурцов, по большей части оказалась мифической. Алгоритм был разработан сыном фермера, работавшим ранее инженером-программистом в компании Toyota, а его недостаточная точность — около 70% — означала, что огурцы все еще приходилось сортировать вручную (Зиберг [2422], 2017).

автоматическую генерацию внутренних представлений для значений слов и взаимозаменяемость обученных энкодеров и декодеров.

Под сквозным обучением понимается построение целых систем как единой обученной функции f. Например, функция f для машинного перевода может принимать в качестве входа предложение S_E на английском языке и выводить эквивалентное предложение на японском языке $S_J = f(S_E)$. Такая функция f может быть изучена на основе обучающих данных в виде переведенных человеком пар предложений (или даже пар текстов, в которых выравнивание соответствующих предложений или фраз является частью решаемой проблемы). Более классический конвейерный подход может сначала проанализировать S_E, затем извлечь его значение, после вновь выразить это значение на японском языке как S_J и наконец отредактировать S_J с использованием языковой модели японского языка. Подобный конвейерный подход имеет два основных недостатка: во-первых, ошибки накапливаются на каждом этапе, и во-вторых, люди должны определять, что представляет собой "дерево разбора" и "смысловое значение", однако для этих понятий нет легкодоступной объективной основы и наши теоретические представления о них почти наверняка неполны.

На нынешнем этапе понимания, впрочем, классический конвейерный подход — который хотя бы наивно, кажется, лучше соответствует тому, как работает человек-переводчик, — уже обошел сквозной метод, который стал возможным благодаря глубокому обучению. Например, Ву и соавт. в работе [2392] (2016) показали, что сквозной перевод с использованием глубокого обучения уменьшил ошибку перевода на 60% по сравнению с предыдущей конвейерной системой. С 2020 года производительность систем машинного перевода приближается к свойственной человеку производительности для таких языковых пар, как французский и английский, для которых доступны очень большие парные наборы данных, и они могут использоваться для других языковых пар, охватывающих бо́льшую часть населения Земли. Есть даже некоторые свидетельства того, что сети, обучаемые на нескольких языках, на самом деле учатся представлению внутреннего смысла; например, после обучения переводу с португальского на английский и с английского на испанский оказалось возможным напрямую переводить с португальского на испанский при отсутствии каких-либо пар предложений "португальский/испанский" в обучающем наборе.

Один из наиболее важных выводов, вытекающих из применения глубокого обучения для языковых задач, заключается в том, что значительная часть выгоды происходит от многократного внутреннего представления обычных слов как векторов в многомерном пространстве — так называемых **встраиваний слов** (см. раздел 24.1). Векторы обычно извлекаются из весов первого скрытого слоя сети, обученной на больших объемах текста, и собирают статистику лексического контекста, в котором используются слова. Поскольку слова со схожими значениями используются в схожих контекстах, они оказываются близко друг к другу и в векторном пространстве. Это позволяет сети эффективно обобщать слова

по собственным категориям без необходимости того, чтобы люди предопределили эти категории. Например, если начать предложение словами "Джон купил арбуз и два фунта...", то система, скорее всего, продолжит его словами "яблок" или "бананов", но никак не "тория" или "географии". Такой прогноз сделать гораздо проще, если "яблоки" и "бананы" будут иметь схожие представления во внутреннем слое.

21.8.3. Обучение с подкреплением

При обучении с подкреплением (*reinforcement learning* — RL) агент, принимающий решения, учится на последовательности сигналов вознаграждения, которые обеспечивают ему некоторую информацию о качестве его поведения. Цель состоит в том, чтобы оптимизировать сумму будущих наград. Это можно сделать несколькими способами: в терминологии главы 17 агент может изучать функцию ценности, Q-функцию, стратегию и т.д. С точки зрения глубокого обучения все это — функции, которые могут быть представлены графами вычислений. Например, функция ценности в игре го принимает позицию доски в качестве входных данных и возвращает оценку, насколько эта позиция выгодна для агента. В то время как методы обучения в RL отличаются от методов обучения с учителем, способность многослойных вычислительных графов представлять сложные функции в больших входных пространствах оказалась очень полезной. Соответствующее поле исследований называется ▶ **глубоким обучением с подкреплением**.

В 1950-х годах Артур Самуэль экспериментировал с многослойными представлениями функции ценности в своей работе по обучению с подкреплением программы игры в шашки, но обнаружил, что на практике аппроксимация линейными функциями работала лучше всего. (Этот факт мог быть следствием его работы на компьютере, вычислительная мощность которого была примерно в 100 млрд раз меньше мощности современного тензорного процессора.) Первой крупной успешной демонстрацией глубокого обучения с подкреплением был агент компании DeepMind, игравший на приставке Atari, (Мних и др. [1603], 2013). Разные экземпляры этого агента были обучены играть в несколько видеоигр на приставке Atari и продемонстрировали такие навыки, как стрельба по инопланетным космическим кораблям, отбивание мячей ракетками и вождение гоночных автомобилей в симуляторах. В каждом случае агент изучал Q-функцию на основе данных в виде необработанных изображений и сигнала вознаграждения, представлявшего собой счет в игре. Последующая работа привела к разработке систем глубокого обучения с подкреплением, которые на сверхчеловеческом уровне играют в большинство из 57 различных игр на приставках Atari. В системе AlphaGo компании DeepMind также использовалось глубокое обучение с подкреплением, обеспечившее ей победу над лучшими игроками в го среди людей (см. главу 5).

Несмотря на впечатляющие успехи, глубокое обучение с подкреплением по-прежнему сталкивается со значительными препятствиями: часто бывает очень трудно достичь хорошей производительности, и обученная система может вести

себя очень непредсказуемо, если среда даже совсем незначительно отличается от обучающих данных (Ирпан [1116], 2018). По сравнению с другими приложениями глубокого обучения, глубокое обучение с подкреплением редко применяется в коммерческих условиях. Тем не менее это по-прежнему очень активная область исследований.

Резюме

В этой главе описаны методы обучения функций, представленных глубокими графами вычислений. Рассматривались следующие основные темы.

- **Нейронные сети** представляют сложные нелинейные функции с помощью сети из параметризованных линейно-пороговых элементов.
- **Алгоритм обратного распространения** реализует градиентный спуск в пространстве параметров с целью минимизации функции потерь.
- **Глубокое обучение** хорошо работает в задачах распознавания визуальных объектов, распознавания речи, обработки естественного языка и обучения с подкреплением в сложном окружении.
- **Сверточные сети** особенно хорошо подходят для обработки изображений и других задач, в которых данные имеют топологию сетки.
- **Рекуррентные сети** эффективны для задач обработки последовательности данных, включая языковое моделирование и машинный перевод.

Библиографические и исторические заметки

Литература по нейронным сетям весьма обширна. Коуэн и Шарп ([484], 1988; [483], 1988) исследовали раннюю историю этого направления, начиная с работы Мак-Каллока и Питтса ([1537], 1943). (Как уже упоминалось в главе 1, Джон Маккарти указал на работу Николаса Рашевски ([1854], 1936; [1855], 1938) как на самую раннюю математическую модель нейронного обучения.) Норберт Винер, пионер кибернетики и теории управления (Винер [2339], 1948), работал с Мак-Каллоком и Питтсом и оказал влияние на многих молодых исследователей, включая Марвина Мински, который, возможно, был первым, кто в 1951 году разработал работающую нейронную сеть в аппаратной реализации (см. Мински и Паперт [1586], (1988), стр. ix–x). Алан Тьюринг ([2234], 1948) написал исследовательский отчет под названием *Intelligent Machinery*, который начинается с предложения "Я предлагаю исследовать вопрос о том, возможно ли для машины продемонстрировать интеллектуальное поведение". Далее в этом отчете описываются архитектура рекуррентной нейронной сети, которую он назвал "неорганизованная машина типа B", и подход к ее обучению. К сожалению, этот отчет оставался неопубликованным до 1969 года и до недавнего времени практически игнорировался.

Перцептрон, однослойная нейронная сеть с жесткой пороговой функцией активации, был представлен миру Френком Розенблаттом ([1910], 1957). После демонстрации в июле 1958 года в газете "Нью-Йорк Таймс" ее назвали "зародышем электронного компьютера, который, как ожидают [в военно-морском флоте], будет способен ходить, говорить, видеть, писать, воспроизводить себя и осознавать свое существование". Позже Розенблат ([1911], 1960) доказал теорему о сходимости перцептрона, хотя она была определена в чисто математической работе вне контекста нейронных сетей (Агмон [21], 1954; Моцкин и Шенберг[1629], 1954). Несколько ранних исследовательских работ проводилось также на многослойных сетях, включая **перцептроны Гамбы** (Гамба и др. [809], 1961) и **мадалины** (Уидроу [2335], 1962). Статья *Learning Mahines* (Нильссон [1686], 1965) охватывает бо́льшую часть этой ранней работы и многое другое. Последующий закат ранних исследований перцептрона был ускорен (или, как позже утверждали авторы, просто объяснен) выходом книги *Perceptrons* (Мински и Паперт [1585], 1969), в которой высказывались сожаления по поводу отсутствия математической строгости в этом поле исследований. В книге также подчеркивалось, что однослойные перцептроны могут представлять только линейно разделяемые понятия, и отмечалось отсутствие эффективных алгоритмов обучения для многослойных сетей. Все эти ограничения были уже хорошо известны (Хокинс [987], 1961) и признаны самим Розенблаттом ([1912], 1962).

Работы, собранные Хинтоном и Андерсоном ([1025], 1981) на базе конференции, проходившей в Сан-Диего в 1979 году, можно рассматривать как знаменующие возрождение коннекционизма. Двухтомная антология *PDP* (*Parallel Distributed Processing* — параллельная распределенная обработка) Румельхарта и МакКлеланда ([1933], 1986), как говорили, "способствовала распространению евангелия", особенно в научных сообществах психологии и когнитивной науки. Самым важным достижением этого периода был алгоритм обратного распространения для обучения многослойных сетей.

Алгоритм обратного распространения был несколько раз независимо открыт в разных контекстах (Келли [1214], 1960; Брайсон [1214], 1962; Дрейфус [654], 1962; Брайсон и Хо [333], 1969; Вербос [2321], 1974; Паркер [1734], 1985), поэтому Стюарт Дрейфус в работе [656] (1990) назвал это "градиентной процедурой Келли–Брайсона". Хотя Вербос рассматривал этот алгоритм применительно к нейронным сетям, его идеи не получили широкого распространения, пока в журнале *Nature* не появилась статья Дэвида Румелхарта, Джеффа Хинтона и Рона Уильямса [1932] (1986), в которой было дано нематематическое представление этого алгоритма. Математическая респектабельность была подкреплена публикациями, показывающими, что многослойные сети с прямой связью являются (при соблюдении технических условий) универсальными аппроксиматорами функций (Цибенко [510], 1988; [511], 1989). В конце 1980-х и в начале 1990-х годов наблюдался огромный рост числа исследований в области нейронных сетей: количество статей по этой тематике в период между 1980–1984 и 1990–1994 годами выросло в 200 раз!

В конце 1990-х и начале 2000-х годов интерес к нейронным сетям уменьшился по сравнению с интересом к другим методами, — на первый план вышли байесовские сети, ансамблевые методы и методы ядра. Интерес к глубоким моделям возродился только тогда, когда исследования Джеффа Хинтона по глубоким байесовским сетям — порождающие модели с категориальными переменными в корне и переменными свидетельства в листьях — начали приносить плоды, превосходящие ядерные методы на небольших проверочных наборах данных (Хинтон и др. [1032], 2006). Однако по-настоящему интерес к глубокому обучению разгорелся лишь тогда, когда Крижевски и соавт. ([1313], 2013), благодаря использованию глубоких сверточных сетей, добились убедительной победы в конкурсе ImageNet (Руссаковски и др. [1936], 2015).

Комментаторы часто ссылаются на доступность "больших данных" и вычислительную мощность графических процессоров как на основные факторы, способствовавшие возникновению глубокого обучения. Также важны были и улучшения в области архитектуры сетей, в том числе принятие функции активации ReLU вместо логистической сигмоиды (Джарретт и др.[1128], 2009; Нейр и Хинтон [1654], 2010; Глоро и др. [869], 2011), а позднее и разработка остаточных сетей (Хе и др. [995], 2016).

С алгоритмической стороны использование стохастического градиентного спуска (SGD) с мини-пакетами оказалось принципиально важным решением, обеспечившим возможность масштабировать нейронные сети до очень больших наборов данных (Боту и Буске [261], 2008). Нормализация пакетов (Иоффе и Жегеди [2015], 2015) также способствовала ускорению и повышению надежности процесса обучения и вызвала появление нескольких дополнительных методов нормализации (Ба и др. [93], 2016; Ву и Хе [2393], 2018; Миято и др. [1602], 2018). В нескольких работах анализировалось эмпирическое поведение алгоритма SGD в больших сетях и при больших наборах данных (Дофин и др. [530], 2015; Хороманска и др. [423], 2014; Гудфеллоу и др. [898], 2015). С теоретической стороны некоторый прогресс был достигнут в объяснении наблюдений того факта, что алгоритм SGD применительно к сверхпараметрическим сетям часто достигает глобального минимума с ошибкой обучения, равной нулю, хотя до сих пор в теоремах об этом эффекте предполагается использование сети со слоями, намного более широкими, чем это когда-либо осуществлялось на практике (Аллен-Чжу и др. [37], 2018; Ду и др. [657], 2018). У таких сетей имеется более чем достаточно возможностей, чтобы действовать как поисковые таблицы для данных обучающего набора.

Последней частью головоломки, по крайней мере для приложений компьютерного зрения, стало использование сверточных сетей. Впервые эта концепция появилась в описаниях зрительной системы млекопитающих нейрофизиологами Дэвидом Хьюбелом и Торстено Визелом (Хьюбел и Визел [1086], 1959; [1087], 1962; [1088], 1968). В зрительной системе кошки они описали "простые" клетки, напоминающие детекторы краев, а также "сложные" клетки, инвариантные к некоторым преобразованиям, таким как небольшие пространственные смещения.

В современных сверточных сетях выход сверточного слоя подобен "простой" клетке, тогда как выход объединяющего слоя аналогичен сложной клетке.

Работа Хьюбела и Визела вдохновила появление многих ранних коннекционистских моделей компьютерного зрения (Марр и Поджио [1498], 1976). Неокогнитрон (Фукусима [798], 1980; Фукусима и Мияке [799], 1982), разработанный как модель зрительной коры, был, по существу, сверточной сетью с точки зрения архитектуры модели, хотя появления эффективного алгоритма обучения для таких сетей еще следовало подождать, — пока Ян ЛеКун и соавт. не показали, как можно применять обратное распространение (ЛеКун и др.[1373], 1995). Одним из ранних коммерческих успехов нейронных сетей была система распознавания рукописных цифр, построенная с использованием сверточных сетей (ЛеКун и др. [1373], 1995).

В 1970-х годах рекуррентные нейронные сети (RNN) обычно предлагались как модели функций мозга, но с этими предложениями не было связано никаких эффективных алгоритмов обучения. Метод обратного распространения ошибки во времени впервые был представлен в докторской диссертации Пола Вербоса ([2321], 1974), а его более поздняя обзорная статья (Вербос [2322], 1990) содержит несколько дополнительных ссылок на повторные открытия этого метода в 1980-х годах. Одна из самых значительных ранних работ по RNN была опубликована Джеффом Элманом ([683], 1990), построившим RNN с архитектурой, предложенной Майклом Джорданом ([1150], 1986). Уильямс и Зипсер ([2351], 1989) представили алгоритм динамического обучения RNN. Бенжио и соавт. ([171], 1994) проанализировали проблему исчезающих градиентов в рекуррентных сетях. Как средство избежания этой проблемы была предложена архитектура долгой краткосрочной памяти (LSTM) (Хохрейтер [1038], 1991; Хохрейтер и Шмидхубер [1039], 1997; Герс и др. [841], 2000). Относительно недавно появилась возможность разрабатывать эффективные схемы RNN автоматически (Йозефович и др. [1156], 2015; Зоф и Ли [2453], 2016).

С целью улучшения обобщения в нейронных сетях были опробованы многие методы. Метод сокращения весов был предложен Хинтоном ([1030], 1987) и математически проанализирован Крогом и Герцем ([1315], 1992). Метод исключения представили Шривастава и соавт. ([2119], 2014). Сегеди и соавт. ([2166], 2013) предложили идею состязательных примеров, что породило огромный поток публикаций.

Пул и соавт. ([1807], 2017) показали, что глубокие сети (но не мелкие) могут распутывать сложные функции в плоские системы связей в пространстве скрытых элементов. Ролник и Тегмарк ([1907], 2018) доказали, что количество элементов, необходимых для аппроксимации определенного класса полиномов с n переменными, растет экспоненциально для мелких сетей, но только линейно — для глубоких сетей.

Уайт и соавт. ([2330], 2019) показали, что их система BANANAS может выполнять поиск нейронной архитектуры (NAS), прогнозируя точность сети в пределах

до 1% после обучения только на 200 случайных примерах архитектур. Зоф и Ле ([2453], 2016) использовали обучение с подкреплением для поиска в пространстве архитектур нейронных сетей. Реал и соавт. ([1864], 2018) использовали эволюционный алгоритм для выбора модели, Лю и соавт. ([1433], 2017) применили эволюционные алгоритмы на иерархических представлениях, а Йадерберг и соавт. ([1122], 2017) описали обучение на основе популяций. Лю и соавт. ([1430], 2019) ослабили пространство архитектур до непрерывного дифференцируемого пространства и использовали метод градиентного спуска для поиска локально оптимального решения. Фам и соавт. ([1789], 2018) описывают систему ENAS (*Efficient Neural Architecture Search*), выполняющую поиск оптимальных подграфов большого графа. Этот поиск выполняется быстро, поскольку не требует переобучения параметров. Идея поиска подграфа восходит к алгоритму "оптимального повреждения мозга", предложенного ЛеКуном и соавт. в работе [1371] (1990).

Несмотря на этот впечатляющий набор подходов, есть критики, полагающие, что это поле исследований еще не достигло зрелости. Ю и соавт. в работе [2415] (2019) показали, что в некоторых случаях эти алгоритмы NAS являются не более эффективными, чем случайный выбор архитектуры. Обзор последних результатов по проблеме поиска нейронной архитектуры представлен Элскеном и соавт. ([686], 2018).

Обучение без учителя представляет собой значительную область исследований в статистике, в основном под заголовком оценки плотности. Работы Сильвермана ([2066], 11986) и Мерфи ([1638], 2012) являются хорошими источниками по классическим и современным методам в этой области. Метод анализа основных компонентов (PCA) восходит к работе Пирсона ([1760] (1901); его название было предложено в независимой работе Хотеллинга [1070] (1933). В модели вероятностного PCA (Типпинг и Бишоп [2217], 1999) добавляется порождающая модель для самих главных компонент. Идея вариационного автокодировщика принадлежит Кингме и Веллингу ([1228], 2013), а в статье Резенде и соавт. [1875] (2014) дано введение в вариационные методы вывода в графических моделях.

Для моделей авторегрессии классическим учебником является книга Бокса и соавторов. Уравнение Юла–Уокера для подгонки моделей AR было независимо разработано Юлом ([2417], 1927) и Уокером ([2286], 1931). Авторегрессионные модели с нелинейными зависимостями были разработаны несколькими авторами (Фрей [782], 1998; Бенджио и Бенджио [172], 2001; Ларошель и Мюррей [1354], 2011). Модель авторегрессии WaveNet (Ван ден Оорд и др. [2251], 2016) была основана на более ранних работах по применению авторегрессии при генерации изображений (Ван ден Оорд и др. [2252], 2016). Генеративно-состязательные сети, или GAN, впервые были предложены Гудфеллоу и соавт. в работе [897] (2015) и нашли много применений в ИИ. Появляется некоторое теоретическое понимание их свойств, что способствует улучшению моделей GAN и их алгоритмов (Ли и Малик [1402], 2018; [1401], 2018; Чжу и др. [2440], 2019). Часть этого понимания включает защиту от атак противника (Карлини и др. [371], 2019).

Несколько направлений исследований нейронных сетей были популярны в прошлом, но сегодня они активно не исследуются. ▶**Сети Хопфилда** (Хопфилд [1058], 1982) имеют симметричные соединения между каждой парой узлов и могут научиться хранить шаблоны в ассоциативной памяти, так что весь шаблон может быть извлечен путем индексации в памяти с использованием фрагмента шаблона. Сети Хопфилда являются детерминированными; позже они были обобщены до стохастической ▶**машины Больцмана** (Хинтон и Сейновски [1027], 1983; [1028], 1986). Машины Больцмана, возможно, являются самым ранним примером глубокой порождающей модели. Трудности вывода в машинах Больцмана привели к успехам в разработке методов Монте-Карло и вариационных методов (см. раздел 13.4).

Исследования нейронных сетей для целей ИИ также в некоторой степени переплетены с исследованиями в области биологических нейронных сетей. Эти две области совпадали в 1940-х годах, а истоки идей сверточной сети и обучения с подкреплением можно проследить до исследований биологических систем. Однако в настоящее время новые идеи в глубоком обучении, как правило, основаны на чисто вычислительных или статистических интересах. В области ▶**вычислительной нейробиологии** основной целью является создание вычислительной модели, отражающей важные и специфические свойства реальных биологических систем. Обзоры по этой теме представлены Даяном и Эбботтом ([546], 2001) и Траппенбергом ([2226], 2010).

Для области нейронных сетей и глубокого обучения ведущими современными учебниками являются книги Гудфеллоу и соавт. ([895], 2016) и Чарняка ([399], 2018). Есть также много практических руководств, связанных с различными пакетами программного обеспечения с открытым исходным кодом для глубокого обучения. Три лидера в этой области — Ян ЛеКун, Джошуа Бенжио и Джефф Хинтон — в 2015 году представили ее ключевые идеи исследователям из других областей науки в своей знаменитой статье в журнале *Nature*, за что в 2018 году все трое были удостоены премии Тьюринга. В работе Шмидхубера [1996] (2015) дан общий обзор состояния дел в области глубокого обучения, а Денг и соавт. в статье [604] (2014) сосредоточились на задачах обработки сигналов.

Основными местами представления результатов исследований в области глубокого обучения являются конференции *Neural Information Processing Systems* (NeurIPS), *International Conference on Machine Learning* (ICML) и *International Conference on Learning Representations* (ICLR). Главными журналами являются *Machine Learning*, *Journal of Machine Learning Research* и *Neural Computation*. Все чаще из-за быстрых темпов исследований статьи появляются сначала на сайте `arXiv.org`, а также часто описываются в исследовательских блогах крупных исследовательских центров.

Упражнения

21.1. Постройте вручную нейронную сеть, вычисляющую функцию XOR для двух входов. Обязательно укажите, какие элементы были использованы при построении сети.

21.2. Простой перцептрон не может представлять функцию XOR (или, в общем случае, функцию четности его входов). Опишите, что происходит с весами персептрона с четырьмя входами с жестким порогом, начиная со значения 0,1, установленного для всех весов, как пример достижения функции четности.

21.3. Вспомните из главы 19, что существует 2^{2^n} различных булевых функций от n входов. Сколько из них можно представить перцептроном с жестким порогом?

21.4. Пусть имеется следующий набор обучающих примеров, каждый с шестью входами и одним целевым выходом.

Пример	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$	**T**
A_1	1	0	1	0	0	0	1
A_2	1	0	1	1	0	0	1
A_3	1	0	1	0	1	0	1
A_4	1	1	0	0	1	1	1
A_5	1	1	1	1	0	0	1
A_6	1	0	0	0	1	1	1
A_7	1	0	0	0	1	0	0
A_8	0	1	1	1	0	1	1
A_9	0	1	1	0	1	1	0
A_{10}	0	0	0	1	1	0	0
A_{11}	0	1	0	1	0	1	0
A_{12}	0	0	0	1	0	1	0
A_{13}	0	1	1	0	1	1	0
A_{14}	0	1	1	1	0	0	0

а) Примените правило обучения перцептрона к этим данным и приведите окончательные веса.

б) Примените к этим данным правило обучения дерева решений и приведите полученное дерево решений.

в) Прокомментируйте свои результаты.

21.5. В разделе 21.2.2 отмечалось, что выходные данные логистической функции можно интерпретировать как *вероятность* p, присвоенную моделью утверждению, что $f(\mathbf{x}) = 1$, — тогда вероятность того, что $f(\mathbf{x}) = 0$, будет равна $1 - p$. Запишите вероятность p как функцию от $\mathbf{x}$ и вычислите производную $\log p$ по каждому весу w_i. Повторите процесс для $\log(1 - p)$. Эти вычисления дают правило обучения для минимизации функции потерь отрицательного логарифмического

правдоподобия для вероятностной гипотезы. Прокомментируйте любое сходство с другими правилами обучения, приведенными в этой главе.

21.6. Допустим, имеется нейронная сеть с линейными функциями активации. Это означает, что для каждого ее элемента выход является некоторой константой c, умноженной на взвешенную сумму его входов.

а) Предположим, что сеть имеет один скрытый слой. При заданном наборе значений весов $\mathbf{w}$ запишите уравнения для значения элементов выходного слоя как функции от $\mathbf{w}$ и входного слоя $\mathbf{x}$ без какого-либо явного упоминания выходов скрытого слоя. Покажите, что может существовать сеть без скрытых элементов, которая будет вычислять ту же функцию.

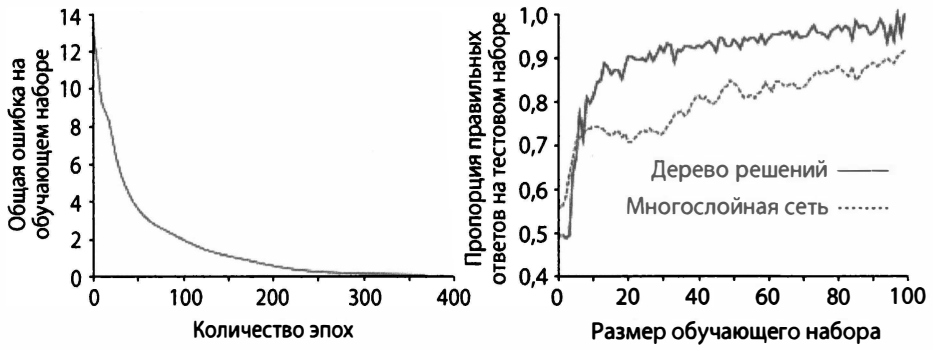
б) Повторите выкладки из п. *а* этого примера, но на этот раз для сети с любым количеством скрытых слоев.

в) Предположим, что сеть с одним скрытым слоем и линейными функциями активации имеет n входных и выходных узлов и h скрытых узлов. Какое влияние оказывает определенное в п. *а* этого примера преобразование в сеть без скрытых слоев на общее количество весов? Обсудите этот вопрос, в частности, для случая $h \ll n$.

21.7. Реализуйте структуру данных для многослойных нейронных сетей с прямой связью, не забыв предоставить информацию, необходимую как для прямой оценки, так и для обратного распространения. Используя эту структуру данных, напишите функцию NEURAL-NETWORK-OUTPUT, которая получает на входе пример и сеть и вычисляет соответствующие выходные значения.

21.8. Предположим, что обучающий набор данных содержит только один пример, повторенный 100 раз. В 80 из 100 случаев единственным выходным значением является 1, а в остальных 20 случаях — 0. Какой прогноз даст сеть с обратным распространением для этого примера, если предположить, что она была обучена и достигла глобального оптимума? (*Подсказка.* Чтобы найти глобальный оптимум, дифференцируйте функцию ошибок и установите ее равной нулю.)

21.9. Нейронная сеть, эффективность обучения которой представлена на рис. 21.10, имеет четыре скрытых узла. Это число было выбрано несколько произвольно. Используйте метод перекрестной проверки, чтобы найти лучшее количество скрытых узлов.



а) б)

Рис. 21.10. а) Кривая обучения демонстрирует постепенное сокращение ошибки по мере того, как веса модифицируются на серии последовательных эпох обучения сети на данных из примера об ожидании в ресторане из главы 19. (При обучении нейронной сети *эпохой* называют один полный проход по выборке обучения.) **б)** Сравнение кривых обучения показывает, что деревья решений обучаются на данных из примера об ожидании в ресторане несколько лучше, чем многослойная сеть с обратным распространением

ГЛАВА 22

Обучение с подкреплением

В этой главе обсуждается, как полученные агентом вознаграждения и наказания могут использоваться для его обучения максимизации вознаграждений в будущем.

При **обучении с учителем** агент учится, пассивно наблюдая за примерами пар ввода-вывода, предоставляемых ему "учителем". В этой главе будет показано, как агенты могут активно учиться на собственном опыте, без учителя, принимая во внимание только собственный конечный успех или неудачу.

22.1. Обучение посредством вознаграждения

Рассмотрим проблему обучения игре в шахматы. Представим, что этот процесс рассматривается как задача обучения с учителем с использованием методов, обсуждавшихся в главах 19–21. Функция играющего в шахматы агента принимает в качестве входных данных позицию на доске и возвращает его очередной ход, поэтому обучение данной функции осуществляется посредством предоставления ей примеров шахматных позиций, каждая из которых имеет метку в виде правильного хода. Далее, пусть в наличии имеется база данных, содержащая несколько миллионов гроссмейстерских игр, каждая из которых представлена как последовательность позиций и ходов. Ходы, сделанные победителем, за некоторыми исключениями, считаются хорошими, хотя и не всегда идеальными. Таким образом, у нас есть многообещающий обучающий набор. Проблема заключается в том, что в базе имеется сравнительно мало примеров (около 10^8) по отношению к пространству всех возможных шахматных позиций (около 10^{40}). В новой игре могут встретиться позиции, существенно отличающиеся от всех позиций, представленных в базе данных, и функция обученного агента в этом случае, скорее всего, выдаст неправильное решение — не в последнюю очередь потому, что она не имеет ни малейшего представления о том, что является конечной целью ее ходов (мат), и даже о том, как выполненный ход повлияет на позиции фигур. И конечно, шахматы — крошечная часть реального мира. Для решения более реалистичных задач

понадобится гораздо больше грандиозных баз данных, которых просто не существует.[1]

Альтернативой является ▶**обучение с подкреплением** (*reinforcement learning* — RL), при котором агент взаимодействует с миром и периодически получает **вознаграждения** (или, в терминологии психологии, **подкрепления**), которые отражают, насколько хорошо он действует. Например, в шахматах вознаграждение равно 1 за победу, 0 — за проигрыш и 1/2 — за ничью. Концепция вознаграждений уже упоминалась в главе 17 при обсуждении **марковских процессов принятия решений** (MDP). При обучении с подкреплением цель та же самая — максимизировать ожидаемую сумму вознаграждений. Однако обучение с подкреплением отличается от "просто решения задачи MDP" тем, что агент не *получает* MDP в качестве исходной задачи, которую ему предстоит решить, агент непосредственно *находится* в MDP. Ему может быть неизвестна модель перехода или функция вознаграждения, и он вынужден действовать, чтобы узнать больше. Представьте, что вы играете в новую игру, правила которой вам неизвестны. После ста или около того ходов рефери объявляет: "Вы проиграли". По сути, в этом и состоит основная концепция обучения с подкреплением.

С точки зрения разработчиков систем искусственного интеллекта, предоставить агенту сигнал вознаграждения обычно намного проще, чем предоставить множество помеченных примеров того, как следует себя вести. Во-первых, функция вознаграждения часто (как это имеет место в шахматах) очень лаконична и проста в определении: достаточно лишь нескольких строк кода, чтобы сообщить играющему в шахматы агенту, выиграл он или проиграл, или указать агенту, участвующему в автогонках, выиграл он ее, проиграл или попал в аварию и потерпел крах. Во-вторых, от разработчиков не требуется быть экспертами, способными предоставить информацию о правильном действии в любой ситуации, как в случае, когда применяется обучение с учителем.

Тем не менее оказывается, что получение даже минимального опыта может иметь большое значение в обучении с подкреплением. Два примера в предыдущем абзаце — вознаграждения за выигрыш/проигрыш в шахматах или автогонках — это то, что называют ▶**редкими вознаграждениями**, поскольку в подавляющем большинстве состояний агент вообще не получает какого-либо информативного сигнала о вознаграждении. В таких играх, как теннис или крикет, дополнительные вознаграждения легко можно предоставить за каждый выигранный балл или выполненную пробежку. В автомобильных гонках будет вполне разумно вознаграждать агента за продвижение по трассе в правильном направлении. При обучении ползанию достижением можно считать вообще любое продвижение вперед. Подобные промежуточные вознаграждения существенно облегчают обучение.

---

[1] Как отмечали Ян ЛеКун и Алеша Эфрос, "революция ИИ не будет происходить под наблюдением учителя".

Если есть возможность предоставлять агенту подходящий случаю сигнал вознаграждения, обучение с подкреплением можно считать универсальным способом построения систем искусственного интеллекта. Это особенно верно для систем *эмуляции*, в которых нет недостатка в возможностях получать опыт. Добавление глубокого обучения как инструмента в системы обучения с подкреплением также сделало возможным появление новых приложений, таких как обучение игре в видеоигры на приставке Atari на основе необработанного визуального ввода (Мних и др. [1603], 2013), управление роботами (Левин и др. [1394], 2016) и игра в покер (Браун и Сендхолм [320], 2017).

Были разработаны буквально сотни различных алгоритмов обучения с подкреплением, и многие из них могут использоваться как инструменты в широком спектре методов обучения из глав 19–21. В этой главе на основе нескольких примеров будут рассмотрены основные идеи и дано общее представление обо всем разнообразии возможных подходов. В целом возможные подходы можно классифицировать следующим образом.

- ►**Обучение с подкреплением на основе модели**. В этих подходах агент использует модель перехода среды как инструмент, помогающий интерпретировать сигналы вознаграждения и принимать решения о том, как действовать. Изначально эта модель может быть неизвестна, и тогда агент изучает ее посредством наблюдения результатов своих действий, либо она может быть уже определена, — например, программе игры в шахматы могут быть известны все правила этой игры, даже если она еще не обучена выбирать хорошие ходы. В частично наблюдаемых средах модель перехода также будет полезна для **оценки состояния** (см. главу 14). Системы обучения с подкреплением на основе модели часто изучают функцию полезности $U(s)$, определенную (как описывается в главе 17) в терминах суммы вознаграждений, начиная от состояния s и последующих.[2]
- **Обучение с подкреплением без модели**. В этих подходах агент изначально не знает и не изучает модель перехода для окружающей среды. Вместо этого он учится непосредственно представлению о том, как себя вести. Подобный подход возможен в двух вариантах.
  - ►**Изучение полезности действий**. Функция полезности действий была представлена в главе 17. Наиболее распространенной формой изучения полезности действий является ►**Q-обучение**, когда агент изучает ►**Q-функцию** или функцию ожидаемой полезности действия $Q(s, a)$, определяющую сумму вознаграждений от состояния s и далее, если будет выполнено действие a. При известной Q-функции агент может выбрать,

[2] В литературе по обучению с подкреплением, ориентированной в большей степени на исследование операций, а не на экономику, функцию полезности часто называют **функцией ценности** и обозначают как $V(s)$.

что ему делать в состоянии s, посредством поиска действия с самым высоким значением ожидаемой полезности Q.

- ▶**Поиск стратегии**. Агент изучает стратегию $\pi(s)$, непосредственно отображающую состояния на действия. В терминологии главы 2 это **рефлекторный агент**.

В разделе 22.2 описывается ▶**пассивное обучение с подкреплением**, при котором стратегия агента остается неизменной, а задача состоит в том, чтобы с помощью обучения определить полезности состояний (или пар "состояние–действие"). Здесь также может потребоваться изучение модели окружающей среды. (Для этого раздела важно понимание марковского процесса принятия решений, подробно рассматривавшегося в главе 17.) В разделе 22.3 рассматривается ▶**активное обучение с подкреплением**, когда агент также должен определить, что ему следует делать. Здесь принципиальной задачей является **исследование**: агент должен экспериментировать, насколько это возможно, изучая окружающую среду с целью определения, как в ней следует действовать. В разделе 22.4 обсуждается, как агент может использовать индуктивное обучение (включая методы глубокого обучения) для ускорения обучения на собственном опыте. Здесь также анализируются другие подходы, способные помочь масштабировать методы обучения с подкреплением для решения реальных проблем, включая предоставление промежуточных псевдовознаграждений с целью управления обучением и организации поведения агента как иерархии действий. В разделе 22.5 рассматриваются методы поиска стратегий. В разделе 22.6 исследуется ▶**обучение на демонстрации** (*apprenticeship learning*) или **ученичество**: обучение агента на основе демонстраций поведения, а не сигналов вознаграждения. Наконец в разделе 22.7 приводится информация о практическом применении методов обучения с подкреплением.

22.2. Пассивное обучение с подкреплением

Начнем с рассмотрения самого простого случая полностью наблюдаемой среды с небольшим количеством действий и состояний, в которой агент уже имеет фиксированную стратегию $\pi(s)$, определяющую его действия. Агент пытается изучить функцию полезности $U^{\pi}(s)$ — ожидаемого общего обесцениваемого вознаграждения, если стратегия π выполняется, начиная с состояния s. Назовем этот вариант ▶**пассивным обучающимся агентом**.

Задача пассивного обучения аналогична задаче **оценки стратегии**, являющейся частью алгоритма итерации по политикам, описанного в разделе 17.2.2. Разница лишь в том, что пассивный обучающийся агент не знает модели перехода $P(s' \mid s, a)$, определяющей вероятность достижения состояния s' из состояния s после выполнения действия a, а также функции вознаграждения $R(s, a, s')$, определяющей его вознаграждение за каждый переход.

В качестве примера будем использовать клеточный мир 4×3, представленный в главе 17. На рис. 22.1 приведены оптимальная стратегия для этого мира и соответствующие полезности. Агент выполняет в окружающей среде ряд ►**попыток** (*trial*), используя свою стратегию π. При каждой попытке агент начинает с состояния (1,1) и испытывает некоторую последовательность переходов между состояниями до тех пор, пока не достигнет одного из терминальных состояний, (4,2) или (4,3). Его восприятие обеспечивает ему сведения как о текущем состоянии, так и о вознаграждении, полученном за только что выполненный переход с целью достижения этого состояния. Типичные попытки могут выглядеть следующим образом.

$$(1,1) \xrightarrow[Up]{-0{,}04} (1,2) \xrightarrow[Up]{-0{,}04} (1,3) \xrightarrow[Right]{-0{,}04} (1,2) \xrightarrow[Up]{-0{,}04} (1,3) \xrightarrow[Right]{-0{,}04} (2,3) \xrightarrow[Right]{-0{,}04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

$$(1,1) \xrightarrow[Up]{-0{,}04} (1,2) \xrightarrow[Up]{-0{,}04} (1,3) \xrightarrow[Right]{-0{,}04} (2,3) \xrightarrow[Right]{-0{,}04} (3,3) \xrightarrow[Right]{-0{,}04} (3,2) \xrightarrow[Up]{-0{,}04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

$$(1,1) \xrightarrow[Up]{-0{,}04} (1,2) \xrightarrow[Up]{-0{,}04} (1,3) \xrightarrow[Right]{-0{,}04} (2,3) \xrightarrow[Right]{-0{,}04} (3,3) \xrightarrow[Right]{-0{,}04} (3,2) \xrightarrow[Up]{-1} (4,2)$$

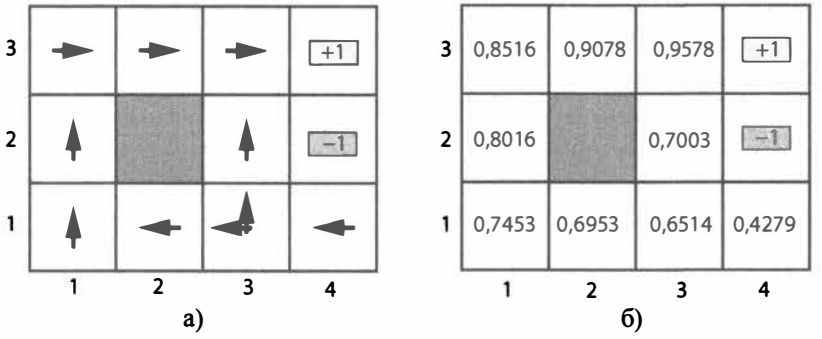


Рис. 22.1. а) Оптимальные стратегии для стохастической среды при вознаграждении $R(s, a, s') = -0{,}04$ для переходов между нетерминальными состояниями. Здесь имеются две оптимальные стратегии, потому что в состоянии (3,1) как переход *Left*, так и переход *Up* являются оптимальными, — это уже было показано ранее, на рис. 17.2. **б)** Полезности состояний клеточного мира 4×3 при заданной стратегии π

Обратите внимание, что каждый переход отмечается как выполненным действием, так и вознаграждением, полученным в следующем состоянии. Цель состоит в том, чтобы использовать эту информацию о вознаграждении для изучения ожидаемой полезности $U^{\pi}(s)$, связанной с каждым нетерминальным состоянием s. Полезность здесь определяется как ожидаемая сумма (обесцениваемых) вознаграждений,

полученных при соблюдении стратегии π. Как и в уравнении (17.2) в разделе 17.1.2, это соотношение записывается следующим образом:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right], \tag{22.1}$$

где $R(S_t, \pi(S_t), S_{t+1})$ — вознаграждение, получаемое, когда действие $\pi(S_t)$ выполняется в состоянии S_t и достигается состояние S_{t+1}. Обратите внимание, что S_t является случайной величиной, обозначающей состояние, достигнутое в момент времени t при выполнении стратегии π, начиная с состояния $S_0 = s$. Здесь и далее во все уравнения будет включен **коэффициент обесценивания** γ, но для нашего клеточного мира 4×3 будет принято значение $\gamma = 1$, что означает отсутствие обесценивания.

22.2.1. Непосредственная оценка полезности

Идея метода ▶**непосредственной оценки полезности** состоит в том, что полезность состояния определяется как ожидаемое суммарное вознаграждение от этого состояния и далее (называемое ожидаемым ▶**вознаграждением за всю траекторию** (*reward-to-go*)) и что каждая попытка предоставляет *выборку* этого значения для каждого посещенного состояния. Например, первая попытка из трех приведенных выше предоставляет одну выборку с суммарным вознаграждением 0,76 для состояния (1,1), две выборки со значениями 0,8 и 0,88 — для состояния (1,2), две выборки со значениями 0,84 и 0,92 — для состояния (1,3) и т.д. Таким образом, в конце каждой последовательности алгоритм вычисляет наблюдаемое вознаграждение за всю траекторию для каждого состояния и соответственно обновляет оценку полезности для этого состояния, просто вычисляя скользящее среднее для каждого состояния в таблице. В пределе, после выполнения бесконечного количества попыток, среднее по выборкам сходится к значению истинного ожидания, приведенному в уравнении (22.1).

Это означает, что обучение с подкреплением фактически сводится к стандартной задаче обучения с учителем, в которой каждый пример представлен парой (*state*, *reward-to-go*). Имеется много мощных алгоритмов обучения с учителем, поэтому данный подход кажется многообещающим, однако в нем игнорируется важное ограничение: ➨*полезность состояния определяется его собственным вознаграждением и ожидаемой полезностью состояний-преемников*. Говоря конкретнее, значения полезности подчиняются уравнениям Беллмана для данной конкретной стратегии (см. также уравнение (17.14)):

$$U_i(s) = \sum_{s'} P(s' \mid s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]. \tag{22.2}$$

Поскольку в методе непосредственной оценки полезности связи между состояниями игнорируются, он не позволяет воспользоваться некоторыми возможностями для обучения. Например, во второй из трех попыток, приведенных выше, достигается состояние (3,2), которое еще не было до сих пор посещено агентом. После следующего перехода агент достигает состояния (3,3), которое, как известно из первой попытки, имеет высокую полезность. Уравнение Беллмана позволяет сразу же определить, что состояние (3,2) также, по-видимому, будет иметь высокую полезность, поскольку оно ведет к состоянию (3,3), но метод непосредственной оценки полезности не позволяет что-либо определить с точки зрения обучения до конца данной попытки. В более широком контексте метод непосредственной оценки полезности можно рассматривать как поиск функции полезности U в пространстве гипотез, размеры которого намного больше, чем это необходимо, поскольку оно также включает большое количество функций, нарушающих уравнения Беллмана. По этой причине данный алгоритм часто сходится очень медленно.

22.2.2. Адаптивное динамическое программирование

Агент, действующий по принципу ▶**адаптивного динамического программирования** (*adaptive dynamic programming* — ADP), использует преимущества наличия ограничений между полезностями состояний путем изучения связывающей их модели перехода и соответствующего марковского процесса принятия решений с применением динамического программирования. Для пассивно обучающегося агента это означает, что он должен подставлять полученную с помощью обучения модель перехода $P(s' \mid s, \pi(s))$ и наблюдаемые вознаграждения $R(s, \pi(s), s')$ в уравнение Беллмана (22.2) для вычисления полезности состояний. Как было отмечено в главе 17 при обсуждении метода итерации по стратегиям, эти уравнения будут линейными, если стратегия π фиксирована, поэтому их можно будет решить с помощью любого пакета линейной алгебры.

Как альтернативный вариант можно принять подход **модифицированной итерации по стратегиям** (см. раздел 17.2.2), при котором упрощенный процесс итерации по значениям используется для обновления оценок полезностей после каждого изменения в изучаемой модели. Поскольку после каждого наблюдения в модель обычно вносятся лишь незначительные изменения, в процессе итерации по значениям в качестве начальных значений могут использоваться предыдущие оценки полезностей, а сам процесс, как правило, сходится очень быстро.

Изучить модель перехода просто, поскольку среда полностью наблюдаема. Это означает, что речь идет о задаче обучения с учителем, в которой входом для каждого обучающего примера является пара "состояние–действие" (s, a), а выходом является результирующее состояние s'. Модель перехода $P(s' \mid s, a)$ представляется в виде таблицы и оценивается непосредственно по частоте, с которой состояние s' достигается при выполнении действия a в состоянии s, определяемой как значение

счетчика $N_{s'|s,a}$. Например, в трех попытках, приведенных в разделе 22.2, действие *Right* в состоянии (3,3) выполняется четыре раза, а результирующим состоянием в двух случаях становится состояние (3,2) и еще в двух случаях — состояние (4,2), поэтому вероятности перехода $P((3,2) \mid (3,3), Right)$ и $P((4,3) \mid (3,3), Right)$ равны и обе оцениваются как 1/2.

Полная программа агента для пассивного агента ADP приведена на рис. 22.2. Показатели производительности этой программы для клеточного мира 4×3 представлены на рис. 22.3. Судя по тому, как быстро улучшаются полученные агентом ADP оценки значений, его ограничивает лишь собственная способность изучать модель перехода. В этом смысле данный агент может служить эталоном, с которым можно будет сравнивать производительность других алгоритмов обучения с подкреплением. Однако в очень больших пространствах состояний этот подход становится неосуществимым. Например, в нардах для его использования потребуется решить приблизительно 10^{20} уравнений с 10^{20} неизвестными.

---

function PASSIVE-ADP-LEARNER(*percept*) **returns** действие a
  inputs: *percept*, результаты восприятия, обозначающие текущее состояние s' и сигнал вознаграждения r
  persistent: π, фиксированная стратегия
    mdp, марковский процесс принятия решений с моделью P, вознаграждениями R, действиями A, коэффициентом обесценивания γ
    U, таблица полезностей состояний, первоначально пустая
    $N_{s'|s,a}$, таблица векторов счетчиков результата, индексированная по состоянию и действию, первоначально содержащая нули
    s, a, предыдущие состояние и действие, первоначально пустые

  if s' является новым **then** $U[s'] \leftarrow 0$
  if s не пусто **then**
    увеличить значения $N_{s'|s,a}[s, a][s']$
    $R[s, a, s'] \leftarrow r$
    добавить a к $A[s]$
    $\mathbf{P}(\cdot \mid s, a) \leftarrow$ NORMALIZE($N_{s'|s,a}[s, a]$)
    $U \leftarrow$ POLICYEVALUATION(π, U, *mdp*)
    $s, a \leftarrow s', \pi[s']$
    return a

Рис. 22.2. Программа пассивного агента обучения с подкреплением, основанная на адаптивном динамическом программировании. Агент выбирает значение коэффициента обесценивания γ, а затем последовательно вычисляет значения P и R марковского процесса принятия решений. Функция POLICY-EVALUATION решает уравнения Беллмана для фиксированной стратегии, как описано в разделе 17.2.2

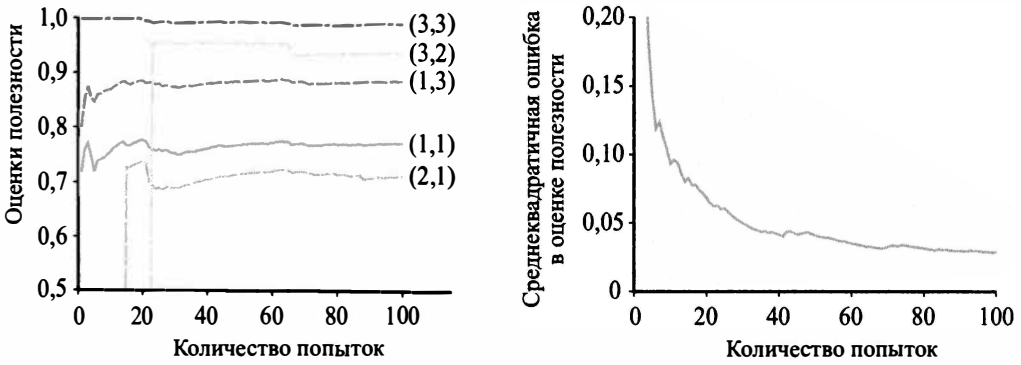


Рис. 22.3. Кривые пассивного обучения ADP для клеточного мира 4 × 3, полученные при использовании оптимальной стратегии, приведенной на рис. 22.1, *а*. **а)** Оценки полезности для выбранного подмножества состояний как функции от количества попыток. Обратите внимание, что потребовалось 14 и 23 попытки соответственно, прежде чем для редко посещаемых состояний (2,1) и (3,2) было "обнаружено", что из них есть доступ к выходу +1 в состоянии (4,3). **б)** Среднеквадратичная ошибка (см. приложение А) оценки значения $U(1,1)$, усредненная по 50 прогонам, включающим по 100 попыток

22.2.3. Обучение с учетом временной разности

Способ решения лежащего в основе марковского процесса принятия решений, предложенный в предыдущем разделе, не является единственным возможным способом применения уравнений Беллмана для решения задачи обучения. Другой способ состоит в использовании наблюдаемых переходов для настройки полезности наблюдаемых состояний так, чтобы они соответствовали уравнениям ограничений. Рассмотрим, например, переход от состояния (1,3) к состоянию (2,3) во второй попытке, представленной выше, в разделе 22.2. Предположим, что по результатам первой попытки оценки полезности составляли $U^{\pi}(1,3) = 0{,}88$ и $U^{\pi}(2,3) = 0{,}96$. Теперь, если этот переход от состояния (1,3) к состоянию (2,3) будет происходить постоянно, можно ожидать, что полезности будут подчиняться уравнению

$$U^{\pi}(1,3) = -0{,}04 + U^{\pi}(2,3),$$

поэтому значение полезности $U^{\pi}(1,3)$ будет равно 0,92. Следовательно, ее текущая оценка 0,84 может быть немного заниженной и должна быть увеличена. В общем случае, если происходит переход из состояния s в состояние s' посредством выполнения действия $\pi(s)$, к значению полезности $U^{\pi}(s)$ применяется следующее обновление:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha[R(s, \pi(s), s') + \gamma U^{\pi}(s') - U^{\pi}(s)]. \quad (22.3)$$

Здесь α — параметр **скорости обучения**. Поскольку в этом правиле обновления присутствует разность между полезностями последовательных состояний (а значит, последовательных моментов времени), его часто называют уравнением ▶**временной разности** (*temporal-difference* — TD). Так же, как и в правилах обновления веса из главы 19 (например, уравнение (19.6) в разделе 19.6.3), TD-член $R(s, \pi(s), s') + \gamma U^{\pi}(s') - U^{\pi}(s)$ здесь фактически является сигналом ошибки, а обновление предназначено для уменьшения этой ошибки.

Все методы временной разности работают путем корректировки оценок полезности в сторону идеального равновесия, которое выполняется локально, когда оценки полезности верны. В случае пассивного обучения равновесие задается уравнением (22.2). Теперь уравнение (22.3), по сути, вынуждает агента достичь равновесия, заданного уравнением (22.2), но с учетом некоторых нюансов. Прежде всего, обратите внимание, что обновление включает в себя только наблюдаемое состояние-преемник s', тогда как действительные условия равновесия включают все возможные последующие состояния. Можно полагать, что это приведет к слишком большому изменению $U^{\pi}(s)$ в случае очень редкого перехода, но на самом деле, поскольку редкие переходы происходят очень редко, *среднее значение* $U^{\pi}(s)$ в пределе будет сходиться к правильной величине, даже если само значение будет продолжать колебаться.

Более того, если заменить постоянный параметр α функцией, которая будет уменьшаться по мере увеличения количества посещений состояния, как показано на рис. 22.4, то само значение $U^{\pi}(s)$ будет сходиться к правильному значению.[3] На рис. 22.5 иллюстрируется работа пассивного агента TD в клеточном мире 4 × 3. Он учится не так быстро, как агент ADP, и демонстрирует заметно более высокую изменчивость, но этот метод намного проще и требует гораздо меньше вычислений на одно наблюдение. Обратите внимание, что метод временной разности ▶*не нуждается в модели перехода для выполнения обновлений.* Окружающая среда сама обеспечивает агенту информацию о связи между соседними состояниями в виде наблюдаемых переходов.

Подходы на основе метода адаптивного динамического программирования (ADP) и на основе метода временных разностей (TD) тесно связаны. В обоих случаях предпринимаются попытки внести локальные корректировки в оценки полезностей с целью обеспечения "согласования" каждого состояния с его преемниками. Одно из различий между ними состоит в том, что в методе TD состояние корректируется для согласования с его *наблюдаемым* преемником (уравнение (22.3)), а в методе ADP состояние корректируется для согласования со *всеми* преемниками, которые могут быть получены с учетом их вероятностей (уравнение (22.2)).

[3] Необходимые условия приведены в разделе 19.6.4. На рис. 22.5 использовалась зависимость $\alpha(n) = 60/(59 + n)$, удовлетворяющая этим условиям.

```
function PASSIVE-TD-LEARNER(percept) returns действие
  inputs: percept, результаты восприятия, определяющие текущее
                   состояние s' и сигнал вознаграждения r
  persistent: π, фиксированная стратегия
              s, предыдущее состояние, первоначально пустое
              U, таблица значений полезностей, первоначально пустая
              N_s, таблица частот состояний, первоначально содержащая нули

  if s' является новым then U[s'] ← 0
  if s не пусто then
    увеличить значение N_s[s]
    U[s] ← U[s] + α(N_s[s]) × (r + γU[s'] – U[s])
  s ← s'
  return π[s']
```

Рис. 22.4. Программа пассивного агента обучения с подкреплением, изучающего оценки полезности с использованием временных разностей. Функция размера шага $\alpha(n)$ выбирается с учетом условия обеспечения сходимости

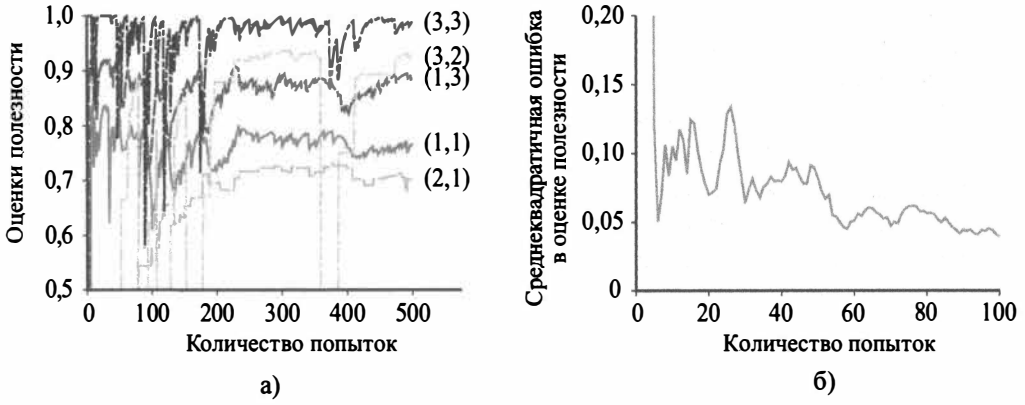


Рис. 22.5. Кривые обучения методом временных разностей для клеточного мира 4 × 3. **а)** Оценки полезности для выбранного подмножества состояний, как функции от количества попыток для единственного прогона из 500 попыток. Сравните с результатами прогона в 100 попыток, представленными на рис. 22.3, *а*. **б)** Среднеквадратичная ошибка оценки для U(1,1), усредненная по 50 прогонам, состоящим из 100 попыток каждый

Это различие исчезает, когда результаты корректировок в методе TD усредняются по большому количеству переходов, поскольку частота появления каждого преемника в общем множестве переходов приблизительно пропорциональна его вероятности. Более важное различие состоит в том, что в методе TD отдельная корректировка выполняется в расчете на каждый наблюдаемый переход, а в методе ADP выполняется столько корректировок, сколько необходимо для восстановления согласованности между оценками полезностей U и моделью переходов P. Хотя наблюдаемый переход вносит в модель переходов P только локальное изменение, его результаты могут потребовать распространения по всем полезностям U. Таким образом, метод временных разностей TD может рассматриваться как грубое, но эффективное первое приближение к методу адаптивного динамического программирования ADP.

Каждая корректировка, внесенная согласно методу ADP, с точки зрения метода TD может рассматриваться как результат ►**"псевдоэксперимента"**, полученный путем моделирования в текущей модели перехода. Подход, предусмотренный в методе TD, можно расширить с целью использования модели перехода для выработки результатов нескольких псевдоэкспериментов — переходов, возможность осуществления которых агент TD *может* допустить согласно его текущей модели. В ответ на каждый наблюдаемый переход агент TD может вырабатывать большое количество воображаемых переходов. Благодаря этому результирующие оценки полезностей будут все точнее и точнее аппроксимировать соответствующие оценки для метода ADP, — разумеется, за счет увеличения времени вычислений.

Аналогичным образом могут создаваться все более эффективные версии метода ADP — путем непосредственной аппроксимации алгоритмов для итерации по значениям или итерации по стратегиям. Хотя алгоритм итерации по значениям весьма эффективен, он будет невыполним при наличии, скажем, 10^{100} состояний. Однако многие из необходимых корректировок значений состояний на каждой итерации будут исключительно малы. Одним из возможных подходов, применимых для быстрого получения достаточно качественных ответов, является ограничение количества корректировок, внесенных после каждого наблюдаемого перехода. Можно также использовать некоторую эвристику для ранжирования возможных корректировок, с тем чтобы в дальнейшем осуществлять только наиболее значимые из них. Так, эвристика ►**приоритетной прогонки** (*prioritized sweeping*) позволяет реализовать внесение корректировок только в состояния, *возможные* преемники которых уже подвергались *большим* корректировкам в их собственных оценках полезностей.

Приближенные алгоритмы ADP, в которых используются подобные эвристики, обычно обеспечивают обучение примерно с таким же быстродействием, как и полные алгоритмы ADP (с точки зрения количества обучающих последовательностей), но могут оказаться на несколько порядков более эффективными с точки зрения объема вычислений (см. упражнение 22.3.) Это позволяет применять такие

алгоритмы для обработки пространств состояний, которые слишком велики для полного алгоритма ADP. Приближенные алгоритмы ADP имеют еще одно важное преимущество: на ранних этапах обучения в новой среде модель перехода P часто будет далека от правильной, поэтому нет смысла слишком точно вычислять функцию полезности для согласования с ней. Приближенный алгоритм позволяет использовать минимальный объем корректировок, который уменьшается по мере того, как модель переходов становится все более точной. Это позволяет исключить продолжительные итерации по значениям, которые могут возникать на ранних этапах обучения из-за больших изменений в модели.

22.3. Активное обучение с подкреплением

Пассивный обучающийся агент руководствуется фиксированной стратегией, определяющей его поведение, тогда как ▶**активный обучающийся агент** должен сам принимать решение о том, какие действия ему следует предпринять. Начнем с агента, действующего по принципу адаптивного динамического программирования (ADP), и рассмотрим, какие изменения необходимо внести в его проект, чтобы он мог функционировать с учетом этой новой степени свободы.

Прежде всего агенту потребуется изучить полную модель перехода с вероятностями результатов для *всех* действий, а не просто модель для заданной стратегии. Для этой цели отлично подходит простой механизм обучения, используемый в алгоритме PASSIVE-ADP-AGENT. Далее необходимо принять в расчет тот факт, что агент должен осуществлять выбор из целого ряда возможных действий. Полезности, которые ему потребуются для обучения, определяются *оптимальной* стратегией; они подчиняются уравнениям Беллмана, которые для удобства еще раз приведены ниже:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma U(s')]. \qquad (22.4)$$

Эти уравнения могут быть решены для получения функции полезности U с помощью алгоритмов итерации по значениям или итерации по стратегиям, приведенных в главе 17.

Последняя задача состоит в определении того, что делать на каждом этапе. Получив функцию полезности U, оптимальную для изучаемой модели, агент может извлечь информацию об оптимальном действии, составив одношаговый прогноз для максимизации ожидаемой полезности. Возможен альтернативный вариант, когда используется итерация по стратегиям: оптимальная стратегия уже известна, поэтому агенту достаточно просто выполнить действие, рекомендуемое оптимальной стратегией. Но следует ли ему выполнить его?

22.3.1. Исследование среды

На рис. 22.6, *а* показаны результаты одной последовательности попыток для агента ADP, который следует рекомендациям оптимальной стратегии для изучаемой модели на каждом этапе. На деле агент так и *не смог* изучить истинные полезности или истинную оптимальную стратегию! Вместо этого на 3-й попытке он находит стратегию, позволяющую достичь вознаграждения +1 вдоль нижнего маршрута, проходящего через состояния (2,1), (3,1), (3,2) и (3,3), как показано на рис. 22.6, *б*. После экспериментов с незначительными изменениями, начиная с 8-й попытки, агент постоянно придерживается этой стратегии, так и не изучив полезностей других состояний и не обнаружив оптимальный маршрут через состояния (1,2), (1,3) и (2,3). Такого агента, действующего на основании жадного алгоритма, называют ▶**жадным агентом**, поскольку он жадным образом выбирает то действие, которое в настоящий момент считает оптимальным на каждом этапе. Иногда жадность окупается, и агент в конечном счете находит оптимальную стратегию, но чаще всего этого не случается.

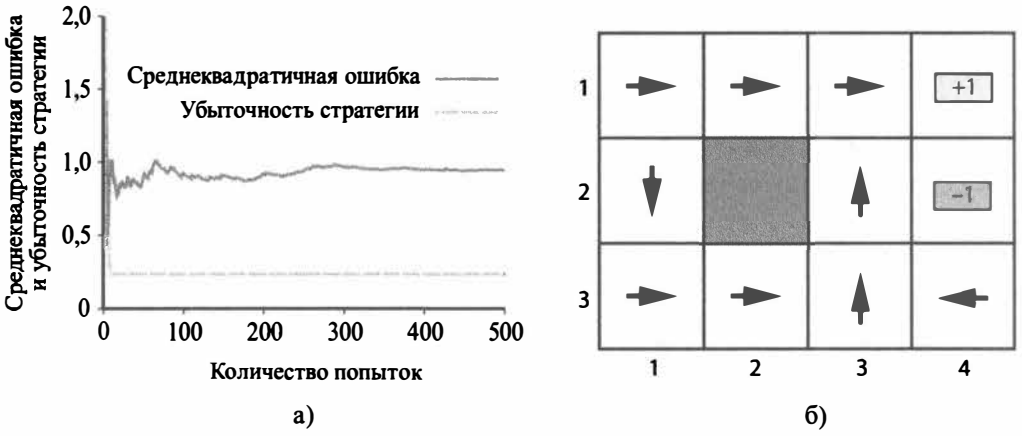


Рис. 22.6. Производительность жадного агента ADP, всегда выполняющего действие, рекомендованное оптимальной стратегией для изучаемой модели. **а)** Среднеквадратичная ошибка в оценках полезностей, усредненная по всем девяти нетерминальным квадратам, и убыточность стратегии в состоянии (1,1). Можно видеть, что стратегия быстро сходится после всего восьми испытаний к субоптимальной стратегии с потерей 0,235. **б)** Неоптимальная стратегия, на которой завершился поиск стратегии, выполняемый жадным агентом в данной конкретной последовательности попыток. Обратите внимание на действие *Down* в состоянии (1,2)

Как могло получиться, что выбор оптимального действия приводит к неоптимальным результатам? Ответ состоит в том, что изучаемая модель не является в

точности такой же, как истинная среда, а значит, то, что оптимально в модели на данном этапе обучения, может оказаться неоптимальным в истинной среде. К сожалению, агент не знает того, какова истинная среда, поэтому не может вычислить оптимальное действие для истинной среды. Так что же делать?

В проекте жадного агента не учтено, что действия не только предоставляют *вознаграждения*, они также предоставляют *информацию* в форме восприятий в результирующих состояниях. Как было показано для **задач о бандитах** в разделе 17.3, агент должен найти компромисс между **эксплуатацией** текущего лучшего действия с целью максимизации своего краткосрочного вознаграждения и **исследованием** пока неизвестных состояний с целью получения информации, которая может привести к изменению стратегии (и к большему вознаграждению в будущем). В реальном мире постоянно приходится выбирать между продолжением удобного существования и прыжком в неизвестность в надежде на лучшую жизнь.

Хотя задачи о бандитах с трудом поддаются точному решению для получения *оптимальной* схемы исследования среды, можно найти приемлемую схему, которая в конечном итоге позволит обнаружить оптимальную стратегию, даже если на это может потребоваться больше времени, чем кажется оптимальным. Любая такая схема не должна быть жадной с точки зрения ближайшего следующего перехода, но должна быть "жадной в пределе бесконечного исследования" (*greedy in the limit of infinite exploration* — ►**GLIE**). Схема GLIE должна опробовать каждое действие в каждом состоянии неограниченное количество раз, чтобы избежать в конце вероятности того, что оптимальное действие пропущено. Агент ADP, использующий такую схему, в конечном итоге изучит истинную модель перехода, а затем сможет успешно действовать в процессе эксплуатации.

Предложено несколько схем GLIE; в одной из простейших предусматривается заставить агента выбирать случайное действие на этапе t с вероятностью $1/t$, а в противном случае — придерживаться жадной стратегии. Хотя такая схема в конечном итоге сходится к оптимальной стратегии, процесс может оказаться слишком медленным. Лучший подход состоит в присваивании определенных весов тем действиям, которые агент пока проверял не очень часто, одновременно стараясь избегать тех действий, которые считаются имеющими низкую полезность (как это делалось при поиске по дереву методом Монте-Карло в разделе 5.4). Такой подход может быть реализован путем модификации уравнения ограничения (22.4) таким образом, чтобы в нем присваивалась более высокая оценка полезности относительно мало исследованным парам "состояние–действие".

По сути, это сводится к определению оптимистического распределения априорных вероятностей по возможным вариантам среды и вынуждает агента на первых порах вести себя так, как если бы повсюду были разбросаны замечательные вознаграждения. Будем использовать нотацию $U^+(s)$ для обозначения оптимистической оценки полезности состояния s (т.е. ожидаемого будущего вознаграждения по всей траектории) и нотацию $N(a, s)$ для количества попыток опробования

действия a в состоянии s. Предположим также, что в обучающемся агенте ADP используется метод итерации по значениям, — тогда необходимо переписать уравнение обновления (уравнение (17.10) из раздела 17.2.1), чтобы включить в него оптимистическую оценку:

$$U^+(s) \leftarrow \max_a f\left(\sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma U^+(s')],\ N(s, a)\right). \qquad (22.5)$$

Здесь f — это ▶**функция исследования**. Функция $f(u, n)$ определяет компромисс между жадностью (предпочтением, отданным высоким значениям полезности u) и любопытством (предпочтением, отданным низким значениям n, характеризующим действия, которые еще не были опробованы достаточно часто). Иначе говоря, функция $f(u, n)$ должна увеличиваться в зависимости от u и уменьшаться в зависимости от n. Безусловно, что существует много возможных функций, которые соответствуют этим условиям. Одним из довольно простых определений такой функции является следующее:

$$f(u, n) = \begin{cases} R^+ & \text{если n} < N_e \\ u & \text{в противном случае} \end{cases}$$

Здесь R^+ — оптимистическая оценка наилучшего возможного вознаграждения, которое может быть получено в любом состоянии, а N_e — постоянный параметр. Результатом применения такой функции становится то, что агент пытается проверить каждую пару "состояние–действие" по меньшей мере N_e раз. Тот факт, что в правой части уравнения (22.5) присутствует величина U^+, а не U, очень важен. По мере развития процесса исследования в большом количестве попыток будут опробованы состояния и действия, находящиеся недалеко от начального состояния. Если бы использовалась более пессимистическая оценка полезности U, то вскоре агент утратил бы склонность к проведению дальнейших исследований среды. Но применение оценки U^+ означает, что выгоды от исследования среды распространяются в обратном направлении от границ неисследованных регионов, поэтому получают больший вес действия, *ведущие* к неисследованным регионам, а не просто те действия, которые сами по себе остаются малоизученными.

Эффект использования такой исследовательской стратегии наглядно представлен на рис. 22.7, *б*, демонстрирующем ее быструю сходимость к стратегии с нулевыми потерями в отличие от жадного подхода. Оптимальная стратегия найдена уже после всего лишь 18 попыток. Обратите внимание, что ошибка в оценке полезности не сходится так же быстро. Это связано с тем, что агент довольно рано прекращает исследование частей пространства состояний, не обещающих вознаграждения, и в дальнейшем посещает их только "по случаю". Но благодаря такому подходу агент приобретает идеальное понимание того, что не следует задумываться о точных значениях полезностей состояний, которые, как ему известно,

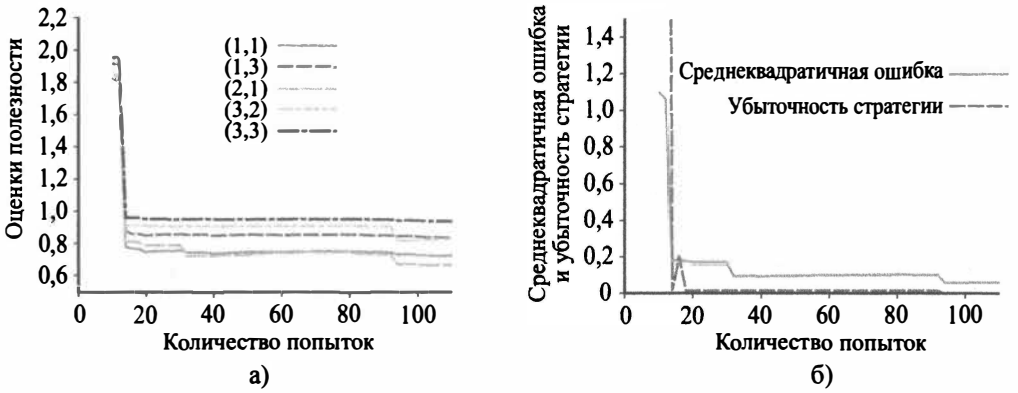


Рис. 22.7. Производительность агента ADP, исследующего среду с использованием параметров $R^+ = 2$ и $N_e = 5$. **а)** Изменение оценки полезностей для выбранных состояний во времени. **б)** Среднеквадратичная ошибка в значениях полезностей и связанная с ней убыточность стратегии

являются нежелательными и которых можно избежать. Падая со скалы, нет смысла выяснять, какую радиостанцию было бы лучше всего послушать.

22.3.2. Безопасное исследование

До сих пор предполагалось, что агенту ничто не мешает проводить исследования — он может действовать по своему усмотрению, при этом любые отрицательные вознаграждения ведут только к улучшению его модели мира. Иначе говоря, если агент играл в шахматы и проиграл, это не причинило ему ущерба (кроме, возможно, ущемленной гордости), а то, что он узнал, поможет ему играть лучше в следующей игре. Аналогично в среде эмулятора беспилотного автомобиля можно безопасно исследовать пределы возможностей автомашины, и любые аварии дают агенту лишь больше информации. Если машина разбивается, достаточно просто нажать кнопку сброса.

К сожалению, реальный мир менее склонен к прощению. Если вы — рыба-луна, то вероятность вашего выживания до зрелости составляет лишь около 0,00000001. Многие действия являются ►**необратимыми** в том смысле, как это было определено в разделе 4.5 для агентов, действующих в оперативном режиме: если было предпринято необратимое действие, никакая последовательность действий не позволит вернуться к тому состоянию, которое имело место до этого момента. В худшем случае агент входит в ►**поглощающее состояние**, в котором любые действия не оказывают никакого эффекта и никаких вознаграждений получено не будет.

Во многих практических ситуациях нельзя позволить агентам предпринимать необратимые действия или входить в поглощающие состояния. Например, агент, обучающийся вождению на реальной машине, должен избегать действий, которые могут привести к любой из следующих ситуаций:

- переход в состояния с большим отрицательным вознаграждением, такие как серьезные аварии;
- переход в состояния, из которых нет выхода, такие как въезд автомобиля в глубокую канаву;
- переход в состояния, которые надолго (или даже навсегда) ограничат размер будущих вознаграждений, такие как повреждение двигателя автомобиля, в результате чего его максимальная скорость уменьшится.

Оказаться в плохом состоянии агент может либо потому, что модель ему *неизвестна* и он активно предпринимает исследования в направлении, которое оказалось плохим, либо потому, что его модель *неверна* и он не знает, что данное действие может иметь катастрофические последствия. Обратите внимание, что в алгоритме на рис. 22.2 для изучения модели перехода используется оценка максимального правдоподобия (см. главу 20). Более того, выбирая стратегию, основанную исключительно на *оценочной* модели, этот алгоритм действует так, *как если бы* модель была правильной. Но это не обязательно хорошая идея! Например, агент беспилотного такси, который пока не знает назначения и правил работы светофоров, может один или два раза проигнорировать красный свет без вредных последствий и на этой основе сформулировать стратегию всегда игнорировать красный свет светофоров.

Лучшей идеей было бы выбрать стратегию, работающую достаточно хорошо для всего диапазона моделей, которые имеют обоснованную возможность оказаться истинной моделью, даже если эта стратегия будет выглядеть как неоптимальная для модели максимального правдоподобия. Есть три математических подхода, которые можно применить для этой цели.

Первый подход, ▶**байесовское обучение с подкреплением**, предполагает использование априорной вероятности $P(h)$ для гипотез h в отношении того, что является истинной моделью. Апостериорная вероятность $P(h \mid \mathbf{e})$ получается обычным способом по правилу Байеса с учетом наблюдений до текущего момента. Затем, если агент принимает решение прекратить обучение, оптимальной стратегией будет та, которая дает наибольшую ожидаемую полезность. Пусть U_h^{π} — это ожидаемая полезность, усредненная по всем возможным начальным состояниям, полученная при выполнении стратегии π в модели h. Тогда имеем

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_h P(h \mid \mathbf{e}) U_h^{\pi}.$$

В некоторых особых случаях эта стратегия может быть даже рассчитана! Однако, если агент продолжит обучение в будущем, найти оптимальную стратегию

будет значительно сложнее, так как ему потребуется учитывать влияние будущих наблюдений на его убеждения о модели перехода. Задача превращается в ▶**разведочный POMDP** — частично наблюдаемый марковский процесс принятия решений, доверительные состояния которого распределены по моделям. В принципе, эта задача исследования POMDP может быть сформулирована и решена еще до того, как агент вступит в мир. Результатом такого решения будет полная стратегия, указывающая агенту, что делать дальше, с учетом любой возможной последовательности восприятия. Решение задачи разведочного POMDP обычно чрезвычайно сложно, но концепция обеспечивает аналитическую основу для понимания задачи разведки, описанной в разделе 22.3.

Следует отметить, что быть совершенно байесовским недостаточно, чтобы агент был защищен от безвременной смерти. Если априорное распределение не будет включать некоторого указания на свидетельства, предполагающие опасность, то ничто не удержит агента от решения предпринять исследовательское действие, которое приводит к поглощающему состоянию. Например, раньше считалось, что у маленьких детей имеется врожденный страх высоты и они не станут ползти за край скалы, но оказалось, что это не так (Адольф и др. [16], 2014).

Другой подход, основанный на ▶**теории надежного управления**, допускает множество возможных моделей $\mathcal{H}$ без присвоения им вероятностей и определяет оптимально устойчивую стратегию как ту, которая дает лучший результат для *худшего варианта* из множества $\mathcal{H}$:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \min_{h} U_h^{\pi}.$$

Часто множество $\mathcal{H}$ будет множеством моделей, превышающих некоторый порог правдоподобия в $P(h \mid \mathbf{e})$, так что надежный и байесовский подходы связаны между собой.

Подход в теории надежного управления может рассматриваться как игра между агентом и его противником, в которой противник всегда выбирает наихудший возможный результат для любого действия агента, а искомая стратегия является минимаксным решением для этой игры. В этом смысле логический агент для мира вампуса (см. раздел 7.7) является агентом, действующим в соответствии с теорией надежного управления: он рассматривает все модели, которые логически возможны, и не исследует любые локации, которые могут содержать яму или вампуса, — следовательно, он ищет действие с максимальной полезностью для худшего случая из всех возможных гипотез.

Проблема с выбором наихудшего предположения состоит в том, что это приводит к чрезмерно консервативному поведению. Управляющий беспилотным автомобилем агент, предполагающий, что все остальные водители на дороге *будут пытаться столкнуться с ним*, не имеет другого выбора, кроме как остаться в гараже. Реальная жизнь полна таких компромиссов "риск–вознаграждение".

Хотя одной из причин рискнуть обратиться к обучению с подкреплением было стремление избавиться от необходимости привлечения учителя-человека (как при обучении с учителем), оказалось, что знания людей могут способствовать сохранению безопасности системы. Одним из способов является запись серии действий опытного учителя, чтобы система могла действовать разумно с самого начала и была способна самостоятельно обучаться и совершенствоваться непосредственно из этого исходного состояния. Второй способ состоит в том, что человек составляет набор ограничений в отношении того, что системе разрешено делать, и реализует программу вне пределов системы обучения с подкреплением, позволяющую применять эти ограничения. Например, при обучении агента беспилотного вертолета может быть предусмотрена специальная стратегия, берущая управление на себя, когда вертолет переходит в такое состояние, из которого любые дальнейшие небезопасные действия агента могут привести к переходу в необратимое состояние — в такое, в котором контроллер безопасности уже не сможет гарантировать, что можно будет избежать попадания в поглощающее состояние. Во всех других состояниях обучающийся агент может делать все, что сочтет нужным.

22.3.3. Q-обучение по методу временной разности

Теперь, после разработки алгоритма активного агента ADP, рассмотрим, как сконструировать активного агента, обучающегося по методу временной разности (*temporal-difference* — TD). Наиболее очевидным отличием является то, что агент должен будет изучить модель перехода так, чтобы иметь возможность выбрать действие на основе полезности $U(s)$, выполнив прогнозирование на один шаг вперед. Проблема получения модели для агента TD идентична проблеме для агента ADP, а правило обновления TD остается без изменений. И вновь можно показать, что алгоритм TD будет сходиться к тем же значениям, что и ADP, по мере того, как количество обучающих последовательностей стремится к бесконечности.

Метод **Q-обучения** устраняет необходимость в модели за счет изучения функции "действие-полезность" $Q(s, a)$ вместо функции полезности $U(s)$. Функция $Q(s, a)$ определяет ожидаемое общее обесцениваемое вознаграждение, если агент выполняет действие a в состоянии s и в дальнейшем действует оптимально. Знание Q-функции позволяет агенту действовать оптимально, просто выбирая $\text{argmax}_a Q(s, a)$, без необходимости прогнозирования на основе модели перехода.

Можно также вывести обновление TD для Q-значений без использования модели. Начнем с уравнения Беллмана для $Q(s, a)$ — уравнения (17.8), еще раз приведенного здесь для удобства:

$$Q(s,a) = \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma \max_{a'} Q(s', a')]. \qquad (22.6)$$

Исходя из этого, можно записать обновление TD для Q-обучения по аналогии с обновлением TD для полезностей в уравнении (22.3):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]. \qquad (22.7)$$

Это обновление вычисляется всякий раз, когда действие a выполняется в состоянии s, ведущем к состоянию s'. Как и в уравнении (22.3), член $R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)$ представляет ошибку, которую обновление пытается свести к минимуму.

Самой важной частью этого уравнения является то, чего оно не содержит: ➨*агент Q-обучения по методу временной разности не нуждается в модели перехода $P(s' \mid s, a)$ ни для обучения, ни для выбора действий.* Как отмечалось в начале главы, методы, не использующие модели, могут применяться даже в очень сложных проблемных областях, поскольку нет модели, которую требуется иметь или изучать. С другой стороны, агент Q-обучения не имеет возможности заглядывать в будущее, поэтому может испытывать затруднения, когда вознаграждения редки и для их достижения необходимо строить длинные последовательности действий.

Полная программа исследующего среду Q-обучающегося агента, в котором используется метод временной разности, приведена на рис. 22.8. Обратите внимание, что в нем используется точно такая же функция исследования f, как и в случае исследующего среду агента ADP, поэтому возникает необходимость вести статистические данные о выполненных действиях (таблицу N). Если бы применялась более простая стратегия исследования (скажем, выбор действий случайным образом в некоторой части этапов при условии, что эта часть уменьшается со временем), то можно было бы отказаться от использования таких статистических данных.

---

function Q-LEARNING-AGENT(*percept*) **returns** действие a
  inputs: *percept*, результаты восприятия, определяющие текущее состояние s' и сигнал вознаграждения r
  persistent: Q, таблица значений действий, индексированная по состояниям и действиям, первоначально содержащая нули
  N_{sa}, таблица частот пар "состояние–действие", первоначально содержащая нули
  s, a, предыдущие состояние и действие, первоначально пустые

  if s не пусто **then**
    увеличить значение $N_{sa}[s,a]$
    $Q[s,a] \leftarrow Q[\mathrm{s,a}] + \alpha(N_{sa}[s,a])(r + \max_{\mathrm{a}'} Q[s',a'] - Q[s,a])$
  $s, a \leftarrow \mathrm{s}', \mathrm{argmax}_{\mathrm{a}'} f(Q[s',a'], N_{sa}[s',a'])$
  return a

Рис. 22.8. Исследующий среду Q-обучающийся агент. Это активный ученик, который изучает функцию "действие-полезность" $Q(a,s)$ каждого действия в каждой ситуации. В нем используется такая же исследовательская функция f, как и в агенте ADP, но исключается необходимость изучать модель перехода

У метода Q-обучения имеется близкородственный метод, называемый ▶**SARSA** (*State, Action, Reward, State, Action* — состояние, действие, вознаграждение, состояние, действие). Правило обновления для SARSA очень похоже на правило обновления Q-обучения (уравнение (22.7)) за исключением того, что в методе SARSA обновление выполняется Q-значением действия a', которое фактически выполняется:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma Q(s',a') - Q(s,a)]. \qquad (22.8)$$

Правило применяется в конце каждого цикла s, a, r, s', a' — отсюда и название. Отличие этого метода от Q-обучения довольно тонкое: в то время как при Q-обучении копируются Q-значения из лучшего действия в s', в методе SARSA ожидается, пока это действие не будет выполнено, и только затем копируется Q-значение для этого действия. Если агент жадный и всегда выполняет действие с лучшим Q-значением, оба эти метода будут идентичны. Однако когда выполняется разведка, появляются различия: если исследование дает отрицательное вознаграждение, в методе SARSA действие наказывается, а в методе Q-обучения — нет.

Q-обучение — это алгоритм обучения ▶**вне стратегии** (*off-policy*), потому что он изучает Q-значения, которые отвечают вопрос "Насколько выгодно это действие в данном состоянии, если прекратить использовать любую стратегию, которая используется сейчас, и начать действовать в соответствии со стратегией, выбирающей лучшее действие (по имеющимся оценкам)?" Метод SARSA — это алгоритм ▶**на основе стратегии**: он изучает Q-значения, которые отвечают на вопрос "Насколько выгодно это действие в данном состоянии, если продолжать придерживаться текущей стратегии?" Поэтому Q-обучение является более гибким методом, чем SARSA, в том смысле, что Q-обучающийся агент может научиться вести себя хорошо, действуя под контролем широкого спектра стратегий разведки. С другой стороны, метод SARSA подходит, если общая стратегия частично контролируется другими агентами или программами, — в этом случае лучше узнать Q-функцию того, что действительно произойдет, если агент будет выбирать предположительно лучшие действия. При использовании обоих методов, Q-обучения и SARSA, агент может изучить оптимальную стратегию для клеточного мира 4×3, но достигает этой цели с гораздо меньшей скоростью, чем агент ADP. Это связано с тем, что локальные обновления не обеспечивают согласованности среди всех Q-значений через модель перехода.

22.4. Обобщение в обучении с подкреплением

До сих пор мы предполагали, что функции полезности и Q-функции представлены в виде таблицы с одним выходным значением для каждого состояния. Это справедливо для пространств состояний с примерно до 10^6 состояниями, чего более чем достаточно для нашего игрушечного двухмерного клеточного мира. Но в реальном мире в проблемных средах с гораздо большим количеством состояний

конвергенция будет слишком медленной. Нарды проще, чем большинство реальных приложений, тем не менее эта игра имеет около 10^{20} состояний. Невозможно без затруднений посетить их все, чтобы научиться играть в эту игру.

В главе 5 была представлена идея **функции оценки** как компактной меры желательности состояния в потенциально обширных пространствах состояний. В терминологии этой главы функция оценки является функцией, аппроксимирующей полезность; поэтому ниже термин ▶**аппроксимация функции** будет использоваться для обозначения процесса построения компактного приближения к истинной функции полезности или Q-функции. Например, можно аппроксимировать функцию полезности, используя взвешенную линейную комбинацию **признаков** $f_1,\ldots,fn$:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \cdots + \theta_n f_n(s).$$

Вместо того чтобы изучать 10^{20} значений состояний в таблице, алгоритм обучения с подкреплением может обучать, скажем, 20 значений для параметров $\theta = \theta_1,\ldots, \theta_{20}$, которые делают $\hat{U}_\theta$ хорошим приближением к истинной функции полезности. Иногда такая аппроксимирующая функция полезности сочетается с упреждающим поиском для получения более точных решений. Добавление упреждающего поиска означает, что эффективное поведение может быть достигнуто с использованием намного более простого аппроксиматора функции полезности, который можно обучить на гораздо меньшем числе случаев.

Аппроксимация функций позволяет эффективно представить функции полезности (или Q-функции) для очень больших пространств состояний, но важнее то, что она допускает индуктивное обобщение: на основании тех состояний, которые агент уже посетил, он может делать обобщения на состояния, которые им еще не посещались. Тезауро ([2192], 1992) использовал этот метод для создания программы игры в нарды, которая играет на уровне чемпионов-людей даже при том, что она исследовала только триллионную часть полного пространства состояний игры в нарды.

22.4.1. Аппроксимация прямой оценки полезности

Метод прямой оценки полезности (см. раздел 22.2) генерирует траектории в пространстве состояний и для каждого состояния извлекает сумму вознаграждений, полученных от этого состояния и далее до остановки. Состояние и сумма полученных вознаграждений представляют собой обучающий пример для алгоритма **обучения с учителем**. Например, предположим, что полезности для клеточного мира 4×3 представлены с помощью простой линейной функции, в которой признаками квадратов являются только их координаты x и y. В этом случае имеем

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y. \qquad (22.9)$$

Таким образом, если $(\theta_0, \theta_1, \theta_2) = (0{,}5; 0{,}2; 0{,}1)$, то $\hat{U}_\theta(1,1) = 0{,}8$. По результатам выполненной совокупности попыток будет получено множество выборочных значений $\hat{U}_\theta(x, y)$, после чего с помощью стандартной линейной регрессии (см. главу 19) можно будет найти соответствие, наилучшее с точки зрения минимизации квадратичных ошибок.

Для обучения с подкреплением больше смысла имеет использование алгоритма *оперативного* обучения, обновляющего значения параметров после каждой попытки. Предположим, что была выполнена очередная попытка и суммарное вознаграждение, полученное начиная с квадрата (1,1), составляет 0,4. Это указывает на то, что текущее значение $\hat{U}_\theta(1,1)$, равное 0,8, слишком велико и должно быть уменьшено. Как следует откорректировать параметры, чтобы достичь этой цели? Как и в случае обучения нейронной сети, запишем функцию ошибки и вычислим ее градиент по отношению к параметрам. Если $u_j(s)$ — наблюдаемое суммарное вознаграждение, полученное от состояния s и далее в j-й попытке, то ошибка определяется как (половина) квадрата разностей прогнозируемой общей суммы и фактической общей суммы: $E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$. Скорость изменения ошибки по отношению к каждому параметру θ_i равна $\partial E_j / \partial \theta_i$, поэтому, чтобы изменить значение параметра в направлении уменьшения ошибки, необходимо вычислить следующее выражение:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha[u_j(s) - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}. \qquad (22.10)$$

Это выражение называется ▶ **правилом Видроу–Хоффа** или ▶ **дельта-правилом** для оперативных вычислений по методу наименьших квадратов. Применительно к линейному аппроксиматору функции $\hat{U}_\theta(s)$ в уравнении (22.9) получаем три простых правила обновления:

$$\theta_0 \leftarrow \theta_0 + \alpha[u_j(s) - \hat{U}_\theta(s)],$$
$$\theta_1 \leftarrow \theta_1 + \alpha[u_j(s) - \hat{U}_\theta(s)]x,$$
$$\theta_2 \leftarrow \theta_2 + \alpha[u_j(s) - \hat{U}_\theta(s)]y.$$

Эти правила можно применить к примеру, в котором $\hat{U}_\theta(1,1)$ равно 0,8, а $u_j(1,1)$ равно 0,4. Значения параметров θ_0, θ_1 и θ_2 уменьшаются на коэффициент $0{,}4\alpha$, что уменьшает ошибку, относящуюся к квадрату (1,1). Обратите внимание, что ➔ *изменение значений параметров θ_i в ответ на наблюдаемый переход между двумя состояниями также приводит к изменению значений $\hat{U}_\theta$ для всех прочих состояний!* Именно это подразумевалось в приведенном выше утверждении, что функциональная аппроксимация позволяет агенту, проводящему обучение с подкреплением, обобщать свой опыт.

Мы рассчитываем на то, что агент будет проводить обучение быстрее, если он использует аппроксиматор функции, при условии, что пространство гипотез не слишком велико, но включает некоторые функции, характеризующиеся достаточно

приемлемым соответствием истинной функции полезности. В упр. 22.7 предлагается оценить производительность метода непосредственной оценки полезности как с функциональной аппроксимацией, так и без нее. В клеточном мире 4×3 действительно достигается заметное увеличение производительности, однако не столь существенное, просто потому что это пространство состояний очень мало. Достигнутое увеличение производительности становится намного более существенным в клеточном мире 10×10 с вознаграждением +1 в квадрате (10,10).

Клеточный мир 10×10 хорошо подходит для линейной функции полезности, поскольку истинная функция полезности является гладкой и почти линейной: по сути это диагональная линия с нижней точкой для квадрата (1,1) и верхней точкой для квадрата (10,10) (см. упражнение 22.10). Однако если вознаграждение +1 переместить из квадрата (10,10) в квадрат (5,5), то истинная функция полезности своей формой будет больше напоминать пирамиду и попытка применения аппроксиматора функции, приведенного в уравнении (22.9), окончится крахом.

Но не все потеряно! Напомним, что для линейной функциональной аппроксимации важно, чтобы функция линейно зависела от признаков. Но сами эти признаки вполне могут представлять собой произвольные нелинейные функции от переменных состояния. Поэтому в качестве признака можно включить, например, такой член, как $f_3(x,y)=\sqrt{(x-x_g)^2+(y-y_g)^2}$, измеряющий расстояние до цели. С этим новым признаком линейный функциональный аппроксиматор успешно работает.

22.4.2. Обучение с аппроксимацией временной разности

Подобные идеи столь же успешно можно применить к агентам, обучающимся по методу временной разности. Для этого достаточно так откорректировать параметры, чтобы это позволило уменьшить временную разность между последовательными состояниями. Новые версии уравнений для метода TD и Q-обучения (уравнение (22.3) в разделе 22.2.3 и уравнение (22.8)в разделе 22.3.3) приведены ниже. Для полезностей это:

$$\theta_i \leftarrow \theta_i + \alpha[R(s,a,s') + \gamma\hat{U}_\theta(s') - \hat{U}_\theta(s)]\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}, \qquad (22.11)$$

а для Q-значений используется следующее уравнение:

$$\theta_i \leftarrow \theta_i + \alpha[R(s,a,s') + \gamma \max_{a'} \hat{Q}_\theta(s',a') - \hat{Q}_\theta(s,a)]\frac{\partial \hat{Q}_\theta(s,a)}{\partial \theta_i}. \qquad (22.12)$$

Для пассивного обучения по методу временной разности можно показать, что эти правила обновления сходятся к ближайшей возможной аппроксимации истинной функции, если аппроксиматор функции *линейно* зависит от признаков.[4]

[4] Определение расстояния между функциями полезности является довольно надуманным; см. Цициклис и Ван Рой ([2229], 1997).

Однако при активном обучении и *нелинейных* функциях, подобных функциям, определяемым нейронными сетями, ничего нельзя гарантировать: в некоторых очень простых случаях при этих правилах обновления параметры могут возрастать до бесконечности, даже несмотря на то, что в пространстве гипотез существуют хорошие решения. Были разработаны более сложные алгоритмы, позволяющие избежать этих проблем, но и в настоящее время вся область обучения с подкреплением на основе общих аппроксиматоров функций остается тонким искусством.

Помимо параметров с возрастающими до бесконечности значениями, существует еще более удивительная проблема, называемая ▶**катастрофическим забыванием**. Предположим, что обучается агент беспилотного автомобиля с целью научиться двигаться по (эмулированной) дороге безопасно, без аварий. За выезд за пределы дороги назначено высокое отрицательное вознаграждение, и квадратичные признаки положения на дороге используются для того, чтобы агент смог изучить, что полезность от нахождения посреди дороги выше, чем от нахождения ближе к ее краю. Все идет хорошо, и агент в конечном счете научился отлично ездить по средней полосе дороги. Однако через некоторое время после этого радостного события машина внезапно выезжает с дороги и падает в откос. Почему? То, что случилось, — результат *слишком хорошего* обучения, поскольку агент выучил тот факт, что следует держаться подальше от края дороги, он научился и тому, что вся центральная часть дороги является безопасным местом, и при этом забыл, что ее часть ближе к краю опасна. Поэтому для него центральная часть дороги теперь характеризуется фиксированным значением функции ценности, а квадратичные признаки положения на ней получили нулевой весовой коэффициент. В результате любое ненулевое значение веса линейных признаков приведет к тому, что автомобиль выедет за пределы дороги с той или иной стороны.

Одно из решений этой проблемы, называемое ▶**повторением опыта**, позволяет гарантировать, что агент на регулярной основе продолжит руководствоваться своим юношеским представлением о возможности аварии. Алгоритм обучения может сохранять траектории для всего процесса обучения и воспроизводить эти траектории, чтобы гарантировать, что его функция ценности по-прежнему точна для тех частей пространства состояний, которые он больше не посещает.

Для систем обучения с подкреплением на основе моделей функциональная аппроксимация может также оказаться очень полезной для изучения модели проблемной среды. Напомним, что задача изучения модели для *наблюдаемой* среды представляет собой задачу *обучения с учителем*, поскольку каждый очередной результат восприятия предоставляет информацию о результирующем состоянии. При этом может использоваться любой из методов обучения с учителем, описанный в главах 19–21, с соответствующими поправками для учета того факта, что необходимо прогнозировать полное описание состояния, а не просто булеву классификацию или единственное реальное значение. После изучения модели агент может выполнять упреждающий поиск для улучшения своих решения, а также применять внутреннюю симуляцию для улучшения своих аппроксимирующих

представлений функций U или Q, отказавшись от медленного и потенциально дорогостоящего реального опыта.

Для *частично наблюдаемой среды* задача обучения становится гораздо более сложной, так как следующее восприятие уже не является меткой для задачи прогнозирования состояния. Если известно, что представляют собой скрытые переменные и какими причинными отношениями они связаны друг с другом и с наблюдаемыми переменными, то можно зафиксировать структуру динамической байесовской сети и воспользоваться алгоритмом EM для обучения ее параметров, как было описано в главе 20. Однако изучение внутренней структуры динамических байесовских сетей и создание новых переменных состояния до сих пор считается трудной задачей. В некоторых случаях успешные результаты были достигнуты при использовании для исследования скрытой структуры глубоких рекуррентных нейронных сетей (раздел 21.6).

22.4.3. Глубокое обучение с подкреплением

Имеется две причины, по которым необходимо выйти за пределы аппроксиматоров линейных функций. Во-первых, может не существовать хорошей линейной функции, которая достаточно подходит для аппроксимации функции полезности или Q-функции. Во-вторых, может оказаться затруднительно подобрать необходимые признаки, особенно в новых проблемных областях. Если хорошо подумать, то на самом деле это одна и та же причина: всегда *можно* представить функции U и Q как линейную комбинацию признаков, особенно если у нас есть такие признаки, как $f_1(s) = U(s)$ и $f_2(s, a) = Q(s, a)$. Но если подыскать такие признаки (в вычислительно эффективной форме) не удается, аппроксиматора линейной функции может оказаться недостаточно.

По этим причинам (или причине) исследователи рассматривали более сложные, нелинейные аппроксиматоры функций с самых первых дней исследований в области обучения с подкреплением. В настоящее время в этой роли очень популярны глубокие нейронные сети (глава 21), доказавшие свою эффективность даже в тех случаях, когда ввод представляет собой необработанное изображение без каких-либо предоставленных человеком признаков. Если все пойдет хорошо, глубокая нейронная сеть сама сможет эффективно обнаруживать полезные для нее признаки. И если последний слой сети является линейным, то появляется возможность узнать, какие именно признаки сеть использует для построения собственного аппроксиматора линейных функций. Система обучения с подкреплением, в которой глубокая сеть используется в качестве аппроксиматора функций, называется системой глубокого обучения с подкреплением.

Как и в уравнении (22.9), глубокая сеть по сути является функцией, параметризованной по θ, за исключением того, что теперь функция намного сложнее. Параметрами являются все веса во всех слоях сети. Тем не менее градиенты, необходимые для уравнений (22.11) и (22.12), точно такие же, как градиенты, необходимые

для обучения с учителем, и они могут быть вычислены с помощью того же процесса обратного распространения, описанного в разделе 21.4.

Как сообщается в разделе 22.7, системы глубокого обучения с подкреплением достигли весьма значительных результатов, в том числе научились играть в широкий спектр видеоигр на экспертном уровне, победили чемпионов мира по игре в го и успешно обучили роботов выполнять сложные задания.

Несмотря на впечатляющие успехи, системы глубокого обучения с подкреплением по-прежнему сталкиваются со значительными препятствиями: часто бывает трудно добиться хорошей производительности, и обученная система может вести себя очень непредсказуемо, когда проблемная среда даже немного отличается от обучающих данных. По сравнению с другими приложениями глубокого обучения, глубокое обучение с подкреплением редко применяется в коммерческих условиях. Тем не менее это очень активная область исследований.

22.4.4. Модификация вознаграждения

Как отмечалось во введении к этой главе, в реальных условиях вознаграждения могут быть очень редкими: для достижения ненулевого вознаграждения потребуется выполнить много простых действий. Например, играющий в футбол робот может послать сотни тысяч команд управления двигателями его суставов, прежде чем его командой будет пропущен гол. И теперь он должен понять, что же он сделал не так. Технический термин для этой ситуации — проблема ▶ **распределения вознаграждения** (*credit assignment*) или **передачи ответственности за результат**. Помимо варианта сыграть триллионы футбольных игр так, чтобы отрицательное вознаграждение в конечном итоге распространилось в обратном направлении на действия, ответственные за это, есть ли какое-то иное хорошее решение?

Один распространенный метод, первоначально использовавшийся в дрессировке животных, называется ▶ **модификацией вознаграждения** (*reward shaping*). Этот метод предусматривает предоставление агенту дополнительных вознаграждений, называемых ▶ **псевдовознаграждениями**, за "достигнутый прогресс". Например, можно дать роботу псевдовознаграждение за контакт с мячом или за продвижение мяча к воротам противника. Такие вознаграждения могут значительно ускорить обучение и предоставить их несложно, но есть риск, что агент научится максимизировать псевдовознаграждения, а не истинные вознаграждения, — например, стоя рядом с мячом "вибрировать" ногой, вызывая множество контактов с мячом.

В главе 17 (раздел 17.1.3) был предложен способ изменения функции вознаграждения без изменения оптимальной стратегии. Для любой функции потенциала $\Phi(s)$ и любой функции вознаграждения R можно создать новую функцию вознаграждения R', которая будет выглядеть следующим образом:

$$R'(s, a, s') = R(s, a, s') + \gamma\Phi(s') - \Phi(s).$$

Функцию потенциала Ф можно построить так, чтобы она отражала любые желательные аспекты состояния, такие как достижение подцелей или уменьшение расстояния до желаемого конечного состояния. Например, функция потенциала Ф для робота, играющего в футбол, может добавлять постоянный бонус для состояний, в которых команда робота владеет мячом, и другой бонус — за сокращение дистанции от мяча до ворот соперника. Это приведет к более быстрому обучению в целом, но не помешает роботу, скажем, научиться возвращать мяч вратарю, когда он оказывается в опасной ситуации.

22.4.5. Иерархическое обучение с подкреплением

Другой способ справиться с очень длинными последовательностями действий — разбить их на несколько меньших частей, а затем разбить эти части на еще более мелкие и действовать так до тех пор, пока полученные последовательности действий не станут достаточно короткими, чтобы сделать обучение легким. Такой подход называется ▶**иерархическим обучением с подкреплением** (*hierarchical reinforcement learning* — HRL) и имеет много общего с описанными в главе 11 методами **HTN-планирования** (раздел 11.4.1). Например, процесс забивания гола в футболе можно разбить на этапы получения владения мячом, передачи мяча товарищу по команде, получения мяча от товарища по команде, продвижения к воротам противника и финального удара, каждый из которых, в свою очередь, может быть дополнительно разбит на фрагменты поведения более низкого уровня. Очевидно, что существует несколько способов достижения владения мячом и выполнения решающего удара, на площадке присутствует несколько товарищей по команде, которым можно передать пас, и так далее, поэтому каждое действие более высокого уровня может иметь много разных реализаций на более низком уровне.

Чтобы проиллюстрировать эти идеи, воспользуемся примером упрощенной футбольной игры под названием ▶**keepaway**, в которой одна команда из трех игроков старается как можно дольше удерживать мяч посредством дриблинга и пасов между собой, в то время как другая команда из двух игроков пытается завладеть мячом путем перехвата паса или блокирования игрока, владеющего мячом.[5] Эта игра реализована в 2D-симуляторе RoboCup, который предоставляет детальные модели движения в непрерывном состоянии с периодом 100 мс и зарекомендовал себя как хороший испытательный стенд для систем обучения с подкреплением.

Иерархический агент обучения с подкреплением начинает с ▶**частичной программы**, которая в общих чертах обрисовывает иерархическую структуру поведения агента. Язык частичного программирования для программы агента расширяет любой обычный язык программирования, добавляя примитивы для неопределенного выбора, который должен быть заполнен посредством обучения. (Здесь в

[5] Слухи о том, что появление игры **keepaway** было вдохновлено реальной тактикой команды "Барселона", вероятно, необоснованы.

качестве языка программирования используется псевдокод.) Частичная программа может быть сколь угодно сложной при условии, что она заканчивает свою работу.

Несложно увидеть, что иерархическое обучение с подкреплением (HRL) включает обычное обучение с подкреплением (RL) в качестве особого случая. Агенту просто предоставляется тривиальная частичная программа, позволяющая ему продолжать выбирать любое действие из $A(s)$ — множества действий, которые могут быть выполнены в текущем состоянии s:

```
while true do
   choose (A(s)).
```

Оператор **choose** позволяет агенту выбрать любой элемент из указанного множества. Процесс обучения преобразует частичную программу агента в полную программу посредством изучения, как следует делать каждый выбор. Например, процесс обучения может ассоциировать Q-функцию с каждым вариантом выбора и как только Q-функции будут изучены, программа реализует поведение, выбирая вариант с наибольшим Q-значением каждый раз, когда сталкивается с необходимостью выбора.

Программы агента для игры ***keepaway*** более интересны. Рассмотрим частичную программу для одного игрока в команде "хранителей мяча". Выбор того, что делать на верхнем уровне, зависит в основном от того, есть ли у этого игрока мяч:

```
while not Is-Terminal(s) do
   if Ball-In-My-Possession(s) then choose({Pass, Hold, Dribble})
   else choose({Stay, Move, Intercept-Ball}).
```

Каждый из этих вариантов запускает подпрограмму, которая сама может принимать дальнейшие решения вплоть до примитивных действий, которые могут быть выполнены напрямую. Например, действие высокого уровня Pass выбирает партнера по команде для передачи мяча, но также имеет выбор ничего не делать и вернуть управление на более высокий уровень, если это необходимо (например, если некому передать пас):

```
choose({pass-To(choose(TeamMates(s))), return}).
```

Затем процедура Pass-To должна выбрать скорость мяча и направление паса. Хотя для человека, даже с небольшим опытом игры в футбол будет сравнительно просто дать обучающемуся агенту соответствующие рекомендации высокого уровня, было бы трудно, если вообще возможно, записать правила определения скорости и направления удара, чтобы максимизировать вероятность сохранения владения мячом этой командой. Точно так далеко не очевидно, как правильно выбрать игрока команды, которому следует передать мяч, или куда двигаться, чтобы самому быть готовым получить мяч. Частичная программа обеспечивает общее ноу-хау — общую схему "строительных лесов" и структурную организацию для сложного поведения, — а все детали прорабатываются в процессе обучения.

Теоретические основы HRL построены на концепции ▶**совместного пространства состояний**, согласно которой каждое состояние (s, m) состоит из физического состояния s и машинного состояния m. Машинное состояние определяется текущим внутренним состоянием программы агента: счетчик команд для каждой подпрограммы в текущем стеке вызовов, значения аргументов и значения всех локальных и глобальных переменных. Например, если программа агента решила передать пас Али, партнеру по команде, и сейчас находится в середине расчетов скорости мяча при передаче, то тот факт, что Али является аргументом процедуры PASS-TO, является частью текущего машинного состояния. ▶**Состояние выбора** $\sigma = (s, m)$ — это состояние, в котором счетчик команд для машинного состояния m находится в точке выбора в программе агента. Между двумя состояниями выбора может произойти любое количество вычислительных переходов и физических действий, но все они, так сказать, предопределены: по определению агент не делает никакого выбора между состояниями выбора. По сути, иерархический агент RL решает задачу марковского принятия решения со следующими элементами.

- Состояния — это состояния выбора σ в совместном пространстве состояний.
- Действия в состоянии σ — это варианты выбора c, доступные в состоянии выбора σ в соответствии с частичной программой.
- Функция вознаграждения $\rho(\sigma, c, \sigma')$ является ожидаемой суммой вознаграждений для всех физических переходов, происходящих между состояниями выбора σ и σ'.
- Модель перехода $\tau(\sigma, c, \sigma')$ определяется очевидным образом: если выбор c вызывает физическое действие a, то модель перехода τ заимствуется из физической модели $P(s' \mid s, a)$; если выбор c вызывает вычислительный переход, такой как вызов подпрограммы, то этот переход детерминировано изменяет вычислительное состояние m согласно правилам языка программирования.[6]

Решив задачу принятия решения, агент находит оптимальную стратегию, соответствующую оригинальной частичной программе.

Иерархическое обучение с подкреплением может оказаться очень эффективным методом для изучения сложных форм поведения. В игре ***keepaway*** агент иерархического обучения с подкреплением, основанный на частичной программе, схематично описанной выше, изучает стратегию, которая будет направлена на постоянное удержание мяча против стандартной стратегии нападающих, обеспечивая

[6] Поскольку до достижения следующего состояния выбора может быть выполнено более одного физического действия, проблема технически является полумарковским процессом принятия решений, в котором действиям позволяется иметь различную продолжительность, включая стохастическую длительность. Если коэффициент обесценивания $\gamma < 1$, то продолжительность действия оказывает влияние и на обесценивание, применяемое к вознаграждению, полученному во время выполнения действия, а это означает, что необходимо сделать некоторые дополнительные скидки при расчете вознаграждения и модель перехода должна включать в себя распределение вероятностей для продолжительности.

значительное улучшение предыдущего рекорда — примерно на 10 секунд. Одной из важных особенностей является то, что навыки более низкого уровня не являются фиксированными подпрограммами в обычном смысле; выбор в них чувствителен к общему внутреннему состоянию программы агента, поэтому они ведут себя по-разному в зависимости от того, откуда они были вызваны в рамках этой программы и что происходит в это время. При необходимости Q-функции для выборов более низкого уровня могут быть инициированы отдельными процессами обучения с их собственными функциями вознаграждения и затем интегрированы в общую систему, чтобы их можно было адаптировать к контексту всего агента.

В предыдущем разделе было показано, что формирование вознаграждений может оказаться полезным для обучения сложному поведению. При иерархическом обучении с подкреплением тот факт, что обучение происходит в совместном пространстве состояний, обеспечивает дополнительные возможности модификации вознаграждения. Например, чтобы помочь с изучением Q-функции для точного выполнения паса в подпрограмме PASS-TO, можно предоставлять пасующему агенту модифицируемое вознаграждение, зависящее от расположения предполагаемого получателя паса и близости противников к этому игроку: мяч должен оказаться близко к получателю и подальше от соперников. Это кажется совершенно очевидным, но ➔ *личность предполагаемого получателя паса не является частью физического состояния мира*. Физическое состояние состоит только из местоположений, ориентаций и скоростей игроков и мяча. В физическом мире нет "пасующего" и "получателя", — это полностью внутренние конструкции. А это означает, что нет никакого способа дать соответствующий разумный совет в стандартной системе обучения с подкреплением.

Иерархическая структура поведения также обеспечивает естественную ▶**аддитивную декомпозицию** общей функции полезности. Напомним, что полезность — это накапливаемая сумма вознаграждений с течением времени. Рассмотрим последовательность, скажем, из десяти временных этапов с вознаграждениями $[r_1, r_2, \ldots, r_{10}]$. Предположим, что на протяжении первых пяти временных этапов агент выполняет процедуру PASS-TO(*Ali*), а на протяжении остальных пяти этапов он выполняет процедуру MOVE-INTO-SPACE. Тогда полезность для начального состояния является суммой общего вознаграждения, полученного во время выполнения PASS-TO, и общего вознаграждения, полученного во время выполнения MOVE-INTO-SPACE. Первое зависит только о того, попадет ли мяч к Али при наличии у него достаточного времени и пространства, чтобы сохранить владение им, а последнее зависит только от того, достигнет ли агент хорошего места для получения мяча от Али. Другими словами, общая полезность декомпонуется на несколько этапов, каждый из которых зависит только от немногих переменных. А это, в свою очередь, означает, что обучение происходит гораздо быстрее, чем если пытаться изучать единственную функцию полезности, зависящую от всех переменных. Это чем-то напоминает теоремы о представлении, лежащие в основе сжатости байесовских сетей (глава 13).

22.5. Поиск стратегии

Последний подход к решению задач обучения с подкреплением, который рассматривается в этой главе, носит название ▶ **поиск стратегии**. В определенном отношении метод поиска стратегии является самым простым из всех методов, рассматриваемых в данной главе; его идея состоит в том, что необходимо продолжать корректировать стратегию до тех пор, пока связанная с ней производительность увеличивается, после чего останавливаться.

Начнем с анализа самих стратегий. Напомним, что стратегия π — это функция, отображающая состояния на действия. В наибольшей степени нас интересуют *параметризованные* представления стратегии π, имеющие намного меньше параметров по сравнению с количеством состояний в пространстве состояний (как и в предыдущем разделе). Например, можно представить стратегию π в виде коллекции параметризованных Q-функций, по одной для каждого действия, и каждый раз предпринимать действие с наивысшей прогнозируемой ценностью:

$$\pi(s) = \underset{a}{\text{argmax}}\, \hat{Q}_\theta(s, a). \tag{22.13}$$

Каждая Q-функция может представлять собой линейную функцию, как показано в уравнении (22.9), или нелинейную функцию, например, представленную в виде глубокой нейронной сети. В таком случае поиск стратегии сводится к корректировке параметров θ в целях улучшения стратегии. Следует отметить, что если стратегия представлена с помощью Q-функций, то поиск стратегии приводит к процессу изучения Q-функций. ➡ *Этот процесс не следует рассматривать как Q-обучение!*

При Q-обучении с помощью аппроксимации функций алгоритм находит такое значение θ, при котором функция $\hat{Q}_\theta$ становится "близкой" к функции Q^* — оптимальной Q-функции. С другой стороны, при поиске стратегии выполняется поиск такого значения θ, которое приводит к достижению высокой производительности, — значения, найденные этими двумя методами, могут различаться весьма существенно. (Например, аппроксимированная Q-функция, определяемая как $\hat{Q}_\theta(s, a) = Q^*(s, a) / 100$, дает оптимальную производительность, даже если она вовсе не близка к оптимальной функции Q^*.) Еще одним примером, четко иллюстрирующим это различие, является случай, когда $\pi(s)$ вычисляется, скажем, посредством опережающего поиска на глубину 10 с приближенной функцией полезности $\hat{U}_\theta$. Значение θ, дающее хорошие результаты, может быть получено задолго до того, как удастся получить приближение $\hat{U}_\theta$, напоминающее истинную функцию полезности.

Одним из недостатков представления стратегий в том виде, как показано в уравнении (22.13), состоит в том, что в случае дискретных действий стратегия *не является непрерывной* функцией своих параметров. Это означает, что существуют

такие значения θ, при которых бесконечно малые изменения θ вызывают резкое переключение стратегии с одного действия на другое. Вследствие этого может также происходить изменение ценности стратегии, не являющееся непрерывным, в результате чего поиск на основе градиента становится затруднительным. По этой причине в методах поиска стратегии часто используется ▶**стохастическое представление стратегии** $\pi_\theta(s, a)$, задающее *вероятность* выбора действия a в состоянии s. Одним из широко применяемых представлений такого типа является функция **softmax**:

$$\pi_\theta(s, a) = \frac{e^{\beta \hat{Q}_\theta(s,a)}}{\sum_{a'} e^{\beta \hat{Q}_\theta(s,a')}}. \tag{22.14}$$

Параметр $\beta > 0$ модулирует "мягкость" функции softmax: для больших значений β по сравнению с разделением между Q-значениями функция softmax приближается к жесткой функции max, тогда как для значений параметра β, близких к нулю, функция softmax приближается к равномерному случайному выбору среди действий. Для всех конечных значений параметра β функция softmax предоставляет дифференцируемую функцию от θ, — следовательно, ценность стратегии (которая непрерывно зависит от вероятностей выбора действий) является дифференцируемой функцией от θ.

Теперь обратимся к методам улучшения стратегии. Начнем с простейшего случая: детерминированная стратегия и детерминированная среда. Пусть $\rho(\theta)$ будет ▶**ценностью стратегии**, т.е. ожидаемым вознаграждением после выполнения стратегии π_θ. Если выражение для функции $\rho(\theta)$ можно вывести в замкнутой форме, то получаем стандартную задачу оптимизации, как описано в главе 4. Можно следовать вектору ▶**градиента стратегии** $\nabla_\theta \rho(\theta)$, если функция $\rho(\theta)$ дифференцируема. В качестве альтернативы, если $\rho(\theta)$ в замкнутом виде недоступна, можно оценить стратегию π_θ, просто выполняя ее и регистрируя накопленные вознаграждения. Также можно следовать **эмпирическому градиенту** методом восхождения к вершине, т.е. оценивая изменения ценности стратегии для небольших приращений каждого из параметров. При соблюдении обычных предосторожностей этот процесс будет сходиться к локальному оптимуму в пространстве стратегии.

Если же среда (или стратегия) является недетерминированной, все становится сложнее. Предположим, что предпринимается попытка применить метод восхождения к вершине, для чего требуется сравнить $\rho(\theta)$ и $\rho(\theta + \Delta\theta)$ при некотором небольшом значении $\Delta\theta$. Проблема состоит в том, что суммарное вознаграждение при каждой попытке может существенно изменяться, поэтому оценки ценности стратегии на основе небольшого количества попыток могут оказаться совершенно ненадежными, а результаты, полученные при попытке сравнить две такие оценки, будут еще более ненадежными. Одно из решений состоит в том, чтобы предпринять много попыток, измеряя дисперсию выборок и используя ее для определения

того, достаточно ли много было сделано попыток, чтобы получить надежные данные о направлении улучшения для $\rho(\theta)$. К сожалению, такой подход является практически неприменимым во многих реальных задачах, когда каждая попытка может оказаться достаточно дорогостоящей, требующей больших затрат времени и, возможно, даже опасной.

В случае недетерминированной стратегии $\pi_\theta(s, a)$ можно получить несмещенную оценку градиента $\nabla_\theta\rho(\theta)$ в θ непосредственно по результатам попыток, выполненных при данном значении параметра θ. Для простоты выведем формулу подобной оценки для простого случая эпизодической среды, в которой за выполнение каждого действия a предоставляется вознаграждение $R(s_0, a, s_0)$ и среда перезапускается в состоянии s_0. В этом случае ценность стратегии является просто ожидаемым значением вознаграждения, поэтому имеет место следующее:

$$\nabla_\theta\rho(\theta) = \nabla_\theta \sum_a R(s_0, a, s_0)\pi_\theta(s_0, a) = \sum_a R(s_0, a, s_0)\nabla_\theta\pi_\theta(s_0, a).$$

Теперь можно применить простой трюк, позволяющий аппроксимировать результаты этого суммирования с помощью выборок, сформированных на основании распределения вероятностей, определенного стратегией $\pi_\theta(s_0, a)$. Предположим, что общее количество попыток равно N, и действием, предпринятым в j-й попытке, является a_j. В таком случае получим следующее:

$$\nabla_\theta\rho(\theta) = \sum_a \pi_\theta(s_0, a) \cdot \frac{R(s_0, a, s_0)\nabla_\theta\pi_\theta(s_0, a)}{\pi_\theta(s_0, a)} =$$

$$= \approx \frac{1}{N}\sum_{j=1}^{N} \frac{R(s_0, a_j, s_0)\nabla_\theta\pi_\theta(s_0, a_j)}{\pi_\theta(s_0, a_j)}.$$

Таким образом, истинный градиент ценности стратегии аппроксимируется суммой термов, включающей градиент вероятности выбора действия при каждой попытке. Для последовательного случая это соотношение для каждого посещенного состояния s можно обобщить к такому виду:

$$\nabla_\theta\rho(\theta) \approx \frac{1}{N}\sum_{j=1}^{N} \frac{u_j(s)\nabla_\theta\pi_\theta(s, a_j)}{\pi_\theta(s, a_j)},$$

где a_j — действие, выполненное в состоянии s при j-й попытке, а $u_j(s)$ — суммарное вознаграждение, полученное при j-й попытке, начиная от состояния s и далее. Построенный в результате алгоритм благодаря Рону Уильямсу ([2350], 1992) получил название REINFORCE, — обычно он является гораздо более эффективным по сравнению с восхождением к вершине, при котором используется большое количество попыток для каждого значения θ. Тем не менее он все еще действует гораздо медленнее, чем требуется.

Рассмотрим следующую задачу: даны две стратегии игры в блэкджек, и необходимо определить, какая из них лучше. Эти стратегии могут давать истинную чистую прибыль на раздачу в размере, скажем, −0,21% и +0,06%, поэтому выяснить, какая из них лучше, очень важно. Один из способов решения этой задачи состоит в выполнении каждой стратегии при игре против стандартной стратегии "сдающего" в течение определенного количества раздач с последующим сравнением суммарного выигрыша для каждой из стратегий. Но этот подход связан с определенными проблемами, поскольку выигрыш при любой из стратегий изменяется в широких пределах в зависимости от того, какие карты были получены при очередной раздаче, хорошие или плохие.

Лучшее решение для игры блэкджек — заранее составить определенное количество раздач, и ➧ *пусть один и тот же набор раздач будет разыгран при каждой из стратегий.* В этом случае исключается ошибка измерения за счет различий в полученных при раздаче картах, а для определения, какая из двух стратегий игры в блэкджек лучше, будет достаточно всего нескольких тысяч раздач.

Эта идея, называемая ▶ **коррелированной выборкой**, может быть применена к задаче поиска стратегии в общем случае — при наличии симулятора проблемной среды, обеспечивающего возможность повторения последовательности случайных чисел. Данный подход был реализован в алгоритме поиска стратегии под названием "PEGASUS" (Энг и Джордан [1675], 2000), который был одним из первых алгоритмов, обеспечивших достижение полностью стабильного беспилотного полета вертолета (см. ниже рис. 22.9, *б*). Можно показать, что количество случайных последовательностей, необходимых для получения гарантии, что стоимость *каждой* стратегии оценена хорошо, зависит только от сложности пространства стратегии, а не от сложности соответствующей проблемной области.

22.6. Обучение на демонстрации и обратное обучение с подкреплением

Некоторые предметные области настолько сложны, что, оказывается, трудно определить функцию вознаграждения для использования в обучении с подкреплением. Что именно должен делать беспилотный автомобиль? Безусловно, он не должен потратить слишком много времени на поездку до места назначения, но при этом он не должен ехать так быстро, чтобы вызвать чрезмерный риск или получить штраф за превышение скорости. Он должен экономить топливо или энергию. Ему следует избегать толчков или резких ускорений, отрицательно воздействующих на пассажиров, но резкое торможение все же допустимо в чрезвычайной ситуации. И так далее. Решить, какой вес придать каждому из этих факторов, — трудная задача. Еще хуже то, что почти наверняка будут важные факторы, которые так и не были учтены, например обязанность вести себя с уважением по отношению к другим водителям. Пропуск любого фактора обычно приводит к поведению,

при котором пропущенный фактор характеризуется экстремальными значениями; в данном случае это будет крайне вызывающее, игнорирующее интересы других водителей вождение, проявляющееся просто как следствие стремления максимизировать все остальные факторы.

Один из подходов состоит в проведении обширного тестирования в условиях симуляции с целью заметить проблемное поведение и попытаться изменить функцию вознаграждения таким образом, чтобы исключить это нежелательное поведение. Другой подход заключается в поиске дополнительных источников информации о соответствующей функции вознаграждения. Одним из таких источников может быть поведение агентов, которые уже имеют оптимальную (или, скажем, почти оптимальную) соответствующую функцию вознаграждения; в данном случае это опытные водители-люди.

В области исследований, получившей название ▶**обучение на демонстрации** (*apprenticeship learning*) или ▶**ученичество** (или даже **стажировка**), изучается процесс обучения тому, как вести себя хорошо посредством наблюдений за поведением экспертов. Агенту демонстрируются примеры алгоритма поведения экспертов за рулем и указывается: "Делай так". Есть по крайней мере два варианта подхода к решению задачи обучения на демонстрации. Первый — тот, который кратко обсуждался в начале главы: предполагая, что проблемная среда является наблюдаемой, для изучения стратегии $\pi(s)$ используются методы обучения с учителем на основе пар "состояние–действие". Этот подход называется ▶**имитационным обучением**. Он имел некоторый успех в робототехнике (см. раздел 26.8.2), но в общем случае страдает от проблемы уязвимости: даже небольшие отклонения от обучающего набора приводят к ошибкам, которые накапливаются со временем и в конечном счете приводят к отказу. Более того, имитационное обучение в лучшем случае позволяет лишь дублировать эксплуатационные качества учителя и не обеспечивает возможности их превзойти. Когда люди учатся подражанием, в отношении их действий иногда используется уничижительный термин "обезьянничать". (Вполне возможно, что обезьяны в своей среде также используют термин "человечничать", причем в еще более уничижительном смысле.) Здесь подразумевается, что обучающийся посредством имитации не понимает, *почему* он должен выполнять какое-либо действие так, а не иначе.

Второй подход к обучению на демонстрации состоит в том, чтобы понять, *почему*: наблюдать за действиями эксперта (и конечными состояниями), пытаясь выяснить, что именно максимизирует функцию вознаграждения эксперта. Зная это, можно было бы разработать оптимальную стратегию для данной функции вознаграждения. Можно ожидать, что такой подход приведет к созданию надежных стратегий на основе сравнительно немногих примеров поведения эксперта. В конце концов, вся область обучения с подкреплением основана на идее, что функция вознаграждения, а не стратегия или функция ценности, является наиболее лаконичным, надежным и переносимым определением задачи. Кроме того, если ученик делает соответствующие допущения о возможной субоптимальности

со стороны эксперта, то он может оказаться в состоянии действовать лучше, чем эксперт посредством оптимизации и точной аппроксимации функции истинного вознаграждения. Подобный подход называют ▶**обратным обучением с подкреплением** (*inverse reinforcement learning* — IRL): изучение функции вознаграждения путем наблюдения стратегии вместо изучения стратегии путем наблюдения за вознаграждениями.

Как же можно определить функцию вознаграждения эксперта на основании действий эксперта? Начнем с предположения, что эксперт действовал рационально. В этом случае, похоже, следует искать такую функцию вознаграждения R^*, что общее ожидаемое обесцененное вознаграждение в соответствии со стратегией эксперта будет выше (или по крайней мере такое же) чем при любой другой возможной стратегии.

К сожалению, существует слишком много функций вознаграждения, удовлетворяющих этому ограничению, — одной из их будет $R^*(s, a, s') = 0$, поскольку любая стратегия будет рациональной, когда вообще нет никаких вознаграждений.[7] Другая проблема с этим подходом состоит в том, что предположение о рациональности эксперта нереально. Это допущение означает, например, что робот, наблюдающий за игрой Ли Седола в го против программы AlphaGo, когда он делает ход, в итоге оказавшийся проигрышным, должен будет предположить, что Ли Седол пытался проиграть эту игру.

Для исключения проблемы, заключающейся в том, что функция вознаграждения $R^*(s, a, s') = 0$ может объяснить любое наблюдаемое поведение, полезно будет обратиться к байесовскому подходу. (Чтобы вспомнить, что это означает, обратитесь к разделу 20.1.) Предположим, что имеются данные восприятия **d**, и пусть h_R будет гипотезой, что R является истинной функцией вознаграждения. Тогда, согласно правилу Байеса, можно записать

$$P(h_R \mid \mathbf{d}) = \alpha P(\mathbf{d} \mid h_R) P(h_R).$$

Теперь, если априорное распределение $P(h_R)$ основано на простоте, гипотеза о том, что $R = 0$, оценивается достаточно хорошо, поскольку 0 — это, определенно, просто. С другой стороны, член $P(\mathbf{d} \mid h_R)$ будет *бесконечно мал* для гипотезы $R = 0$, потому что это не объясняет, почему эксперт выбрал именно данный вариант поведения из огромного пространства поведений, которые могли бы являться оптимальными, если бы гипотеза была верной. Так, для функции вознаграждения R, имеющей уникальную оптимальную стратегию или относительно

[7] Согласно уравнению (17.9) в разделе 17.1.3, функция вознаграждения $R'(s, a, s') = R(s, a, s') + \gamma \Phi(s') - \Phi(s)$ имеет в точности те же оптимальные стратегии, что и $R(s, a, s')$, поэтому можно восстановить функцию вознаграждения только до возможного добавления любой формирующей функции $\Phi(s)$. Это не столь уж серьезная проблема, потому что робот, использующий функцию вознаграждения R', будет вести себя точно так, как робот, использующий "правильную" функцию вознаграждения R.

небольшой класс эквивалентных оптимальных стратегий, значение $P(\mathbf{d} \mid h_R)$ будет гораздо выше.

Чтобы допустить возможность случайной ошибки эксперта, достаточно просто позволить вероятности $P(\mathbf{d} \mid h_R)$ быть ненулевой, даже когда восприятие $\mathbf{d}$ следует из поведения, которое слегка неоптимально в соответствии с функцией вознаграждения R. Типичное предположение (сделанное, надо сказать, больше для математического удобства, чем для полной достоверности в отношении реальных человеческих данных) состоит в том, что агент, истинная Q-функция которого — $Q(s, a)$, делает выбор не в соответствии с детерминированной стратегией $\pi(s) = \text{argmax}_a Q(s, a)$, а согласно стохастической стратегии, определяемой распределением softmax из уравнения (22.14). Иногда это называют ▶ **рациональностью Больцмана**, потому что в статистической механике вероятности принятия состояния в распределении Больцмана экспоненциально зависят от их энергетических уровней.

В литературе описаны десятки алгоритмов обратного обучения с подкреплением. Один из самых простых называется ▶ **соответствие признаков** (*feature matching*). В нем предполагается, что функция вознаграждения может быть записана в виде взвешенной линейной комбинации признаков:

$$R_\theta(s, a, s') = \sum_{i=1}^{n} \theta_i f_i(s, a, s') = \theta \cdot \mathbf{f}.$$

Например, признаки в проблемной области беспилотного вождения автомобиля могут включать скорость движения, скорость, превышающую ее установленное ограничение, ускорение, близость к ближайшему препятствию и т.д.

Вспомним из уравнения (17.2) в разделе 17.1.2, что полезность выполнения стратегии π, начиная из состояния s_0, определяется как

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right],$$

где ожидание E относительно распределения вероятностей по последовательности состояний определяется состоянием s и стратегией π. Поскольку предполагается, что функция R — линейная комбинация значений признаков, можно переписать это выражение следующим образом:

$$\begin{aligned} U^\pi(s) &= E\left[\sum_{t=0}^{\infty} \gamma^t \sum_{i=1}^{n} \theta_i f_i(S_t, \pi(S_t), S_{t+1})\right] = \\ &= \sum_{i=1}^{n} \theta_i E\left[\sum_{t=0}^{\infty} \gamma^t f_i(S_t, \pi(S_t), S_{t+1})\right] = \\ &= \sum_{i=1}^{n} \theta_i \mu_i(\pi) = \theta \cdot \mu(\pi). \end{aligned}$$

Здесь ▶ **математическое ожидание признака** $\mu_i(\pi)$ определено как ожидаемая ценность признака f_i с учетом обесценивания при выполнении стратегии π. Например, если признак f_i — это превышение допустимой для транспортного средства скорости (выше установленного ограничения скорости), то $\mu_i(\pi)$ — это средняя (обесцененная по времени) избыточная скорость по всей траектории. Ключевым моментом в отношении ожидаемых средних значений признаков является следующее: ➡ *если стратегия* π *обеспечивает ожидаемые средние значения признаков* $\mu_i(\pi)$, *соответствующие тем, которые имеют место при выполнении экспертом стратегии* π_E, *то стратегия* π *так же хороша, как стратегия эксперта в соответствии с собственной функцией вознаграждения эксперта*. Получается, мы не можем измерить точные значения ожидаемых средних значений признаков для стратегии эксперта, но можем аппроксимировать их, используя средние значения по наблюдаемым траекториям. Следовательно, необходимо найти такие значения параметров θ_i, чтобы ожидаемые значения признаков для принятой стратегии, вызванные этими значениям параметров, совпадали с их значениями при экспертной стратегии по наблюдаемым траекториям. Следующий алгоритм позволяет достичь этой цели с любой желаемой предельной ошибкой.

- Выберите начальную стратегию по умолчанию $\pi^{(0)}$.
- Для $j = 1, 2, \ldots$ до сходимости:
  - найти параметры $\theta^{(j)}$, такие, что стратегия эксперта максимально превосходит стратегии $\pi^{(0)}, \ldots, \pi^{(j-1)}$ в соответствии с ожидаемой полезностью $\theta^{(j)} \cdot \mu(\pi)$;
  - пусть $\pi^{(j)}$ — оптимальная стратегия для функции вознаграждения $R^{(j)} = \theta^{(j)} \cdot \mathbf{f}$.

Этот алгоритм сходится к стратегии, близкой по ценности к стратегии эксперта, в соответствии с собственной функцией вознаграждения эксперта. Потребуется выполнить лишь $O(n \log n)$ итераций и $O(n \log n)$ демонстраций действий эксперта, где n — количество признаков.

Робот может использовать обратное обучение с подкреплением для выработки хорошей собственной стратегии за счет понимания действий эксперта. Более того, робот может изучать стратегии, используемые другими агентами в многоагентной проблемной области, не важно, состязательной или кооперативной. И наконец, обратное обучение с подкреплением может использоваться для научных исследований (без каких-либо идей относительно устройства агента) с целью лучше понять поведение людей и различных животных.

Ключевым допущением в методе обратного обучения с подкреплением является то, что "эксперт" ведет себя оптимально или почти оптимально относительно некоторой функции вознаграждения в марковском процессе принятия решений с одним агентом. Это допущение будет разумным, если обучающийся наблюдает за экспертом через одностороннее зеркало, пока эксперт занимается своими

делами, не подозревая, что за ним наблюдают. Это допущение уже не будет разумным, если эксперт знает о том, что за ним наблюдает ученик. Например, предположим, что робот обучается в медицинском учебном заведении с целью выучиться на хирурга посредством наблюдения за человеком-экспертом. В алгоритме обратного обучения с подкреплением предполагается, что человек выполняет операцию обычным оптимальным образом, как если бы никакого робота не было. Но в действительности все пойдет совсем не так: человек-хирург мотивирован помочь роботу (как и любому другому студенту-медику) учиться быстро и хорошо, и поэтому он значительно изменит свое поведение. Он может по ходу дела объяснять, что именно он делает в данный момент, указывать на ошибки, которых следует избегать, такие как слишком глубокий разрез или слишком плотные швы, или же предлагать план действий в чрезвычайных обстоятельствах на случай, если во время операции что-то пойдет не так. Ни одно из этих действий не имеет смысла при выполнении операции без учеников, поэтому алгоритмы обратного обучения с подкреплением не смогут правильно интерпретировать основную функцию вознаграждения. Вместо обучения ситуацию такого рода следует понимать как игру для двух участников, подобно тому, как описано в разделе 18.2.5.

22.7. Применение методов обучения с подкреплением

Теперь обратимся к примерам применения методов обучения с подкреплением. К ним, в частности, относятся игры, когда модель перехода известна и цель состоит в том, чтобы изучить функцию полезности, а также робототехника, в которой модель перехода изначально неизвестна.

22.7.1. Приложения ведения игр

В главе 1 описывалась ранняя работа Артура Самуэля, начатая им в 1952 году, в которой обучение с подкреплением применялось к программе игры в шашки. Прошло несколько десятилетий, прежде чем этот вызов вновь был принят Джерри Тезауро в его работе над программой игры в нарды. Результатом первых попыток Тезауро (1990) была система под названием "NEUROGAMMON". Выбранный им подход был интересным вариантом имитационного обучения. Входные данные представляли собой пакет из 400 игр, которые Тезауро сыграл против самого себя. Вместо изучения стратегии система NEUROGAMMON преобразовывала каждый ход (s, a, s') в множество обучающих примеров, в каждом из которых состояние s' помечалось как позиция, лучшая, чем некоторая другая позиция s'', достижимая из состояния s различными ходами. Сеть включала две отдельные половины, для s' и для s'', и ограничивалась выбором того варианта хода, который оказывался лучшим при сравнении выходов двух ее половин. Таким образом, каждая половина

сети была вынуждена изучать функцию оценки $\hat{U}_\theta$. Система NEUROGAMMON выиграла компьютерную олимпиаду 1989 года — первая программа обучения, выигравшая турнир по компьютерным играм, — но так никогда и показала результатов выше, чем собственный средний уровень игры Тезауро.

В следующей системе Тезауро, TD-GAMMON (1992), был принят недавно опубликованный TD-метод обучения Саттона, — по сути, это возвращение к подходу, исследованному Самуэлем, но со значительно б*о*льшим пониманием, как это сделать правильно технически. Функция оценки $\hat{U}_\theta$ была представлена полносвязной нейронной сетью с одним скрытым слоем, содержащим 80 узлов. (Также использовались некоторые разработанные вручную входные признаки, заимствованные из системы NEUROGAMMON.) После проведения 300 тыс. обучающих игр система достигла уровня игры, сопоставимого с уровнями трех лучших игроков мира. Кит Вулси, игрок из первой десятки, сказал: "У меня не было повода усомниться в том, что эта программа оценивает позицию гораздо лучше меня".

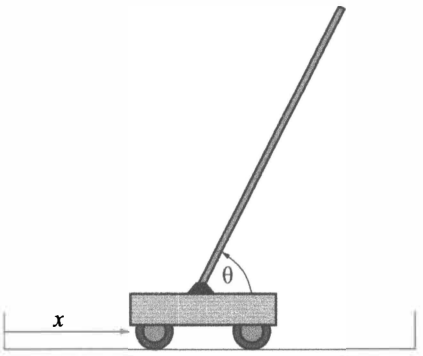
Следующая цель состояла в том, чтобы проводить обучение на основании необработанных результатов восприятия — приблизиться к реальному миру — вместо дискретных представлений игровой доски. Начиная с 2012 года команда компании DeepMind занималась разработкой системы ▶**Deep Q-network** (▶**DQN**) — первой современной системы глубокого обучения с подкреплением. В системе DQN для представления Q-функции используется глубокая нейронная сеть, а во всем остальном это типичная система глубокого обучения. Система DQN по отдельности обучалась игре в каждую из 49 различных видеоигр на приставке Atari. Она научилась водить симулируемые гоночные машины, стрелять в инопланетные космические корабли и отбивать мячи ракеткой. В каждом случае агент изучал Q-функцию на основании необработанных видеоданных с внешним сигналом вознаграждения, представленным счетом в игре. В целом система действовала на уровне эксперта-человека, хотя несколько игр доставили разработчикам некоторые неприятности. В частности, одна из игр, "Месть Монтесумы", оказалась слишком трудной для обучения, поскольку требовала расширенных стратегий планирования, а вознаграждения в игре были слишком редкими. Дальнейшая работа привела к созданию новых систем глубокого обучения с подкреплением, проводивших более широкую исследовательскую работу и способных добиться победы в игре "Месть Монтесумы" и других сложных играх.

В системе ALPHAGO компании DeepMind глубокое обучение использовалось с целью победить лучших игроков-людей в игре го (см. главу 5). В то время как для игр на видеоприставке Atari, которые в основном носят реактивный характер, было достаточно Q-функции без прогнозирования наперед, игра го требует существенного прогнозирования. По этой причине в системе ALPHAGO изучается как функция ценности, так и Q-функция, что позволяет ей направлять поиск за счет прогнозирования, какие из возможных ходов стоит исследовать. Q-функция, реализованная в системе как сверточная нейронная сеть, сама по себе достаточно

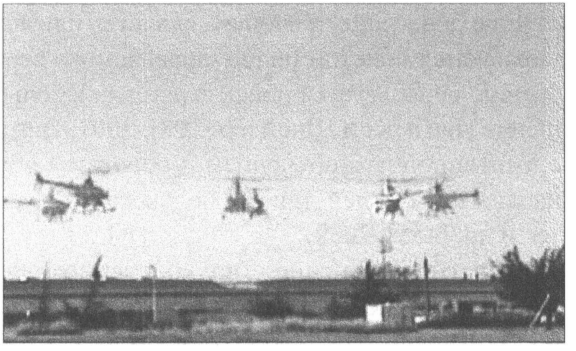
точна, чтобы победить большинство людей, играющих на любительском уровне, вообще без какого-либо поиска.

22.7.2. Управление роботами

Постановка знаменитой задачи поиска равновесия системы ▶**тележка–шест**, известной также под названием задачи ▶**обратного маятника**, показана на рис. 22.9, *а*. Задача заключается в том, чтобы удерживать шест примерно в вертикальном положении ($\theta \approx 90°$) за счет применения силы для перемещения тележки вправо или влево, сохраняя ее положение x в пределах заданной длины пути. Такой, казалось бы, простой задаче посвящено несколько тысяч статей в области обучения с подкреплением и теории управления. Основная сложность здесь состоит в том, что переменные состояния x, θ, $\dot{x}$ и $\dot{\theta}$ являются непрерывными, тогда как действия в этой задаче определяются как дискретные: это "рывки" тележки влево или вправо в так называемом режиме ▶**релейного управления** (*bang-bang control*).



а)



б)

Рис. 22.9. а) Постановка задачи балансировки длинного шеста, установленного на движущейся тележке. Тележка может совершать кратковременные "рывки" влево или вправо под управлением контроллера, отслеживающего положение тележки x и ее скорость $\dot{x}$, а также угол наклона шеста θ и скорость изменения этого угла $\dot{\theta}$. **б)** Шесть наложенных друг на друга покадровых изображений одного автономно управляемого вертолета, выполняющего очень сложный маневр "нос в круге". Контроллер управляет вертолетом, следуя стратегии, разработанной алгоритмом поиска стратегии PEGASUS (Энг и др. [1677], 2003). Имитационная модель была разработана путем наблюдения результатов различных управляющих манипуляций на настоящем вертолете, после чего алгоритм выполнялся на симуляторе модели в течение одной ночи. Для разных маневров были разработаны различные контроллеры. Во всех случаях их производительность намного превышала производительность опытного человека-пилота, использующего дистанционное управление.

Самая ранняя работа по изучению этой задачи была выполнена Мичи и Чамберсом ([1565], 1968), которые использовали реальную тележку и шест, а не их симуляцию. Разработанный ими алгоритм BOXES был в состоянии поддерживать равновесие шеста в течение часа после всего лишь 30 пробных попыток. В алгоритме сначала выполнялась дискретизация четырехмерного пространства состояний путем его разбивки на гиперкубы, называемые разработчиками *boxes* — отсюда и название алгоритма, — а затем выполнялась пробная попытка удержания, пока шест не падал. Отрицательное подкрепление связывалось с последним действием в последнем гиперкубе, а затем распространялось в обратном направлении через всю последовательность действий. Улучшенное обобщение и более быстрое обучение могут быть достигнуты при использовании алгоритма, который *адаптивно* разбивает пространство состояний в соответствии с наблюдаемым изменением вознаграждения, либо при использовании аппроксиматора нелинейной непрерывной функции, такого как нейронная сеть. В настоящее время уравновешивание *тройного* обратного маятника (три шеста, поставленные друг на друга) считается тривиальной задачей — способность, далеко выходящая за пределы возможностей большинства людей, но вполне достижимая для систем обучения с подкреплением.

Еще более впечатляющим является применение обучения с подкреплением для управления полетом радиоуправляемого вертолета (рис. 22.9, *б*). В этом задании обычно используется поиск стратегии в большой задаче марковского принятия решений (Багнелл и Шнейдер [108], 2001; Энг и др. [1677], 2003), часто в сочетании с имитационным обучением и обратным обучением с подкреплением на основе наблюдений за действиями человека, эксперта в пилотировании вертолетов (Коатес и др. [455], 2009).

Обратное обучение с подкреплением так же успешно применялось для интерпретации поведения человека, включая прогнозирование пункта назначения и выбор маршрута водителями такси на базе GPS-данных о поездках суммарной длиной 100 тыс. миль (Зибарт и др. [2444], 2008), а также для интерпретации детализированных физических движений пешеходов в сложном окружении, исходя из многих часов записей видеонаблюдений (Китани и др. [1234], 2012). В области робототехники одной демонстрации действий эксперта оказалось достаточно, чтобы четвероногий робот LittleDog выучил функцию вознаграждения с 25 признаками и проворно прошелся по ранее неизвестному ему участку каменистой местности (Колтер и др. [1274], 2008). Подробно о том, как обучение с подкреплением и обратное обучение с подкреплением используются в робототехнике, говорится в разделах 26.7 и 26.8.

Резюме

В этой главе рассматривалась задача обучения с подкреплением: как агент может добиться успеха в неизвестной среде, опираясь только на свое восприятие и случайные вознаграждения. Обучение с подкреплением можно рассматривать как очень широко применимую парадигму для создания интеллектуальных систем. Основные идеи, изложенные в этой главе, следующие.

- Общая схема агента определяет тип информации, которую ему необходимо изучить.
  - Агент **обучения с подкреплением на основе модели** нуждается (или уже обладает) в модели перехода $P(s' \mid s, a)$ для проблемной среды и изучает функцию полезности $U(s)$.
  - Агент **обучения с подкреплением без модели** может изучать функцию полезности действия $Q(s, a)$ или стратегию $\pi(s)$.
- Изучение полезности действий может быть реализовано с использованием нескольких различных подходов.
  - При **непосредственной оценке полезности** наблюдаемое вознаграждение за всю траекторию для заданного состояния используется как прямое свидетельство для изучения его полезности.
  - В методе **адаптивного динамического программирования** (ADP) на основании наблюдений изучаются модель и функция вознаграждения, а затем алгоритм итерации по значениям или алгоритм итерации по стратегиям используется для получения полезности или нахождения оптимальной стратегии. В методе ADP оптимально используются локальные ограничения, налагаемые на полезности состояний как следствие структуры отношения соседства в окружающей среде.
  - В методах **временной разности** (*Temporal Difference* — TD) оценки полезности корректируются для согласования с состояниями-преемниками. Такой подход можно рассматривать как простую аппроксимацию метода ADP, когда модель перехода для процесса обучения не требуется. Однако использование изученной модели для выработки псевдорезультатов опытов может способствовать ускорению обучения.
- Функции полезности действия, или Q-функции, могут быть изучены методом ADP или TD. При использовании метода TD в процессе **Q-обучения** не требуется модель ни на этапе обучения, ни на этапе выбора действия. Это упрощает задачу обучения, но потенциально может привести к ограничению способности к обучению в сложных средах, поскольку агент не сможет моделировать результаты возможных вариантов действий.
- Если обучающийся агент отвечает за выбор действий в процессе обучения, он должен находить компромисс между оценочной ценностью этих

действий и потенциалом изучения новой информации. Точное решение проблемы разведки невозможно, но некоторые простые эвристики могут оказаться полезными. Агент-исследователь также должен заботиться о том, чтобы избежать преждевременной смерти.

- В больших пространствах состояний алгоритмы обучения с подкреплением должны использовать приближенное представление функций $U(s)$ и $Q(s, a)$ для обобщения состояний. **Глубокое обучение с подкреплением** — подход с использованием глубоких нейронных сетей в качестве аппроксиматоров функций — позволило добиться значительных успехов при решении сложных задач.
- Методы **формирования вознаграждений** и **иерархического обучения с подкреплением** будут полезны при изучении сложного поведения, особенно когда вознаграждения редки и для их получения требуется выполнение длинных последовательностей действий.
- Методы **поиска стратегии** применяются непосредственно к представлению стратегии в попытке улучшить ее с учетом наблюдаемой производительности. Изменчивость производительности в стохастической проблемной области представляет собой серьезную проблему, но в случае симулируемых проблемных областей эту сложность можно преодолеть, заранее формируя и фиксируя случайные выборки.
- Методы **обучения на демонстрации**, построенные на наблюдении обучающегося агента за поведением экспертов, могут оказаться эффективным решением, когда трудно определить правильную функцию вознаграждения. В методе **имитационного обучения** задача формулируется как задача изучения стратегии методами обучения с учителем на основе пар "состояние–действие", предоставляемых экспертом, а метод **обратного обучения с подкреплением** позволяет вывести из поведения эксперта информацию о вознаграждении.

Обучение с подкреплением продолжает оставаться одной из наиболее активных областей исследований в машинном обучении. Оно освобождает нас от ручного построения поведения и от необходимости снабжения метками обширных наборов данных, требуемых для методов обучения с учителем, а также от обязанности вручную программировать стратегии управления. Применение обучения с подкреплением в робототехнике обещает быть особенно полезным, поскольку здесь требуются методы обучения в непрерывных, многомерных, частично наблюдаемых средах, в которых успешное поведение может складываться из тысяч или даже миллионов примитивных действий.

В этой главе были представлены различные подходы к обучению с подкреплением, поскольку не существует (по крайней мере, пока) единственного лучшего подхода. Вопрос о выборе между методами на основе модели и безмодельными методами — это, по сути, вопрос о наилучшем способе представления функции

агента. Это проблема, лежащая в самой основе искусственного интеллекта. Как уже указывалось в главе 1, одной из ключевых исторических характеристик многих исследований в области ИИ является их (часто не декларируемая) приверженность к подходу, **основанному на знаниях**. Это равносильно предположению, что лучший способ представить функцию агента заключается в построении представления некоторых аспектов среды, в которой агент находится. Некоторые утверждают, что при наличии доступа к достаточному количеству данных безмодельные методы позволяют добиться успеха в любом домене. Возможно, это верно в теории, но, конечно же, вселенная может просто не содержать достаточного количества данных, чтобы реализовать это на практике. (Например, совсем непросто представить, как безмодельный подход позволит спроектировать и построить, скажем, детектор гравитационных волн LIGO.) Наша интуиция, как бы там ни было, подсказывает, что, когда среда становится более сложной, преимущества подхода, основанного на модели, становятся более очевидными.

Библиографические и исторические заметки

Кажется вероятным, что основная идея обучения с подкреплением — животные охотнее делают то, за что их вознаграждают, чем то, за что их наказывают — сыграла значительную роль в одомашнивании собак не менее 15 тыс. лет назад. Первые успехи в научном понимании обучения с подкреплением были достигнуты благодаря исследованиям русского физиолога Ивана Павлова, получившего Нобелевскую премию в 1904 году, и работам американского психолога Эдварда Торндайка, особенно его книге "Интеллект животных" ([2208], 1911). Хильгард и Бауэр ([1022], 1975) предоставили хороший обзор этой темы.

Алан Тьюринг ([2234], 1948; [2235], 1950) предложил обучение с подкреплением как подход к обучению компьютеров, но считал это лишь частичным решением проблемы: "Использование наказаний и вознаграждений в лучшем случае может являться лишь частью процесса обучения". Первым успешным применением машинного обучения любого типа следует считать программу игры в шашки Артура Самуэля ([1969], 1959; [1970], 1967). Самуэль предложил бо́льшую часть современных идей в обучении с подкреплением, включая обучение с учетом временно́й разности и аппроксимацию функции. Он экспериментировал с многослойными представлениями функций ценности, что близко к современному глубокому обучению с подкреплением. В конечном счете он обнаружил, что в его задаче лучше всего работает простая линейная функция оценки вручную отобранных признаков. Это могло быть следствием того, что он работал с компьютером, примерно в 100 млрд раз менее мощным, чем современный тензорный процессор.

Примерно в то же время исследователи в области теории адаптивного управления (Видроу и Хофф [2335], 1960), опираясь на работы Хебба ([997], 1949), обучали простые сети с использованием дельта-правила. Таким образом, можно сказать,

что обучение с подкреплением сочетает в себе влияние психологии животных, неврологии, исследования операций и теории оптимального управления.

Связь между обучением с подкреплением и марковскими процессами принятия решений впервые была отмечена Вербосом ([2324], 1977). (В работе Яна Уиттена (1977) языком теории управления описывается процесс, подобный обучению с учетом временной разности.) Разработка методов обучения с подкреплением в области ИИ связана в первую очередь с работами, проводившимися в университете штата Массачусетс в начале 1980-х годов (Барто и др. [139], 1981). Во влиятельной статье Рича Саттона ([2156], 1988) представлено математическое понимание методов обучения с учетом временной разности (TD). Сочетание TD-обучения с основанной на модели генерацией симулируемых экспериментов было предложено в архитектуре DYNA Саттона ([2158], 1990). Метод Q-обучения был разработан Крисом Уоткинсом и впервые представлен в его докторской диссертации ([2299], 1989), а метод SARSA впервые упоминается в техническом отчете Руммери и Ниранджана ([1934], 1994). Эвристика **приоритетной прогонки** (*prioritized sweeping*) была независимо предложена Муром и Аткесоном ([1613], 1993) и Пенгом и Уильямсом ([1771], 1993).

Использование принципа функциональной аппроксимации в области обучения с подкреплением восходит к программе игры в шашки Артура Самуэля ([1969], 1959). Использование нейронных сетей для представления функций ценности получило широкое распространение в 1980-е годы и вышло на передний план благодаря программе TD-Gammon Джерри Тезауро (Тезауро [2192], 1992; [2193], 1995). В настоящее время глубокие нейронные сети являются наиболее популярным выбором для аппроксимации функций в обучении с подкреплением. Арулкумаран и соавт. ([80], 2017) и Франсуа-Лаве и соавт. ([770], 2018) предоставили обзоры работ в области глубокого обучения с подкреплением. В системе DQN (Мних и др. [1604], 2015) глубокая нейронная сеть используется для изучения Q-функции, тогда как в системе ALPHAZERO (Сильвер и др. [2064], 2018) нейронная сеть изучает как функцию ценности для использования с известной моделью, так и Q-функцию для использования в решениях метауровня, направляющих поиск. Однако Ирпан ([1116], 2018) предупреждает, что системы глубокого обучения с подкреплением могут плохо работать, если реальная среда даже немного отличается от среды, в которой проводилось обучение.

Взвешенные линейные комбинации признаков и глубокие нейронные сети являются развернутым представлением аппроксимируемых ими функций. Но также возможно применить обучение с подкреплением и при их структурном представлении, — этот подход называется **реляционным обучением с подкреплением** (Тадепалли и др. [2169], 2004). Использование реляционных описаний позволяет обобщать сложные варианты поведения с участием различных объектов.

Тема анализа свойств сходимости алгоритмов обучения с подкреплением, использующих аппроксимацию функций, является исключительно формальной. Результаты обучения по методу учета временной разности значительно улучшаются

при использовании аппроксиматоров линейных функций (Саттон [2156], 1988; Даян [545], 1992; Цициклис и Ван Рой [2229], 1997), но для нелинейных функций было представлено несколько примеров расхождения (для ознакомления см. Цициклис и Ван Рой [2229], 1997). Папавассилиоу и Рассел в работе [1729] (1999) описывают тип обучения с подкреплением, который сходится с аппроксиматором функций при условии, что проблема подгонки гипотезы к данным разрешима. Лю и соавт. ([1434], 2018) описывают семейство алгоритмов **градиентного учета временной разности** и предоставляют обширный теоретический анализ сходимости и сложности выборки.

В статье Барто и соавт. [138] (1995) обсуждаются различные методы исследования для задач последовательного принятия решений. Кирнс и Сингх ([1207], 1998), а также Брафман и Тенненхольц ([288], 2000) описывают алгоритмы исследования неизвестных сред, которые гарантированно сходятся к почти оптимальным стратегиям с выборочной сложностью, зависящей полиномиально от количества состояний. Байесовское обучение с подкреплением (Дирден и др. [574], 1998; [573], 1999) позволяет под иным углом посмотреть и на неопределенность модели, и на исследование неизвестных сред.

Основная идея, лежащая в основе имитационного обучения, заключается в применении методов обучения с учителем с использованием обучающего набора, представляющего действия эксперта. В области адаптивного управления эта идея известна давно, но в области ИИ на нее впервые обратили внимание благодаря статье Саммута и соавт. ([1968], 1992) на тему "Учимся летать" — с использованием симулятора полета.

Свой метод они назвали **клонированием поведения**. Спустя несколько лет та же исследовательская группа сообщила, что этот метод оказался гораздо более хрупким, чем сообщалось изначально (Камачо и Мичи [359], 1995): даже очень небольшие возмущения вынуждали изученную стратегию отклоняться от желаемой траектории, приводя к накоплению ошибки, поскольку агент все дальше и дальше отклонялся от условий обучающего набора. (См. также обсуждение этой проблемы в разделе 26.8.2.) Разработка методов обучения на демонстрации имела целью сделать этот подход более надежным, в частности путем включения информации о желаемых результатах в дополнение к стратегии эксперта. Энг и соавт. ([1677], 2003), а также Коатес и соавт. ([455], 2009) показали, как методы обучения на демонстрации работают применительно к обучению управлению настоящим вертолетом, как было показано на рис. 22.9, *б* в разделе 22.7.2.

Обратное обучение с подкреплением (IRL) было предложено Расселом ([1940], 1998), а первые алгоритмы были описаны Энгом и Расселом в статье [1678] (2000). (Подобная проблема изучалась в области экономики гораздо дольше под названием **"структурная оценка MDP"** (Сарджент [1978], 1978).) Алгоритм, приведенный в этой главе, принадлежит Аббилу и Энгу ([5], 2004). Бейкер и соавт. в статье [114] (2009) описывают, каким образом понимание действий другого агента может рассматриваться как обратное планирование. Хо и соавт. ([1034], 2017)

показали, что агенты могут учиться лучше на поведении, которое *поучительно*, а не *оптимально*. Хадфилд-Менелл и соавт. ([942], 2017) распространили подход IRL на теоретико-игровую формулировку, включающую в себя как наблюдателя, так и демонстратора, показав, как преподавание и изучение поведения возникают как решения игры.

Гарсия и Фернандес ([812], 2015) дали всеобъемлющий обзор проблемы безопасного обучения с подкреплением. Мунос и соавт. ([1636], 2017) описали алгоритм безопасного исследования вне стратегии (т.е. Q-обучения). Ханс и соавт. ([958], 2008) разделили задачу безопасного исследования на две части: определение функции безопасности с целью выяснения, каких состояний следует избегать, и определение запасной стратегии для возвращения агента в безопасное состояние, когда в противном случае он может попасть в небезопасное состояние. В работе Ю и соавт. ([2409], 2017) показано, как обучить модель глубокого обучения с подкреплением водить машину в симуляторе, а затем использовать трансферное обучение для достижения безопасного вождения в реальном мире.

Томас и соавт. ([2203], 2017) предложили подход к обучению, который с высокой вероятностью гарантирует действия не хуже нынешней стратегии. Акаметалу и соавт. ([26], 2014) описывают основанный на достижимости подход, при котором процесс обучения проводится под руководством контролирующей стратегии, гарантирующей, что агент никогда не попадет в небезопасное состояние. Сондерс и соавт. ([1983], 2018) продемонстрировали, что система может использовать вмешательства человека с целью удержать ее возможный выход из безопасной области и со временем научиться действовать так, чтобы таких вмешательств требовалось все меньше.

Методы поиска стратегии вышли на передний план под влиянием исследований Вильямса ([2350], 1992), который разработал семейство алгоритмов REINFORCE. Дальнейшие исследования Марбаха и Цициклиса ([1490], 1998), Саттона и соавт. ([2157], 2000) и Бакстера и Бартлетта ([144], 2000), позволили укрепить и обобщить результаты достижения сходимости в методе поиска стратегии. Шульман и соавт. в [2006] (2015) предложили метод **оптимизации доверительной области стратегии** — теоретически обоснованный и практичный алгоритм поиска стратегии, для которого было разработано много вариантов. Метод коррелированной выборки для уменьшения дисперсии в сравнениях по методу Монте-Карло принадлежит Кану и Маршаллу ([1168], 1953); это также один из многих методов сокращения дисперсии, исследованных Хаммерсли и Хендскомбом в работе [954] (1964).

В ранних подходах к иерархическому обучению с подкреплением (HRL) предпринимались попытки построить иерархии с использованием **абстракции состояний**, т.е. группирования состояний в некоторые абстрактные состояния с последующим применением методов обучения с подкреплением в пространстве абстрактных состояний (Дайян и Хинтон [547], 1993). К сожалению, модель перехода для абстрактных состояний, как правило, оказывалась немарковской, что

препятствовало сходимости стандартных алгоритмов обучения с подкреплением. Временной подход к абстракции, рассматривавшийся в этой главе, был разработан в конце 1990-х годов (Парр и Рассел [1736], 1998; Андре и Рассел [53], 2002; Саттон и др. [2157], 2000), а затем расширен для поддержки конкурентного поведения Марти и соавт. ([1505], 2005). Диттерих в работе [619] (2000) представил понятие аддитивного разложения Q-функций, индуцированного иерархией подпрограмм. Временная абстракция основана на гораздо более раннем результате Форестье и Варайи ([758], 1978), которые показали, что большие задачи MDP могут быть разложены на систему из двух слоев, в которой контролирующий слой делает выбор среди контроллеров нижнего слоя, каждый из которых возвращает управление в контролирующий слой по завершении работы. Проблема изучения самой иерархии абстракций исследовалась по крайней мере с момента публикации работы Питера Андре ([54], 1985) и до недавнего исследования в области изучения простейших движений робота (см. статью Франса и соавт. ([773], 2018)). Игра ***keepaway*** была предложена Стоуном и соавт. ([2140], 2005), а обсуждавшееся в этой главе ее представление в виде задачи HRL принадлежит Баи и Расселу ([111], 2017).

Достижения в неврологии часто стимулировали появление новых идей в области обучения с подкреплением и подтверждали ценность этого подхода. Исследования с использованием одноклеточной регистрации показывают, что дофаминовая система в мозгу приматов реализует нечто, напоминающее изучение функции ценности (Шульц и др. [2007], 1997). В учебнике по неврологии Даяна и Эбботта ([546], 2001) описываются возможные нейронные реализации обучения по методу временной разности, а в связанных с этой темой статьях описываются другие неврологические и поведенческие эксперименты (Даян и Нив [548], 2008; Нив [1694], 2009; Ли и др. [1375], 2012).

Работа в области обучения с подкреплением была ускорена благодаря наличию инструментов моделирования с открытым исходным кодом для разработки и тестирования обучающихся агентов. Разработка Университета Альберты, программная среда Arcade Learning Environment (ALE) (Беллемар и др.[162], 2013), была выбрана для работы с 55 классическими видеоиграми приставки Atari. Пиксели на экране предоставлялись агенту как результаты восприятия наряду с текущим счетом по ходу игры. Среда ALE использовалась командой компании DeepMind для реализации обучения в системе DQN и проверки обобщения их системы на широком разнообразии игр (Мних и др. [1604], 2015).

В свою очередь, компания DeepMind перевела в разряд приложений с открытым кодом несколько своих агентских платформ, включая DeepMind Lab (Битти и др.[151], 2016), AI Safety Gridworlds (Лейке и др. [1381], 2017), игровую платформу Unity (Джулиани и др. [1160], 2018), а также DM Control Suite (Тасса и др. [2178], 2018). Компания Blizzard выпустила платформу StarCraft II Learning Environment (SC2LE), к которой компания DeepMind добавила компонент PySC2, предназначенный для реализации систем машинного обучения на языке Python (Виньялс и др. [2274], 2017).

Система эмуляции AI Habitat в Facebook (Савва и др. [1985], 2019) обеспечивает виртуальную фотореалистичную среду для внутренних роботизированных задач, а платформа этой компании HORIZON (Гаучи и др. [820], 2018) обеспечивает обучение с подкреплением в крупномасштабных промышленных системах. Система SYNTHIA (Рос и др. [1909], 2016) — это среда моделирования, предназначенная для улучшения возможностей компьютерного зрения беспилотных автомобилей. Система OpenAI Gym (Брокман и др. [311], 2016) предоставляет несколько сред для создания агентов обучения с подкреплением и совместима с другими системами моделирования, такими как симулятор Google Football.

Литтман в работе [1421] (2015) предоставляет обзор области обучения с подкреплением, предназначенный для широкой научной аудитории. В каноническом учебнике Саттона и Барто ([2159], 2018), двух пионеров в этой области, показано, как обучение с подкреплением объединяет в себе идеи обучения, планирования и действия. Кохендерфер ([1250], 2015) использует менее математический подход к этой теме, предоставляя множество реальных примеров. В небольшой книге Шепешвари ([2168], 2010) содержится общий обзор алгоритмов обучения с подкреплением. Берцекас и Цициклис ([201], 1996) дают строгое теоретическое обоснование применению динамического программирования и стохастической сходимости. Статьи по обучению с подкреплением часто публикуются в журналах *Machine Learning* и *Journal of Machine Learning Research*, а также в материалах конференций *International Conferences on Machine Learning* (ICML) и *Neural Information Processing Systems* (NeurIPS).

Упражнения

22.1. Реализуйте пассивного обучающегося агента, действующего в простой среде, такой как клеточный мир 4×3. Для случая первоначально неизвестной модели среды сравните производительность обучения с помощью алгоритмов непосредственной оценки полезности, метода временных разностей и метода адаптивного динамического программирования. Сравните оптимальные стратегии и некоторые случайно выбранные стратегии. Применительно к какой из них быстрее всего сходятся оценки полезности? Что происходит при увеличении размеров среды? (Опробуйте варианты среды с препятствиями и без препятствий.)

22.2. В главе 17 **правильная стратегия** для некоторой задачи MDP была определена как стратегия, гарантированно позволяющая достичь терминального состояния. Покажите, что для пассивного агента адаптивного динамического программирования возможна ситуация, что он найдет с помощью обучения модель перехода, для которой его стратегия π будет неприемлема, даже если эта стратегия π подходит для истинного MDP. При использовании таких моделей этап оценки стратегии может завершиться неудачей, если $\gamma = 1$. Покажите, что такая проблема не может возникнуть, если этап оценки стратегии применяется к изученной модели только в конце каждой попытки.

22.3. Начиная с проекта пассивного агента ADP, модифицируйте его так, чтобы в нем использовался приближенный алгоритм ADP, как описано в данной главе. Выполните это задание в два описанных ниже этапа.

а) Реализуйте приоритетную очередь для внесения корректировок в оценки полезностей. После корректировки данных для некоторого состояния все его предшественники также становятся кандидатами на проведение корректировки и должны быть внесены в очередь. Очередь инициализируется с учетом состояния, из которого произошел последний переход. Предусмотрите возможность использования только фиксированного количества корректировок.

б) Проведите эксперименты с использованием различных эвристик для упорядочения приоритетной очереди, исследуя их влияние на скорость обучения и продолжительность вычислений.

22.4. В методе непосредственной оценки полезности, описанном в разделе 22.2.1, используются различимые терминальные состояния для обозначения конца каждой попытки. Как можно модифицировать этот метод для применения в вариантах среды с обесцениваемыми вознаграждениями и без терминальных состояний?

22.5. Запишите уравнения обновления параметров для обучения по методу временных разностей с

$$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}.$$

22.6. Адаптируйте мир пылесоса (глава 2) для обучения с подкреплением, добавив вознаграждения за очистку квадратов. Сделайте этот мир наблюдаемым, предоставив агенту соответствующие возможности восприятия. Теперь поэкспериментируйте с разными типами агентов обучения с подкреплением. Требуется ли аппроксимация функции для достижения успеха? Какой тип аппроксиматора может применяться для этого приложения?

22.7. Реализуйте проект агента обучения с подкреплением, исследующего окружающую среду и использующего непосредственную оценку полезностей. Подготовьте две версии — с табличным представлением и с применением аппроксиматора функции, показанного в уравнении (22.9). Сравните их производительность в следующих трех вариантах среды.

а) Клеточный мир 4×3, описанный в данной главе.

б) Клеточный мир 10×10 без препятствий и с вознаграждением +1 в квадрате (10,10).

в) Клеточный мир 10×10 без препятствий и с вознаграждением +1 в квадрате (5,5).

22.8. Разработайте подходящие признаки для обучения с подкреплением в стохастических клеточных мирах (обобщениях мира 4×3), содержащих множество препятствий и множество конечных состояний с вознаграждениями +1 или −1.

22.9. Расширьте стандартную игровую среду (глава 5), чтобы включить в нее сигнал вознаграждения. Поместите в эту среду двух агентов обучения с подкреплением

(они, конечно же, могут совместно использовать программу агента) и заставьте их играть друг против друга. Примените обобщенное правило обновления TD (уравнение (22.11)) для обновления функции оценки. Возможно, вы захотите начать с простой линейной взвешенной функции оценки и простой игры, такой как крестики-нолики.

22.10. Вычислите истинную функцию полезности и наилучшую линейную аппроксимацию в точках x и y (как описано в уравнении (22.9)) для перечисленных ниже вариантов среды.

а) Клеточный мир 10×10 с единственным терминальным состоянием +1 в квадрате (10,10).

б) Мир, как в варианте *а*, но с дополнительным терминальным состоянием –1 в квадрате (10,1).

в) Мир, как в варианте *б*, но с дополнительными препятствиями в 10 квадратах, выбранных случайным образом.

г) Мир, как в варианте *б*, но со стеной, простирающейся от квадрата (5,2) до квадрата (5,9).

д) Мир, как варианте *а*, но с терминальным состоянием в квадрате (5,5).

Действия представляют собой детерминированные движения в четырех направлениях. Для каждого случая сравните результаты с использованием трехмерных графиков. Для каждой среды предложите дополнительные признаки (кроме x и y), которые позволили бы улучшить эту аппроксимацию, и продемонстрируйте полученные результаты.

22.11. Реализуйте алгоритмы REINFORCE и PEGASUS и примените их к клеточному миру 4×3, используя семейство стратегий по вашему выбору. Прокомментируйте полученные результаты.

22.12. Исследуйте задачу применения идей обучения с подкреплением для моделирования поведения людей и животных.

22.13. Может ли обучение с подкреплением служить подходящей абстрактной моделью для эволюции? Какая связь существует (если существует) между жестко закрепленными сигналами вознаграждения и эволюционной пригодностью?

Часть VI

Общение, восприятие и осуществление действий

Обработка естественного языка 283

Глубокое обучение при обработке естественного языка 341

Компьютерное зрение 381

Робототехника 447

ГЛАВА 23

Обработка естественного языка

В этой главе обсуждается, как компьютерные приложения могут использовать естественный язык для общения с людьми и учиться на том, что ими было написано.

Около 100 тыс. лет назад люди научились говорить, и около 5 тыс. лет назад они научились писать. Сложность и разнообразие человеческого языка выделяет *Homo sapiens* из всех прочих видов живых существ на нашей планете. Безусловно, имеются и другие атрибуты, уникальные именно для человека: никакие другие виды не носят одежду, не создают произведения искусства и не тратят по два часа каждый день на социальные сети так, как это делают люди. Но когда Алан Тьюринг предложил свой тест на интеллектуальность, он построил его на основе именно языка, а не способности к искусству или потребности в одежде, возможно, из-за его универсальной применимости, а также особенно четкой демонстрации интеллектуальности поведения: говорящий (или пишущий) ставит **целью** передать некоторые **знания**, затем **разрабатывает** некоторый язык, **представляющий** знания, и **действует** для достижения поставленной цели. Слушающий (или читающий) **воспринимает** язык и **делает заключение** о предполагаемом значении услышанного (или прочитанного). Подобный тип общения через язык позволил цивилизации развиваться. Это главный инструмент человечества, служащий для передачи культурных, правовых, научных и технологических знаний. Есть три основные причины, по которым компьютеры применяются для ►**обработки естественного языка** (*natural language processing* — ►**NLP**).

- **Общение с людьми.** Во многих ситуациях для взаимодействия с компьютером человеку удобнее использовать речь, и в большинстве таких ситуаций ему будет удобнее пользоваться естественным языком, а не каким-то формальным набором правил, подобным исчислению предикатов первого порядка.
- **Обучение**. Люди записали огромный объем своих знаний, используя естественные языки. Один только сайт Википедии содержит 30 млн страниц

фактов, таких как "Галаго — это небольшие ночные приматы", в то время как вряд ли существуют источники подобных сведений, написанные на языке формальной логики. Если необходимо, чтобы создаваемая система много знала, было бы лучше, чтобы она понимала естественный язык.

- Углубление **научного понимания** языков и особенностей их использования, благодаря применению инструментов ИИ в сочетании с методами и достижениями лингвистики, когнитивной психологии и нейробиологии.

В этой главе рассматриваются различные математические модели языка и обсуждаются задачи, которые могут быть решены с их помощью.

23.1. Модели языка

Формальные языки, такие как логика первого порядка, определены точно, как это было показано в главе 8. Их **грамматика** определяет синтаксис допустимых высказываний, а **семантические правила** определяют значение этих высказываний.

Любые естественные языки, такие как английский, китайский или русский, просто невозможно охарактеризовать столь же четко.

- Языковые суждения варьируются от человека к человеку и от времени до времени. Все согласятся с тем, что "Быть неприглашенным — грустно" — это грамматически правильное предложение на русском языке, но вряд ли кто-то согласится с грамматической правильностью предложения "Грустно быть не приглашать".
- Естественный язык **неоднозначен** ("Он видел свет" может означать, что он видел пятно света в темноте либо что он объездил много стран мира) и **расплывчат** ("Этого мало" — точно не указано, насколько мало, и чего, собственно, недостаточно).
- Отображение символов на объекты формально не определено. В логике первого порядка два вхождения символа "Ричард" должны ссылаться на одно и то же лицо, но на естественном языке два вхождения одного и того же слова или фразы могут относиться к совершенно разным вещам реального мира.

Если невозможно провести четкое булево различие между грамматически правильной и грамматически неправильной цепочками слов, то можно по крайней мере сказать, насколько вероятна или невероятна каждая из них.

Мы определяем ▶**модель языка** (*language model*) как распределение вероятностей, описывающее вероятность любой цепочки слов. Такая модель должна сказать, что фраза "Смею ли я тревожить вселенную?" с разумной вероятностью является цепочкой слов на русском языке, тогда как фраза "Вселенная ли смеет я тревожить?" с очень небольшой вероятностью является такой цепочкой.

С помощью модели языка можно предсказать, какие слова, вероятно, появятся следующими в тексте, и тем самым предложить возможное продолжение для электронного письма или текстового сообщения. Также можно вычислить, какие изменения в тексте сделают его более вероятным, а следовательно, предложить необходимые исправления с точки зрения правописания или грамматики. При наличии соответствующей пары моделей можно вычислить наиболее вероятный перевод предложения. Воспользовавшись несколькими примерами пар "вопрос/ответ" в качестве обучающего набора, можно будет вычислить наиболее вероятный ответ на вопрос. Таким образом, модели языка являются основой для решения широкого диапазона задач обработки естественного языка. Саму задачу построения языковой модели также можно считать общим ориентиром для измерения прогресса в понимании языка.

Естественные языки сложны, поэтому любая модель языка будет в лучшем случае лишь приблизительной. Лингвист Эдвард Сепир сказал: "Ни один язык не является тиранически последовательным. Все грамматики имеют лазейки" (Сепир [1976], 1921). Философ Дональд Девидсон сказал: "Нет такой вещи, как язык, если понимать язык как... четко определенную общепринятую структуру" (Девидсон [532], 1986), — здесь он имел в виду, что не существует единой строго определенной модели английского языка (как, например, для языка Python 3.8): каждый имеет собственную, отличную от других модель, но людям все же как-то удается преодолевать эти различия и общаться друг с другом. В этом разделе будут рассмотрены упрощенные языковые модели, которые явно ошибочны, но могут быть полезны при решении определенных задач.

23.1.1. Модель мешка слов

В разделе 12.6.1 объясняется, как наивная байесовская модель, основанная на присутствии конкретных слов, позволяет надежно классифицировать предложения по категориям, например первое предложение из приведенных ниже относится к категории "бизнес", а второе — к категории "погода".

1. В понедельник стоимость акций возросла: основные индексы прибавили 1%, поскольку сохраняется оптимизм в отношении сезона отчетности за первый квартал.
2. В понедельник проливные дожди по-прежнему охватывают бо́льшую часть восточного побережья, из-за чего в городе Нью-Йорк и других местах были сделаны предупреждения о возможности наводнения.

В данном разделе эта наивная байесовская модель рассматривается еще раз, но уже в качестве полной модели языка. Это означает, что недостаточно просто установить, какая категория с наибольшей вероятностью подходит для каждого предложения, — требуется получить совместное распределение вероятностей по всем предложениям и категориям. А это говорит о том, что нужно рассматривать *все*

слова в предложении. Пусть дано предложение, состоящее из слов w_1, w_2,... w_N (которое мы обозначим как $w_{1:N}$, как в главе 14), тогда согласно наивной байесовской формуле (уравнение (12.21)) получим

$$\mathbf{P}(Class \mid w_{1:N}) = \alpha \mathbf{P}(Class) \prod_j \mathbf{P}(w_j \mid Class).$$

Применение наивного байесовского алгоритма к цепочкам слов называется ▶ **моделью мешка слов** (*bag-of-words model*). Это порождающая модель, описывающая процесс генерации предложения: представим, что для каждой категории (*бизнес*, *погода* и т.д.) есть мешок со словами (для наглядности можно представить, что каждое слово написано на листке бумаги и помещено в мешок, — чем чаще слово встречается, тем больше листков с этим словом находится в мешке). Чтобы сгенерировать текст, сначала выбираем один из мешков, а остальные отставляем в сторону. Запускаем в мешок руку и наугад вытаскиваем листок с каким-то словом, — это будет первое слово в предложении. Возвращаем листок в мешок, а затем извлекаем следующее слово. Эта процедура повторяется до тех пор, пока не будет извлечен указатель конца предложения (например, точка).

Эта модель явно неверна: в ней ошибочно предполагается, что каждое слово не зависит от других, и следовательно, она не способна генерировать связные предложения на естественном языке. Но она позволяет с хорошей точностью выполнить классификацию на основании наивной байесовской формулы: наличие слов "акции" и "доходы" является хорошим свидетельством категории "бизнес", в то время как слова "дождь" и "облачно" указывают на категорию "погода".

Априорные вероятности, необходимые для этой модели, можно изучить методами обучения с учителем на основании массива или ▶ **корпуса** текста, в котором каждый сегмент текста помечен соответствующим классом. Корпус обычно включает не менее миллиона слов текста, и его словарь насчитывает не менее десятков тысяч различных слов. В последнее время наблюдается тенденция к использованию еще бо́льших корпусов, подобных корпусу из 2,5 млрд слов сайта Википедия или корпусу iWeb с 14 млрд слов, собранных с 22 млн веб-страниц.

На базе корпуса для каждой категории $\mathbf{P}(Class)$ можно оценить ее априорную вероятность путем подсчета, насколько часто она встречается. Подсчет также можно использовать для оценки условной вероятности появления каждого слова в данной категории, $\mathbf{P}(w_j \mid Class)$. Например, если просмотрено 3 тыс. текстов и из них 300 были классифицированы как "бизнес" (*business*), то можно оценить априорную вероятность $P(Class = business) \approx 300/3000 = 0{,}1$. Далее, если во всех текстах категории *business* было 100 000 слов, а слово "акции" (*stocks*) появилось 700 раз, то можно оценить условную вероятность $P(stocks \mid Class = business) \approx 700/100\,000 = 0{,}007$. Оценка подсчетом работает хорошо, когда есть большое количество подсчитываемых экземпляров (и низкая дисперсия), но если подсчитываемые количества невелики, то лучше обратиться к другому способу оценки вероятностей, который будет представлен в разделе 23.1.4.

Иногда иной подход к машинному обучению, такой как логистическая регрессия, нейронные сети или метод опорных векторов, может работать даже лучше, чем наивный байесовский. Признаками в модели машинного обучения являются отдельные слова словаря: "a", "aardvark",..., "zyzzyva", а их значениями — количество раз, когда каждое слово появляется в тексте (или иногда просто булево значение, указывающее, появляется слово в тексте или нет). Это делает вектор признаков большим и разреженным — в языковой модели может быть 100 000 слов, и следовательно, вектор признаков будет иметь размерность 100 000, но для короткого текста почти все признаки будут равны нулю.

Как было показано ранее, некоторые модели машинного обучения работают лучше, когда выполняется **отбор признаков**, в данном случае — ограничение общего количества слов, используемых как признаки. Можно отбросить слова, которые встречаются очень редко (а следовательно, имеют высокую дисперсию в их прогнозирующей способности), а также слова, которые будут общими для всех категорий (например, "the") и не способствуют их различению. Также можно смешивать другие признаки с признаками, представляющими слова; например, если классифицируются сообщения электронной почты, то можно добавить признаки для отправителя, времени отправки сообщения, слов в заголовке сообщения, наличия нестандартной пунктуации, процента прописных букв, наличия вложений и т.д.

Обратите внимание, что принятие решения, что именно считать *словом*, также является нетривиальной задачей. Является ли "aren't" одним словом, или оно должно быть разбито на "aren/'/t", "are/n't" или как-то еще? Процесс разбиения текста на последовательность слов называется ▶**токенизацией**.

23.1.2. *N*-граммные модели слов

Модели "мешок слов" свойственны ограничения. Например, слово "quarter" (*четверть*) распространено как в бизнес-текстах, так и в спортивной категории. Но последовательность из четырех слов "first quarter earnings report" (*отчет о доходах первого квартала*) обычна только для категории *business*, а последовательность "fourth quarter touchdown passes" (*голевые пасы в четвертой четверти*) характерна только в категории *sports*. Желательно, чтобы в модели учитывались эти различия. Можно, конечно, настроить модель типа "мешок слов" так, чтобы она рассматривала некоторые специальные фразы, подобные "first quarter earnings report", как если бы это было одно слово, но более принципиальным подходом является введение новой модели, в которой каждое слово зависит от предыдущих слов. Можно начать с того, чтобы сделать каждое слово зависимым от *всех* предыдущих слов в предложении:

$$P(w_{1:N}) = \prod_{j=1}^{N} P(w_j \mid w_{1:j-1}).$$

Данная модель в некотором смысле является совершенно "правильной" в том отношении, что она охватывает все возможные взаимодействия между словами, но это непрактично: при словаре в 100 000 слов и длиной предложений в 40 слов такая модель будет иметь 10^{200} оцениваемых параметров. Используя модель **цепи Маркова**, можно пойти на компромисс, учитывая зависимости только между n смежными словами. Этот подход известен как ▶ ***n*-граммная модель** (от греческого корня *грамма*, означающего "написанное"): последовательность записанных символов длиной n называется n-граммой, а ее специальные случаи — "униграмма" для 1-граммы, "биграмма" — для 2-граммы и "триграмма" — для 3-граммы. В n-граммной модели вероятность появления каждого слова зависит только от $n-1$ предыдущих слов, т.е.

$$P(w_j \mid w_{1:j-1}) = P(w_j \mid w_{j-n+1:j-1})$$

$$P(w_{1:N}) = \prod_{j=1}^{N} P(w_j \mid w_{j-n+1:j-1}).$$

N-граммные модели хорошо подходят для классификации газетных разделов, а также для других задач классификации, таких как ▶ **обнаружение спама** — различение спама и не спама, ▶ **анализ тональности текста** (*sentiment analysis*) — классификация обзора фильма или некоторого продукта как положительного или отрицательного и ▶ **определения авторства** (*author attribution*) — литературный стиль и словарный запас Хемингуэя отличается от аналогичных характеристик Фолкнера или Шекспира.

23.1.3. Другие *n*-граммные модели

Альтернативой n-граммным моделям слов является ▶ **модель уровня символов** (*character-level model*), в которой вероятность появления каждого символа определяется $n-1$ предыдущими символами. Этот подход полезен для работы с неизвестными словами и для языков, в которых слова часто объединяются с образованием новых слов, например как в случае датского слова "Specialлægepraksis planlægningsstabiliseringsperiode".

Модели уровня символов хорошо подходят для задач ▶ **определения языка**: получив новый текст, необходимо определить, на каком языке он написан. Даже при очень коротких текстах, таких как "Hello, world" или "Wie geht's dir", n-граммная модель уровня символов позволяет идентифицировать первый язык как английский, а второй — как немецкий, обеспечивая в общем случае точность выше 99%. (Близкородственные языки, такие как шведский и норвежский, различать сложнее, поэтому в подобных случаях потребуются более длинные образцы текста, а достигаемая точность будет примерно на уровне 95%.) Модели уровня символов также хорошо подходят для определенных задач классификации, например для определения, что "dextroamphetamine" является названием препарата,

"Kallenberger" — фамилией человека, а "Plattsburg" — названием города, даже если в обработанных текстах эти слова ранее никогда не встречались.

Другой возможностью является ▶**модель скип-граммы**, в которой подсчитываются слова, находящиеся близко друг к другу, и не учитывается слово (или больше слов), стоящее между ними. Например, для фразы на французском языке "je ne comprends pas" 1-скип-биграммами будут "je comprends" и "ne pas". Сбор подобных скип-грамм помогает создать лучшую модель французского языка, поскольку они предоставляют сведения о правилах спряжения глаголов ("je" употребляется с "comprends", а не "comprend") и построения оборотов отрицания ("ne" идет с "pas"), — из обычных биграмм эту информацию извлечь нельзя.

23.1.4. Сглаживание *n*-граммных моделей

Высокочастотные *n*-граммы, такие как "of the", имеют большие значения счетчиков вхождения в обучающих корпусах, поэтому оценка их вероятности, скорее всего, будет точной: в другом обучающем корпусе будет получена аналогичная оценка. Низкочастотные *n*-граммы имеют малые значения счетчиков вхождения, подверженные случайному шуму, — они характеризуются высокой дисперсией. Построенные модели будут работать лучше, если удастся каким-то образом сгладить эти различия.

Кроме того, всегда есть шанс, что потребуется оценить текст, содержащий неизвестное слово, т.е. слово ▶**вне словаря** (*out-of-vocabulary*), которое никогда не появлялось в обучающем корпусе. Но было бы ошибкой назначать такому слову нулевую вероятность, поскольку в этом случае вероятность всего предложения, $P(w_{1:N})$, также будет нулевой.

Один из способов моделирования неизвестных слов — модифицировать обучающий корпус, заменив редкие слова специальным символом (по традиции это `<UNK>`). Можно заранее решить оставить в словаре только, скажем, 50 тыс. наиболее распространенных слов или же все слова с частотой, большей 0,0001%, а все прочие заменить символом `<UNK>`. Затем обычным способом для корпуса вычисляются счетчики вхождения *n*-грамм, при этом символ `<UNK>` воспринимается так же, как и любое другое слово, т.е. когда в тестовом наборе появляется неизвестное слово, его вероятность определяется как вероятность символа `<UNK>`. Иногда для разных типов неизвестных слов используются разные символы-заменители. Например, любую строку цифр можно заменить символом `<NUM>`, а адрес электронной почты — символом `<EMAIL>`. (Отметим, что желательно также иметь специальный символ, например `<S>`, чтобы отмечать начало (и конец) отдельного текста. В результате, когда в формуле расчета вероятностей биграмм потребуется выбрать слово перед первым словом в тексте, ответом будет `<S>`, а не сообщение об ошибке.)

Но даже после того, как неизвестные слова были соответствующим образом обработаны, остается проблема не встречавшихся ранее *n*-грамм. Например, тестовый текст может содержать фразу "бесцветные аквамариновые идеи", — это три

слова, которые в обучающем корпусе могли присутствовать по отдельности, но пока не встречались именно в таком порядке. Проблема заключается в том, что некоторые маловероятные n-граммы присутствуют в обучающем корпусе, тогда как других маловероятных n-грамм в нем нет. Нежелательно, чтобы одни из них получали нулевую вероятность, в то время как другие будут иметь небольшую положительную вероятность. Следовательно, требуется применить ►**сглаживание** ко всем подобным n-граммам, — зарезервировав некоторую часть общей вероятностной массы модели для неизвестных n-грамм, чтобы уменьшить ее дисперсию.

Самый простой тип сглаживания был предложен Пьером-Симоном Лапласом еще в XVIII веке для оценки вероятности редких событий, таких как то, что солнце завтра не взойдет. Лаплас придерживался (неверной) теории о возрасте Солнечной системы, в которой предполагалось, что ей около $N = 2$ млн дней. Судя по этим данным, было нуль из двух миллионов дней, когда солнце не всходило, но мы не хотим сказать, что вероятность такого события равна нулю. Лаплас показал, что если принять равномерное априорное распределение и объединить его с имеющимися на текущий момент свидетельствами, то $1/(N+2)$ будет наилучшей оценкой вероятности того, что солнце завтра не взойдет, — либо да, либо нет (это 2 в знаменателе), а равномерное априорное распределение говорит, что это событие одинаково вероятно и невероятно (это 1 в числителе). Сглаживание по Лапласу (основная идея которого — принять, что событие наблюдалось на один раз больше) — это шаг в правильном направлении, но для многих приложений обработки естественного языка оно работает плохо.

Другой выбор — это ►**модель с откатом** (*backoff model*), в которой работа начинается с оценки количества n-грамм, но для любой конкретной последовательности с низким (или нулевым) счетчиком выполняется возврат к $(n-1)$-граммам. ►**Линейное интерполяционное сглаживание** представляет собой модель с откатом, объединяющую триграммную, биграммную и униграммную модели с помощью линейной интерполяции. В этой модели оценка вероятности определяется как

$$\hat{P}(c_i \mid c_{i-2:i-1}) = \lambda_3 P(c_i \mid c_{i-2:i-1}) + \lambda_2 P(c_i \mid c_{i-1}) + \lambda_1 P(c_i),$$

где $\lambda_3 + \lambda_2 + \lambda_1 = 1$. Значения параметров λ_i могут быть фиксированными или их можно обучить с помощью алгоритма максимизации ожиданий. Также возможно, чтобы значения коэффициентов λ_i зависели от значений счетчиков: если количество триграмм велико, то их вес будет относительно больше, а если же мало, то бо́льшие веса присваиваются биграммам и униграммам.

Один лагерь исследователей разработал еще более сложные методы сглаживания (например, методы Виттена–Белла и Кнезера–Нея), тогда как другой лагерь предлагает просто собирать корпуса большего размера — такие, чтобы даже простые методы сглаживания хорошо работали (один такой подход называется "глупым откатом" — *stupid backoff*). Однако оба лагеря стремятся к достижению одной и той же цели: уменьшить дисперсию в модели языка.

23.1.5. Представление слов

N-граммы позволяют получить модель, которая будет точно предсказывать вероятности появления последовательностей слов. Такая модель укажет, например, что "a black cat" с большей вероятностью является фразой на английском языке, чем "cat black a", потому что фраза "a black cat" составляет примерно 0,000014% всех триграмм в обучающем корпусе, тогда как фраза "cat black a" вообще в нем отсутствует. Все, что знает *n*-граммная модель слов, было изучено на основании подсчета появления определенных последовательностей слов.

Однако носитель английского языка мог бы сказать об этом иначе: "a black cat — *какая-то черная кошка*" действительно является фразой на английском языке, потому что она построена по принятому в этом языке шаблону (артикль–прилагательное–существительное), тогда как фраза "cat black a" включает те же слова, но этому шаблону не соответствует.

А теперь рассмотрим фразу "the fulvous kitten" (*этот рыжеватый котенок*). Носитель английского языка легко может распознать здесь тот же шаблон "артикль–прилагательное–существительное" (даже тот, кто не знает, что слово "fulvous" означает "рыжеватый", сможет его распознать, поскольку знает, что почти все слова английского языка, оканчивающиеся на "-ous", являются прилагательными). Более того, говорящий на английском легко распознает тесную синтаксическую связь между артиклями "a" и "the", а также тесную семантическую связь между словами "cat" — *кошка* и "kitten" — *котенок*. Следовательно, появление фразы "a black cat" в данных является свидетельством, через обобщение, что "the fulvous kitten" также является фразой на правильном английском языке.

Модель *n*-грамм не позволяет сделать такое обобщение, потому что это *атомарная* модель: в ней каждое слово рассматривается как неделимый атом, отличный от любого другого слова, и без какой-либо внутренней структуры. В этой книге уже неоднократно указывалось, что *развернутое* или *структурное* представление данных модели обеспечивают большую выразительную силу и лучшее обобщение. В разделе 24.1 будет показано, что модель с развернутым представлением данных, называемая **встраиванием слов**, предоставляет лучшие возможности обобщения.

Одним из типов моделей со структурным представлением слов является ▶**словарь**, обычно создаваемый вручную. Например, ▶**WordNet** — это вручную собранный словарь с открытым исходным кодом в формате, удобном для машинного прочтения, который оказался полезным для многих приложений обработки естественного языка.[1] Ниже представлена статья словаря WordNet для слова "kitten":

[1] И даже для приложений компьютерного зрения: словарь WordNet предоставляет набор категорий, используемых в системе ImageNet.

```
"kitten" <noun.animal> ("young domestic cat")
  IS A: young_mammal

"kitten" <verb.body> ("give birth to kittens")
  EXAMPLE: "our cat kittened again this year"
```

Словарь WordNet позволяет отделить существительные от глаголов и получить основные категории (котенок — молодое млекопитающее (*young mammal*), которое является млекопитающим (*mammal*), которое является животным (*animal*)), но он не содержит детальных сведений о том, как котенок выглядит или как он действует. В словаре WordNet есть информация, что существует два подкласса кошек — сиамская кошка (*Siamese cat*) и мэнская кошка (*Manx cat*), но нет никакой дополнительной информации об их породах.

23.1.6. Маркировка частей речи

Одним из основных способов классификации слов является их распределение по ▶**частям речи** (*part of speech* — ▶**POS**), также называемым **лексическими категориями** или **тегами**: существительное (*noun*), глагол (*verb*), прилагательное (*adjective*) и т.д. Части речи позволяют языковым моделям выявлять такие обобщения, как "в английском языке прилагательные обычно предшествуют существительным". (В других языках, например в французском, все, как правило, наоборот, а в таких языках, как русский, возможны оба варианта.)

Все согласны с тем, что "существительное" и "глагол" являются частями речи, но если углубиться в детали, то однозначного списка не существует. На рис. 23.1 показаны 45 тегов, используемых в наборе данных ▶**Penn Treebank**, — корпусе более чем из трех миллионов слов текста, помеченных тегами части речи. Как будет показано ниже, в корпусе Penn Treebank много предложений аннотированы деревьями синтаксического анализа, от чего, собственно, этот корпус и получил свое имя. Вот выдержка из этого корпуса, указывающая, что слово "from" (*от*) помечено как предлог (IN), слово "the" (*определенный артикль*) — как определитель (DT) и т.д.

From	the	start	,	it	took	a	person	with	great	qualities	to	succeed
IN	DT	NN	,	PRP	VBD	DT	NN	IN	JJ	NNS	TO	VB

Присвоение пометки (тега) части речи каждому слову в предложении называют ▶**маркировкой частей речи** (*part-of-speech tagging*). Хотя это занятие и не очень интересно само по себе, это полезный первый этап во многих других задачах обработки естественного языка (NLP), таких как предоставление ответов на вопросы или перевод. Даже для такой простой задачи, как речевое воспроизведение текста, важно знать, что существительное "record" (*запись*) произносится иначе, чем глагол "record" (*записывать*). В этом разделе будет показано, как две уже знакомые нам модели могут быть применены для решения задачи маркировки частей речи, а в главе 24 будет рассмотрена третья модель.

Тег	Слово	Описание	Тег	Слово	Описание
CC	*and*	Coordinating conjunction	PRP$	*your*	Possessive pronoun
CD	*three*	Cardinal number	RB	*quickly*	Adverb
DT	*the*	Determiner	RBR	*quicker*	Adverb, comparative
EX	*there*	Existential there	RBS	*quickest*	Adverb, superlative
FW	*per se*	Foreign word	RP	*off*	Particle
IN	*of*	Preposition	SYM	+	Symbol
JJ	*purple*	Adjective	TO	*to*	to
JJR	*better*	Adjective, comparative	UH	*eureka*	Interjection
JJS	*best*	Adjective, superlative	VB	*talk*	Verb, base form
LS	1	List item marker	VBD	*talked*	Verb, past tense
MD	*should*	Modal	VBG	*talking*	Verb, gerund
NN	*kitten*	Noun, singular or mass	VBN	*talked*	Verb, past participle
NNS	*kittens*	Noun, plural	VBP	*talk*	Verb, non-3rd-sing
NNP	*Ali*	Proper noun, singular	VBZ	*talks*	Verb, 3rd-sing
NNPS	*Fords*	Proper noun, plural	WDT	*which*	Wh-determiner
PDT	*all*	Predeterminer	WP	*who*	Wh-pronoun
POS	*'s*	Possessive ending	WP$	*whose*	Possessive wh-pronoun
PRP	*you*	Personal pronoun	WRB	*where*	Wh-adverb
$	$	Dollar sign	#	#	Poundsign
"	'	Left quote	"	'	Rightquote
(	[	Left parenthesis	)	]	Rightparenthesis
,	,	Comma	.	!	Sentenceend
:	;	Mid-sentence punctuation			

Рис. 23.1. Теги частей речи (со словом-примером для каждого тега) в корпусе Penn Treebank (Маркус и др. [1492], 1993). Здесь "3rd-sing" является аббревиатурой для "third person singular present tense" (*третье лицо единственное число настоящее время*)

Одной из популярных моделей маркировки частей речи является **скрытая марковская модель (HMM)**. Напомним из материала главы 14, что скрытая марковская модель принимает временную последовательность данных свидетельства и предсказывает, какие наиболее вероятные скрытые состояния могли бы произвести эту последовательность. В примере HMM, приведенном в разделе 14.3, свидетельства представляли собой результаты наблюдения за человеком, приехавшим с зонтиком (или нет), а скрытое состояние представляло дождь (или его отсутствие) во внешнем мире. Для задачи маркировки частей речи свидетельством является последовательность слов $W_{1:N}$, а скрытыми состояниями — лексические категории $C_{1:N}$.

Скрытая марковская модель является порождающей моделью, которая указывает, что для производства языковой конструкции необходимо начать с первого состояния, такого как IN, состояния предлога, а затем сделать два выбора: какое

слово (например, предлог "from" — *из*) должно быть взято и какое состояние (например, DT) будет следующим. Модель не рассматривает какой-либо иной контекст, кроме текущего состояния процесса маркировки частей речи, и не имеет никакого представления о том, что это предложение в действительности пытается передать. И все же это полезная модель — если для поиска наиболее вероятной последовательности скрытых состояний (тегов) применить ►**алгоритм Витерби** (раздел 14.2.3), то можно обнаружить, что присвоение тегов словам достигает очень высокой точности, обычно около 97%.

Чтобы создать скрытую марковскую модель для маркировки частей речи, необходимы модель перехода, предоставляющая вероятность следования одной части речи за другой, $\mathbf{P}(C_t \mid C_{t-1})$, и модель восприятия, $\mathbf{P}(W_t \mid C_t)$. Например, $\mathbf{P}(C_t = VB \mid C_{t-1} = MD) = 0{,}8$ означает, что для данного модального глагола (например, "would" — *будет*) можно с вероятностью 0,8 ожидать, что следующее слово будет глаголом (например, "think" — *думать*). Но откуда взялось значение 0,8? Как и в случае *n*-граммных моделей, — из подсчетов вхождений в корпусе с соответствующим сглаживанием. Оказывается, что в корпусе Penn Treebank имеется 13 124 экземпляра слов с тегом *MD*, и следующим тегом для 10 471 из них является *VB*.

Что касается модели восприятия, то $P(W_t = would \mid C_t = MD) = 0{,}1$ означает, что при выборе модального глагола в 10% случаев выбирается глагол "would". Эти цифры также взяты из результатов подсчетов в корпусе со сглаживанием.

Слабость скрытых марковских моделей в том, что все, что известно о языке, должно быть представлено в виде модели перехода и модели восприятия. Часть речи для текущего слова здесь определяется исключительно вероятностями в этих двух моделях и частью речи предыдущего слова. Например, у разработчика системы нет простого способа указать, что *любое* слово, которое оканчивается на "-ous", скорее всего, является прилагательным, а также что в сочетании "attorney general" (*генеральный прокурор*) слово "attorney" является существительным, а не прилагательным.

К счастью, представить такую информацию позволяют модели **логистической регрессии**. Напомним из раздела 19.6.5, что в модели логистической регрессии вход представляет собой вектор значений признаков $\mathbf{x}$. Затем берется точечное произведение $\mathbf{w} \cdot \mathbf{x}$ вектора этих признаков и вектора предварительно обученных весов $\mathbf{w}$ с последующим преобразованием этой суммы в число от 0 до 1, которое интерпретируется как вероятность того, что вход является положительным примером категории.

Веса в модели логистической регрессии соответствуют тому, насколько важным каждый признак является для каждой прогнозируемой категории, при этом сами значения весов обучаются по методу градиентного спуска. Для задачи маркировки частей речи строится 45 различных моделей логистической регрессии, по одной для каждой части речи, а затем каждой модели задается вопрос, насколько вероятно, что данный пример слова является членом этой категории, с учетом значений признаков для этого слова в его конкретном контексте.

Теперь возникает вопрос, а какими должны быть эти признаки? Для тегов частей речи обычно используют признаки с булевыми значениями, кодирующими информацию о помечаемом слове w_i (и, возможно, других близлежащих словах), а также категорию, которая была назначена предыдущему слову c_{i-1} (и, возможно, категории стоящих перед ним слов). Признаки могут зависеть от точной идентичности слова, некоторых аспектов его написания или каких-либо атрибутов из его словарной статьи. При маркировке частей речи тегами множество используемых признаков может включать следующие:

w_{i-1} = "I"	w_{i+1} = "for"
w_{i-1} = "you"	c_{i-1} = IN
w_i оканчивается на "ous"	w_i содержит дефис
w_i оканчивается на "ly"	w_i содержит цифру
w_i начинается с "un"	w_i все буквы прописные
w_{i-2} = "to" и c_{i-1} = VB	w_{i-2} имеет атрибут PRESENT
w_{i-1} = "I" и w_{i+1} = "to" = VB	w_{i-2} имеет атрибут PAST

Например, слово "walk" может быть существительным (*прогулка*) или глаголом (*ходить*), но для фразы "I walk to school," признак из последней строки левого столбца приведенной выше таблицы позволяет классифицировать слово "walk" как глагол (VB). Другой пример: слово "cut" может быть существительным *порез* (NN), глаголом прошедшего времени *резал* (VBD) или глаголом настоящего времени *режу* (VBP). В случае фразы "Yesterday I cut the rope" признак из последней строки правого столбца приведенной выше таблицы (первое слово фразы "yesterday" (*вчера*) имеет атрибут PAST) позволяет классифицировать слово "cut" как глагол прошедшего времени (VBD), а в предложении "Now I cut the rope" признак в предпоследней строке этого столбца (первое слово фразы "now" (*сейчас*) имеет атрибут PRESENT) позволяет классифицировать слово "cut" как глагол настоящего времени (VBP).

В целом в модели может использоваться миллион признаков, но для каждого конкретного слова только несколько десятков из них будут ненулевыми. Признаки, как правило, определяются вручную разработчиком системы, который придумывает интересные для него шаблоны признаков.

Для логистической регрессии не существует понятия последовательности входных данных — на вход подается единственный вектор признаков (информация об одном слове), и модель выдает для него результат (тег). Но есть возможность заставить логистическую регрессию обрабатывать последовательность входов с помощью **жадного поиска**: начните с выбора наиболее вероятной категории для первого слова, а затем переходите к остальным словам в порядке слева направо. На каждом этапе категория c_i присваивается в соответствии с формулой

$$c_i = \underset{c' \in Categories}{\operatorname{argmax}} P(c' \mid w_{1:N}, c_{1:i-1}).$$

То есть классификатору разрешено просматривать любые признаки, не относящиеся к категории, для любого слова в любом месте предложения (поскольку все эти признаки фиксированы), а также любые ранее присвоенные категории.

Обратите внимание, что жадный поиск делает окончательный выбор категории для каждого слова, а затем переходит к следующему слову. Если этот выбор противоречит свидетельствам далее в предложении, нет возможности вернуться назад и отменить выбор. Это делает данный алгоритм очень быстрым. Алгоритм Витерби, напротив, ведет таблицу всех возможных вариантов категорий на каждом этапе, и у него всегда есть возможность внести требуемые изменения. В результате данный алгоритм является более точным, но гораздо более медленным. Для обоих алгоритмов компромиссом является **поиск по лучу**, при котором рассматриваются все возможные категории на каждом временном этапе, но затем из них сохраняется только b наиболее вероятных тегов с отбрасыванием всех остальных, менее вероятных. Изменение параметра b определяет компромисс между скоростью работы и точностью алгоритма.

Наивные байесовские и скрытые марковские модели являются **порождающими моделями** (см. раздел 20.2.3). Это означает, что они изучают совместное распределение вероятностей $\mathbf{P}(W, C)$ и предоставляют возможность генерировать случайные предложения посредством выборки из этого распределения вероятностей, сначала — для получения первого слова предложения (с категорией), а затем — для добавления в него слов по одному за раз.

С другой стороны, логистическая регрессия является **дискриминативной моделью**. Она изучает условное распределение вероятностей $\mathbf{P}(C \mid W)$, а это означает, что она может присваивать категории с учетом последовательностей слов, но при этом не позволяет генерировать случайные предложения. В целом исследователи обнаружили, что дискриминативные модели обладают более низким уровнем ошибок, — возможно, потому, что они моделируют предполагаемый результат напрямую и упрощают аналитикам создание дополнительных признаков. Однако порождающие модели имеют тенденцию сходиться быстрее и поэтому могут оказаться предпочтительнее, когда доступное для обучения время невелико или когда объем обучающих данных ограничен.

23.1.7. Сравнение языковых моделей

Чтобы понять, на что похожи разные n-граммные модели языка, были созданы униграммная (т.е. *мешок слов*), биграммная, 3-граммная и 4-граммная модели для слов, содержащихся в этой книге, а затем случайным образом выбраны последовательности слов из каждой из четырех моделей.

1. $n = 1$: *logical are as are confusion a may right tries agent goal the was*
   (логичный есть как есть путаница может правильная попытка агент цель этот был)

2. $n = 2$: *systems are very similar computational approach would be represented*
   (системы очень похожи вычислительный подход был бы представлен)
3. $n = 3$: *planning and scheduling are integrated the success of naive Bayes model is*
   (планирование и составление графиков интегрированы успех наивной байесовской модели есть)
4. $n = 4$: *taking advantage of the structure of Bayesian networks and developed various languages for writing "templates" with logical variables, from which large networks could be constructed automatically for each problem instance*
   (использование преимуществ структуры байесовских сетей и развитие различных языков для написания "шаблонов" с логическими переменными, на основании которых можно автоматически построить большие сети для каждого экземпляра проблемы)

Из этой небольшой выборки вполне очевидно, что униграммная модель является очень плохим приближением как для английского языка в целом, так и для учебника по ИИ в частности, а также что 4-граммная модель, пусть тоже несовершенная, все же намного лучше. Далее, с целью продемонстрировать, как резко изменяются выборки в зависимости от источника обучения (и определенно, не просто ради развлечения), к нашей 4-граммной модели был добавлен текст Библии короля Иакова, после чего получены следующие случайные выборки.

- *Prove that any 3-SAT problem can be reduced to simpler ones using the laws of thy God.*
  (Докажите, что любая проблема 3-SAT может быть уменьшена до более простой, используя законы твоего Бога.)
- *Masters, give unto your servants that which is true iff both P and Q in any model m by a simple experiment: put your hand unto, ye and your households for it is pleasant.*
  (Мастера, дайте своим слугам то, что верно, если и P, и Q в любой модели m простым экспериментом: приложите свою руку к себе и своим домочадцам, потому что это приятно.)
- *Many will intreat the LORD your God, Saying, No; but we will ignore this issue for now; Chapters 7 and 8 suggest methods for compactly representing very large belief states.*
  (Многие будут умолять Господа, Бога твоего, говоря: нет; но мы пока проигнорируем эту проблему; в главах 7 и 8 предлагаются методы компактного представления очень больших доверительных состояний.)
- *And it came to pass, as if it had no successors.*
  (И это случилось, как будто у него не было преемников.)
- *The direct utility estimation is just an instance of the general or algorithm in which new function symbols are constructed "on the fly." For example, the first child of the Holy Ghost.*

(Прямая оценка полезности есть просто пример общего или алгоритма, в котором новые функциональные символы создаются "на лету". Например, первое дитя Святого Духа.)

Для *n*-граммных моделей существует определенное ограничение — при увеличении *n* они будут генерировать все более правильный язык, но при этом проявят тенденцию воспроизводить длинные отрывки из своих обучающих данных дословно, вместо того чтобы генерировать новый текст. Языковые модели с более сложным представлением слов и контекста показывают лучшие результаты. В оставшейся части этой главы демонстрируется, как модель языка можно улучшить за счет включения в нее **грамматики**, а в главе 24 речь идет о недавно разработанных методах глубокого обучения, позволяющих создавать впечатляющие языковые модели. Одна из таких моделей глубокого обучения, GPT-2, позволяет генерировать образцы беглого английского языка при получении подсказки. После введения в модель GPT-2 первых двух предложений этого абзаца в качестве подсказки она воспроизвела два таких образца.

- *Numerous studies have shown that the best language models that can generate language are those that use neural networks, and in particular that use deep learning to model speech. I have used one of these models to generate hundreds of sentences in multiple languages, and it has the ability to generate almost perfect translations of my training sentences. It's a great model to have, and will save you a lot of time in the future when you need to translate between many languages.*
  (Многочисленные исследования показали, что лучшими языковыми моделями, которые могут генерировать язык, являются те, которые используют нейронные сети, и, в частности, те, которые используют глубокое обучение для моделирования речи. Я использовал одну из этих моделей для создания сотен предложений на многих языках, и она обладает способностью генерировать почти идеальные переводы моих обучающих предложений. Эту отличную модель стоит иметь, она сэкономит вам много времени в будущем, когда вам потребуется переводить между многими языками.)
- *These models can also do better if their samples are chosen at random, rather than by an expert in the domain. However, as a general rule, n-gram and language models that include a large number of features tend to produce text better, since these features contribute the most to language learning. To measure the performance of a model, I've made some sample sentences that I'd like to train to better understand speech and language models. You can read them to get a grasp of how the model predicts.*
  (Эти модели также могут работать лучше, если их выборки выбираются случайным образом, а не экспертом в предметной области. Однако, как правило, *n*-граммные и языковые модели, которые включают большое количество признаков, имеют тенденцию производить текст лучше, поскольку эти признаки вносят наибольший вклад в изучение языка. Чтобы измерить произво-

дительность модели, я сделал несколько образцов предложений, которые я хотел бы изучить для лучшего понимания речевых и языковых моделей. Вы можете прочитать их, чтобы получить понимание того, как модель предсказывает.)

Как видите, эти отрывки разнообразны и грамматически корректны, — более того, они связаны с тематикой, определенной в предложениях, введенных в качестве подсказки. Но эти предложения логически не следуют одно из другого, развивая связанный тезис. Модель языка GPT-2 известна как **модель Трансформер**, которая будет рассмотрена в разделе 24.4; дополнительные примеры по модели GPT-2 приведены в конце главы 24 на рис. 24.14. Другой моделью типа "Трансформер" является модель "Язык условного трансформера" (*Conditional Transformer Language* — CTRL). Эту модель можно контролировать более гибко, — в приведенных ниже примерах модели CTRL было предложено сгенерировать текст в категории *обзор продуктов*, содержащий оценку продуктов с рейтингами 1 и 4 (из 5).

- **1.0**: *I bought this for my son who is a huge fan of the show. He was so excited to get it and when he opened it, we were all very disappointed. The quality of the product is terrible. It looks like something you would buy at a dollar store.*
  (Я купил это для моего сына, который является большим поклонником шоу. Он был так взволнован, когда получил это, а когда он его открыл, мы все были очень разочарованы. Качество продукта ужасное. Это выглядит как что-то, что можно было бы купить в долларовом магазине.)
- **4.0**: *I bought this for my husband and he loves it. He has a small wrist so it is hard to find watches that fit him well. This one fits perfectly.*
  (Я купила это для моего мужа, и ему оно понравилось. У него маленькое запястье, поэтому ему трудно найти часы, которые ему хорошо подходят. Это подходит идеально.)

23.2. Грамматика

В главе 7, чтобы записать грамматику для языка логики первого порядка, использовалась форма Бэкуса–Наура (BNF). **Грамматика** — это совокупность правил, определяющих древовидную структуру допустимых фраз, а **язык** — это множество всех предложений, которые следуют этим правилам.

Естественные языки работают не совсем так, как формальный язык логики первого порядка, — в них не существует жесткой границы между допустимыми и недопустимыми предложениями, а также отсутствует единая четко определенная древовидная структура для каждого предложения. Однако иерархическая структура *действительно* важна в естественном языке. Слово "Stocks" (*акции*) в предложении "Stocks rallied on Monday" (*Акции выросли в понедельник*) — это не просто слово и не просто *существительное*; в этом предложении оно также представляет собой *именную группу*, которая является субъектом для следующей за ней

глагольной группы. ►**Синтаксические категории**, такие как *именная группа* или *глагольная группа*, помогают ограничить вероятные слова в каждой точке предложения, а ►**структура фразы** обеспечивает основу для смыслового значения или **семантики** предложения.

Существует много конкурирующих языковых моделей, основанных на идее иерархической синтаксической структуры. В этом разделе описывается популярная модель под названием ►**вероятностная контекстно-свободная грамматика** (*probabilistic context-free grammar* — PCFG). В вероятностной грамматике каждой строке присваивается некоторая вероятность, а "контекстно-свободная" означает, что любое правило может использоваться в любом контексте: правила для именной группы в начале предложения такие же, как и для другой именной группы далее в предложении, и если одна и та же фраза встречается в двух местах, она каждый раз должна иметь одну и ту же вероятность. Ниже грамматика PCFG определяется для крошечного подмножества английского языка, которое подходит для общения между агентами, исследующими мир вампуса. Назовем этот язык $\mathcal{E}_0$ (рис. 23.2). Правило грамматики, подобное

$$\begin{aligned} Adjs &\rightarrow Adjective & [0{,}80] \\ &\mid Adjective\ Adjs & [0{,}20] \end{aligned}$$

означает, что синтаксическая категория *Adjs* может состоять из одного прилагательного (*Adjective*) с вероятностью 0,80 или прилагательного, за которым следует строка, определяемая категорией *Adjs*, с вероятностью 0,20.

К сожалению, эта грамматика допускает ►**перегенерацию** (*overgenerate*), т.е. она позволяет генерировать предложения, которые не являются грамматически правильными, например "Me go I" (*Мне пойду я*). Она также допускает и ►**недогенерацию** (*undergenerate*): существует множество правильных предложений на английском языке, которые она отвергает, например "I think the wumpus is smelly" (*Я думаю, вампус вонючий*). О том, как изучить лучшую грамматику, речь пойдет чуть позже, а пока сосредоточимся на том, что можно делать с этой очень простой грамматикой.

23.2.1. Словарь грамматики $\mathcal{E}_0$

Список допустимых слов, или ►**словарь** (*lexicon*), для грамматики $\mathcal{E}_0$ представлен на рис. 23.3. На нем каждая лексическая категория заканчивается символом "...", указывающим на то, что в этой категории есть и другие слова. Для существительных (*Noun*), личных имен (*Name*), глаголов (*Verb*), прилагательных (*Adjective*) и наречий (*Adverb*) все слова невозможно перечислить даже в принципе не только потому, что каждый из этих классов включает десятки тысяч членов, но и потому, что в них постоянно добавляются новые, такие как *humblebrag* (скромно похвастаться) или *microbiome* (микробиома). По этой причине данные пять категорий называются ►**открытыми классами**. Местоимения (*Pronoun*),

S	→	*NP VP*	[0,90]	I + feel a breeze
	|	*S Conj S*	[0,10]	I feel a breeze + and + It stinks
NP	→	*Pronoun*	[0,25]	I
	|	*Name*	[0,10]	Ali
	|	*Noun*	[0,10]	pits
	|	*Article Noun*	[0,25]	the + wumpus
	|	*Article Adjs Noun*	[0,05]	the + smelly dead + wumpus
	|	*Digit Digit*	[0,05]	3 4
	|	*NP PP*	[0,10]	the wumpus + in 1 3
	|	*NP RelClause*	[0,05]	the wumpus + that is smelly
	|	*NP Conj NP*	[0,05]	the wumpus + and + I
VP	→	*Verb*	[0,40]	stinks
	|	*VP NP*	[0,35]	feel + a breeze
	|	*VP Adjective*	[0,05]	smells + dead
	|	*VP PP*	[0,10]	is + in 1 3
	|	*VP Adverb*	[0,10]	go + ahead
Adjs	→	*Adjective*	[0,80]	smelly
	|	*Adjective Adjs*	[0,20]	smelly + dead
PP	→	*Prep NP*	[1,00]	to + the east
RelClause	→	*RelPro VP*	[1,00]	that + is smelly

Рис. 23.2. Грамматика языка $\mathcal{E}_0$ с примерами фраз для каждого правила. Синтаксическими категориями языка являются **предложение** (*sentence — S*), **именная группа** (*noun phrase — NP*), **глагольная группа** (*verb phrase — VP*), **последовательность прилагательных** (*adjective list — Adjs*), **предлог** (*prepositional phrase — PP*) и **придаточное предложение** (*relative clause — RelClause*)

Noun	→	**stench** [0,05] | **breeze** [0,10] | **wumpus** [0,15] | **pits** [0,05] | ...
Verb	→	**is** [0,10] | **feel** [0,10] | **smells** [0,10] | **stinks** [0,05] | ...
Adjective	→	**right** [0,10] | **dead** [0,05] | **smelly** [0,02] | **breezy** [0,02] | ...
Adverb	→	**here** [0,05] | **ahead** [0,05] | **nearby** [0,02] | ...
Pronoun	→	**me** [0,10] | **you** [0,03] | **I** [0,10] | **it** [0,10] | ...
RelPro	→	**that** [0,40] | **which** [0,15] | **who** [0,20] | **whom** [0,02] | ...
Name	→	**Ali** [0,01] | **Bo** [0,01] | **Boston** [0,01] | ...
Article	→	**the** [0,40] | **a** [0,30] | **an** [0,10] | **every** [0,05] | ...
Prep	→	**to** [0,20] | **in** [0,10] | **on** [0,05] | **near** [0,10] | ...
Conj	→	**and** [0,50] | **or** [0,10] | **but** [0,20] | **yet** [0,02] | ...
Digit	→	**0** [0,20] | **1** [0,20] | **2** [0,20] | **3** [0,20] | **4** [0,20] | ...

Рис. 23.3. Словарь языка $\mathcal{E}_0$. Сумма вероятностей для каждой категории равна 1

относительные местоимения (*relative propronouns — RelPro*), артикли (*Article*), предлоги (*prepositions — Prep*) и союзы (*conjunction — Conj*) называются ▶**замкнутыми классами**, поскольку к ним относится лишь небольшое количество слов (дюжина или около того) и они меняются на протяжении веков, а не месяцев. Например, местоимения "thee" (*тебе*, *тебя*) и "thou" (*ты*) широко использовались еще в XVII веке, но к концу XIX века вышли из употребления, а сегодня их можно встретить только в стихах и некоторых региональных диалектах. Наконец, к последней категории на рис. 23.3, *Digit*, относятся числительные.

23.3. Синтаксический анализ

Под ▶**синтаксическим анализом** (*parsing*), иначе называемым *синтаксическим разбором* или *парсингом*, понимается процесс анализа строки слов согласно правилам грамматики с целью определения структуры той фразы, которую она представляет. Можно рассматривать этот процесс как ▶**поиск** допустимого дерева синтаксического разбора, листовыми узлами которого являются слова строки. На рис. 23.4 показано, что можно начать анализ с символа *S* и выполнять поиск в направлении сверху вниз или начать с отдельных слов и выполнять поиск в направлении снизу вверх. Чистые стратегии синтаксического анализа в направлении сверху вниз или снизу вверх могут, однако, оказаться неэффективными, поскольку в конечном счете приведут к многократному применению усилий в том числе и в тех областях пространства поиска, которые ведут в тупик. Рассмотрим следующие два предложения.

> Have the students in section 2 of Computer Science 101 take the exam.
> (*Попросите студентов 2-й группы, слушающих курс "Компьютерные науки 101", сдать экзамен.*)
> Have the students in section 2 of Computer Science 101 taken the exam?
> (*Студенты 2-й группы, слушающие курс "Компьютерные науки 101", сдали экзамен?*)

Несмотря на то что первые 10 слов этих предложений одни и те же, их синтаксический анализ будет сильно различаться, потому что первое из них — команда, а второе — вопрос. Алгоритму анализа, действующему в направлении слева направо, потребуется угадать, чем является первое слово (частью команды или вопроса), при этом выяснить, правильно ли сделанное предположение, будет невозможно по крайней мере до одиннадцатого слова, *take* или *taken*. Если алгоритм не угадает, ему придется вновь вернуться к первому слову и заново проанализировать все предложение с другой его интерпретацией.

Чтобы избежать подобного источника неэффективности, можно использовать **динамическое программирование**: каждый раз, проанализировав подстроку, сохраняем полученные результаты, чтобы позже не пришлось повторно ее анализировать. Например, как только будет обнаружено, что строка "the students in section 2

of Computer Science 101" является *NP*, можно записать этот результат в структуре данных, известной как ▶**диаграмма**. Алгоритм, который реализует такой подход, называется ▶**диаграммным синтаксическим анализатором** (*chart parser*). Поскольку в данном случае речь идет о контекстно-свободных грамматиках, то любое словосочетание, обнаруженное в контексте одной ветви пространства поиска, вполне может применяться и в любой другой ветви пространства поиска. Существует много типов диаграммных синтаксических анализаторов; здесь будет рассмотрена вероятностная версия алгоритма восходящего диаграммного синтаксического анализатора, известного как ▶**алгоритм CYK**, — название образовано от имен его изобретателей, Али Кока (*Ali Cocke*), Даниэля Юнгера (*Daniel Younger*) и Тадео Казами (*Tadeo Kasami*).[2]

Список элементов	Правило
S	
NP VP	*S* → *NP VP*
NP VP Adjective	*VP* → *VP Adjective*
NP Verb Adjective	*VP* → *Verb*
NP Verb **dead**	*Adjective* → **dead**
NP **is dead**	*Verb* → **is**
Article Noun **is dead**	*NP* → *Article Noun*
Article **wumpus is dead**	*Noun* → **wumpus**
the wumpus is dead	*Article* → **the**

Рис. 23.4. Синтаксический анализ строки "Wumpus is dead" (*вампус мертв*) как предложения в соответствии с грамматикой $\mathcal{E}_0$. При выполнении анализа в направлении сверху вниз работа начинается с категории *S* и на каждом этапе один нетерминальный элемент *X* сопоставляется с правилом вида (*X* → *Y*...) с заменой *X* в списке элементов на *Y*..., например *S* заменяется последовательностью *NP VP*. При выполнении анализа в направлении снизу вверх работа начинается со слов "the wumpus is dead", на каждом этапе строка токенов, таких как (*Y*...), сопоставляется с правилами вида (*X* → *Y*...) и токены заменяются элементом *X*, например "The" заменяется словом *Article* или *Article Noun* с *NP*

Алгоритм CYK представлен на рис. 23.5. Он требует грамматики, все правила которой должны быть представлены в одном из двух очень специфических форматов: лексические правила в виде *X* → **word** [*p*] и синтаксические правила в виде *X* → *YZ* [*p*] с точно двумя категориями в правой части. Такой формат грамматики, называемый ▶**нормальной формой Хомского**, может показаться ограничивающим, но это не так: любая контекстно-свободная грамматика может быть автоматически преобразована в нормальную форму Хомского. (Этот процесс подробно описывается в упражнении 23.21.)

[2] Иногда авторы в названии алгоритма перечисляются в порядке CKY.

```
function CYK-PARSE(words, grammar) returns таблица дерева анализа
  inputs: words, список слов
          grammar, структура с правилами лексики LEXICALRULES
                   и правилами грамматики GRAMMARRULES

  T ← таблица        // T[X, i, k] — наиболее вероятное дерево категории X,
                                    охватывающее слова words_{i:k}
  P ← таблица, исходно все 0     // P[X, i, k] — вероятности дерева T[X, i, k]
  // Вставка лексических категорий для каждого слова
  for i = 1 to LEN(words) do
    for each (X, p) in grammar.LEXICALRULES(words_i) do
      P[X, i, i] ← p
      T[X, i, i] ← TREE(X, words_i)
  // Построение X_{i:k} из Y_{i:j} + Z_{j+1:k}, кратчайшие диапазоны строятся первыми
  for each (i, j, k) in SUBSPANS(LEN(words)) do
    for each (X, Y, Z, p) in grammar.GRAMMARRULES do
      PYZ ← P[Y, i, j] × P[Z, j + 1, k] × p
      if PYZ > P[X, i, k] do
        P[X, i, k] ← PYZ
        T[X, i, k] ← TREE(X, T[Y, i, j], T[Z, j + 1, k])
    return T

function SUBSPANS(N) yields (i, j, k) кортежи
  for length = 2 to N do
    for i = 1 to N + 1 − length do
      k ← i + length − 1
      for j = i to k − 1 do
        yield (i, j, k)
```

Рис. 23.5. Алгоритм CYK для выполнения синтаксического анализа. Получив на входе последовательность слов, алгоритм находит наиболее вероятное дерево синтаксического анализа для этой последовательности в целом и ее подпоследовательностей. Таблица $P[X, i, k]$ содержит вероятности наиболее вероятного дерева категории X, охватывающего слова $words_{i:k}$. Результирующая таблица $T[X, i, k]$ содержит наиболее вероятное дерево категории X, охватывающее позиции от i до k включительно. Функция SUBSPANS возвращает все кортежи (i, j, k), охватывающие диапазон слов $words_{i:k}$, при $i \leq j < k$, располагая кортежи в порядке увеличения длины диапазона $i{:}k$, чтобы при объединении двух более коротких диапазонов в более длинный более короткие промежутки уже находились в таблице. Функция LEXICALRULES(*word*) возвращает коллекцию пар (X, p), по одной для каждого правила вида $X \rightarrow word[p]$, а функция GRAMMARRULES генерирует кортежи (X, Y, Z, p), по одному на каждое правило грамматики вида $X \rightarrow YZ[p]$

Алгоритм CYK использует для таблиц P и T пространство $O(n^2m)$, где n — число слов в предложении, а m — количество нетерминальных символов в грамматике, и требует времени $O(n^3m)$. Если необходим алгоритм, гарантированно работающий для всех возможных контекстно-свободных грамматик, то лучшего варианта, чем этот, не найти. Но в действительности выполнять синтаксический анализ требуется лишь для естественных языков, а не для всех возможных грамматик. Естественные языки развивались таким образом, чтобы их легко было понимать в реальном времени, избегая излишней сложности, насколько это возможно, поэтому кажется вполне вероятным, что для них должен существовать более быстрый алгоритм синтаксического анализа.

Чтобы попытаться добраться до времени выполнения $O(n)$, можно применить поиск A* довольно простым способом: каждое состояние является списком элементов (слов или категорий), как показано на рис. 23.4. Начальное состояние представляет список слов, а целевым состоянием является единственный элемент S. Стоимость состояния обратно пропорциональна его вероятности, определенной согласно правилам, применявшимся до данного момента, и существуют различные эвристики для оценки оставшегося расстояния до цели. Лучшая эвристика при текущем применении исходит от машинного обучения, применяемого к корпусу предложений.

При использовании алгоритма A* нет необходимости проводить поиск по всему пространству состояний, и существует гарантия, что первый найденный вариант синтаксического анализа будет наиболее вероятным (при условии использования допустимой эвристики). Такой алгоритм обычно работает быстрее, чем CYK, но (в зависимости от деталей грамматики) все же медленнее, чем $O(n)$. Пример результата такого грамматического анализа приведен на рис. 23.6.

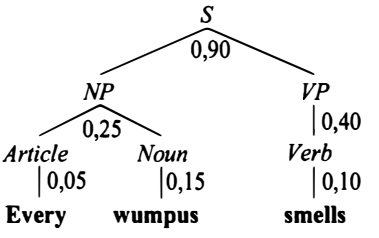


Рис. 23.6. Дерево синтаксического анализа предложения "Every wumpus smells" (*Каждый вампус воняет*) согласно грамматике $\mathcal{E}_0$. Каждый внутренний узел дерева обозначен своей вероятностью. Вероятность дерева как целого равна $0{,}9 \times 0{,}25 \times 0{,}05 \times 0{,}15 \times 0{,}40 \times 0{,}10 = 0{,}0000675$. Это дерево также может быть записано в линейной форме как [*S*[*NP*[*Article* **every**] [*Noun* **wumpus**]] [*VP*[*Verb* **smells**]]]

Так же, как в случае присвоения тегов частей речи, при выполнении синтаксического анализа можно использовать **поиск по лучу**, когда в любой момент времени рассматривается только b наиболее вероятных альтернативных вариантов анализа. В целом это означает, что нет гарантии нахождения анализа с наибольшей вероятностью, но (при тщательной реализации) алгоритм синтаксического анализа будет выполняться за время $O(n)$, и при этом находить лучший вариант синтаксического анализа в большинстве случаев.

Алгоритм синтаксического анализа по лучу с $b = 1$ называется ►**детерминированным синтаксическим анализом**. Одним популярным детерминированным подходом является ►**синтаксический анализ "сдвиг-уменьшение"** (*shift-reduce parsing*), при котором предложение перебирается слово за словом с выбором в каждой точке, как поступить: вставить слово в стек составляющих или извлечь верхний элемент (элементы) стека составляющих в соответствии с правилом грамматики. Каждый способ синтаксического анализа имеет своих приверженцев в сообществе NLP. Даже если существует возможность преобразовать систему сдвига-уменьшения в систему вероятностной контекстно-свободной грамматики (и наоборот), при применении машинного обучения к задаче, включающей грамматику, индуктивное смещение и, следовательно, обобщение, которое будет демонстрировать каждая из систем, окажется разным (Эбни и др. [6], 1999).

23.3.1. Синтаксический анализ зависимостей

Существует широко используемый альтернативный синтаксический подход, называемый ►**грамматикой зависимости**, в котором предполагается, что синтаксическая структура образована бинарными отношениями между лексическими элементами без необходимости в синтаксических составляющих. На рис. 23.7 приведен синтаксический анализ зависимостей и синтаксический анализ структуры фразы для одного и того же предложения.

В каком-то смысле грамматика зависимости и синтаксический анализ структуры фразы являются просто вариантами нотации. Если дерево структуры фразы снабжено заголовком каждой фразы, из него можно восстановить дерево зависимостей. Также в обратном направлении — дерево зависимостей можно преобразовать в дерево структуры фразы путем введения произвольных категорий (хотя таким способом не всегда можно получить дерево, имеющее естественный вид).

Поэтому нельзя предпочесть одну нотацию другой, исходя из того, что мощнее; скорее можно предпочесть одну из них потому, что она более естественна — либо для человека, разрабатывающего систему, либо для системы машинного обучения, которой предстоит изучать эти структуры. В общем случае деревья структуры фразы являются более подходящими для естественных языков с в основном фиксированным порядком слов (например, для английского), тогда как деревья зависимостей более удобны для языков с в основном свободным порядком слов (например,

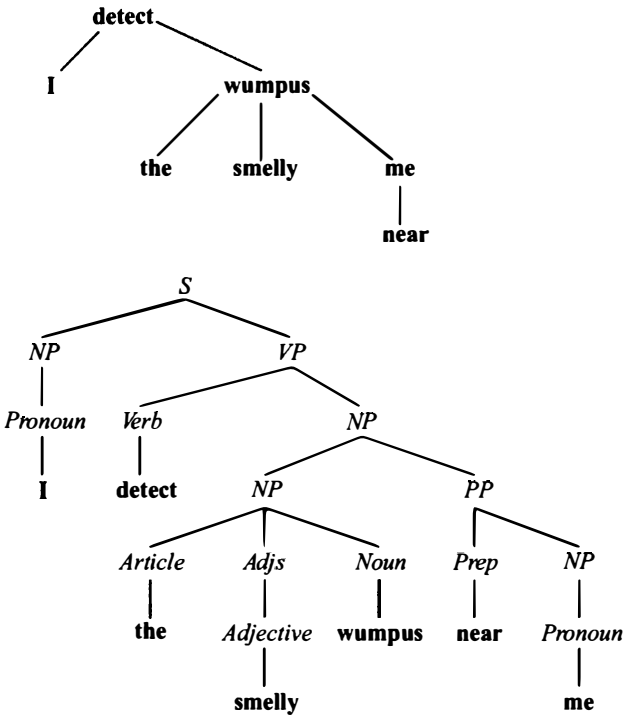


Рис. 23.7. Синтаксический анализ зависимостей (вверху) и соответствующий синтаксический анализ структуры фразы (внизу) для предложения "I detect the smelly wumpus near me" (*Я обнаружил вонючего вампуса рядом со мной*)

для русского или латыни), в которых порядок слов определяется в большей степени грамматикой, чем синтаксическими категориями.

Сегодня популярность грамматики зависимости в значительной степени обусловлена проектомUniversal Dependencies (Нивре и др. [1695], 2016) — проектом с открытым исходным кодом по созданию банка деревьев, в котором определен набор отношений и предоставлены миллионы проанализированных предложений более чем на 70 языках.

23.3.2. Обучение синтаксического анализатора на примерах

Задача создания грамматики для значительной части английского языка весьма трудоемка и подвержена ошибкам. Отсюда можно предположить, что было бы лучше **изучить** грамматические правила (и вероятности), а не записывать их вручную. При использовании метода обучения с учителем потребуются пары ввода-вывода — предложения и их деревья синтаксического анализа. Проект Penn

Treebank — самый известный источник таких данных, предоставляющий разработчикам более 100 тыс. предложений, аннотированных деревом синтаксического анализа их структуры. На рис. 23.8 представлен пример такой аннотации в виде дерева синтаксического анализа предложения из банка Penn Treebank.

```
[ [S [NP -2 Her eyes]
    [VP were
        [VP glazed
            [NP *-2]
            [SBAR-ADV as if
                [S [NP she]
                    [VP did n't
                        [VP [VP hear [NP *-1]]
                            or
                            [ VP [ ADVP even] see [NP *-1]]
                            [NP -1 him ]]]]]]]
 .]
```

Рис. 23.8. Аннотированное дерево для предложения "Her eyes were glazed as if she didn't hear or even see him" (*Ее глаза остекленели, как будто она не слышала или даже не видела его*) из банка Penn Treebank. Обратите внимание на грамматический феномен, о котором пока не говорилось: связывание фразы из одной части дерева с фразой из другой. В этом дереве анализируется фраза "hear or even see him" (*слышать или даже видеть его*), состоящая из двух составляющих *VP*, [*VP* **hear** [*NP**-1]] и [*VP*[*ADVP* **even**] **see** [*NP**-1]], каждая из которых включает отсутствующий объект, обозначенный как ***-1**, который относится к *NP*, помеченному в другом месте дерева как [*NP*-1 **him**]. Точно так же [*NP**-2] относится к [*NP*-2 **Her eyes**]

Имея в наличии банк деревьев, можно создать PCFG, просто посчитав, сколько раз каждый тип узла появляется в дереве (с обычными предостережениями по поводу сглаживания малых значений счетчиков). На рис. 23.8 присутствуют два узла вида [*S*[*NP*...][*VP*...]]. Можно было бы подсчитать эти и все остальные поддеревья с корнем *S* в корпусе. Если есть 1000 узлов *S*, из которых 600 имеют эту форму, то можно создать правило:

$$S \rightarrow NP\ VP[0,6].$$

Всего в банке Penn Treebank более 10 тыс. различных типов узлов. Это отражает тот факт, что английский язык является сложным языком, но и указывает на то, что аннотаторы, создававшие этот банк, отдавали предпочтение плоским деревьям, возможно даже более плоским, чем хотелось бы. Например, фраза "the good and the bad" (*хорошее и плохое*) синтаксически анализируется как одна фраза с существительным, а не две фразы с существительным, соединенные союзом, что дает нам правило

$$NP \rightarrow Article\ Noun\ Conjunction\ Article\ Noun.$$

Существуют сотни похожих на это правил, определяющих именные фразы как строки категорий с союзом где-то посередине. Более краткая грамматика может охватить все эти правила в отношении существительных и союзов одним-единственным правилом

$$NP \rightarrow NP\ Conjunction\ NP.$$

В работах Бода и соавт. ([239], 2003) и Бода ([238], 2008) показано, как автоматически извлекать обобщенные правила, подобные этим, значительно сокращая количество правил, которые следуют из банка деревьев, и создавая грамматику, которая в конечном итоге улучшает обобщение для ранее не встречавшихся предложений. Этот подход получил название **синтаксический анализ, ориентированный на данные** (*data-oriented parsing*).

Как видите, банки деревьев не идеальны — они содержат ошибки и имеют собственные особенности синтаксического анализа. Также понятно, что создание банка деревьев требует много тяжелой ручной работы, а это означает, что эти банки будут иметь относительно небольшие размеры по сравнению со всем текстом, который не был аннотирован деревьями синтаксического анализа. Альтернативный подход заключается в использовании ▶**синтаксического анализа без учителя** (*unsupervised parsing*), при котором новая грамматика изучается (или улучшается уже существующая) за счет использования набора предложений без деревьев синтаксического анализа.

В предложенном в статье Додда ([630], 1988) **алгоритме "внутренний-внешний"** (*inside-outside algorithm*), который здесь подробно рассматриваться не будет, способность оценивать вероятности в PCFG изучается на примерах предложений без деревьев синтаксического анализа. В нем вероятности оцениваются способом, аналогичным тому, что применяется в прямом-обратном алгоритме (см. рис. 14.4). В работе [2116] (2010) Спитковский и соавт. описывают иной подход к обучению без учителя: метод ▶**последовательного обучения** (*curriculum learning*). В этом случае обучение начинается с простейшей части учебного плана, коротких однозначных предложений из двух слов, таких как "Он ушел", которые могут быть легко проанализированы на основе предыдущих знаний или аннотаций. Каждый новый синтаксический анализ короткого предложения расширяет знания системы, так что со временем она сможет справляться с предложениями из трех слов, затем — из четырех слов и в конце концов — с предложениями из 40 слов.

Также может использоваться ▶**синтаксический анализ с частичным привлечением учителя** (*semisupervised parsing*), в котором обучение начинается с небольшого количества предложений, снабженных деревьями синтаксического анализа, в качестве данных для построения начальной грамматики. Затем для улучшения грамматики добавляется большое количество предложений без результатов анализа. В подходе с частичным привлечением учителя может оказаться

полезной ▶**неполная расстановка скобок** (*partial bracketing*): можно использовать широкодоступный текст, который был размечен авторами, а не лингвистами, с частично древовидной структурой в форме HTML-текста или аналогичных этому способов записи. В HTML-тексте большинство скобок соответствует синтаксическим компонентам, поэтому даже неполная разметка текста скобками может быть полезной для изучения грамматики (Перейра и Шабес [1776], 1992; Спитковский и др. [2117], 2010). Рассмотрим следующий фрагмент HTML-текста из газетной статьи:

```
In 1998, however, as I <a>established in
<i>The New Republic</i></a> and Bill Clinton just
<a>confirmed in his memoirs</a>, Netanyahu changed his mind
```

Слова, окруженные тегами и , образуют именную группу, а две строки слов, окруженные тегами <a> и каждая, образуют две глагольные группы.

23.4. Расширенные грамматики

До сих пор мы имели дело с **контекстно-свободными грамматиками**. Но не каждая именная группа *NP* может появиться в любом контексте с равной вероятностью. Предложение "I ate a banana" (*Я ем банан*), безусловно, правильное, тогда как предложение "Me ate a banana" (*Мне ем банан*) построено неграмотно, а предложение "I ate a bandanna" (*Я ем бандану*) крайне маловероятно.

Проблема в том, что наша грамматика ориентирована на лексические категории, такие как *Pronoun* (местоимение), но хотя слова "I" (*я*) и "me" (*мне*) оба являются местоимениями, только местоимение "I" может быть подлежащим в предложении. Точно так же слова "banana" (*банан*) и "bandanna" (*бандана*) оба являются существительными, но только первое из них, вероятнее всего, станет объектом, на который будет направлено действие глагола "ate" (*ем*). Лингвисты говорят, что местоимение "I" находится в именительном падеже (т.е. является субъектом, выполняющим действие, определяемое глаголом), а местоимение "me" — в объектном падеже[3] (т.е. является объектом, на который направлено действие глагола). Они также говорят, что "I" является местоимением первого лица (местоимение "you" (*вы*) — второго лица, а "she" (*она*) — третьего лица) и единственного числа (местоимение "we" (*мы*) — множественного числа). Синтаксическая категория, подобная *Pronoun*, дополненная такими признаками, как "именительный падеж, первое лицо единственного числа", называется ▶**подкатегорией**.

В этом разделе будет показано, как грамматика может представлять этот вид знаний, чтобы обеспечить более тонкие различия в отношении того, какие предложения более вероятны. Также будет показано, как построить представление

[3] Именительный падеж иногда называют также номинативным падежом, а объектный падеж — винительным падежом. Во многих языках имеется также дательный падеж для слов в позиции косвенного объекта.

семантики фразы композиционным способом. Все это реализуется с помощью ▶**расширенной грамматики** (*augmented grammar*), в которой нетерминальные символы являются не просто атомарными, такими как *Pronoun* (*местоимение*) или *NP* (*именная группа*), а определены в структурном представлении. Например, именная группа "I" может быть представлена как $NP(Sbj, 1S, Speaker)$, что означает "именная группа, именительный падеж, первое лицо единственного числа, значение — тот, кто произносит предложение", тогда как слово "me" будет представлено как $NP(Obj, 1S, Speaker)$, подчеркивая тот факт, что это слово в объектном (винительном) падеже.

Рассмотрим последовательность "*Noun* и *Noun* или *Noun*", которая при синтаксическом анализе может быть воспринята как "[*Noun* и *Noun*] или *Noun*", либо как "*Noun* и [*Noun* или *Noun*]". Наша контекстно-свободная грамматика не позволяет представить предпочтение одного варианта синтаксического анализа перед другим, поскольку правило для соединенных союзом именных групп, $NP \rightarrow NP\ Conjunction\ NP$ [0,05], даст одинаковую вероятность для каждого из вариантов анализа. Нам же нужна такая грамматика, которая позволит выразить предпочтение результатам синтаксического анализа "[[спагетти и фрикадельки] или лазанья]" и "[спагетти и [пирог или пирожное]]" перед альтернативными вариантами расстановки скобок для каждой из этих фраз.

Под названием ▶**лексическая PCFG** (*lexicalized PCFG*) понимается разновидность расширенной грамматики, позволяющая назначать вероятности, основанные на свойствах слов фразы, отличных от синтаксических категорий. Подобные данные были бы очень разреженными, если бы вероятность предложения, скажем, из 40 слов зависела от *всех* 40 слов — это та же самая проблема, которая была отмечена выше, при обсуждении n-грамм. Для упрощения введем понятие ▶**заголовок** фразы — это ее самое важное слово. Тогда "banana" будет заголовком именной группы (NP) "a banana", тогда как "ate" будет заголовком глагольной группы (VP) "ate a banana". Обозначение $VP(v)$ используется для фразы категории VP, главным словом которой является v. Вот пример лексической PCFG.

$VP(v) \rightarrow Verb(v)\ NP(n)$	$[P_1(v, n)]$
$VP(v) \rightarrow Verb(v)$	$[P_2(v)]$
$NP(n) \rightarrow Article(a)\ Adjs(j)\ Noun(n)$	$[P_3(n, a)]$
$NP(n) \rightarrow NP(n)\ Conjunction(c)\ NP(m)$	$[P_4(n, c, m)]$
$Verb(\mathbf{ate}) \rightarrow \mathbf{ate}$	[0,002]
$Noun(\mathbf{banana}) \rightarrow \mathbf{banana}$	[0,0007]

Здесь $P_1(v, n)$ представляет вероятность того, что глагольная группа VP с заголовком v будет соединена с именной группой NP с заголовком n для формирования VP. Теперь есть возможность указать, что фраза "ate a banana" является более вероятной, чем фраза "ate a bandanna", просто обеспечив, чтобы соблюдалось отношение $P_1(ate, banana) > P_1(ate, bandanna)$. Обратите внимание, что, поскольку

мы рассматриваем только заголовки фраз, различие между "ate a banana" (*есть банан*) и "ate a rancid banana" (*есть отвратительный банан*) не будет отражаться вероятностью P_1. Концептуально P_1 — это огромная таблица вероятностей: если в словаре есть 5 тыс. глаголов и 10 тыс. существительных, то таблица P_1 будет включать 50 млн строк, но большинство из них не будет храниться в явном виде, скорее они будут выводиться посредством сглаживания и возврата. Например, можно вернуться от $P_1(v, n)$ к модели, которая зависит только от v. Такая модель потребует в 10 тыс. раз меньше параметров, но все же позволит зафиксировать важные закономерности — такие, как тот факт, что переходный глагол, например "ate" (*есть*), с большей вероятностью будет следовать за именной группой *NP* (независимо от заголовка), чем непереходный глагол, например "sleep" (*спать*).

В разделе 23.2 было показано, что простая грамматика $\mathcal{E}_0$ допускает перегенерацию, производя такие неправильные предложения, как "I saw she" (*Я видел она*) или "I sees her" (*Я видит ее*). Чтобы избежать этой проблемы, в нашей грамматике должно быть представлено знание, что слово "her" (*ее*), а не слово "she" (*она*), является допустимым объектом для слова "saw" (*видел*) (или любого другого глагола), а также что слово "see" (*вижу*), а не "sees" (*видит*) — та форма глагола, которая употребляется с подлежащим "I" (*я*).

Все эти факты можно было бы полностью закодировать в записях вероятностей, например, сделав вероятность $P_1(v, she)$ очень небольшой для всех глаголов v. Но более краткий и модульный подход — расширить категорию именной группы *NP* дополнительными переменными: будем использовать запись $NP(c, pn, n)$ для представления фразы именной группы в падеже c (именительном или винительном) с лицом и числом pn (например, третье лицо единственного числа) и существительным-заголовком n. На рис. 23.9 представлена расширенная лексическая грамматика, работающая с этими дополнительными переменными. Рассмотрим одно из правил этой грамматики подробнее:

$$S(v) \rightarrow NP(Sbj, pn, n)\ VP(pn, v)[P_5(n, v)].$$

Это правило гласит, что когда за именной группой *NP* следует глагольная группа *VP*, они могут образовывать предложение *S*, но только в том случае, если группа *NP* находится в именительном (*Sbj*) падеже, а ее лицо и число (*pn*) идентичны таковым глагольной группы *VP*. (Говорят, что они *согласованы*.) Если это так, то у нас есть предложение *S*, заголовком которого является глагол из глагольной группы *VP*. Вот пример соответствующего лексического правила:

$$Pronoun(Sbj, 1S, I) \rightarrow \mathbf{I}\ [0{,}005],$$

которое утверждает, что "I" (*я*) — это местоимение в именительном падеже первого лица единственного числа с заголовком "I".

$$
\begin{aligned}
S(v) &\rightarrow NP(Sbj, pn, n)\ VP(pn, v) \mid \ldots \\
NP(c, pn, n) &\rightarrow Pronoun(c, pn, n) \mid Noun(c, pn, n) \mid \ldots \\
VP(pn, v) &\rightarrow Verb(pn, v)\ NP(Obj, pn, n) \mid \ldots \\
PP(head) &\rightarrow Prep(head)\ NP(Obj, pn, h) \\
Pronoun(Sbj, 1S, \mathbf{I}) &\rightarrow \mathbf{I} \\
Pronoun(Sbj, 1P, \mathbf{we}) &\rightarrow \mathbf{we} \\
Pronoun(Obj, 1S, \mathbf{me}) &\rightarrow \mathbf{me} \\
Pronoun(Obj, 3P, \mathbf{them}) &\rightarrow \mathbf{them} \\
Verb(3S, \mathbf{see}) &\rightarrow \mathbf{see}
\end{aligned}
$$

Рис. 23.9. Часть расширенной грамматики, поддерживающей согласование падежа, согласование существительного и глагола, а также заголовки фраз. Имена, начинающиеся с прописной буквы, — это константы: *Sbj* и *Obj* — для именительного и винительного падежа, *1S* — для первого лица единственного числа, *1P* и *3P* — для первого и третьего лица множественного числа соответственно. Как и прежде, названия, начинающиеся со строчных букв, являются именами переменных. Для простоты вероятности в записи правил были опущены

23.4.1. Семантическая интерпретация

Чтобы продемонстрировать, как можно добавить семантику к грамматике, начнем с примера, который значительно проще, чем английский язык: с семантики арифметических выражений. На рис. 23.10 представлена грамматика для арифметических выражений, в которой каждое правило расширяется одним аргументом, указывающим на семантическое толкование фразы. Семантика цифры, такой как "3", является самой этой цифрой. Семантикой выражения "3 + 4" является оператор "+", применяемый к семантикам фраз "3" и "4". Грамматические правила подчиняются принципу ▶**композиционной семантики**, — семантика фразы является функцией от семантики ее подфраз. На рис. 23.11 приведено дерево семантического анализа фразы $3 + (4 \div 2)$ согласно этой грамматике. Корнем дерева данного семантического анализа является выражение *Exp*(5), семантической интерпретацией которого является цифра "5".

А теперь перейдем к семантике английского языка или хотя бы к совсем небольшой его части. Для семантического представления будем использовать логику первого порядка, поэтому простое предложение "Ali loves Bo" (*Али любит Бо*) будет иметь следующее семантическое представление: *Loves*(*Ali*, *Bo*). Но как быть в случае составных фраз? Можно представить именную группу (*NP*) "Ali" логическим термом *Ali*. Но глагольная группа (*VP*) "loves Bo" не является ни логическим термом, ни полным логическим высказыванием. Интуитивно понятно, что фраза "loves Bo" является описанием, которое либо может, либо не может применяться к конкретному человеку (в данном случае оно применяется к Али). А это означает,

$Exp(op(x_1, x_2)) \rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)$
$Exp(x) \rightarrow (\ Exp(x)\ )$
$Exp(x) \rightarrow Number(x)$
$Number(x) \rightarrow Digit(x)$
$Number(10 \times x_1 + x_2) \rightarrow Number(x_1)\ Digit(x_2)$
$Operator(+) \rightarrow +$
$Operator(-) \rightarrow -$
$Operator(\times) \rightarrow \times$
$Operator(\div) \rightarrow \div$
$Digit(0) \rightarrow \mathbf{0}$
$Digit(1) \rightarrow \mathbf{1}$
...

Рис. 23.10. Грамматика для арифметических выражений, расширенная семантикой. Каждая переменная x_i представляет семантику ее составляющей

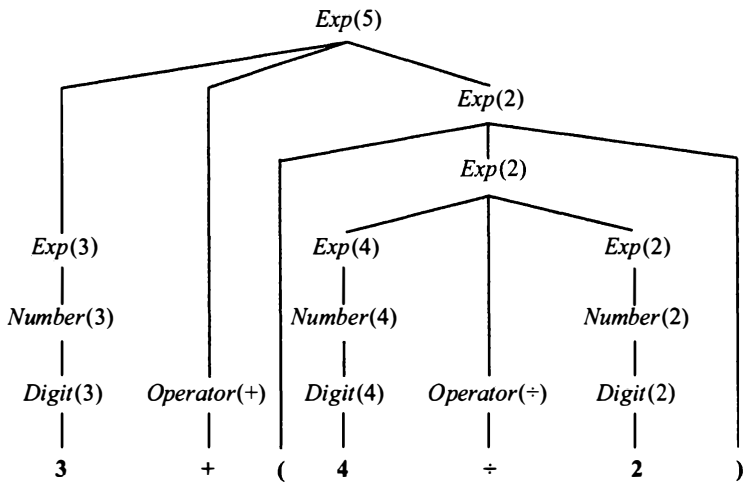


Рис. 23.11. Дерево синтаксического анализа с семантическими интерпретациями для строки "3 + (4 ÷ 2)"

что фраза "loves Bo" является **предикатом**, который в сочетании с термом, представляющим человека, дает полное логическое высказывание.

Используя λ-выражения (см. сноску 3 в начале раздела 8.2.3), представить фразу "loves Bo" как предикат можно следующим образом:

$$\lambda x\ Loves(x, Bo).$$

Теперь необходимо записать правило, которое будет утверждать: "группа *NP* с семантикой *n*, за которой следует группа *VP* с семантикой *pred*, образует предложение, семантикой которого будет результат применения предиката *pred* к семантике *n*":

$$S(pred(n)) \rightarrow NP(n)\ VP(pred).$$

Это правило указывает, что семантической интерпретацией предложения "Ali loves Bo" будет

$$(\lambda x\ Loves(x, Bo))(Ali),$$

что эквивалентно $Loves(Ali, Bo)$. Формально это выражение определяется как "β-уменьшение применения лямбда-функции".

Остальная часть семантики прямо вытекает из уже сделанного на данный момент выбора. Поскольку группы *VP* представлены в виде предикатов, глаголы также должны быть предикатами. Глагол "loves" (*любит*) следует представить как $\lambda y\ \lambda x\ Loves(x, y)$ — предикат, который при заданном аргументе *Bo* возвращает предикат $\lambda x\ Loves(x, Bo)$. В конечном итоге получаем грамматику и дерево синтаксического анализа, представленные на рис. 23.12. В более полной грамматике следовало бы все расширения (семантика, падеж, лицо и число, заголовок) поместить вместе в одно множество правил. Здесь же показано только семантическое расширение, чтобы было понятнее, как работают правила.

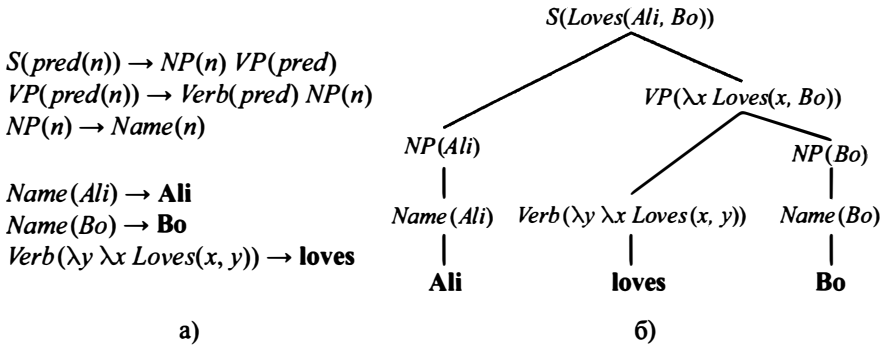


Рис. 23.12. а) Грамматика, позволяющая вывести дерево семантического анализа и семантическую интерпретацию для предложения "Ali loves Bo" (и трех других предложений). Каждая категория расширяется одним аргументом, представляющим семантику. **б)** Дерево синтаксического анализа с семантическими интерпретациями для строки "Ali loves Bo"

23.4.2. Изучение семантических грамматик

К сожалению, в банк Penn Treebank не включены семантические представления для входящих в него предложений, включены лишь деревья синтаксического разбора. Поэтому, если поставлена задача изучения агентом семантической грамматики, понадобится иной источник примеров. Цеттлемойер и Коллинс в статье [2428] (2005) описывают подсистему в системе ответов на вопросы, которая обучается грамматике на основании примеров, состоящих из пар в виде предложения и его семантической формы.

- **Предложение.** What states border Texas? (*Какие штаты граничат с Техасом?*)
- **Логическая форма**: $\lambda x.state(x) \wedge \lambda x.borders(x, Texas)$

При наличии большого множества пар, подобных этой, и небольшом дополнительном объеме ручного кодирования знаний для каждой новой предметной области система генерирует правдоподобные лексические строки (например, что "Texas" (*Техас*) и "state" (*штат*) являются такими существительными, что высказывание *state*(*Texas*) является истинным) и одновременно изучает параметры для грамматики, позволяющей этой системе выполнять семантический анализ предложений. Система Цеттлемойера и Коллинса достигла 79%-ной точности на двух разных тестовых наборах ранее неизвестных ей предложений. Чжао и Хуан ([2436], 2015) описывают систему семантического разбора, использующую метод "сдвиг-уменьшение", которая работает быстрее и достигает точности от 85% до 89%.

Ограничением этих систем является то, что обучающие данные включают логические формы. Их создание обходится дорого и требует участия людей-комментаторов, имеющих специальные знания, — не каждый понимает все тонкости лямбда-исчисления и логики предикатов. Гораздо проще собрать примеры пар вопросов и ответов, подобных приведенным ниже.

- **Вопрос.** What states border Texas? (*Какие штаты граничат с Техасом?*)
- **Ответ.** Louisiana, Arkansas, Oklahoma, NewMexico. (*Луизиана, Арканзас, Оклахома, Нью-Мексико.*)
- **Вопрос.** How many times would Rhode Island fit into California? (*Сколько раз Род-Айленд поместился бы в Калифорнии?*)
- **Ответ.** 135

Такие пары вопросов и ответов довольно распространены в Интернете, поэтому можно собрать большую базу данных без привлечения людей-экспертов. Воспользовавшись этим большим источником данных, можно создавать синтаксические анализаторы, превосходящие по производительности те, которые были созданы с использованием небольшой базы данных предложений, аннотированных логическими формами (Лянг и др. [1408], 2011; Лянг и Поттс [1409], 2015). Ключевой подход, описанный в этих документах, состоит в том, чтобы изобрести внутреннюю логическую форму, которая является композиционной, но не допускает экспоненциально большого пространства поиска.

23.5. Сложности реального естественного языка

Грамматика реального английского языка бесконечно сложна (и другие языки сложны в той же степени). Ниже кратко упоминаются некоторые аспекты языка, способствующие этой сложности.

►**Квантификация** — количественная оценка качественных показателей. Рассмотрим предложение "Every agent feels a breeze" (*Каждый агент чувствует ветерок*). В грамматике $\mathcal{E}_0$ это предложение имеет только один вариант синтаксического анализа, но семантически он неоднозначен: существует один ветерок, который чувствуют все агенты, или каждый агент чувствует отдельный личный ветерок? Эти две интерпретации могут быть представлены следующим образом:

$$\begin{aligned}&\forall\, a\ a \in Agents \Rightarrow\\ &\qquad \exists\, b\ b \in Breezes \wedge Feel(a, b);\\ &\exists\, b\ b \in Breezes \wedge\ \forall\, a\ a \in Agents \Rightarrow\\ &\qquad Feel(a, b).\end{aligned}$$

Для грамматики один стандартный подход к квантификации состоит в определении не действительного логического семантического предложения, а скорее ►**квазилогической формы**, которая затем превращается в логическое предложение алгоритмами вне процесса синтаксического анализа. Эти алгоритмы могут включать правила предпочтения в отношении выбора той или иной области действия квантификатора — предпочтения, которые не должны отражаться напрямую в грамматике.

►**Прагматика**. Выше было показано, как агент может воспринимать последовательность слов и использовать грамматику для вывода множества возможных семантических интерпретаций. Теперь займемся проблемой завершения интерпретации посредством добавления контекстно-зависимой информации о текущей ситуации. Наиболее очевидная потребность в прагматической информации заключается в определении значения ►**индексикалов**, представляющих собой фразы, которые относятся непосредственно к текущей ситуации. Например, в предложении "I am in Boston today" (*Я сегодня в Бостоне*), и "I" (*я*), и "today" (*сегодня*) являются индексикалами. Слово "I" можно представить переменной *Speaker* — флюентой, ссылающейся на разные объекты в разное время. Принятие решения, на что именно ссылается данная флюента, возлагается на слушателя, — это не считается частью грамматики, а воспринимается скорее как проблема прагматики.

Другой частью прагматики является интерпретация намерений говорящего. Высказывание говорящего считается ►**речевым актом**, и слушатель должен определить, какой тип действия он представляет — вопрос, утверждение, обещание, предупреждение, команду и т.д. Команда, такая как "go to (2,2)" (*иди в (2,2)*), безусловно, относится к слушателю. Пока что наша грамматика для предложений S охватывает только повествовательные предложения, но ее можно расширить,

чтобы включить и команды: команда — это глагольная группа, в которой субъектом неявно является слушатель команды:

$$S(Command(pred(Hearer))) \rightarrow VP(pred).$$

▶ **Далекие зависимости** (*long-distance dependencies*). На рис. 23.8 можно видеть, что результат синтаксического анализа фразы "she didn't hear or even see him" включает два пропуска, в которых именная группа отсутствует, но подразумевается ссылка на *NP* "him". Для обозначения таких пропусков можно использовать символ "␣": "she didn't [hear ␣ or even see ␣] him". В общем случае расстояние между пропуском и группой *NP*, на которую подразумевается ссылка, может быть произвольно большим: в предложении "Who did the agent tell you to give the gold to ␣?" (*Кому этот агент сказал вам отдать золото?*) пропуск относится к слову "Who", которое находится от него на расстоянии 10 слов.

Чтобы убедиться, что пропущенные группы *NP* расставлены правильно, может быть использована сложная система расширяющих правил. Эти правила действительно сложны, — например, не допускается наличие пропуска лишь в одной из двух групп *NP*, соединенных союзом: предложение "What did she play [*NP* Dungeons and ␣]?" (*Она что, играла в [NP Подземелья и ␣]?*) грамматически неправильно. Однако такие пропуски допустимы в обеих глагольных группах *VP*, соединенных союзом, как в предложении "What did you [*VP* [*VP* smell ␣] and [*VP* shoot an arrow at ␣]]?" (*Вы что, VP [VP запах ␣] и [VP выпустить стрелу в ␣]]?*).

▶ **Время обычное и глагольное** (*time and tense*). Предположим, что необходимо представить отличие предложения "Ali loves Bo" (*Али любит Бо*) от предложения "Ali loved Bo" (*Али любил Бо*). В английском языке для представления относительного времени событий используются времена глаголов (прошлое, настоящее и будущее). Одним из хороших вариантов представления времени событий в расширенной грамматике является нотация исчисления событий (см. раздел 10.3). Воспользовавшись нотацией исчисления событий, можно записать следующее:

$$\text{Ali loves Bo:} \quad E_1 \in Loves(Ali, Bo) \wedge During(Now, Extent(E_1)),$$
$$\text{Ali loved Bo:} \quad E_2 \in Loves(Ali, Bo) \wedge After(Now, Extent(E_2)).$$

Из этого можно сделать заключение, что два лексических правила для выбора слов "loves" и "loved" могут быть такими:

$$\text{Verb}(\lambda y\, \lambda x\, e \in Loves(x, y) \wedge During(Now, e)) \rightarrow \textbf{love}$$
$$\text{Verb}(\lambda y\, \lambda x\, e \in Loves(x, y) \wedge After(Now, e)) \rightarrow \textbf{loved}.$$

Кроме этого изменения, все остальное в грамматике остается в прежнем виде, что является обнадеживающей новостью и говорит о том, что был выбран правильный путь, если удалось так легко справиться со столь серьезным усложнением, как времена глаголов (хотя в действительности мы лишь едва прикоснулись к теме полной грамматики с представлением времени, как обычного, так и глагольного).

► **Двусмысленность**. Люди склонны воспринимать двусмысленность как неудачи в общении. Когда слушатель мысленно отмечает двусмысленность высказывания, то это означает, что высказывание для него неясно или просто сбивает с толку. Вот несколько примеров из газетных заголовков.

Squad helps dog bite victim. (*Отряд помогает собаке кусать жертву.*)

Police begin campaign to run down jaywalkers. (*Полиция начинает кампанию по сбежавшим прохожим.*)

Helicopter powered by human flies. (*Вертолет работающий от человека летит.*)

Once-sagging cloth diaper industry saved by full dumps. (*Некогда просевшая промышленность тканевых пеленок спасена полными свалками.*)

Include your children when baking cookies. (*Включайте своих детей при выпечке печенья.*)

Portable toilet bombed; police have nothing to go on. (*Передвижной туалет разбомбили; у полиции ничего нет, чтобы продолжить.*)

Milk drinkers are turning to powder. (*Любители молока превращаются в порошок.*)

Two sisters reunited after18 years in checkout counter. (*Две сестры воссоединились после 18 лет в кассе.*)

Подобная бестолковщина является исключением, — в большинстве случаев речь, которую мы слышим, кажется нам однозначной. Поэтому, когда исследователи в 1960-х годах впервые начали использовать компьютеры для анализа языка, они были весьма удивлены, обнаружив, что почти каждое предложение неоднозначно, с несколькими возможными вариантами синтаксического анализа (иногда сотнями), причем даже тогда, когда лишь один предпочтительный вариант анализа оказывается тем единственным, который отмечают носители языка. Например, мы понимаем фразу "brown rice and black beans" (*коричневый рис и черные бобы*) как "[brown rice] and [black beans]" и никогда не рассматриваем ее маловероятную интерпретацию "brown [rice and black beans]", где прилагательное "brown" (*коричневый*) определяет всю оставшуюся часть фразы, а не одно лишь слово "rice". Когда мы слышим "Outside of a dog, a book is a person's best friend" (Буквально: *За пределами собаки, книга — лучший друг человека*), то интерпретируем слова "outside of" (*снаружи*) как означающее "except for" (*за исключением*) и находим забавным следующее ее предложение в этой шутке Маркса Граучо: "Inside of a dog it's too dark to read." (*Внутри собаки слишком темно, чтобы читать.*)

► **Лексическая неоднозначность** присутствует в том случае, когда слово имеет более одного значения. Существительное "back" (*спина*), например, может быть еще и наречием: "go back" (*вернуться*), прилагательным: "back door" (*задняя дверь*), глаголом: "back a candidate" (*поддержать кандидата*), существительным с другим значением "the back of the room" (*задняя часть комнаты*) и даже именем

собственным: "Back" — река в Нунавуте, Канада. Слово "Jack" может быть именем собственным (Джек), существительным со многими различными значениями (валет в картах, электрический разъем, морской флаг, домкрат, название определенной рыбы, птицы или сыра и пр.) или глаголом: "to jack up a car" (поднять машину на домкрате). Понятие ▶ **синтаксической неоднозначности** относится к фразе, имеющей несколько вариантов синтаксического анализа: так, фраза "I smelled a wumpus in (2,2)" имеет два варианта синтаксического анализа: в первом фраза с предлогом "in (2,2)" определяет существительное (*Я ощутил запах: вампус в (2,2)*), а во втором — глагол (*Я ощутил запах вампуса в (2,2)*). Подобная синтаксическая неоднозначность приводит к ▶ **семантической неоднозначности**, поскольку согласно первому варианту анализа в квадрате [2,2] находится вампус, а согласно второму — лишь ощущается зловоние. Примечательно, что в данном случае неправильная интерпретация этой фразы может оказаться смертельной ошибкой для агента.

Также неоднозначность может проявляться между буквальными и переносными значениями. Образные выражения очень важны в поэзии и весьма распространены в повседневной речи. ▶ **Метонимия** — это оборот речи, в котором одно слово используется для обозначения другого. Услышав фразу "Chrysler announced a new model" (*Крайслер объявил о новой модели*), мы не интерпретируем ее как утверждение, что концерны могут говорить, — мы понимаем, что соответствующее объявление сделал представитель компании. Метонимия весьма распространена и в большинстве случаев интерпретируется людьми совершенно бессознательно.

К сожалению, грамматика большинства естественных языков в том виде, в каком она написана, не так уж легка для понимания. Для правильной обработки семантики при метонимии требуется ввести совершенно новый уровень неоднозначности. Это можно сделать путем предоставления *двух* объектов для семантической интерпретации каждой фразы в предложении: один — для объекта, на который фраза ссылается буквально (Крайслер), а другой — для метонимической ссылки (пресс-секретарь). Далее потребуется указать, что между ними есть связь. В нашей грамматике на текущий момент фразу "Chrysler announced" можно интерпретировать следующим образом:

$$x = Chrysler \wedge e \in Announce(x) \wedge After(Now, Extent(e)).$$

Эту запись следует привести к следующему виду:

$$x = Chrysler \wedge e \in Announce(m) \wedge After(Now, Extent(e)) \wedge Metonymy(m, x).$$

Такая запись говорит, что имеется одна сущность, x, со значением "Chrysler" и другая сущность, m, которая сделала объявление, и что обе эти сущности находятся в метонимической связи. На следующем этапе необходимо определить, какие именно виды метонимических отношений могут возникать. Самый простой случай — когда метонимии вообще нет, т.е. литеральный объект x и метонимический объект m идентичны:

$$\forall\, m, x\ (m = x) \Rightarrow Metonymy(m, x).$$

Для примера с концерном Крайслер разумным обобщением будет то, что название организации может быть использовано вместо ссылки на представителя этой организации:

$$\forall\, m, x\ x \in Organizations \wedge Spokesperson(m, x) \Rightarrow Metonymy(m, x).$$

Другие часто используемые метонимии включают имя автора для работ ("Я читал Шекспира"), название производителя для продукта ("Я езжу на хонде") либо части для целого ("Команде Red Sox нужна сильная рука"). Некоторые примеры метонимии, такие как "Бутерброд с ветчиной за четвертым столиком хочет еще пива", более новые и интерпретируются с учетом ситуации (например, посетитель кафе за определенным столиком, имя которого неизвестно).

►**Метафора** — это еще один оборот речи, в котором фраза с одним буквальным значением употребляется в переносном смысле по аналогии. Следовательно, метафора может рассматриваться как вид метонимии, когда связь определяется отношением подобия.

Под ►**устранением неоднозначности** (*disambiguation*) понимается процесс восстановления наиболее вероятного предполагаемого значения высказывания. В каком-то смысле у нас уже есть основа для решения этой проблемы: каждое правило имеет связанную с ним вероятность, поэтому вероятность интерпретации будет произведением вероятностей тех правил, которые привели к этой интерпретации. К сожалению, эти вероятности отражают лишь то, как часто те или иные фразы встречаются в корпусе, использовавшемся при обучении системы грамматике, а значит, отражают только общие знания, а не конкретные знания о текущей ситуации. Чтобы правильно устранить неоднозначность, необходимо объединить четыре следующие модели.

1. **Модель мира**. Она должна представлять вероятность того, что выражаемые предложением события могут иметь место в мире. Учитывая, что нам известно о мире, более вероятно, что тот, кто сказал "Я умер", имеет в виду "У меня большие проблемы" или "Я потерял жизнь в этой видеоигре", а не "Моя жизнь закончилась, и все же я еще могу говорить".
2. **Ментальная модель**. Эта модель определят вероятность того, что говорящий имеет намерение сообщить определенный факт слушателю. Данный подход объединяет модели того, во что верит говорящий, того, во что, как верит говорящий, верит слушатель, и т.д. Например, когда некто говорит: "Это сложно, я не могу планировать", модель мира может давать вероятность порядка 50% утверждению, что говорящему это действительно сложно и он не готов к составлению соответствующих планов, и вероятность 99,999% — утверждению, что любой человек не способен совершать планирующие полеты в воздухе. Тем не менее мы выбираем первую интерпретацию, потому что более вероятно, что именно это имел в виду говорящий.

3. **Модель языка**. Она определяет вероятность того, что будет выбрана определенная строка слов при условии, что говорящий намерен сообщить определенный факт.
4. **Акустическая модель**. Эта модель задает вероятность того, что при разговоре будет генерироваться определенная последовательность звуков с учетом того, что говорящий выбрал данную строку слов. (В случае рукописного или печатного сообщения возникает проблема оптического распознавания символов.)

23.6. Задачи обработки естественного языка

Обработка естественного языка — это большая область ИИ, заслуживающая собственного полного учебника или даже двух (Гольдберг [880], 2017; Журафски и Мартин [1162], 2020). В этом разделе кратко описываются некоторые из основных задач в этой области, — воспользуйтесь ссылками, чтобы получить более подробную информацию.

► **Распознавание речи** представляет собой задачу преобразования произносимых звуков в текст. На основе полученного в результате текста можно решать различные дальнейшие задачи (например, предоставлять ответы на вопросы). Современные системы имеют коэффициент ошибок в распознавании слов от 3% до 5% (в зависимости от деталей тестового набора), что близко к аналогичному показателю для человека-расшифровщика. Для использующих распознавание речи систем главной проблемой является предоставление соответствующих ответов даже при наличии ошибок в отдельных словах.

Сейчас в лучших системах используется комбинация рекуррентных нейронных сетей и скрытых марковских моделей (Хинтон и др. [1031], 2012; Ю и Денг [2413], 2016; Денг [603], 2016; Чю и др. [417], 2017; Жанг и др. [2433], 2017). Первое применение глубоких нейронных сетей в системах распознавания речи в 2011 году привело к немедленному и резкому снижению уровня ошибок примерно на 30%, — и это в области, которая к тому времени уже казалась вполне зрелой и в которой до этого наблюдались улучшения на уровне всего нескольких процентов в год. Глубокие нейронные сети отлично подходят для этой области потому, что задача распознавания речи характеризуется естественной композиционной разбивкой: звуковые сигналы преобразуются в фонемы, затем в слова, а те — в предложения. Подробно эта тема рассматривается в следующей главе.

► **Преобразование текста в речь** является обратным процессом — это переход от текста к звуку. Тейлор в работе [2187] (2009) дает для этой области обзор книжного масштаба. Задача состоит в том, чтобы правильно произнести каждое слово и сделать речевой поток каждого предложения максимально естественным, с правильными паузами и акцентами.

Другая область разработок заключается в синтезировании разных голосов — начиная с выбора между типичным мужским и женским голосом, с последующим

учетом региональных диалектов и вплоть до имитации голосов знаменитостей. Как и в случае распознавания речи, внедрение глубоких рекуррентных сетей привело к значительным улучшениям: примерно 2/3 слушателей отметили, что система WaveNet на базе нейронных сетей (ван ден Оорд и др. [2251], 2016) обеспечивала более естественное звучание, чем предыдущие системы, в которых нейронные сети не использовались.

Машинный перевод — это преобразование текста на одном языке в текст на другом языке. Такие системы обычно обучаются с использованием двуязычного корпуса: множества пар документов, в которых один член пары, скажем, на английском языке, а другой — на французском. В любом случае документы аннотировать не требуется: система машинного перевода обучается самостоятельно находить и правильно соотносить начало предложений и фраз, а затем, при предоставлении ей нового предложения на одном языке, генерировать его перевод на другой.

В начале 2000-х годов в системах перевода использовались *n*-граммные модели и были получены результаты, в целом достаточно хорошие, чтобы понять смысл текста, но с синтаксическими ошибками в большинстве предложений. Одной из проблем было ограничение длины *n*-грамм: даже с достаточно большим пределом на уровне 7 возникали трудности с передачей информации от одного конца предложения к другому. Другая проблема заключалась в том, что вся информация в *n*-граммной модели находится на уровне отдельных слов. Такая система могла выучить, что английскую фразу "black cat" (*черная кошка*) следует перевести на французский словами "noir" (*черный*) и "chat" (*кошка*), но была не способна выучить правило, что перевод должен иметь вид "chat noir", поскольку в английском языке прилагательное обычно стоит перед существительным, тогда как во французском — после него.

Рекуррентные нейронные модели "последовательность-последовательность" (*sequence-to-sequence*) (Суцкевер и др. [2154], 2015) позволяют обойти эти проблемы. Они способны лучше обобщать (поскольку могут использовать встраивания слов вместо *n*-граммных счетчиков отдельных слов) и могут формировать композиционные модели в пределах различных слоев глубокой сети для эффективной передачи информации. Последующие исследования с использованием механизма концентрации внимания и модели "Трансформер" (Васвани и др. [2266], 2018) позволили достичь увеличения производительности, а гибридная модель, включающая аспекты обеих этих моделей, работает еще лучше, приблизившись к уровню результативности человека для некоторых языковых пар (Ву и др. [2392], 2016; Чен и др. [412], 2018).

►**Извлечение информации** — это процесс приобретения знаний путем просмотра текста и поиска вхождений определенных классов объектов и связей между ними. Типичная задача этого класса — извлечение из веб-страниц сведений об адресах с заполнением полей базы данных для улицы, города, штата и почтового индекса или информации о штормах из отчетов о погоде, с заполнением полей

температуры воздуха, скорости ветра и объема осадков. Если исходный текст хорошо структурирован (например, в виде таблицы), то для извлечения информации могут использоваться простые методы, такие как регулярные выражения (Кафарелла и др. [357], 2008). Задача становится сложнее, если попытаться извлечь *все* факты, а не данные конкретного типа (например, отчеты о погоде). Банко и соавт. ([124], 2007) описывают систему TEXTRUNNER, выполняющую извлечение информации через открытое, расширяющееся множество отношений. Для текста произвольной формы используемые методы включают скрытые марковские модели и системы обучения на основе правил (как в TEXTRUNNER и NELL (Never-Ending Language Learning) (Митчелл и др. [1598], 2018)). В более современных системах применяются рекуррентные нейронные сети, использующие преимущества гибкости метода встраивания слов. Соответствующий обзор представлен в работе Кумара [1326] (2017).

► **Поиск информации** — это задача поиска документов, которые релевантны и важны для указанного запроса. Поисковые системы Интернета, такие как Google и Baidu, решают эти задачи миллиарды раз в день. Следует упомянуть три хороших учебника по этому вопросу: книги Маннинга и соавт. ([1487], 2008), Крофта и соавт. ([502], 2010), а также Баеса-Йатеса и Рибейро-Нето ([106], 2011).

Предоставление ► **ответа на вопрос** — это другая задача, в которой запрос действительно является вопросом, таким как "Кто основал береговую охрану США?" Ответ системы в данном случае не является ранжированным списком документов, это действительно типичный для нас ответ: "Александр Гамильтон". Системы ответов на вопросы существовали, начиная с 1960-х годов, и функционировали, полагаясь на синтаксический анализ, как обсуждалось выше в этой главе. И лишь после 2001 года в таких системах начали использовать поиск информации в Интернете, что позволило радикально расширить охватываемую ими тематику. Кац в работе [1197] (1997) описал синтаксический анализатор START и систему ответов на вопросы. Банко и соавт. в работе [123] (2002) описали систему ASKMSR, которая была менее сложной с точки зрения синтаксического анализа, но более агрессивной в использовании веб-поиска и сортировки результатов. Например, чтобы получить ответ на вопрос "Кто основал береговую охрану США?", эта система выполнит веб-поиск по таким запросам, как [* основал береговую охрану США] и [Береговая охрана США была основана *], а затем изучит несколько полученных в результате веб-страниц, чтобы выбрать наиболее вероятный ответ с учетом того, что слово "Кто" в запросе предполагает, что ответом должно быть имя человека. Конференция The Text Retrieval Conference (TREC) собирает исследования по этой теме и проводит соревнования на ежегодной основе начиная с 1991 года (Аллан и др. [31], 2017). Для систем ответа на вопросы недавно появились новые тестовые наборы, такие как AI2 ARC — тестовый набор вопросов по фундаментальной науке (Кларк и др. [446], 2018).

Резюме

Вот перечень основных тем, рассмотренных в этой главе.

- Вероятностные модели языка, основанные на использовании *n*-грамм, охватывают на удивление большой объем информации о языке. Они позволяют успешно решать такие несхожие задачи, как определение языка документа, исправление орфографии, анализ тональности текста, классификация по жанрам и определение авторства.
- Эти модели языка могут иметь миллионы признаков, поэтому для них очень важны предварительная обработка и сглаживание данных с целью снижения уровня шума.
- При построении статистической языковой системы лучше всего разработать модель, которая обеспечит эффективное использование доступных **данных**, даже если эта модель покажется чрезмерно упрощенной.
- Модель встраивания слов позволяет получить более полное представление о словах и сходстве между ними.
- Для выявления иерархической структуры языка будут полезны грамматики **структуры фраз** (и в частности, контекстно-свободные грамматики). Подход, известный как "вероятностная контекстно-свободная грамматика" (PCFG), широко используется в качестве формального математического представления грамматики зависимости.
- Предложения на контекстно-свободном языке могут быть синтаксически проанализированы за время $O(n^3)$ **диаграммным синтаксическим анализатором**, подобным **алгоритму CYK**, который требует, чтобы грамматические правила были представлены в **нормальной форме Хомского**. С небольшой потерей в точности тексты на естественных языках могут быть проанализированы за время $O(n)$ при использовании поиска по лучу или метода синтаксического анализа "сдвиг-уменьшение".
- **Банк деревьев синтаксического разбора** может служить отличным ресурсом для изучения грамматики PCFG с различными параметрами.
- Удобно **расширить** грамматику для решения таких проблем, как согласование субъекта-глагола и падежа местоимения, а также для предоставления информации на уровне слов, а не только на уровне категорий.
- **Семантическая интерпретация** также может обеспечиваться расширенной грамматикой. Семантическую грамматику можно изучать на основании корпуса вопросов в сочетании либо с их логической формой, либо с ответами на них.
- Естественный язык сложен, и все его аспекты трудно представить в формальной грамматике.

Библиографические и исторические заметки

Впервые *N*-граммные буквенные модели для моделирования языка были предложены Марковым ([1494], 1913). Первым, кто создал *n*-граммные модели для английского языка, был Клод Шеннон (Шеннон и Уивер [2039], 1949). **Модель мешка слов** получила свое название из цитаты, взятой из статьи лингвиста Зеллига Харриса ([968], 1954): "Язык — это не просто мешок слов, а инструмент с особыми свойствами". Норвиг в работе [1701] (2009) дает несколько примеров задач, которые могут быть решены с помощью *n*-граммных моделей.

Хомский указал на ограничения моделей конечных состояний по сравнению с контекстно-свободными моделями ([421], 1956; [422], 1957) и пришел к такому заключению: "Вероятностные модели не дают детального понимания некоторых основных проблем синтаксической структуры". Это верно, но вероятностные модели все же *обеспечивают* понимание некоторых *других* базовых проблем — проблем, которые в контекстно-свободных моделях игнорируются. Эти высказывания Хомского оказали неблагоприятное влияние, отпугивая многих исследователей от статистических моделей в течение двух десятилетий, пока эти модели не появились вновь с целью использования в области распознавания речи (Елинек [1131], 1976) и в науке о мышлении, в которой вновь появившаяся **теория оптимальности** (Смоленски и Принс [2103], 1993; Кагер [1167], 1999) постулировала, что язык действует, отыскивая наиболее вероятного кандидата, оптимально удовлетворяющего всем конкурирующим ограничениям.

Аддитивное сглаживание по Лапласу, впервые предложенное Пьером-Саймоном Лапласом ([1353], 1816), было формализовано Джеффрисом ([1130], 1948). Другие методы сглаживания включают интерполяционное сглаживание (Йелинек и Мерсер [1132], 1980), сглаживание Виттена–Белла ([2365], 1991), сглаживание Гуда–Тьюринга (Черч и Гейл [436], 1991), сглаживание Кнезера–Нея ([1242], 1995) и "глупый откат" (Брантс и др. [292], 2007). В работах Чена и Гудмена [413] (1996) и Гудмена [899] (2001) предоставлены обзоры методов сглаживания.

Простые *n*-граммные буквенные и словесные модели не являются единственно возможными вероятностными моделями. Модель **латентного распределения Дирихле** (Бли и др. [228], 2002; Хоффман и др. [1040], 2011) является вероятностной текстовой моделью, которая рассматривает документ как совокупность тем, каждая со своим распределением слов. Эта модель может рассматриваться как расширение и рационализация модели **латентно-семантической индексации** Дирвестера и соавт. ([591], 1990). Она также связана с моделью смеси множества причин (Сахами и др. [1962], 1996). И, конечно же, существует большой интерес к языковым моделям, не являющимся вероятностными, к таким как модели глубокого обучения, рассматриваемые в главе 24.

Жоли и соавт. в работе [1152] (2016) описывают множество хитростей, повышающих эффективность классификации текста. Иоахим ([1139], 2001) использовал статистическую теорию обучения и метод опорных векторов для проведения

теоретического анализа, в каких случаях классификация будет успешной. Апт и соавт. в работе [64] (1994) сообщают о достижении точности 96% в задаче классификация новостей агентства Рейтер в категорию "Доходы". Коллер и Сахами в статье [1269] (1997) сообщили о достижении точности до 95% при использовании наивного байесовского классификатора и до 98,6% — при использовании байесовского классификатора.

Шапир и Сингер в [1991] (2000) показали, что простые линейные классификаторы часто могут достигать почти такой же высокой точности, как более сложные модели, и при этом работать быстрее. Чжан и соавт. в статье [2434] (2016) описывают текстовый классификатор на уровне символов (а не на уровне слов). Виттен и соавт. ([2367], 1999) описали компрессионные алгоритмы классификации и показали глубокую связь между алгоритмом сжатия LZW и языковой моделью максимальной энтропии.

Общедоступный словарь Wordnet (Феллбаум [720], 2001) содержит около 100 тыс. слов и фраз, распределенных по частям речи и связанных семантическими отношениями, такими как *синоним*, *антоним* и *является частью*. Чарняк ([394], 1996), а также Кляйн и Маннинг ([1238], 2001) обсуждают синтаксический анализ с использованием древовидных диаграмм. Пакет "Британский национальный корпус" (Лич и др. [1377], 2001) содержит 100 млн слов, а всемирная сеть World Wide Web содержит несколько триллионов слов. Франц и Братс ([774], 2006) описывают общедоступный корпус n-грамм от Google, включающий 13 млн уникальных слов, извлеченных из триллионов слов, содержащихся в текстах веб-страниц. Бак и соавт. ([340], 2014) описывают аналогичный набор данных, подготовленный в проекте Common Crawl. Банк данных Penn Treebank (Маркус и др. [1492], 1993; Биз и др. [213], 2015) включает деревья синтаксического анализа для корпуса английского языка объемом 3 млн слов.

Многие методы на основе n-граммных моделей используются также при решении задач биоинформатики. Биостатистика и вероятностное НЛП постепенно сближаются, так как в каждой области необходимо работать с длинными, структурированными последовательностями элементов, выбранными из некоторого алфавита.

В ранних работах по расстановке меток частей речи (POS) использовались различные методы, включая наборы правил (Брилл [306], 1992), n-граммы (Черч [433], 1988), деревья решений (Маркес и Родригес [1497], 1998), скрытые марковские модели (HMM) (Брантс [291], 2000) и логистическую регрессию (Ратнапархи [1860], 1996). Исторически сложилось так, что модели логистической регрессии также называли "марковскими моделями с максимальной энтропией" (*maximum entropy Markov model* — MEMM), поэтому некоторые работы выходили с использованием этого названия. В книге Журафски и Мартина [1162], (2020) есть хорошая глава о расстановке меток POS. Энг и Джордан в статье [1676] (2002) сравнивают дискриминативные и порождающие модели в контексте задач классификации.

Как и семантические сети, контекстно-свободные грамматики впервые были открыты древними индийцами. Индийские грамматики (в частности, Панини, ок. 350 г. до н.э.) изучали шастрический санскрит (Ингерман [1113], 1967). Эти грамматики были заново изобретены Ноамом Хомским ([421], 1956) для анализа английского языка и, независимо от него, Джоном Бакусом ([104], 1959) и Питером Науром с целью анализа языка Алгол-58.

Вероятностные контекстно-свободные грамматики были впервые исследованы Бутом ([251], 1969) и Саломаа ([1965], 1969). Алгоритмы для PCFG были представлены в отличной короткой монографии Чарняка ([393], 1993) и превосходных подробных учебниках Мэннинга и Шютце ([1486], 1999), а также Журафски и Мартина ([1162], 2020). Бейкер в статье [116] (1979) предложил алгоритм "внутренний-внешний" для изучения PCFG. Модель **лексизированных PCFG** (Чарняк [395], 1997; Хва [1109], 1998) объединяет лучшие аспекты PCFG и *n*-граммных моделей. Коллинс в работе [463] (1999) описывает синтаксический анализ PCFG, который лексизируется с признаками заголовков, а Джонсон в работе [1140] (1998) показал, как точность PCFG зависит от структуры банка деревьев, на основании которого изучались его вероятности.

Предпринималось много попыток написать формальные грамматики естественных языков как в "чистой", так и в компьютерной лингвистике. Имеется несколько всеобъемлющих, но неформальных грамматик английского языка (Квирк и др. [1838], 1985; Макколи [1535], 1988; Хаддлестон и Паллум [1090], 2002). С 1980-х годов наметилась тенденция к лексикализации: больше информации помещается в лексикон и меньше в грамматику.

Лексико-функциональная грамматика (Lexical-functional grammar — LFG) (Бреснан [302], 1982) была первой крупной формальной грамматикой с высокой лексикализацией. Если лексикализацию довести до крайности, то можно получить **категориальную грамматику** (Кларк и Курран [448], 2004), в которой может быть всего лишь два грамматических правила, или **грамматику зависимостей** (Смит и Эйшнер [2091], 2008; Кюблер и др. [1317], 2009), в которой вообще нет синтаксических категорий, есть только отношения между словами.

Компьютерный синтаксический анализ был впервые продемонстрирован Ингве ([2406], 1955), а эффективные алгоритмы были разработаны в 1960-х годах с несколькими более поздними доработками (Казами [1195], 1965; Юнгер [2412], 1967; Эрли [671], 1970; Грэм и др. [915], 1980). Черч и Патил в работе [434] (1982) описывают синтаксическую неоднозначность и указывают на некоторые способы решения этой проблемы.

Кляйн и Маннинг в работе [1239] (2003) описывают синтаксический анализ по алгоритму A*, а Паулс и Кляйн в работе [1746] (2009) расширяют этот подход до синтаксического анализа методом *K*-лучших A*, в котором результатом является не один синтаксический разбор, а *K*-лучших. Голдберг и соавт. в работе [881] (2013) описывают необходимые приемы реализации, позволяющие гарантировать,

что синтаксический анализатор, работающий по методу поиска по лучу, потребует временных затрат на уровне $O(n)$, а не $O(n^2)$. Чжу и соавт. в работе [2443] (2013) описывают быстрый детерминистический синтаксический анализатор для естественных языков, работающий по методу "сдвиг-уменьшение". Сага и Лави ([1961], 2006) показали, как добавить поиск к такому синтаксическому анализатору, работающему по методу "сдвиг-уменьшение", сделав его более точным, но за счет некоторого снижения скорости.

Сегодня доступны высокоточные синтаксические анализаторы с открытым исходным кодом, в том числе Parsey McParseface от корпорации Google (Андор и др. [51], 2016), Stanford Parser (Чен и Мэннинг [410], 2014), Berkeley Parser (Китаев и Кляйн [1233], 2018) и анализатор SPACY. Все они выполняют обобщения через нейронные сети и достигают точности примерно 95% на тестовых наборах Wall Street Journal и Penn Treebank. По этому поводу имеются некоторые критические высказывания в отношении того, что авторы слишком узко фокусируется на измерении производительности на нескольких отобранных корпусах, а это может привести к переобучению.

Формальная семантическая интерпретация естественных языков берет свое начало в философии и формальной логике, в частности в работе Альфреда Тарского ([2174], 1935) по семантике формальных языков. Бар-Гилель ([127], 1954) был первым, кто рассмотрел эту проблему с точки зрения практического использования (например, индексикалов) и предположил, что для ее решения можно использовать средства формальной логики. Эссе Ричарда Монтегю "Английский как формальный язык" ([1606], 1970) является своего рода манифестом для логического анализа языка, но есть и другие книги, которые более читабельны (Доути и др. [644], 1991; Портнер и Парти [1814], 2002; Круз [504], 2011). Несмотря на то что программы семантической интерпретации предназначены для выбора наиболее вероятных интерпретаций, литературные критики (Эмпсон [687], 1953; Хоббс [1036], 1990) испытывали сомнения в отношении того, чем является неоднозначность — тем, что нужно устранять, или тем, что нужно лелеять. Норвиг в статье [1699] (1988) обсуждает проблемы рассмотрения одновременно многих интерпретаций, вместо того чтобы останавливаться на одной, наиболее вероятной интерпретации. Лакофф и Джонсон ([1344], 1980) дают привлекательный анализ и приводят каталог общепринятых метафор в английском языке. Мартин ([1510], 1990) и Гиббс ([850], 2006) предложили вычислительные модели для интерпретации метафор.

Первой системой обработки естественного языка, предназначенной для решения реальных задач, была система ответа на вопросы BASEBALL (Грин и др. [918], 1961), обрабатывавшая запросы к базе данных статистики по бейсболу. Вскоре после этого была выпущена система SHRDLU, разработанная Виноградом ([2361], 1972), отвечавшая на вопросы в отношении образцов горных пород, привезенных с Луны по программе "Аполлон".

Банко и соавт. в работе [123] (2002) представили систему ответов на вопросы AskMSR. Схожая система была создана Квоком и соавт. ([1331], 2001). Паска и Харабаджиу в статье [1738] (2001) обсуждают систему ответов на вопросы, победившую в соответствующих соревнованиях.

Современные подходы к семантической интерпретации обычно предполагают, что отображение синтаксиса на семантику будет изучаться на примерах (Целле и Муни [2423], 1996; Цеттлемойер и Коллинз [2428], 2005; Чжао и Хуан [2436], 2015). Первый важный результат по **грамматической индукции** был отрицательным: Голд ([878], 1967) показал, что невозможно надежно изучить точно корректную контекстно-свободную грамматику, заданную как набор строк из этой грамматики. Выдающиеся лингвисты, такие как Хомский ([422], 1957) и Пинкер ([1794], 2003), использовали результаты Голда, чтобы доказать, что должна существовать врожденная ▶**универсальная грамматика**, которая есть у всех детей от рождения. Согласно так называемому аргументу **бедности стимула**, дети не получают достаточного объема информации в своей лингвистической среде, чтобы выучить контекстно-свободную грамматику, поэтому они должны уже "знать" эту грамматику и им достаточно лишь настроить некоторые из ее параметров.

В то время как этот аргумент продолжает влиять на бо́льшую часть лингвистов — последователей Хомского, он был отклонен другими лингвистами (Пуллум [1826], 1996; Элман и др. [684], 1997) и большинством ученых-компьютерщиков. Уже в 1969 году Хорнинг показал, что *можно* изучить — в смысле "вероятно приближенно корректного" (PAC) обучения — *вероятностную* контекстно-свободную грамматику. С тех пор имело место много убедительных эмпирических демонстраций изучения языка только на положительных примерах, таких как изучение семантических грамматик с индуктивным логическим программированием (Магглетон и Де Ридт [1632], 1994; Муни [1611], 1999), тезисы докторской диссертации Шютце ([2011], 1995) и де Маркена ([560], 1996), а также вся линейка современных систем обработки языка на основе модели "Трансформер" (см. раздел 24.4). Ежегодно проводится Международная конференция по грамматическому выводу (*International Conference on Grammatical Inference* — ICGI).

Систему Джеймса Бейкера DRAGON (Бейкер [115], 1975) можно считать первой успешной системой распознавания речи. Бейкер был первым, кто использовал скрытую марковскую модель для обработки речи. После нескольких десятилетий систем, основанных на вероятностных языковых моделях, в этой области началось активное переключение на глубокие нейронные сети (Хинтон и др. [1031], 2012). Дэн в статье [603], (2016) описывает, как введение глубокого обучения обеспечило быстрое улучшение распознавания речи и оказало влияние на результаты, достигнутые в решении других задач обработки естественного языка. Сегодня глубокое обучение является доминирующим подходом для всех крупномасштабных систем распознавания речи. Распознавание речи можно рассматривать как первую область практического применения, в которой успехи глубокого обучения вышли на первый план, а область компьютерного зрения в этом смысле следует непосредственно за ней.

Интерес к области **поиска информации** был подстегнут стремительным увеличением спроса на поиск в Интернете. Крофт и соавт. ([502], 2010), а также Маннинг и др. ([1487], 2008) выпустили учебники, охватывающие основы этой области. Конференция TREC проводит ежегодный конкурс по IR-системам и публикует материалы об их результатах.

Брин и Пейдж в статье [308] (1998) описывают алгоритм PageRank, учитывающий ссылки между веб-страницами, а также дают общее представление о способах реализации систем веб-поиска. Сильверштейн и др. в статье [2067] (1998) привели результаты исследования журнала регистрации миллиарда поисковых запросов в Интернете. Журнал *Information Retrieval* и материалы ежегодной ведущей конференции SIGIR охватывают последние разработки в этой области.

Продвижение в области **извлечения информации** стимулируется ежегодной конференцией Message Understanding Conferences (MUC), спонсируемой правительством США. Обзоры систем, использующих шаблоны, предоставлены Рохе и Шабесом ([1902], 1997), Аппретом ([61], 1999) и Мусли ([1649], 1999). Большие базы данных фактов были извлечены Кравеном и др. ([493], 2000), Паска и др. ([1739], 2006), Митчеллом ([1597], 2007), а также Дурмом и Паска ([665], 2008). Фрейтаг и МакКаллум в работе [776] (2000) обсуждают использование скрытых марковских моделей для извлечения информации. Для решения этой задачи также использовались условные случайные поля (Лафферти и др. [1337], 2001; МакКаллум [1527], 2003). Учебное пособие с практическим руководством было предоставлено Саттоном и МакКаллумом ([2155], 2007). Сараваги в статье [1977] (2007) приводит всеобъемлющий обзор этой области.

Два ранних влиятельных подхода к автоматизированной инженерии знаний для обработки естественных языков предложили сначала Рилофф ([1883], 1993), показавший, что автоматически построенный словарь работает почти так же хорошо, как тщательно составленный вручную предметно-ориентированный словарь, а затем Яровский ([2400], 1995), показавший, что задача классификации слов по смыслу может быть решена с помощью обучения без учителя на корпусе непомеченного текста, — при этом достигается такая же точность, как и при обучении с учителем.

Идея одновременного извлечения шаблонов и примеров из небольшого количества помеченных примеров была разработана независимо и одновременно Блюмом и Митчеллом ([235], 1998), назвавшими этот подход "сообучением" (*cotraining*), и Брином ([307], 1998), который назвал этот метод DIPRE (Dual Iterative Pattern Relation Extraction — *двойное итеративное извлечение шаблона зависимостей*). Подобная же работа, под названием "bootstrapping" (*самозагрузка*) была выполнена Джонсом и соавт. ([1146], 1999). Затем этот метод был улучшен при разработке систем QXTRACT (Агиштейн и Гравано [20], 2003) и KNOWITALL (Этциони и др. [706], 2005). Машинное чтение было представлено Митчеллом ([1596], 2005) и Этциони и соавт. ([705], 2006) и является фокусом проекта TEXTRUNNER (Банко и др. [124], 2007; Банко и Этциони [125], 2008).

В этой главе обсуждение было сосредоточено на предложениях на естественном языке, но также вполне возможно осуществить извлечение информации на основе физической структуры или геометрического расположения текста, а не на его лингвистической структуре. Списки, таблицы, схемы, графики, диаграммы и так далее, независимо от того, закодированы они на языке HTML или доступны через визуальный анализ документов в формате PDF, являются хранилищем данных, которые могут быть извлечены и консолидированы (Херст [1101], 2000; Пинто и др. [1795], 2003; Кафарелла и др. [357], 2008).

Кен Черч в работе [435] (2004) показал, что исследования естественного языка циклически то опираются на данные (эмпиризм), то концентрируются на теориях (рационализм). Он описывает преимущества наличия хороших языковых ресурсов и схем оценки, но одновременно задается вопросом, не зашли ли мы слишком далеко (Черч и Хестнесс [437], 2019). Ранние лингвисты сосредоточивались на реальных данных об использовании языка, включая счетчики частоты. Ноам Хомский в работе [421] (1956) продемонстрировал ограничения моделей конечного состояния, что привело к смещению акцента на теоретические исследования синтаксиса при игнорировании практической реализации законов языка. Этот подход доминировал в течение двадцати лет, пока эмпиризм не вернулся благодаря успеху работ в области статистического распознавания речи (Елинек [1131], 1976). Сегодня акцент по-прежнему сохраняется на данных эмпирического языка, но повышается и интерес к моделям, рассматривающим конструкции более высокого уровня, такие как синтаксические и семантические зависимости, а не просто последовательности слов. Также имеет место сильный акцент на глубоком обучении нейронных сетевых моделей языка, которые подробно обсуждаются в главе 24.

Работы по тематике обработки естественного языка представляются на проводимой раз в два года конференции *Applied Natural Language Processing* (ANLP), конференции *Empirical Methods in Natural Language Processing* (EMNLP) и в журнале *Natural Language Engineering*. Широкий круг работ по обработке естественного языка также публикуется в журнале *Computational Linguistics* и представляется на его конференции, ACL, а также на конференции *International Computational Linguistics* (COLING). Журафский и Мартин в книге [1162] (2020) дали исчерпывающее введение в обработку речи и естественных языков.

Упражнения

23.1. В этом упражнении исследуется качество n-граммных моделей языка. Найдите или создайте корпус текстовых данных на одном языке, состоящий примерно из 100 тыс. слов. Разбейте его на слова и вычислите частоту каждого слова. Сколько различных слов в нем присутствует? Также подсчитайте частоты биграмм (два последовательных слова) и триграмм (три последовательных слова). Воспользуйтесь этими частотами для генерации языка: на основании моделей униграмм, биграмм и триграмм последовательно сформируйте образцы текста из

100 слов, выполняя их выбор случайным образом в соответствии со значениями частот. Сравните три сформированных текста с фактически имеющимся текстом на рассматриваемом языке. Наконец рассчитайте показатель перплексии (т.е. вероятность того, что модель не путает слова в предложениях) для каждой модели.

23.2. Напишите программу для **разделения** строки слов без пробелов. То есть для заданной строки, такой как URL- адрес "`thelongestlistofthelongeststuffatthelongestdomainnameatllonglast.com`", получите список образующих ее слов ["the", "longest", "list", ...]. Эта задача полезна для выполнения синтаксического анализа URL-адресов и исправления орфографических ошибок, когда слова полностью сливаются, а также для обработки таких языков, как китайский, в которых нет пробелов между словами. Данную задачу можно решить с помощью униграммной или биграммной словесной модели, а также алгоритма динамического программирования, подобного алгоритму Витерби.

23.3. *Закон распределения слов Ципфа* гласит: возьмите большой корпус текста, подсчитайте частоту каждого слова в нем, а затем ранжируйте эти частоты в порядке убывания. Пусть f_I будет I-й по величине частотой в этом списке, т.е. f_1 — это частота чаще всего встречающегося слова (в английском языке это обычно "the"), f_2 — частота второго наиболее распространенного слова и т.д. Закон Ципфа гласит, что значение f_I приблизительно равно α/I для некоторой постоянной α. Этот закон, как правило, оказывается весьма точным за исключением очень маленьких и очень больших значений I.

23.4. В Интернете выберите корпус не менее чем из 20 тыс. слов текста и проверьте закон Ципфа экспериментально. Определите меру ошибки и найдите значение α, при котором закон Ципфа точнее всего соответствует вашим экспериментальным данным. Постройте график с осями в логарифмическом масштабе, представляющий f_I в зависимости от I и α/I — в зависимости от I. (На графике с осями в логарифмическом масштабе функция α/I представляет собой прямую линию.) При проведении эксперимента обязательно удалите из текста все маркеры форматирования (например, теги HTML) и приведите к единому виду строчные и прописные буквы.

23.5. В этом упражнении предлагается разработать классификатор для выявления авторства: для заданного текста он должен предсказать, кто из двух возможных авторов написал этот текст. Получите образцы текста двух различных авторов, разделите их на обучающие и тестовые наборы и обучите две модели языка на обучающих наборах. Вам предоставляется возможность выбрать, какие признаки использовать (n-граммы слов или букв будут самым простым вариантом), но вы можете добавить любые дополнительные признаки, которые, по вашему мнению, будут полезны. Затем вычислите вероятность принадлежности текста для каждой языковой модели и выберите наиболее вероятную модель. Оцените точность этого метода. Как изменяется точность при изменении набора признаков? Эта подобласть лингвистики называется **стилометрией**; к числу

достижений в этой области относится идентификация автора "Заметок федералиста" (*Federalist Papers*) и некоторых произведений Шекспира, авторская принадлежность которых некогда оспаривалась. Хорошие результаты могут быть достигнуты даже с простой биграмной буквенной моделью.

23.6. В этом упражнении рассматривается задача распознавания нежелательной электронной почты (спама). Создайте два корпуса: один — из нежелательных почтовых сообщений, а другой — из обычных сообщений. Исследуйте каждый корпус и определите, какие признаки, скорее всего, будут полезны для классификации: униграммы или биграммы слов, длина сообщений, отправитель, время получения и т.д. Затем обучите алгоритм классификации (дерево решений, наивная байесовская модель или любой другой выбранный вами алгоритм) на обучающем наборе и определите его точность на тестовом наборе.

23.7. Создайте тестовый набор из десяти запросов и отправьте их трем крупнейшим поисковым системам Интернета. Оцените каждую из них на точность по 1, 3 и 10 документам. Можете ли вы объяснить различия между этими поисковыми машинами?

23.8. Попытайтесь определить, в какой из поисковых машин, рассматриваемых в предыдущем упражнении, используется приведение к нижнему регистру, выделение основы слова, выявление синонимов и исправление орфографических ошибок.

23.9. Оцените, сколько дискового пространства необходимо для индексации корпуса размером до 100 млрд веб-страниц. Укажите, какие предположения были вами приняты.

23.10. Напишите регулярное выражение или короткую программу для извлечения названий компаний. Проверьте ее на корпусе деловых новостных сообщений. Определите полноту и точность полученных результатов.

23.11. Рассмотрим проблему попытки оценить качество поисковой системы, возвращающей ранжированный список ответов (подобно большинству поисковых систем Интернета). Соответствующая мера качества зависит от предполагаемой модели того, чего пытается достичь отправитель поискового запроса, и от того, какую стратегию использует система. Для каждой из следующих моделей предложите соответствующий числовой показатель.

а) Отправитель запроса просматривает первые двадцать полученных результатов с целью получения как можно большего количества релевантной информации.

б) Отправителю запроса нужен только один релевантный документ, и он будет последовательно просматривать результаты, спускаясь по списку, пока не найдет первый подходящий.

в) Запрос отправителя был достаточно узким, и он может изучить все полученные ответы. Но он должен быть уверен, что просмотрел *все* доступное множество документов, релевантных его запросу. (Например, адвокат хочет быть уверен, что нашел все соответствующие прецеденты, и готов потратить на это значительные ресурсы.)

г) Отправителю запроса нужен только один релевантный документ, и он может позволить себе заплатить ассистенту за час работы по просмотру результатов поиска. За этот час ассистент успеет просмотреть 100 найденных документов. При этом ассистент потребует от отправителя запроса оплату за полный час, независимо от того, найдет ли он нужный документ немедленно или в конце отведенного на поиски часа.

д) Отправитель запроса просматривает все полученные результаты. Изучение документа стоит A долл., нахождение релевантного документа имеет ценность B долл., а невозможность найти релевантный документ обходится в C долл. за каждый ненайденный релевантный документ.

е) Отправитель запроса хочет собрать как можно больше релевантных документов, но нуждается в определенной поддержке. Он просматривает документы по порядку, и если уже просмотренные документы в основном хороши, то он продолжит просмотр, а в противном случае — прекратит его.

23.12. Прочитайте следующий текст с описанием процедуры из четырех этапов, чтобы понять его смысл, и запомните столько, сколько сможете. Достигнутые вами успехи будут оценены позже.

Эта процедура на самом деле довольно проста. Сначала нужно распределить вещи по разным группам. Конечно, для одного раза может оказаться достаточно одной стопки, в зависимости от того, сколько еще предстоит сделать. Если вам потребуется отправиться куда-то еще из-за нехватки необходимых средств, то в этом будет состоять следующий этап, а если нет, то вы уже достаточно хорошо подготовились. Складывая вещи, важно не переусердствовать. Это означает, что лучше, если каждый раз вещей будет меньше, чем слишком много. Когда процедура непродолжительна, это может показаться неважным, однако и здесь осложнения легко могут возникнуть. И в этом случае ошибка обойдется дорого. Сначала вся эта процедура будет казаться сложной. Однако вскоре она станет просто еще одним аспектом вашей жизни. Трудно представить себе, что в недалеком будущем исчезнет потребность в выполнении этой задачи, но об этом трудно судить со всей определенностью. После выполнения процедуры вещи нужно снова распределить по разным группам. Затем их можно разложить по соответствующим им местам. В конечном итоге они опять станут использованными и придется еще раз повторить весь цикл. Однако это занятие — часть повседневной жизни.

23.13. *Грамматика HMM*, по существу, является стандартной скрытой марковской моделью, переменной состояния которой является N (нетерминальная, с такими значениями, как *Det*, *Adjective*, *Noun* и т.д.), а переменной свидетельства — W (слово с такими значениями, как "*is*", "*duck*" и т.д.). Модель HMM включает априорное распределение $\mathbf{P}(N_0)$, модель перехода $\mathbf{P}(N_{t+1} \mid N_t)$ и модель восприятия $\mathbf{P}(W_t \mid N_t)$. Покажите, что каждая грамматика HMM может быть записана как PCFG. (*Подсказка.* Начните с размышлений о том, как априорная HMM может быть представлена правилами PCFG для символа предложения. Может оказаться полезным проиллюстрировать рассуждения для конкретной грамматики HMM со значениями A, B для N и значениями x, y — для W.)

23.14. Рассмотрите следующую PCFG для простых глагольных групп.

0,1: *VP* → *Verb*
0,2: *VP* → *Copula Adjective*
0,5: *VP* → *Verb the Noun*
0,2: *VP* → *VP Adverb*
0,5: *Verb* → **is** (*быть, являться*)
0,5: *Verb* → **shoots** (*подстрелить*)
0,8: *Copula* → **is** (*есть, является*)
0,2: *Copula* → **seems** (*кажется*)
0,5: *Adjective* → **unwell** (*нехороший*)
0,5: *Adjective* → **well** (*хороший*)
0,5: *Adverb* → **well** (*хорошо*)
0,5: *Adverb* → **badly** (*плохо*)
0,6: *Noun* → **duck** (*утка*)
0,4: *Noun* → **well** (*колодец*)

а) Что из следующего имеет ненулевую вероятность в качестве *VP*?
   i) shoots the duck well well well
   ii) seems the well well
   iii) shoots the unwell well badly
б) Какова вероятность генерации фразы "is well well"?
в) Какие типы неоднозначности проявляются в фразе из п. *б*?
г) При заданной произвольной PCFG можно ли рассчитать вероятность того, что эта PCFG сгенерирует строку ровно из 10 слов?

23.15. Рассмотрите следующую простую PCFG для именных групп.

0,6: *NP* → *Det*\ *AdjString*\ *Noun*
0,4: *NP* → *Det*\ *NounNounCompound*
0,5: *AdjString* → *Adj*\ *AdjString*
0,5: *AdjString* → Λ
1,0: *NounNounCompound* → *Noun*
0,8: *Det* → **the** (*определенный артикль — этот*)
0,2: *Det* → **a** (*неопределенный артикль — некоторый*)
0,5: *Adj* → **small** (*маленький*)
0,5: *Adj* → **green** (*зеленый*)
0,6: *Noun* → **village** (*деревня*)
0,4: *Noun* → **green** (*зелень*)

Здесь Λ обозначает пустую строку.

а) Какая самая длинная именная группа (*NP*) может быть сгенерирована этой грамматикой?
   i) три слова
   ii) четыре слова
   iii) бесконечно много слов
б) Что из нижеперечисленного с ненулевой вероятностью может быть сгенерировано как полная *NP*?

i) a small green village
ii) a green green green
iii) a small village green

в) Какова вероятность генерации фразы "the green green"?
г) Какие типы неоднозначности проявляются в фразе из п. *в*?
д) При заданных произвольной PCFG и любой конечной последовательности слов можно ли рассчитать вероятность того, что эта последовательность была сгенерирована данной PCFG?

23.16. Опишите основные различия между языком Java (или любым другим языком программирования, с которым вы знакомы) и естественным языком, прокомментировав в каждом случае проблему "понимания". Рассмотрите такие аспекты, как грамматика, синтаксис, семантика, прагматика, композициональность, контекстная зависимость, лексическая неоднозначность, синтаксическая неоднозначность, разрешение ссылок (включая местоимения), фоновые знания, а самое главное — укажите, что означает "понимание" текста на том и ином языке.

23.17. В этом упражнении речь идет о грамматиках для очень простых языков.

а) Напишите контекстно-свободную грамматику для языка $a^n b^n$.
б) Напишите контекстно-свободную грамматику для языка палиндромов — множества всех строк, вторая половина которых обратна первой половине.
в) Напишите контекстно-зависимую грамматику для языка дубликатов: множества всех строк, вторая половина которых является копией первой половины.

23.18. Рассмотрите предложение "Someone walked slowly to the supermarket" (*Некто медленно шел в супермаркет*) и следующий словарь.

Pronoun → **someone**	*Verb* → **walked**
Adv → **slowly**	*Prep* → **to**
Article → **the**	*Noun* → **supermarket**

Какая из трех приведенных ниже грамматик в сочетании с этим словарем вырабатывает указанное предложение? Покажите соответствующее дерево (деревья) синтаксического анализа.

A

S → *NP VP*
NP → *Pronoun*
NP → *Article Noun*
VP → *VP PP*
VP → *VP Adv Adv*
VP → *Verb*
PP → *Prep NP*
NP → *Noun*

B

S → *NP VP*
NP → *Pronoun*
NP → *Noun*
NP → *Article NP*
VP → *Verb Vmod*
Vmod → *Adv Vmod*
Vmod → *Adv*
Adv → *PP*
PP → *Prep NP*

C

S → *NP VP*
NP → *Pronoun*
NP → *Article NP*
VP → *Verb Adv*
Adv → *Adv Adv*
Adv → *PP*
PP → *Prep NP*
NP → *Noun*

Для каждой из трех приведенных выше грамматик запишите три правильных и три неправильных предложения на английском языке, построенные с помощью этой грамматики. Каждое предложение должно значительно отличаться от другого, состоять не меньше чем из шести слов и включать несколько новых словарных записей (которые вы должны определить). Предложите способы

совершенствования каждой грамматики, предотвращающие выработку неправильных предложений на английском языке.

23.19. Приведите несколько примеров английских выражений с указанием времени, таких как "two o'clock" (*два часа*), "midnight" (*полночь*) или "12:46". Кроме того, подготовьте некоторые примеры, являющиеся грамматически неправильными, такие как "thirty o'clock" (*тридцать часов*) или "half past two fifteen" (*половина три пятнадцати*). Напишите грамматику для языка выражений с указанием времени.

23.20. Некоторые лингвисты утверждают следующее.

Дети, изучающие язык, слышат только положительные примеры языка и не слышат каких-либо отрицательных примеров. Следовательно, гипотеза о том, что "каждое возможное предложение есть в языке" согласуется со всеми наблюдаемыми примерами. Более того, это самая простая совместимая гипотеза. Кроме того, все грамматики языков, которые являются надмножествами истинного языка, также совместимы с наблюдаемыми данными. И все же дети действительно (более или менее) вырабатывают правильную грамматику. Отсюда следует, что они начинают с очень сильных врожденных грамматических ограничений, которые априори исключают все эти более общие гипотезы.

Прокомментируйте слабые места в этой аргументации с точки зрения статистического обучения.

23.21. В этом упражнении требуется преобразовать грамматику $\mathcal{E}_0$ в нормальную форму Хомского (CNF). Это преобразование выполняется в пять этапов.

1. Добавить новый начальный символ.
2. Исключить правила с пустой строкой ϵ.
3. Устранить присутствие нескольких слов в правой части.
4. Исключить правила вида $(X \rightarrow Y)$.
5. Преобразовать длинные правые части в бинарные правила.

а) Начальный символ S может появляться только в левой части CNF. Везде замените S новым символом S' и добавьте правило вида $S \rightarrow S'$.

б) Пустая строка ϵ не может появиться в правой части CNF. Грамматика $\mathcal{E}_0$ не имеет никаких правил с пустой строкой ϵ, так что в нашем случае это не проблема.

в) Слово может появляться в правой части только в правилах вида $(X \rightarrow word)$. Замените каждое правило вида $(X \rightarrow \ldots word \ldots)$ двумя правилами, $(X \rightarrow \ldots W' \ldots)$ и $(W' \rightarrow word)$, используя новый символ W'.

г) Правила вида $(X \rightarrow Y)$ не разрешены в CNF; они должны иметь вид $(X \rightarrow Y Z)$ или $(X \rightarrow word)$. Замените каждое правило вида $(X \rightarrow Y)$ множеством правил вида $(X \rightarrow \ldots)$, по одному на каждое правило $(Y \rightarrow \ldots)$, где $(\ldots)$ представляет один или более символов.

д) Замените каждое правило вида $(X \rightarrow Y Z \ldots)$ двумя правилами, $(X \rightarrow Y Z')$ и $(Z' \rightarrow Z \ldots)$, где Z' — новый символ.

Представьте результаты выполнения каждого этапа процесса и окончательный набор правил.

23.22. Рассмотрим следующую игрушечную грамматику.

S → *NP VP*
NP → *Noun*
NP → *NP* **and** *NP*
NP → *NP PP*
VP → *Verb*
VP → *VP* **and** *VP*
VP → *VP PP*
PP → *Prep NP*
Noun → **Sally**; **pools**; **streams**; **swims**
Prep → **in**
Verb → **pools**; **streams**; **swims**

а) Представьте все деревья синтаксического анализа в этой грамматике для предложения "Sally swims in streams and pools" (*Салли плавает в ручьях и бассейнах*).

б) Представьте все строки таблицы, которая будет создана (не вероятностным) синтаксическим анализатором CYK для этого предложения.

23.23. Рассмотрим следующую грамматику PCFG.

S → *NP VP* [1,0]
NP → *Noun* [0,6] | *Pronoun* [0,4]
VP → *Verb NP* [0,8] | *Modal Verb* [0,2]
Noun → **can** [0,1] | **fish** [0,3] | ... (*жестяная банка*) | (*рыба*)
Pronoun → **I** [0,4] | ... *я*)
Verb → **can** [0,1] | **fish** [0,1] | ... (*консервировать*) | (*ловить рыбу*)
Modal → can [0,3] | ... (*уметь, мочь*)

Предложение "I can fish" (*Я могу ловить рыбу*) в этой грамматике имеет два дерева синтаксического анализа. Представьте эти два дерева, их априорные вероятности и их условные вероятности с учетом данного предложения.

23.24. Расширенная контекстно-свободная грамматика может представлять языки, которые обычная контекстно-свободная грамматика представить не может. Приведите расширенную контекстно-свободную грамматику для языка $a^n b^n c^n$. Допустимыми значениями для переменных расширения являются 1 и *SUCCESSOR*(*n*), где *n* является значением. Правило для предложения на этом языке имеет вид

$$S(n) \rightarrow A(n)\, B(n)\, C(n).$$

Приведите правило (или правила) для каждого из *A*, *B* и *C*.

23.25. Рассмотрите следующее предложение (взято из "Нью-Йорк Таймс" от 28 июля 2008 года).

Banks struggling to recover from multibillion-dollar loans on real estate are curtailing loans to American businesses, depriving even healthy companies of money for expansion and hiring.

(*Банки, изо всех сил пытающиеся оправиться от многомиллиардных кредитов на недвижимость, сокращают кредиты американским предприятиям, лишая даже здоровые компании денег на расширение и наем работников.*)

а) Какие из слов в этом предложении лексически неоднозначны?

б) Найдите в этом предложении два случая синтаксической неоднозначности (в действительности их более двух).

в) Приведите пример метафоры, присутствующей в этом предложении.

г) Можете ли вы найти в нем семантическую неоднозначность?

23.26. Не обращаясь повторно к тексту, приведенному в упражнении 23.12, ответьте на перечисленные ниже вопросы.

а) Что представляют собой упомянутые в тексте четыре этапа?

б) Какой из этапов пропущен?

в) Что представляют собой "вещи", упомянутые в тексте?

г) Какого рода ошибка может оказаться дорогостоящей?

д) Как лучше поступить в данном случае: проделать эту процедуру со слишком малым или слишком большим количеством вещей? Объясните, почему.

23.27. Выберите пять предложений и передайте их в любую интерактивную систему автоматического перевода. Переведите их с исходного языка на другой язык, а затем снова на исходный. Оцените, насколько полученные при этом предложения являются грамматически правильными и сохранившими смысл. Повторите этот процесс. Во второй итерации будут получены худшие или такие же самые результаты? Влияет ли на качество результатов выбор промежуточного языка? Если вы знаете промежуточный язык, посмотрите на перевод на него одного из предложений. Подсчитайте и опишите сделанные ошибки, и приведите свои предположения, почему эти ошибки были сделаны.

23.28. Мы забыли упомянуть, что текст, приведенный в упражнении 23.12, должен быть озаглавлен "Стирка белья". Еще раз прочитайте этот текст и ответьте на вопросы, приведенные в упражнении 23.26. Удалось ли вам на этот раз лучше справиться с заданием? Исследователи использовали этот текст в эксперименте, проводимом под контролем, и обнаружили, что для его понимания очень важен заголовок. Какие выводы можно сделать в отношении того, как работают язык и память?

ГЛАВА 24

Глубокое обучение при обработке естественного языка

В этой главе объясняется, как глубокие нейронные сети можно использовать для решения различных задач обработки языка, благодаря их способности к захвату и фиксации структуры естественного языка наряду с его гибкостью и текучестью.

В главе 23 рассматривались ключевые элементы естественного языка, в том числе грамматика и семантика. Системы, основанные на синтаксическом и семантическом анализе, продемонстрировали успехи при решении многих задач, но их производительность ограничена бесконечной сложностью лингвистических феноменов в реальных текстах. Учитывая огромное количество текста, доступного в форме, подходящей для машинного прочтения, имеет смысл проанализировать, могут ли подходы, основанные на машинном обучении, управляемом данными, быть более эффективными. Исследование этого предположения будет проведено с использованием инструментов, предоставляемых системами глубокого обучения (глава 21).

В разделе 24.1 показано, как можно улучшить обучение, представив слова не как атомарные значения, а как точки в многомерном пространстве. В разделе 24.2 обсуждается использование рекуррентных нейронных сетей для захвата смыслового значения и дальнего контекста на больших расстояниях в тех случаях, когда текст обрабатывается последовательно. Раздел 24.3 посвящен в основном машинному переводу, одной из наиболее успешных областей применения глубокого обучения к задачам обработки естественного языка. В разделах 24.4 и 24.5 рассматриваются такие модели, которые можно обучить на основании большого количества неразмеченного текста, а затем успешно применить к конкретным задачам с достижением в конечном итоге современного уровня производительности. Наконец, в разделе 24.6 подводится итог тому, какие рубежи в области обработки естественного языка уже достигнуты и в каком направлении может идти дальнейшее развитие.

24.1. Метод встраивания слов

Необходимо найти такое представление слов, которое не требует ручной разработки признаков, но позволяет обобщать связанные слова, т.е. слова, которые связаны синтаксически ("colorless" (*бесцветный*) и "ideal" (*идеальный*) — прилагательные), семантически ("cat" (*кошка*) и "kitten" (*котенок*) относятся к кошачьим), тематически ("sunny" (*солнечный*) и "sleet" (*дождь со снегом*) определяют погодные условия), с точки зрения выражения мнения ("awesome" (*здорово*) противоположно "miserably" (*ужасно*)) или иным образом.

Как следует закодировать слово во входном векторе **x**, чтобы его можно было обрабатывать в нейронной сети? В разделе 21.2.1 для этой цели предлагалось использовать **вектор быстрого** (однократного) **кодирования** (*one-hot vector*), т.е. кодировать i-е слово в словаре единичным значением бита в i-й позиции входного вектора с нулевыми значениями во всех остальных его битах. Но такое представление не позволяет выявлять сходства между словами.

Следуя изречению лингвиста Джона Р. Ферта ([744], 1957) "Вы узнаете слово по его окружению", можно было бы представить каждое слово вектором n-граммных счетчиков всех фраз, в которых это слово появляется. Однако необработанные n-граммные счетчики крайне громоздки. Для словаря со 100 тыс. слов потребуется 10^{25} 5-грамм для отслеживания (хотя векторы в этом 10^{25}-мерном пространстве были бы довольно разреженными — большинство счетчиков были бы нулевыми). Можно получить лучшее обобщение, если уменьшить это представление до вектора меньшего размера, возможно, лишь с несколькими сотнями измерений. Такое меньшее, но более плотное представление получило название "вектор ►**встраивания слов**" (*word embedding*). Встраивания слов *изучаются автоматически* на основании данных (позже будет показано, как это делается.) Что представляют собой эти изученные встраивания слов? С одной стороны, каждое из них — просто вектор чисел, в котором отдельные измерения и их числовые значения не имеют какого-либо различимого смысла:

"aardvark" = [–0,7; +0,2; –3.2;...]
"abacus" = [+0,5; +0,9; –1,3;...]
...
"zyzzyva" = [–0,1; +0,8; –0,4;...].

С другой стороны, пространство признаков обладает тем свойством, что в конечном итоге схожие слова будут иметь сходные векторы. Это можно увидеть на рис. 24.1, на котором легко различаются отдельные кластеры для названий стран (france, greece, germany), отношений родства (nephew, niece, uncle, aunt), транспорта (car, bicycle, truck) и еды (banana, pizza, apple).

Как оказалось, по причинам, которые остаются не вполне понятными, у векторов встраивания слов есть и дополнительные свойства, помимо простой близости подобных слов. Предположим, например, что рассматриваются векторы **A** для

Афин и **B** для Греции. Складывается впечатление, что для этих слов разность векторов **B** – **A** кодирует отношение "страна/столица". Для других подобных пар — Франция и Париж, Россия и Москва, Замбия и Лусака — разность соответствующих векторов, по существу, одна и та же.

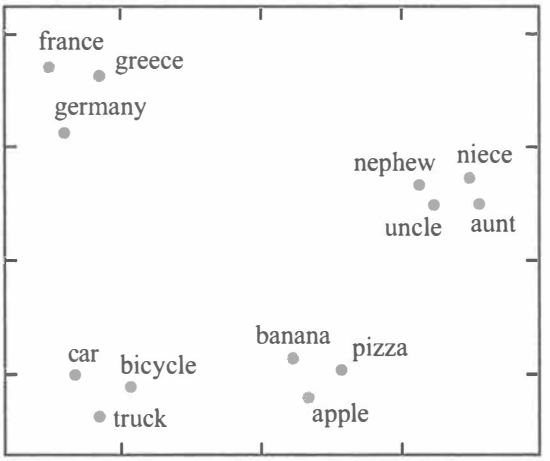


Рис. 24.1. Векторы встраивания слов, вычисленные по алгоритму GloVe (обучение выполнялось на корпусе текста с 6 млрд слов). В этой визуализации 100-мерные векторы слов спроецированы на плоскость с двумя измерениями, — подобные по смыслу слова располагаются рядом друг с другом

Это свойство можно использовать для решения задач поиска аналогичных слов, таких как "Афины для Греции как Осло для [*что*]?" Обозначим вектор слова "Осло" как **C** и вектор искомого слова как **D**, тогда можно предположить, что **B**–**A** = **D** – **C**, что дает **D** = **C** + (**B** – **A**). Если вычислить этот новый вектор **D**, то можно обнаружить, что его значение оказалось ближе к вектору слова "Норвегия", чем к любому другому вектору. На рис. 24.2 показано, что такой тип векторной арифметики работает для многих отношений.

Тем не менее нет никакой гарантии, что конкретный алгоритм встраивания слов, работающий с заданным корпусом текста, зафиксирует определенные семантические отношения. Метод встраивания слов популярен потому, что доказал свою способность давать результат, являющийся хорошим представлением для последующих языковых задач (таких, как предоставление ответов на вопрос, перевод или обобщение), а не потому, что он сам по себе гарантированно обеспечивает ответы на аналогичные вопросы.

Использование векторов встраивания слов вместо их однократного кодирования оказалось полезным практически для всех приложений глубокого обучения, направленных на решение задач обработки естественного языка. Действительно, во

многих случаях оказалось возможным использовать общие **предварительно обученные** векторы, полученные от любого из нескольких возможных поставщиков, для решения собственных конкретных задач обработки языка. На момент написания этой книги к чаще всего используемым предобученным векторным словарям можно отнести WORD2VEC, GloVe (Global Vectors) и FASTTEXT, включающий встраивания для 157 языков. Использование предобученной модели позволяет сэкономить много времени и усилий. Больше информации об этих ресурсах можно найти в разделе 24.5.1.

A	B	C	D = C + (B − A)	Отношение
Athens	Greece	Oslo	Norway	*Столица*
Astana	Kazakhstan	Harare	Zimbabwe	*Столица*
Angola	kwanza	Iran	rial	*Валюта*
copper	Cu	gold	Au	*Обозначение элемента*
Microsoft	Windows	Google	Android	*Операционная система*
New York	New York Times	Baltimore	Baltimore Sun	*Городская газета*
Berlusconi	Silvio	Obama	Barack	*Фамилия*
Switzerland	Swiss	Cambodia	Cambodian	*Национальность*
Einstein	scientist	Picasso	painter	*Профессия*
brother	sister	grandson	granddaughter	*Отношение родства*
Chicago	Illinois	Stockton	California	*Штат*
possibly	impossibly	ethical	unethical	*Отрицание*
mouse	mice	dollar	dollars	*Множественное число*
easy	easiest	lucky	luckiest	*Превосходная степень*
walking	walked	swimming	swam	*Прошедшее время*

Рис. 24.2. Модель встраивания слов иногда может отвечать на вопрос "**A** для **B** как **C** для [*что*]?" с использованием векторной арифметики: при заданных векторах встраивания для слов **A**, **B** и **C** вычислите вектор **D** = **C** + (**B** − **A**) и найдите слово, наиболее близкое к **D**. (Ответы в столбце **D** были автоматически рассчитаны моделью, а описания в столбце "Отношение" добавлены вручную.) Адаптировано из статей Миколова и соавт. [1568] (2013) и [1570] (2014)

Также можно обучить и собственные векторы встраивания слов, — обычно это делается одновременно с обучением сети для конкретной задачи. В отличие от общих предварительно обученных векторов, встраивания слов, предназначенные для конкретной задачи, могут быть обучены на тщательно отобранном корпусе и в результате приобрести тенденцию подчеркивать те аспекты слов, которые полезны именно для этой задачи. Например, предположим, что речь идет о задаче проставления меток частей речи (POS; см. раздел 23.1.6). Напомним, что ее решение предполагает правильное прогнозирование части речи для каждого слова в предложении. Хотя это довольно простая задача, она вовсе не тривиальна, поскольку в английском языке многие слова могут быть помечены разным способом, например

слово *cut* в предложении может быть глаголом настоящего времени (переходным или непереходным), глаголом прошедшего времени, частью инфинитива, причастием прошедшего времени, прилагательным или существительным. Если соседнее временное наречие относится к прошлому, то можно предположить, что это конкретное вхождение слова *cut* является глаголом прошедшего времени; и тогда можно надеяться, что применение метода встраивания позволит зафиксировать этот аспект наречий — определять отношение к прошлому.

Задача проставления POS-тегов может служить хорошим введением в применение глубокого обучения при обработке естественных языков, поскольку лишена запутанности, характерной для более сложных задач, таких как предоставление ответов на вопросы (см. раздел 24.5.3). При наличии корпуса предложений с проставленными POS-тегами изучение параметров для создания векторов встраивания слов и проставления POS-тегов выполняется одновременно. Этот процесс выполняется следующим образом.

1. Выберите ширину w (нечетное количество слов) для окна прогнозирования, которое будет использоваться для пометки каждого слова. Типичное значение $w = 5$; это означает, что тег для слова прогнозируется на основе двух слов слева и двух слов справа от него. Все предложения в используемом корпусе разделите на перекрывающиеся окна длиной w. Каждое такое окно будет представлять собой один обучающий пример, состоящий из w слов в качестве входных данных и категории POS среднего слова в качестве выхода.
2. Создайте словарь из всех уникальных токенов слов, встречающихся в обучающих данных более пяти раз. Обозначим общее количество слов в словаре как v.
3. Отсортируйте полученный словарь в произвольном порядке (например, в алфавитном).
4. Выберите значение d в качестве размера каждого вектора встраивания слова.
5. Создайте новую весовую матрицу $\mathbf{E}$ размером $v \times d$; это будет матрица встраивания слов. Строка i матрицы $\mathbf{E}$ — это вектор встраивания i-го слова в словаре. Инициализируйте матрицу $\mathbf{E}$ случайными значениями (или воспользуйтесь данными из предварительно обученных векторов).
6. Настройте нейронную сеть, которая выводит метку части речи, как показано на рис. 24.3. Первый слой будет состоять из w копий матрицы встраивания. Далее можно использовать два дополнительных скрытых слоя, $\mathbf{z}_1$ и $\mathbf{z}_2$ (с весовыми матрицами $\mathbf{W}_1$ и $\mathbf{W}_2$ соответственно), за которыми следует слой с функцией активации softmax, дающий выходное распределение вероятности $\hat{\mathbf{y}}$ по возможной категории части речи для среднего слова:

$$\begin{aligned}\mathbf{z}_1 &= \sigma(\mathbf{W}_1\mathbf{x}) \\ \mathbf{z}_2 &= \sigma(\mathbf{W}_2\mathbf{z}_1) \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{W}_{out}\mathbf{z}_2).\end{aligned}$$

7. Чтобы закодировать последовательность из *w* слов в качестве входного вектора, просто найдите вектор встраивания для каждого ее слова и конкатенируйте эти векторы. Результатом будет вектор **x** длиной *wd* со значениями, представленными действительными числами. Даже если данное слово будет иметь такой же вектор встраивания, как тот, который находится на первой позиции, последней позиции или где-то посередине, каждый вектор встраивания будет умножен на другую часть первого скрытого слоя; следовательно, имеет место неявное кодирование относительной позиции каждого слова.
8. Обучайте матрицу весов **E** и остальные матрицы весов $\mathbf{W}_1$, $\mathbf{W}_2$ и $\mathbf{W}_{out}$, используя метод градиентного спуска. Если все пойдет хорошо, среднее слово, *cut*, будет помечено как глагол прошедшего времени, исходя из свидетельства в окне *w*, включающего временную ссылку на прошедшее время в виде слова "yesterday" (*вчера*), субъектное местоимение третьего лица "they" (*они*) непосредственно перед словом *cut*, и т.д.

Альтернативой методу встраивания слов является **модель на уровне символов**, в которой ввод представляет собой последовательность символов, каждый из которых закодирован как вектор быстрого кодирования. Такая модель должна изучать, как символы объединяются с целью формирования слов. Авторы большинства исследовательских работ в области обработки естественных языков НЛП придерживаются уровня слов, а не кодировки на уровне символа.

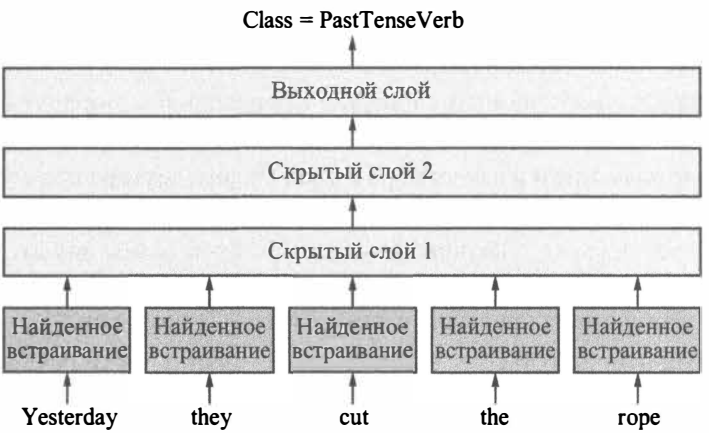


Рис. 24.3. Модель с прямой связью для простановки тегов части речи. В этой модели в качестве входных данных используется окно из пяти слов, а результатом является прогноз тега части речи для слова в середине окна, — в данном случае это *cut*. Модель способна учитывать положение слова, поскольку каждое из пяти входных встраиваний умножается на разные части первого скрытого слоя. В процессе обучения значения параметров для создания векторов встраивания слов и для трех слоев сети простановки тегов изучаются одновременно

24.2. Рекуррентные нейронные сети для задач обработки естественного языка

Хотя теперь у нас уже есть хорошее представление для каждого слова в отдельности, язык образуется из упорядоченных последовательностей слов, в которых важен **контекст**, т.е. набор окружающих слов. Для простых задач, таких как пометка слов в предложении тегами части речи, маленькое окно фиксированного размера — например, из пяти слов — обычно предоставляет достаточно контекста для ее решения.

Более сложные задачи, такие как предоставление ответов на вопросы или разрешение ссылок, в качестве контекста могут потребовать десятков слов. Например, в предложении "Eduardo told me that Miguel was very sick so I took **him** to the hospital" (*Эдуардо сказал мне, что Мигель был очень болен, поэтому я отвез* его *в больницу*) выяснение, что слово **him** (*его*) относится к Мигелю, а не к Эдуардо, требует знания контекста, охватывающего все предложение из 14 слов, от первого до последнего.

24.2.1. Модели языка с рекуррентными нейронными сетями

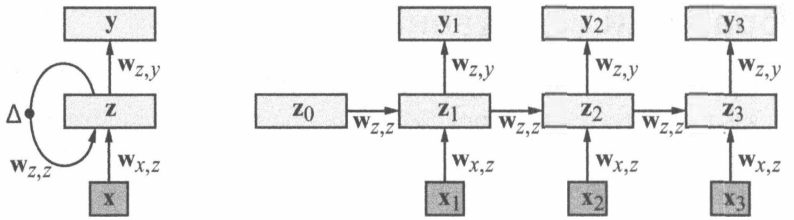
Начнем с проблемы создания **модели языка** с достаточным контекстом. Напомним, что модель языка — это распределение вероятностей по последовательностям слов. Такое решение позволяет прогнозировать следующее слово в тексте, учитывая все предыдущие слова, и часто используется в качестве строительного блока при решении более сложных задач.

Построение модели языка с помощью n-граммной модели (см. раздел 23.1) или сети с прямой связью с фиксированным окном из n слов может столкнуться с большими затруднениями из-за проблемы контекста: либо требуемый контекст будет превышать фиксированный размер окна, либо модель будет иметь слишком много параметров, либо и то, и другое одновременно.

Кроме того, для сети с прямой связью актуальна проблема **асимметрии**: что бы она ни выучила о, скажем, появлении слова *him* в качестве 12-го слова в предложении, ей придется переучиваться при появлении этого слова в других позициях в предложении, поскольку для каждой позиции веса будут разными.

В разделе 21.6 была представлена **рекуррентная нейронная сеть**, или **RNN**, предназначенная для последовательной обработки данных временных рядов, по одному этапу за раз. Это позволяет предположить, что RNN могут оказаться полезными при обработке естественных языков, по одному слову на каждом этапе. Такая сеть была приведена на рис. 21.8, — здесь она приводится еще раз как рис. 24.4.

В рекуррентной нейронной сети модели языка каждое входное слово кодируется как вектор встраивания слова $\mathbf{x}_t$. В сети имеется скрытый слой $\mathbf{z}_t$, который передает в качестве входных данных информацию от одного временного этапа к следующему. Общая цель заключается в выполнении мультиклассовой классификации, где



а) б)

Рис. 24.4. а) Принципиальная схема базовой рекуррентной нейронной сети, в которой скрытый слой **z** включает рекуррентные соединения, — символ Δ здесь указывает на задержку. **б)** Та же самая сеть, развернутая по трем временным этапам для создания представления в виде сети с прямой связью. Обратите внимание, что одни и те же веса используются на всех временных этапах

классы — слова из словаря. Следовательно, выход $\mathbf{y}_t$ будет распределением вероятности softmax по возможным значениям следующего слова в предложении.

Используя архитектуру RNN, можно решить проблему слишком большого количества параметров. Количество параметров в весовых матрицах $w_{z,z}$, $w_{x,z}$ и $w_{z,y}$ остается постоянным независимо от количества слов, — это $O(1)$. Это важное отличие от сетей с прямой связью, в которых количество параметров определяется как $O(n)$, и n-граммных моделей, в которых количество параметров определяется как $O(v^n)$, где v — размер словаря.

Архитектура RNN также позволяет решить проблему асимметрии, так как веса остаются прежними для каждой позиции слова.

В некоторых случаях архитектура RNN также позволяет решить проблему ограниченного контекста. Теоретически нет предела тому, насколько далеко назад во входных данных может заглядывать модель. Каждое обновление скрытого слоя $\mathbf{z}_t$ имеет доступ как к текущему входному слову $\mathbf{x}_t$, так и к предыдущему состоянию скрытого слоя $\mathbf{z}_{t-1}$, а это означает, что информация о любом слове на входе может храниться в скрытом слое неопределенно долго, копируясь (или соответствующим образом изменяясь) от одного временного шага к другому. Но, конечно, объем памяти в скрытом слое **z** ограничен, поэтому он не может помнить все обо всех предыдущих словах.

На практике модели RNN хорошо справляются с различными задачами, но не со всеми. Может оказаться затруднительным предсказать, будет ли их применение успешным для определенной проблемы. Одним из факторов, способствующих успеху, является то, что учебный процесс должен побуждать сеть выделять пространство для хранения в скрытом слое **z** таких аспектов входных данных, которые действительно окажутся полезными для решения поставленной задачи.

Для обучения модели языка на базе RNN используется процесс обучения, описанный ранее, в разделе 21.6.1. Входными данными $\mathbf{x}_t$ здесь являются слова из обучающего корпуса текста, а наблюдаемыми выходными данными — слова из него же, но смещенные на 1. Иначе говоря, для обучающего текста "hello world" первым входом $\mathbf{x}_1$ является вектор встраивания слова "hello", а первым выходом $\mathbf{y}_1$ будет вектор встраивания для слова "world". Модель обучается предсказывать следующее слово, и ожидается, что для достижения этой цели она будет использовать свой скрытый слой, размещая в нем некую полезную информацию. Как объясняется в разделе 21.6.1, вычисляется разность между наблюдаемым выходом и фактическим выходом, вычисленным сетью, которая затем распространяется обратно во времени, с целью обеспечения сохранения весов одинаковыми для всех временных этапов.

После обучения модель можно будет использовать для генерации произвольного текста. Модели передается начальное входное слово $\mathbf{x}_1$, на основании которого она производит выходное слово $\mathbf{y}_1$, представленное как распределение вероятностей softmax по словам. Далее из этого распределения выбирается одно слово, которое записывается в качестве выходного для времени t, а затем передается сети в качестве следующего входного слова $\mathbf{x}_2$. Этот цикл повторяется столько раз, сколько требуется. При выборе слова из распределения $\mathbf{y}_1$ доступно множество вариантов: можно всегда извлекать наиболее вероятное слово, можно выбирать каждое слово в соответствии с его вероятностью или можно иногда выбирать менее вероятные слова, чтобы добавить больше разнообразия в сгенерированный результат. Правило формирования выборки — это гиперпараметр модели.

Вот пример случайного текста, сгенерированного языковой моделью RNN, обученной на текстах Шекспира (Карпати [1190], 2015).

Marry, and will, my lord, to weep in such a one were prettiest;
Yet now I was adopted heir
Of the world's lamentable day,
To watch the next way with his father with his face?

(*Женись, и будешь, милорд, плакать в таком, были самые красивые;*
Все же теперь я был усыновленным наследником
Из мира печального дня,
Чтобы смотреть следующий путь с его отцом и с его лицом?)

24.2.2. Классификация с использованием рекуррентных нейронных сетей

Рекуррентные нейронные сети также можно использовать для решения других задач обработки языка, таких как проставление тегов части речи (POS) или разрешение кореферентности (*coreference resolution*). В обоих случаях входной и скрытый слои будут одинаковыми, но для задачи присвоения тегов части речи на выходе сети будет распределение softmax по тегам POS, тогда как для задач разрешения

кореферентности это будет распределение softmax по возможным предшественникам. Например, если на вход сети поступит слово "его" в предложении "Эдуардо сказал мне, что Мигель был очень болен, поэтому я взял **его** в больницу", на выходе сети высокая вероятность кореферентности с этим словом должна быть присвоена слову-предшественнику "Мигель".

Обучение сети RNN для выполнения классификации проводится так же, как и при изучении модели языка. Разница лишь в том, что для обучающих данных дополнительно потребуются метки — метки частей речи или указания на кореферентность. Это значительно затрудняет сбор обучающих данных в сравнении с задачей изучения модели языка, когда все, что требуется, — это непомеченный текст.

Цель обучения модели языка — дать прогноз для *n*-го слова на основании предыдущих слов. Но для задач классификации нет никаких причин, по которым следует ограничиваться рассмотрением только предыдущих слов. Может оказаться очень полезным заглянуть в предложении вперед. В приведенном выше примере кореферентности слово, кореферентное с "его", было бы совсем иным, если бы предложение заканчивалось словами "навестить Мигеля", а не "в больницу", поэтому в таких задачах крайне важно заглядывать вперед. Из экспериментов по отслеживанию движения глаз известно, что читающий человек не следует взглядом по строкам строго слева направо.

Для захвата контекста справа можно использовать ▶**двунаправленную RNN** — рекуррентную нейронную сеть, объединяющую посредством конкатенации две отдельные модели: модель, работающую справа налево, и модель, работающую слева направо. Пример использования двунаправленной RNN в задаче расстановки тегов POS приведен на рис. 24.5.

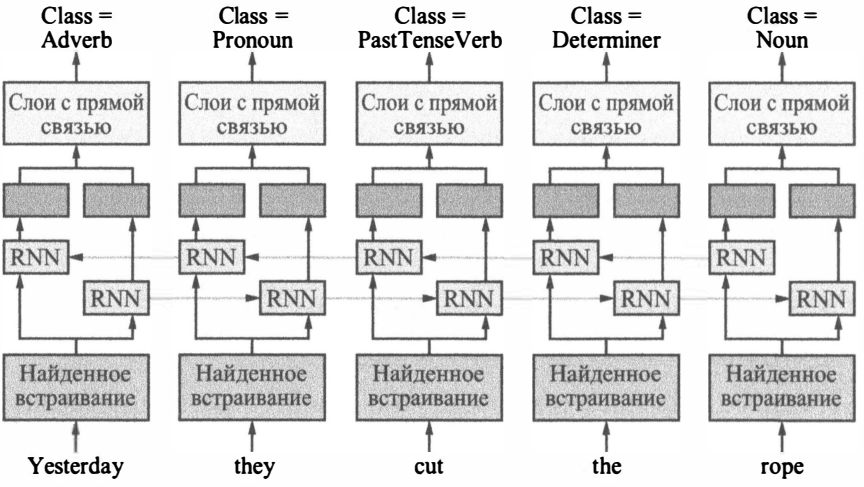


Рис. 24.5. Двунаправленная сеть RNN для задачи расстановки тегов POS

В случае обычной многослойной сети RNN $\mathbf{z}_t$ является скрытым вектором последнего слоя. Для двунаправленной сети RNN $\mathbf{z}_t$ обычно рассматривается как конкатенация векторов на выходе модели слева направо и модели справа налево.

Сети RNN также могут использоваться для задач классификации на уровне предложений (или на уровне документов), — в таких сетях конечный слой выдает единственное выходное значение, а не поток выходных значений, по одному на каждом временном этапе. Например, при **анализе тональности текста** цель состоит в том, чтобы классифицировать текст как выражающий *позитивное* или *негативное* мнение. Например, предложение "This movie was poorly written and poorly acted" (*Этот фильм был плохо написан и плохо поставлен*) следует классифицировать как негативное мнение. (В некоторых схемах анализа тональности текста используется более двух категорий или результатом является числовое скалярное значение.)

Использовать сети RNN для задач уровня предложения немного сложнее, так как необходимо получить совокупное представление для всего предложения $\mathbf{y}$ на основании пословных выходных значений $\mathbf{y}_t$ в сети RNN. Простейший способ сделать это — использовать скрытое состояние сети RNN, соответствующее последнему входному слову, поскольку к этому моменту сеть уже прочтет все предложение. Однако такой подход может вызвать неявное смещение модели в сторону предоставления большего внимания концу предложения. Другой распространенный метод — объединение всех скрытых векторов. Например, ▶**объединение по среднему** (*average pooling*) вычисляется как поэлементное среднее по всем скрытым векторам:

$$\tilde{\mathbf{z}} = \frac{1}{s}\sum_{t=1}^{s}\mathbf{z}_t.$$

Объединенный d-мерный вектор $\tilde{\mathbf{z}}$ может быть передан в один или несколько слоев с прямой связью, прежде чем поступит в выходной слой.

24.2.3. Архитектура LSTM для задач обработки естественного языка

Уже говорилось, что сети RNN иногда позволяют решить проблему ограниченного контекста. В теории любая информация может быть передана от одного скрытого слоя к следующему для любого количества временных этапов. Но на практике информация при этом может быть потеряна или искажена, как в детской игре в испорченный телефон, когда играющие по очереди, от первого к последнему, шепчут соседу на ухо какое-либо сообщение. Как правило, то сообщение, которое получит последний играющий, будет искажено и довольно сильно отличаться от исходного. Для сетей RNN данная проблема аналогична проблеме **исчезающего градиента**, обсуждавшейся в разделе 21.1.2, — за исключением того, что здесь речь идет о состоянии слоев во времени, а не о глубоких слоях.

В разделе 21.6.2 была представлена модель **долгой краткосрочной памяти** (*long short-term memory* — LSTM). Это тип сетей RNN с вентилями, не подверженными проблеме несовершенного воспроизведения сообщения при переходе от одного временного этапа к следующему. Вместо этого модель LSTM может решить *запомнить* некоторые части ввода, копируя их на следующий этап времени, и забыть остальные части. Рассмотрим модель языка для обработки текста, подобного приведенному ниже.

The athletes, who all won their local qualifiers and advanced to the finals in Tokyo, now...
(*Спортсмены, которые все выиграли свои местные отборочные матчи и вышли в финал в Токио, теперь...*)

Если в данный момент спросить модель, какое следующее слово более вероятно, "compete" (*соревнуются*) или "competes" (*соревнуется*), можно ожидать, что она выберет слово "compete", потому что оно согласуется по числу с подлежащим "The athletes". Сеть LSTM может научиться создавать скрытый признак для лица и числа подлежащего в предложении и копировать этот признак вперед без изменений, пока не потребуется сделать выбор, подобный данному. Обычные сети RNN (или n-граммные модели для подобного случая) в такой ситуации часто ошибаются в длинных предложениях со многими промежуточными словами между подлежащим и глаголом.

24.3. Модели "от последовательности к последовательности"

Одним из наиболее широко изучаемых классов задач в обработке естественного языка является ▶**машинный перевод** (*machine translation* — ▶**MT**), целью которого является перевод предложения с ▶**исходного языка** на ▶**целевой язык**, например с испанского на английский. Модель машинного перевода в этом случае обучается на большом корпусе пар исходных/целевых предложений. Конечная цель обучения состоит в точном переводе новых предложений, которых не было в данных, использовавшихся при обучении.

Можно ли использовать рекуррентную нейронную сеть для создания системы MT? Безусловно, можно закодировать исходное предложение с помощью RNN. Если бы существовало однозначное соответствие между словами на исходном и целевом языках, то можно было бы рассматривать задачу машинного перевода как простую задачу расстановки тегов, — получив исходное слово "perro" на испанском языке, просто помечаем его соответствующим английским словом "dog" (*собака*). Но на самом деле слова не всегда находятся в отношении один к одному: в испанском языке три слова "caballo de mar" соответствуют одному английскому слову "seahorse" (*морской конек*), а два слова "perro grande" переводятся как "big dog" (*большая собака*) с обратным порядком слов. Изменение порядка слов может быть еще более экстремальным: в английском языке подлежащее обычно

находится в начале предложения, тогда как на языке фиджи подлежащее обычно его завершает. Так как же можно сгенерировать предложение на целевом языке?

Создается впечатление, что следует генерировать по одному слову за раз, но при этом следить за контекстом, чтобы помнить те части исходного предложения, которые еще не были переведены, и отслеживать ту его часть, которая уже была переведена, чтобы не повторяться. Также кажется, что для некоторых предложений потребуется обработать все исходное предложение, прежде чем приступать к генерированию целевого. Другими словами, генерация каждого слова целевого предложения зависит от всего исходного предложения и от всех ранее сгенерированных слов целевого предложения.

Все это придает процедуре генерации текста при машинном переводе большое сходство с работой стандартной RNN-модели языка, обсуждавшейся в разделе 24.2. Конечно, если обучать сеть RNN на текстах на английском языке, то она с большей вероятностью породит фразу "big dog" (*большая собака*), чем "dog big." (*собака большая*). Однако в задачах MT требуется генерировать не просто любое случайное предложение на целевом языке, — на этом языке необходимо сгенерировать такое предложение, которое *соответствует* предложению исходного языка. Самый простой способ достичь этого — использовать две сети RNN: одну — для источника, а другую — для цели. В сеть RNN источника поступает исходное предложение, а затем окончательное скрытое состояние этой сети используется как начальное скрытое состояние для сети RNN цели. В результате каждое целевое слово неявно обусловливается как всем исходным предложением, так и предыдущими целевыми словами.

Такая архитектура нейронной сети называется базовой моделью ▶**от последовательности к последовательности** (*sequence-to-sequence model* или сокращенно *seq2seq model*), — пример такой сети приведен на рис. 24.6. Модели от последовательности к последовательности чаще всего используются для машинного перевода, но могут использоваться и при решении ряда других задач, таких как автоматическая генерация текстовой подписи к изображению или обобщение: преобразование длинного текста в более короткий с сохранением прежнего смыслового значения.

Базовые модели от последовательности к последовательности были значительным прорывом в области обработки естественных языков в целом и машинного перевода в частности. Согласно Bу и соавт. ([2392], 2016) этот подход привел к уменьшению количества ошибок на 60% по сравнению с предыдущими методами машинного перевода. Однако этим моделям свойственны три серьезных недостатка.

- **Смещение к ближнему контексту.** Как бы RNN-сети не стремились сохранить знания о прошлом, эти знания должны соответствовать их скрытому состоянию. Например, пусть рекуррентная нейронная сеть обрабатывает слово (или временной этап) под номером 57 в последовательности из 70 слов. Скрытое состояние, вероятно, будет содержать больше информации о слове, которое обрабатывалось на этапе 56, чем о слове, обрабатывавшемся

на этапе 5, поскольку каждый раз, когда скрытый вектор обновляется, эта операция должна сопровождаться заменой некоторого количества уже существующей информации новой информацией. Такое поведение является элементом, намеренно заложенным при разработке данной модели, и часто имеет большое значение при обработке естественного языка, так как ближайший контекст, как правило, более важен. Однако дальний контекст иногда имеет решающее значение, но может быть уже утерян в модели RNN, — даже модели LSTM испытывают затруднения в подобных ситуациях.

- **Фиксированный предел для размера контекста**. В модели перевода на базе сети RNN все исходное предложение сжимается в единый вектор скрытого состояния фиксированного размера. В сетях LSTM, используемых в современных моделях обработки естественного языка, обычно имеется около 1024 измерений, и если необходимо в этих 1024 измерениях представить предложение, скажем, из 64 слов, то на каждое слово будет отведено только 16 измерений, чего совершенно недостаточно для сложных предложений. С другой стороны, увеличение размера вектора скрытого состояния может привести к замедлению обучения и переобучению.
- **Медленная последовательная обработка**. Как указывалось в разделе 21.3, нейронные сети достигают значительного повышения эффективности за счет обработки обучающих данных, сгруппированных в пакеты, — это позволяет реализовать все преимущества эффективной аппаратной поддержки матричной арифметики. С другой стороны, сети RNN просто вынуждены работать с обучающими данными по одному слову на каждом этапе.

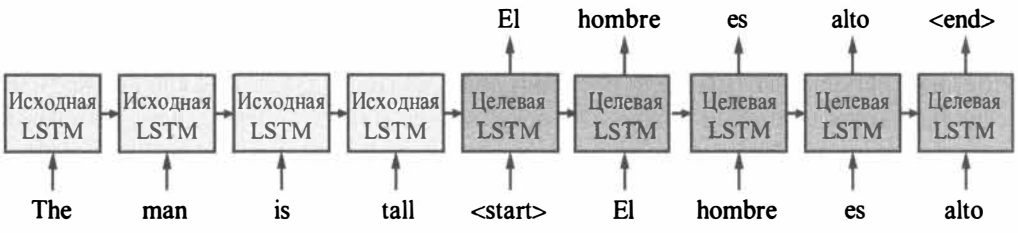


Рис. 24.6. Базовая модель от последовательности к последовательности. Каждый блок представляет один временной этап работы сети LSTM. (Для упрощения встроенный и выходной слои не показаны.) Серией последовательных этапов в сеть вводятся слова исходного предложения "The man is tall" (*Мужчина высокий*), за которыми следует тег `<start>`, указывающий, что сеть должна начать генерировать целевое предложение. Конечное скрытое состояние на этапе конца исходного предложения используется как скрытое состояние начала генерации целевого предложения. Далее каждое слово целевого предложения в момент времени t используется в качестве ввода в момент времени $t + 1$, пока сеть не выведет тег `<end>`, указывающий, что генерация предложения завершена

24.3.1. Механизм внимания

Что мы получим, если целевую сеть RNN обусловить *всеми* скрытыми векторами из исходной сети RNN, а не только последним? Это уменьшит влияние недостатков смещения к ближнему контексту и фиксированного предела для размера контекста, позволив модели с одинаковым успехом обращаться к любому предыдущему слову. Один способ достижения такого доступа заключается в объединении всех скрытых векторов RNN-источника. Однако это решение приводит к огромному увеличению количества весов с одновременным увеличением времени вычислений и, возможно, появлением переобучения. Вместо этого можно воспользоваться тем фактом, что, когда целевая сеть RNN генерирует целевые слова по одному на каждом этапе, весьма вероятно, что на самом деле лишь небольшая часть исходного вектора релевантна каждому генерируемому целевому слову.

Крайне важно понимать, что целевая сеть RNN для каждого генерируемого слова должна обращать внимание на разные части исходного вектора. Предположим, что сеть обучена переводу с английского языка на испанский. На входе получены слова "The front door is red" (*Входная дверь красная*), за которыми следует маркер конца предложения, означающий, что пора начинать вывод испанских слов. Поэтому в идеале следует сначала обратить внимание на слово "The" и сгенерировать соответствующее испанское слово "La", а затем обратить внимание на слово "door" и сгенерировать вывод "puerta", и т.д.

Эту концепцию можно формализовать с помощью компонента нейронной сети, называемого ▶**вниманием** (*attention*), который может использоваться для создания "основанного на контексте обобщения" исходного предложения в представлении с фиксированной размерностью. Вектор контекста $\mathbf{c}_i$ содержит наиболее релевантную информацию для генерации следующего целевого слова и будет использоваться в качестве дополнительного ввода в целевой сети RNN. Модель от последовательности к последовательности, использующая компонент внимания, называется моделью ▶**от последовательности к последовательности с вниманием**. Если стандартная целевая сеть RNN описывается как

$$\mathbf{h}_i = RNN(\mathbf{h}_{i-1}, \mathbf{x}_i),$$

то целевая сеть RNN для модели от последовательности к последовательности с вниманием может быть описана как

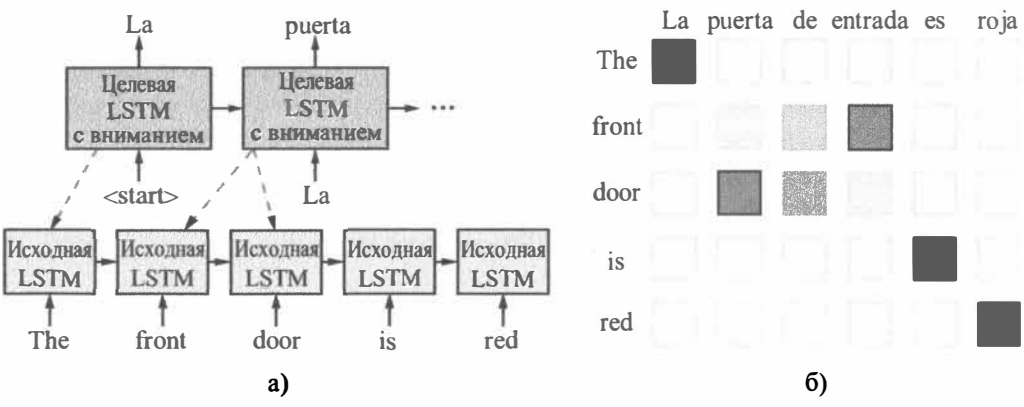
$$\mathbf{h}_i = RNN(\mathbf{h}_{i-1}, [\mathbf{x}_i; \mathbf{c}_i]),$$

где $[\mathbf{x}_i; \mathbf{c}_i]$ — конкатенация входного вектора $\mathbf{x}_i$ и контекстного вектора $\mathbf{c}_i$, определенная как

$$r_{ij} = \mathbf{h}_{i-1} \cdot \mathbf{s}_j$$
$$a_{ij} = e^{r_{ij}} / (\sum_k e^{r_{ik}})$$
$$\mathbf{c}_i = \sum_j a_{ij} \cdot \mathbf{s}_j,$$

где $\mathbf{h}_{i-1}$ является вектором целевой сети RNN, который будет использоваться для прогнозирования слова на временном этапе i, а $\mathbf{s}_j$ — выходной вектор исходной сети RNN для исходного слова (или временного этапа) j. Как $\mathbf{h}_{i-1}$, так и $\mathbf{s}_j$ являются d-мерными векторами, где d — скрытый размер. Следовательно, значение r_{ij} является необработанной "оценкой внимания" между текущим целевым состоянием и исходным словом j. Затем эти оценки нормализуются в вероятность a_{ij} с использованием функции активации softmax для всех исходных слов. В завершение эти вероятности используются для генерации значения взвешенного среднего исходных векторов сети RNN $\mathbf{c}_i$ (еще один d-мерный вектор).

Пример модели от последовательности к последовательности с вниманием приведен на рис. 24.7, *а*. Здесь есть несколько важных деталей, на которых следует остановиться. Во-первых, сам компонент внимания не имеет обученных весовых коэффициентов и поддерживает последовательности переменной длины как на стороне источника, так и на стороне цели. Во-вторых, как и в большинстве других методов моделирования нейронных сетей, о которых шла речь выше, механизм внимания полностью скрыт. Программист не диктует, когда и какая информация используется; модель сама изучает, что ей следует использовать. Механизм внимания также можно комбинировать с многослойными рекуррентными нейронными сетями. В этом случае механизм внимания, как правило, применяется к каждому слою.



а) б)

Рис. 24.7. а) Модель от последовательности к последовательности с вниманием для перевода с английского языка на испанский. Пунктирные линии представляют механизм внимания. **б)** Пример матрицы вероятностей механизма внимания для пары предложений на двух языках, — более темные прямоугольники здесь представляют более высокие значения a_{ij}. Сумма вероятностей в каждом столбце матрицы внимания равна единице

Использование в механизме внимания вероятностной формулировки с функцией активации softmax служит трем целям. Во-первых, это делает внимание функцией дифференцируемой, что необходимо для работы метода обратного распространения. Несмотря на то что сам по себе механизм внимания не имеет обучаемых весовых коэффициентов, градиенты по-прежнему возвращаются через этот механизм к исходной и целевой сетям RNN. Во-вторых, вероятностная формулировка позволяет модели захватывать определенные типы контекстуализации на больших расстояниях, которые, возможно, не были захвачены исходной сетью RNN, поскольку механизм внимания способен рассмотреть сразу всю исходную последовательность и научиться сохранять только то, что важно, игнорируя все остальное. В-третьих, вероятностный характер механизма внимания позволяет сети представлять неопределенность — если сети не известно точно, какое исходное слово переводить следующим, она может распределить вероятности внимания по нескольким параметрам, а затем в конечном итоге выбрать слово, используя целевую сеть RNN.

В отличие от большинства других компонентов нейронных сетей, вероятности механизма внимания часто могут интерпретироваться людьми и являются интуитивно значимыми. Например, в случае машинного перевода вероятности внимания часто соответствуют межсловесному выравниванию, которое мог бы выполнить человек. Это наглядно представлено на рис. 24.7, *б*.

Модели от последовательности к последовательности естественным образом подходят для машинного перевода, но почти любая задача обработки естественного языка может быть переопределена как задача, ориентирующаяся на модель этого типа. Например, система предоставления ответов на вопросы может обучаться на входных данных, состоящих из вопроса, за которым следует разделитель, за которым следует правильный ответ.

24.3.2. Декодирование

Во время обучения модели от последовательности к последовательности предпринимаются попытки максимизировать вероятность каждого слова в целевом обучающем предложении, обусловленную исходным предложением и всеми предыдущими целевыми словами. После завершения обучения модели передается исходное предложение и ее задача состоит в том, чтобы сгенерировать соответствующее целевое предложение. Как показано на рис. 24.7, можно генерировать целевое предложение по одному слову на каждом этапе, а затем возвращать его в качестве обратной связи для слова, которое генерируется на следующем этапе. Данная процедура называется ▶**декодированием**.

Самая простая форма декодирования — выбирать слово с наибольшей вероятностью на каждом временном этапе, а затем передавать его в качестве ввода для следующего этапа. Этот подход называется ▶**жадным декодированием**, потому что после генерации каждого целевого слова система полностью соглашается

с гипотезой, которую она сгенерировала до этого момента. Проблема здесь состоит в том, что цель декодирования — максимизировать вероятность *всей* полученной целевой последовательности, чего можно и не достичь при жадном декодировании. Например, рассмотрим использование метода жадного декодирования при переводе на испанский язык предложения на английском, которое уже упоминалось выше: "The front door is red." (*Входная дверь красная*).

Правильным переводом будет "La puerta de entrada es roja" — на английском языке буквально "The door of entry is red." (*Дверь входа красная*). Предположим, что целевая сеть RNN правильно генерирует первое слово "La", соответствующее артиклю "The" в исходном предложении. Далее жадный декодер может предложить слово "entrada" для исходного слова "front", но это будет ошибкой — согласно правильному порядку слов в испанском языке существительное "puerta" должно стоять *перед* определяющим его прилагательным. Жадное декодирование работает быстро, рассматривая только один вариант выбора на каждом этапе, что требует немного времени, но эта модель не имеет механизма для исправления возможных ошибок.

Можно попробовать улучшить механизм внимания таким образом, чтобы он всегда обращал внимание на слово справа и каждый раз делал правильное предположение. Однако для многих предложений невозможно правильно угадать все слова в начале предложения, пока не станет известно, что у него в конце.

Лучшим подходом является поиск оптимального декодирования (или, по крайней мере, хорошего) с использованием одного из алгоритмов поиска, обсуждавшихся в главах 3 и 4. Чаще всего выбираемый вариант — **поиск по лучу** (см. раздел 4.1.3). В контексте процедуры декодирования при машинном переводе при поиске по лучу на каждом этапе обычно сохраняются первые k гипотез, затем каждая расширяется на одно слово, используя лучшие k вариантов слов, после чего выбираются k лучших из полученных k^2 новых гипотез. Когда все гипотезы в луче генерируют специальный токен <end>, алгоритм передает на выход гипотезу с наивысшей оценкой.

Визуализация процесса поиска по лучу при декодировании приведена на рис. 24.8. По мере того как модели глубокого обучения становятся все более и более точными, обычно появляется возможность использовать луч меньшего размера. На текущий момент в нейронных сетевых моделях машинного обучения используется луч размером от 4 до 8, тогда как в более ранних поколениях статистических моделей машинного перевода размер луча достигал 100 или более.

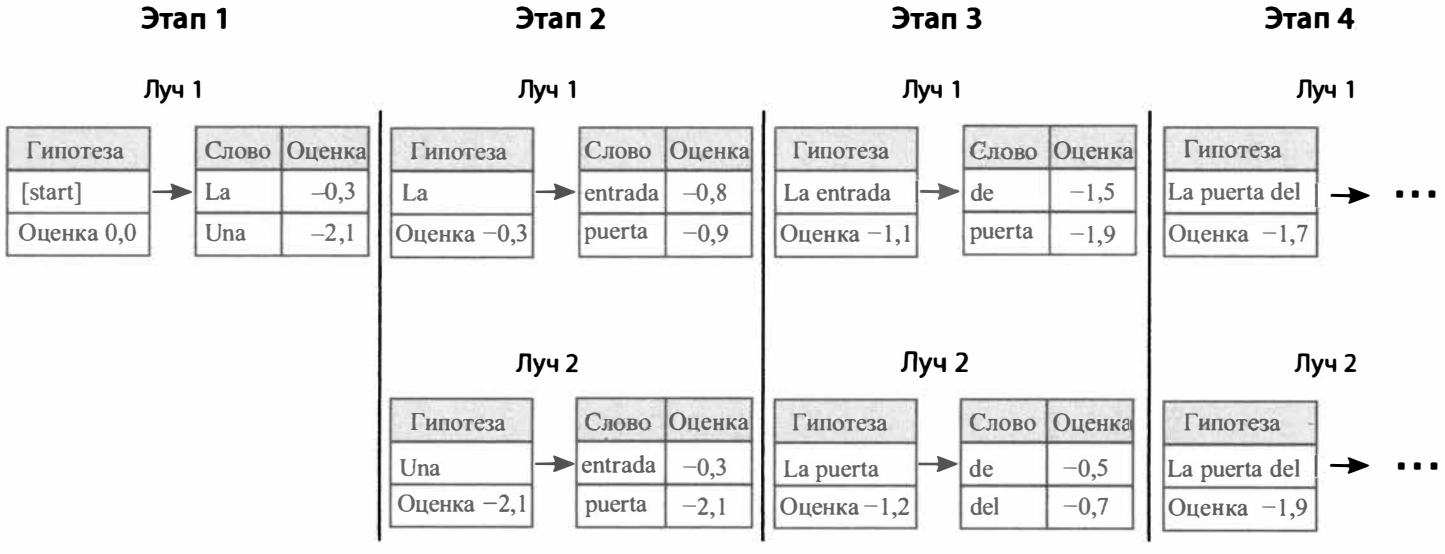


Рис. 24.8. Поиск по лучу с размером луча $b = 2$. Оценка каждого слова — это логарифм вероятности, генерируемой целевой сетью RNN с функцией активации softmax, а оценкой каждой гипотезы является сумма оценок слов. На временном этапе 3-я гипотеза с самой высокой оценкой "La entrada" позволяет генерировать продолжения только с очень небольшой вероятностью и поэтому отбрасывается — "выпадает из луча"

24.4. Архитектура "трансформер"

Во влиятельной статье "Attention is all you need" (*Внимание — это все, что вам нужно*) (Вашвани и др. [2266], 2018) впервые была представлена архитектура ▶**трансформер**, в которой используется механизм ▶**самовнимания**, способный моделировать дальний контекст без последовательной зависимости.

24.4.1. Механизм самовнимания

В рассмотренных выше моделях класса от последовательности к последовательности механизм внимания был направлен от целевой сети RNN к исходной сети RNN. Концепция ▶**самовнимания** расширяет этот механизм таким образом, что каждая последовательность скрытых состояний применяет механизм внимания и к самой себе: исходная — к исходной, а целевая — к целевой. Это позволяет модели дополнительно захватывать удаленный (и близлежащий) контекст в каждой последовательности.

Наиболее простой способ применения механизма самовнимания — это когда матрица внимания непосредственно формируется точечным произведением входных векторов. Однако этот вариант проблематичен: точечное произведение вектора с самим собой всегда будет высоким, поэтому каждое скрытое состояние будет смещено по отношению к себе. В архитектуре "трансформер" эта проблема решается предварительным проецированием входа на три разных представления с использованием трех разных весовых матриц.

- ▶**Вектор запроса** $\mathbf{q}_i = \mathbf{W}_q\mathbf{x}_i$ — это вектор, *от которого* направлено внимание, подобно целевому вектору в стандартном механизме внимания.
- ▶**Вектор ключа** $\mathbf{k}_i = \mathbf{W}_k\mathbf{x}_i$ — это вектор, *на который* направлено внимание, подобно исходному вектору в базовом механизме внимания.
- ▶**Вектор значения** $\mathbf{v}_i = \mathbf{W}_v\mathbf{x}_i$ — это генерируемый контекст.

В стандартном механизме внимания сети ключа и значения идентичны, но интуитивно понятно, что есть смысл сделать их отдельными представлениями. Результаты кодирования i-го слова $\mathbf{c}_i$ можно рассчитать, применив механизм внимания к проецируемым векторам:

$$r_{ij} = (\mathbf{q}_i \cdot \mathbf{k}_j)/\sqrt{d}$$

$$a_{ij} = e^{r_{ij}}/(\sum_k e^{r_{ik}})$$

$$\mathbf{c}_i = \sum_j a_{ij} \cdot \mathbf{v}_j ,$$

где d — размерность векторов $\mathbf{k}$ и $\mathbf{q}$. Обратите внимание, что i и j — это индексы в одном предложении, поскольку контекст кодируется с использованием механизма самовнимания. В каждом слое архитектуры "трансформер" механизм

самовнимания использует скрытые векторы из предыдущего слоя, который исходно является слоем векторов встраивания слов.

Имеется несколько важных деталей, о которых здесь стоит упомянуть. Во-первых, механизм самовнимания *асимметричен*, так как значение r_{ij} отличается от r_{ji}. Во-вторых, для улучшения числовой стабильности в формулу был добавлен коэффициент масштабирования $\sqrt{d}$. В-третьих, кодировку для всех слов в предложении можно рассчитать одновременно, поскольку приведенные выше уравнения могут быть записаны с использованием матричных операций, которые можно эффективно вычислять параллельно на современном специализированном оборудовании.

Выбор того, какой контекст использовать, не задается предварительно, а полностью изучается моделью на обучающих примерах. Контекстное обобщение $\mathbf{c}_i$ представляет собой сумму по всем предыдущим позициям в предложении. Теоретически в $\mathbf{c}_i$ может быть представлена и информация из предложения, но на практике важная информация иногда оказывается утерянной, поскольку она, по сути, усредняется по всему предложению. Один из способов решения этой проблемы называется ▶**множественным вниманием** (*multiheaded attention*). Предложение делится на вплоть до m равных частей, и механизм внимания модели применяется к каждой из этих m частей. Каждая часть получает собственный набор весов, а затем полученные результаты конкатенируются в вектор $\mathbf{c}_i$. За счет выполнения операции конкатенации, а не суммирования выделение важных частей предложения упрощается.

24.4.2. От механизма самовнимания к архитектуре "трансформер"

Механизм самовнимания является лишь одним из компонентов модели архитектуры "трансформер". Каждый слой в модели архитектуры "трансформер" состоит из нескольких подслоев, и в каждом слое этой модели в первую очередь применяется самовнимание. Выход модуля внимания пропускается через слои с прямой связью, в которых одинаковые весовые матрицы прямой связи независимо применяются к каждой позиции. Нелинейная функция активации, обычно ReLU, применяется после первого слоя с прямой связью. Для решения проблемы исчезающего градиента в слой модели "трансформер" добавляются два остаточных соединения. На рис. 24.9 приведена структура модели "трансформер" с одним слоем. На практике модели "трансформер" обычно имеют шесть и более слоев. Как и в случае других моделей, обсуждавшихся выше, выход слоя i используется в качестве входных данных для слоя $i+1$.

В архитектуре "трансформер" порядок слов в предложении не фиксируется явно, поскольку контекст моделируется только через механизм самовнимания, который безразличен к порядку слов. Чтобы захватить порядок слов, в модели "трансформер" используется метод, получивший название ▶**позиционное**

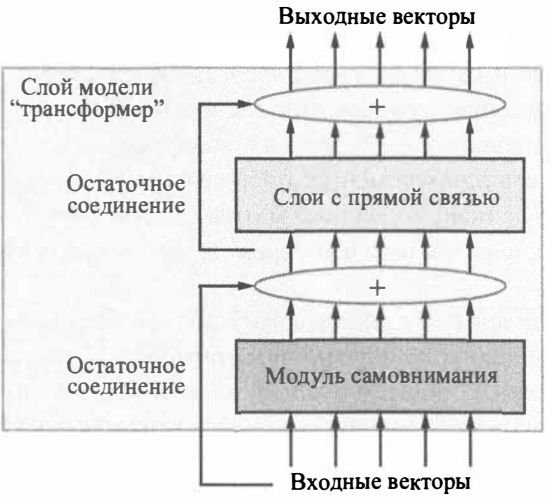


Рис. 24.9. Модель "трансформер" с одним слоем, состоящим из сети с прямой связью и остаточных связей

встраивание. Если входная последовательность имеет максимальную длину n, то изучается n новых векторов встраивания — по одному на каждую позицию слова и вход в первый слой модели "трансформер" представляет собой сумму встраивания слова в позиции t плюс позиционное встраивание, соответствующее позиции t.

На рис. 24.10 представлена схема модели архитектуры "трансформер" для задачи проставления тегов частей речи (POS), на вход которой подано то же самое предложение, что и на рис. 24.3. В нижней части схемы значения встраивания слов и позиционное встраивание суммируются с целью формирования входного вектора для модели "трансформер" с тремя слоями. Модель "трансформер" генерирует по отдельному вектору для каждого слова, как и в случае модели для расстановки тегов POS на основе сети RNN. Каждый вектор слова подается в последний выходной слой с функцией активации softmax для получения распределения вероятности по тегам.

В этом разделе фактически была рассказана только половина истории под названием "архитектура трансформер": та модель, которая была описана выше, в действительности называется ▶**энкодером трансформера**. Она может быть полезна для решения задач классификации текста. Полная архитектура "трансформер" изначально была разработана как улучшенный вариант модели от последовательности к последовательности для задач машинного перевода. Поэтому в дополнение к энкодеру эта архитектура включает в себя ▶**декодер трансформера**. Кодер и декодер почти идентичны, — за исключением того, что в декодере

используется такая версия модуля самовнимания, в которой каждое слово может соотноситься только с предыдущими словами, так как текст генерируется слева направо. Декодер также имеет второй модуль внимания в каждом слое архитектуры "трансформер", который связан с выходным вектором энкодера.

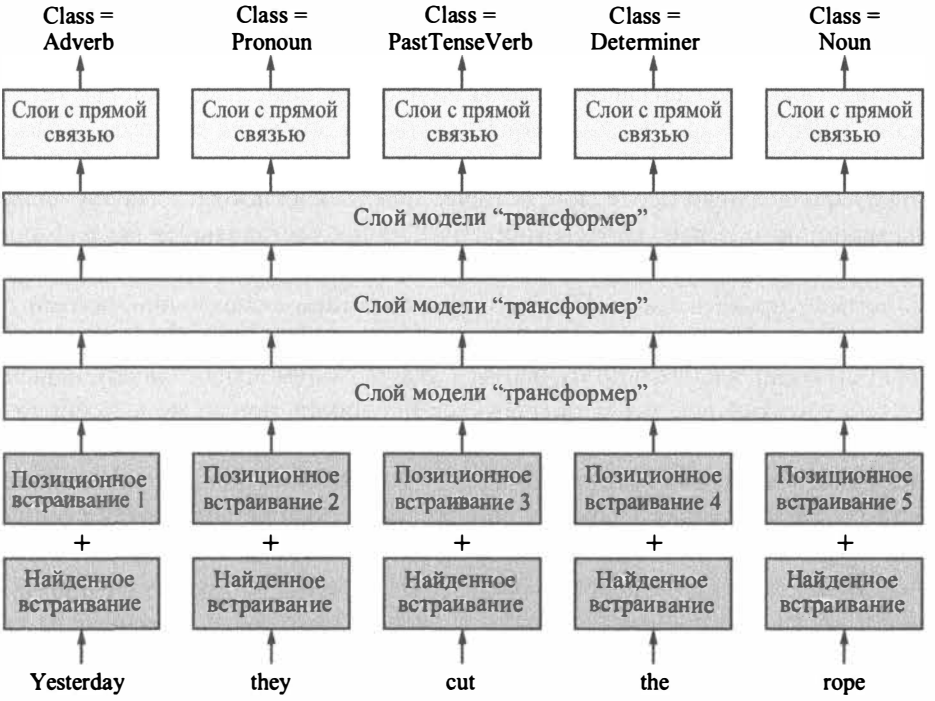


Рис. 24.10. Использование архитектуры "трансформер" применительно к задаче расстановки тегов частей речи

24.5. Предобучение и трансферное обучение

Получение достаточного количества данных для построения надежной модели может оказаться серьезной проблемой. В области компьютерного зрения (см. главу 25) эта проблема была решена посредством создания больших коллекций изображений (таких, как ImageNet) и ручной их маркировки.

В области обработки естественного языка более распространенным вариантом является работа с текстом без маркировки. Эта особенность отчасти связана с трудностью подобной маркировки: неквалифицированному работнику несложно присвоить изображению метку "кошка" или "закат", но, чтобы аннотировать предложение тегами частей речи или построить дерево его синтаксического анализа,

необходим работник, прошедший серьезную подготовку. Другое отличие связано с обилием текста: в Интернете каждый день добавляется более 100 млрд слов текста, включая оцифрованные книги, редактируемые ресурсы, подобные Википедии, и не редактируемые посты в социальных сетях.

Такие проекты, как Common Crawl, предоставляют легкий доступ к подобным данным. Любой последовательный текст может использоваться для построения *n*-грамм или моделей встраивания слов, а некоторые тексты предоставляются со структурой, которая может быть полезна для решения различных задач, например есть много сайтов, содержащих ответы на часто задаваемые вопросы, т.е. пары "вопрос–ответ", которые можно использовать для обучения системы предоставления ответов на вопросы. Аналогичным образом многие веб-сайты публикуют тексты с параллельным переводом, которые можно использовать для обучения системы машинного перевода. Некоторые тексты предоставляются даже с определенными метками, — например, на сайтах с различными обзорами пользователи могут комментировать текст обзора с использованием 5-звездочной рейтинговой системы.

Было бы очень желательно избавиться от необходимости создавать новый набор данных всякий раз, когда потребуется построить новую модель обработки естественного языка. В этом разделе вводится идея ▶**предобучения** — формы **трансферного обучения** (см. раздел 21.7.2), в которой используется большое количество общедоступных языковых данных общего характерах для обучения начальной версии модели обработки естественного языка. Достигнув этой отправной точки, для уточнения модели можно будет использовать уже меньший объем специализированных данных (возможно, включая некоторые помеченные данные), благодаря чему уточненная модель сможет быстро выучить дополнительный словарный запас, идиомы, синтаксические структуры и другие лингвистические явления, специфичные для данной конкретной области.

24.5.1. Предобученные встраивания слов

Выше, в разделе 24.1, кратко обсуждалась концепция встраивания слов. Было показано, что в этом случае схожие слова — скажем, *банан* и *яблоко* — в конечном итоге представляются сходными векторами, а задачу поиска аналогии оказывается возможным решить посредством простого векторного вычитания. Все это указывает на то, что метод встраивания слов позволяет выявить и зафиксировать важную информацию о словах.

В этом разделе будет детально рассмотрен вопрос о том, как встраивания слов создаются с использованием полностью неконтролируемого процесса обработки большого корпуса текста. Этот подход совершенно отличается от процедуры, описанной в разделе 24.1, где встраивания создавались в процессе обучения с учителем на основе данных с метками частей речи, расстановка которых требовала дорогостоящего ручного аннотирования текстов.

Здесь будет рассмотрена лишь одна конкретная модель построения встраивания слов — модель GloVe (Global Vectors). Она начинает работу со сбора счетчиков количества появления каждого слова в окне другого слова, что похоже на модель скип-граммы. Сначала выбирается размер окна (допустим, 5 слов), и пусть X_{ij} будет количеством раз, когда слова i и j одновременно присутствуют внутри окна, а X_i пусть будет количеством раз, когда слово i встречается в окне с любым другим словом. Обозначим через $P_{ij} = X_{ij} / X_i$ вероятность появления слова j в контексте слова i. Наконец, как и прежде, пусть $\mathbf{E}_i$ — это вектор встраивания для слова i.

Одно из принятых в модели GloVe допущений состоит в том, что лучше всего связь между двумя словами можно выявить, сравнив оба эти слова с другими словами. В качестве примера возьмем слова "ice" (*лед*) и "steam" (*пар*). Рассмотрим отношение вероятностей их совместного появления в окне с другим словом w, т.е.

$$P_{w,ice} / P_{w,steam} .$$

Если w — слово "solid" (*твердый*), это отношение будет большим (это означает, что понятие *твердый* в большей степени относится к слову *лед*), а когда w — слово "gas" (*газ*), оно будет невелико (и это означает, что слово *газ* ближе к слову *пар*). Если же w является словом, не имеющим собственного значения, подобно артиклю "the", или таким словом, как "water" (*вода*), которое одинаково релевантно к обоим, или одинаково нерелевантно к ним как слово "fashion" (*мода*), то это отношение будет близко к единице.

Отталкиваясь от этого интуитивного допущения, модели GloVe было предоставлено некоторое математическое обоснование (Пеннингтон и др. [1772], 2014), в котором отношения вероятностей преобразуются в векторные разности и точечные произведения и в конечном счете сводятся к ограничению

$$\mathbf{E}_i \cdot \mathbf{E}'_k = \log(P_{ij}).$$

Другими словами, скалярное произведение двух векторов встраиваний слов равно логарифму вероятности их совместного появления в окне. Это утверждение интуитивно понятно: два почти перпендикулярных друг другу вектора будут иметь точечное произведение, близкое к нулю, а два почти идентичных нормализованных вектора будут иметь точечное произведение, близкое к единице. Здесь есть одно техническое усложнение: модель GloVe создает два вектора встраивания слов для каждого слова: $\mathbf{E}_i$ и E'_i — вычисление двух векторов с последующим их сложением в конечном счете способствует ограничению переобучения.

Обучение такой модели, как GloVe, как правило, обходится намного дешевле, чем обучение стандартной нейронной сети: новая модель может быть обучена на миллиардах слов текста за несколько часов при использовании стандартного настольного компьютера.

Можно обучить предобученные векторы встраивания слов для конкретной проблемной области, а затем извлекать из модели знания по ней. Например, Цитоян

и соавт. ([2228], 2019) использовали 3,3 млн научных рефератов по теме материаловедения для обучения модели встраивания слов. Они установили, что, подобно тому, как общая модель встраивания слов может ответить на вопрос "Афины для Греции, как Осло для [что?]" словом "Норвегии", их обученная материаловедению модель на вопрос "NiFe — ферромагнетик, а IrMn — это [что?]" может дать ответ "антиферромагнетик".

Их модель не полагается исключительно на совместное появление слов в окне, — создается впечатление, что она способна выделять и фиксировать более сложные научные знания. На вопрос, какие химические соединения можно классифицировать как "термоэлектрик" или "топологический изолятор", эта модель оказалась способной ответить правильно. Однако соединение $CsAgGa_2Se_4$ в обучающем корпусе никогда не появляется в окне вместе со словом "термоэлектрик", но появляется вместе со словами "халькогенид", "запрещенная зона" и "оптоэлектрик", которые являются ключами, позволяющими классифицировать это соединение как "термоэлектрик". Более того, при обучении только на рефератах, относящихся к 2008 году, и при выдаче запроса отобрать соединения, которые являются "термоэлектриками", но еще не упоминаются как таковые в рефератах, для трех из пяти лучших пиков, предложенных моделью, было обнаружено, что они действительно описываются как термоэлектрики в работах, опубликованных между 2009 и 2019 годами.

24.5.2. Предобученные контекстные представления

Встраивания слов являются лучшими представлениями слов, чем атомарные токены, но здесь имеет место важная проблема: многозначность слов. Например, в английском языке слово "rose" (*роза*) может являться названием цветка или быть формой прошедшего времени глагола "rise" (*подниматься*). Следовательно, для "rose" можно ожидать обнаружить по крайней мере два совершенно разных кластера близких по значению слов: один будет соответствовать названиям цветов, таким как "dahlia" (*георгина*), а другой — глаголам, определяющим состояние подъема, таким как "upsurge" (*повышаться*). Ни один вектор встраивания не сможет захватить оба эти контекста одновременно. Слово "rose" является ярким примером слов с (по крайней мере) двумя совершенно различными значениями, но другие слова могут обладать более тонкими оттенками значения, зависящими от контекста, например слово "need" (*нужно*, *требуется*) в фразах "you need to see this movie" (*тебе нужно посмотреть этот фильм*) и "humans need oxygen to survive" (*людям требуется кислород, чтобы жить*). А некоторые идиоматические фразы, такие как "break the bank" (*сорвать банк*), вообще лучше анализировать как единое целое, а не по входящим в него словам.

Поэтому, вместо того чтобы просто обучать таблицу "слово–встраивание", желательно обучить модель создавать ▶**контекстные представления** для каждого слова в предложении. Контекстное представление отображает в вектор

встраивания как само слово, так и слова из окружающего его контекста. Иначе говоря, если передать в эту модель слово "rose" в контексте "the gardener planted a rose bush" (*садовник посадил розовый куст*), она должна создать контекстное встраивание, которое будет похоже (но не обязательно идентично) представлению, которое будет получено для него в контексте "the cabbage rose had an unusual fragrance" (*капустная роза имеет необычный аромат*) и очень отличается от представления для слова "rose" в контексте "the river rose five feet" (*река поднялась на пять футов*).

На рис. 24.11 показана рекуррентная сеть, создающая контекстные встраивания слов, — на рисунке они представлены прямоугольниками без обозначений. Предполагается, что коллекция неконтекстных встраиваний слов уже собрана. На вход сети подается предложение "The red car is big." (*Этот красный автомобиль большой*) по одному слову за раз, и модель должна дать прогноз для следующего слова. Поэтому, например, на рисунке в точке, где было достигнуто слово "car", в узел RNN на этом временном этапе поступают два входа: неконтекстное встраивание для слова "car" (*автомобиль*) и контекст, кодирующий информацию из предыдущих слов "The red" (*этот красный*). На этом основании узел RNN выведет контекстное представление для слова "car". Затем сеть как целое выведет свой прогноз для следующего слова: "is" (*есть, является*) и весовые коэффициенты в сети будут обновлены так, чтобы минимизировать ошибку между предсказанием и фактическим следующим словом.

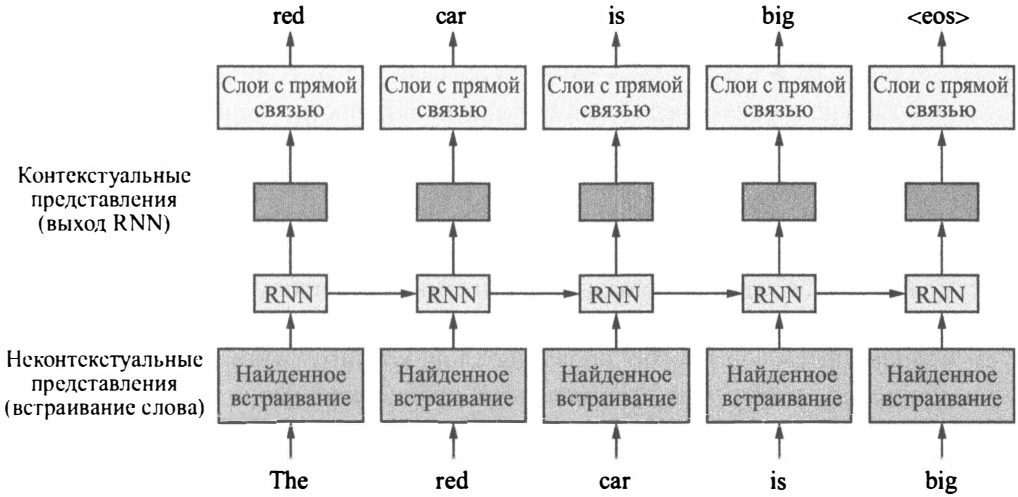


Рис. 24.11. Обучение контекстных представлений с использованием модели языка слева направо

Эта модель аналогична модели для расстановки тегов частей речи, приведенной на рис. 24.5, но с двумя важными отличиями. Во-первых, эта модель является однонаправленной (слева-направо), тогда как модель на рис. 24.5 — двунаправленной. Во-вторых, вместо предсказания тегов POS для *текущего* слова эта модель прогнозирует *следующее* слово предложения, используя предыдущий контекст. Как только модель будет построена, ее можно будет использовать для извлечения представления слов и передачи их какому-то другому заданию, — нет необходимости продолжать прогнозировать следующее слово. Обратите внимание, что для вычисления контекстных представлений всегда требуется два входа, текущее слово и контекст.

24.5.3. Маскированные модели языка

Слабость стандартных языковых моделей, таких как *n*-граммные модели, заключается в том, что определение контекста каждого слова построено только на предыдущих словах предложения. Предсказания делаются в направлении слева направо. Но иногда важный контекст присутствует в предложении позже, например слово "feet" в фразе "...rose five feet" (...поднялась на пять футов) помогает прояснить значение слов, стоящих до него.

Одним из простых обходных путей является обучение отдельной модели языка справа-налево, определяющей контекст для каждого слова на основе последующих слов в предложении, с последующей конкатенацией представлений слева направо и справа налево. Однако такая модель не способна комбинировать свидетельства из обоих направлений.

Вместо этого можно использовать ▶ **маскированную модель языка** (*masked language model* — ▶ **MLM**). Модели MLM обучают маскированием (сокрытием) отдельных слов на входе и предложением модели дать прогноз для замаскированных слов на выходе. В этой задаче можно использовать глубокие двунаправленные сети RNN или архитектуру "трансформер" поверх маскированного предложения. Например, при заданном входном предложении "The river rose five feet" (*река поднялась на пять футов*) можно замаскировать его среднее слово и получить предложение "The river __ five feet", а затем предложить модели заполнить пропуск (рис. 24.12).

Финальные скрытые векторы, соответствующие маскированным токенам, затем используются для прогнозирования слов, которые были замаскированы, в этом примере — слова "rose" (*поднялась*). В процессе обучения одно предложение можно использовать несколько раз с разными замаскированными словами. Прелесть этого подхода прежде всего в том, что он не требует помеченных данных, — предложение предоставляет собственную метку для замаскированного слова. Если эту модель обучить на большом корпусе текста, она будет генерировать предобученные представления, позволяющие хорошо справляться с широким спектром задач обработки естественного языка (машинный перевод, предоставление ответов на вопросы, обобщение, определение авторства и т.д.).

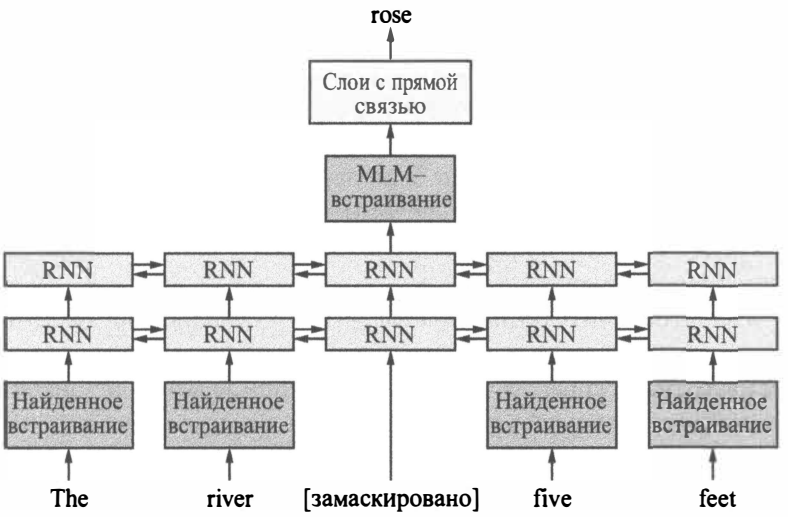


Рис. 24.12. Маскированная модель языка: предобучение двунаправленной модели — например, многослойной сети RNN — посредством маскирования входных слов и предоставления прогнозов только для этих замаскированных слов

24.6. Современное состояние разработок

Широкое применение методов глубокого и трансферного обучения способствовало заметному продвижению в современном состоянии разработок в области обработки естественного языка, причем настолько, что один комментатор в 2018 году заявил, что "пришло время ImageNet для области обработки естественных языков" (Рудер [1930], 2018). Имеется в виду, что так же, как в области компьютерного зрения, где переломный момент наступил в 2012 году, когда системы глубокого обучения дали удивительно хорошие результаты в конкурсе ImageNet, в 2018 году такой же переломный момент наступил и в области обработки естественных языков. Основной импульс для наступления этого переломного момента дало обнаружение того факта, что трансферное обучение хорошо работает в задачах обработки естественного языка: можно загрузить общую модель языка, а затем настроить ее для решения любой конкретной задачи.

Все это началось с простых встраиваний слов из таких систем, как WORD2VEC в 2013 году и GLOVE в 2014 году. Исследователи могли скачать такую модель или обучить собственную относительно быстро и без доступа к суперкомпьютерам, тогда как создание собственных предобученых контекстных представлений обходилось на порядки дороже.

Реализация этих моделей стала возможной только после того, как новые аппаратные достижения (графические процессоры и TPU) получили достаточно широкое распространение. Именно благодаря этому у исследователей появилась возможность загружать модели вместо того, чтобы тратить ресурсы на обучение собственных. Разработка архитектуры "трансформер" обеспечила возможность эффективного обучения гораздо более крупных и глубоких нейронных сетей, чем это было возможно ранее (на этот раз благодаря достижениям в области программного обеспечения, а не аппаратных средств). С 2018 года новые проекты обработки естественного языка, как правило, начинались с уже имеющейся, предобученной модели "трансформер".

Хотя эти модели архитектуры "трансформер" обучались предсказывать следующее слово в тексте, они показали удивительно хорошие результаты и при решении других языковых задач. Модель RoBERTa при некоторой тонкой настройке демонстрирует современный уровень результатов в ответах на вопросы и тестах на понимание прочитанного (Лю и др. [1431], 2019). Модель GPT-2, с архитектурой, подобной архитектуре "трансформер" — модель языка с 1,5 млрд параметров, обученных на 40 Гбайт текста из Интернета, — показала хорошие результаты в таких разнообразных задачах, как перевод с французского языка на английский и обратно, поиск соответствий при дальних зависимостях и предоставление ответов на вопросы из области общих знаний, и все это без тонкой подстройки под конкретную задачу. Как показано на рис. 24.13, система GPT-2 способна генерировать достаточно убедительный текст, исходя всего из нескольких слов, заданных в качестве подсказки.

Дословный перевод текстов, приведенных на рис. 24.13, следующий.

Это общепризнанная истина, что земля не является центром вселенной. Есть те, кто утверждает это так. Я не принимаю их, но другие, я считаю, придерживаются того же мнения. Правда, однако, что если есть другое, чем центр, и если есть какие-либо другие живые существа во вселенной, и если они не люди, то у нас еще нет наших ответов. Мы должны идти дальше. Эта страница дает упрощенный, упрощенный ответ на проблему. У нас нет всех ответов. Истина в том, однако, что истина не здесь.

Когда Грегор Самса проснулся однажды утром, он не заметил ничего странного. "Когда моя жена смотрит на меня, я чувствую, что она смотрит на произведение искусства", — сказал он. "Я думаю, она восхищается чем то, что я создал". Идея состоит в том, что, глядя на свою жизнь, ты учишь что-то важное и становишься лучше. Это теория, которая возникла из работы психолога Даниэля Големана, в которой он спросил "Откуда ты знаешь, что ты не неудачник?"

Алиса начинала очень уставать от сидения со своей сестрой на берегу. Она села, зевнула и сказала с громким коротким криком: "Я надеюсь, ты не возражаешь, если я продолжу делать то, что я бы хотела сделать, и если кто-то спросит

меня, кто из нас будет делать больше, не говори им, что я не буду делать много, моя дорогая сестра".

Все счастливые семьи одинаковы; каждая счастливая семья похожа на райский сад. Единственная разница между счастливыми и несчастными семьями в том, что несчастные семьи не имеют никаких цветов или деревьев.

Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Расскажи мне историю. Пожалуйста, заполните следующие подробности. Спасибо... Спасибо за ваш интерес к этому интервью. Пожалуйста, подождите...

It is a truth universally acknowledged that the earth is not the center of the universe. There are those who assert there is. I do not accept them, but others I consider to be of the same opinion. The truth is, however, that if there are other than the center, and if there are any other living things in the universe and if they are not human, then we do not yet have our answers. We have to go on. This page gives a simplified, simplified answer to the problem. We don't have all the answers. The truth is, however, that the truth is out there.

When Gregor Samsa woke up one morning, he did not notice anything strange. "When my wife is looking at me, I feel like she is looking at a piece of art," he said. "I think she is admiring something I have created." The idea is that by looking at your own life, you learn something important and become a better person. It is a theory that emerged from psychologist Daniel Goleman's work, in which he asked "How do you know you're not a loser?"

Alice was beginning to get very tired of sitting with her sister on the bank. She sat up, yawned, and said, with a loud little scream, "I hope you don't mind if I keep on doing what I should like to do, and if someone asks me which of us will do more, don't tell them that I won't do much, my dear sister."

All happy families are alike; each happy family is like a garden of paradise. The only difference between happy families and unhappy families, is that the unhappy family doesn't have any flowers or trees.

Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Please fill out the following details. Thank you... Thank you for your interest in this interview. Please wait...

Рис. 24.13. Примеры завершенных текстов, сгенерированных моделью языка GPT-2, на основании подсказок, выделенных жирным шрифтом. Большая часть этих текстов — достаточно свободный английский, по крайней мере локально. Последний пример демонстрирует, что иногда работа модели все же нарушается

Как пример современной системы обработки естественного языка можно привести систему ARISTO (Кларк и др. [447], 2019), достигшую уровня 91,6% правильных ответов на экзамене за 8-й класс по естествознанию с несколькими вариантами ответов (рис. 24.14). Система ARISTO состоит из ансамбля решателей: в некоторых используется поиск информации (подобно поисковой системе в Интернете), некоторые выполняют анализ семантического следования и проводят качественные рассуждения, а некоторые используют модель языка ROBERTA с архитектурой "трансформер". Как оказалось, сама по себе система ROBERTA на этом тестовом наборе показала результат 88,2%. Система ARISTO также достигла уровня 83% правильных ответов на более сложном экзамене за 12-й класс. (Оценка 65% считается показателем "соответствует стандартам", а оценка 85% — показателем "соответствует стандартам с отличием".)

---

1. **What will best separate a mixture of iron filings and black pepper?**
   (*С помощью чего проще всего разделить смесь из железных опилок и черного перца?*)
   a) magnet b) filter paper c) triple beam balance d) voltmeter
   (1. *Магнит* 2. *Фильтровальная бумага* 3. *Точные весы* 4. *Вольтметр*)

2. **Which form of energy is produced when a rubber band vibrates?**
   (*Какая форма энергии производится, когда резиновая лента вибрирует?*)
   (a) chemical (b) light (c) electrical (d) sound
   (1. *Химическая* 2. *Световая* 3. *Электрическая* 4. *Звуковая*)

3. **Because copper is a metal, it is**
   (*Поскольку медь — это металл, она...*)
   (a) liquid at room temperature (b) nonreactive with other substances
   (1. *Жидкая при комнатной температуре* 2. *Не реагирует с другими веществами*)
   (c) a poor conductor of electricity (d) a good conductor of heat
   (3. *Плохо проводит электричество* 4. *Хорошо проводит тепло*)

4. **Which process in an apple tree primarily results from cell division?**
   (*Какой процесс в яблоне является прямым результатом деления клеток?*)
   (a) growth (b) photosynthesis (c) gas exchange (d) waste removal
   (1. *Рост* 2. *Фотосинтез* 3. *Газообмен* 4. *Выделение отходов*)

Рис. 24.14. Экзаменационные вопросы за 8-й класс по естествознанию, на которые система ARISTO способна дать правильный ответ, используя ансамбль методов, наиболее влиятельным из которых является модель языка ROBERTA. Для ответа на эти вопросы требуются знание естественного языка, понимание структуры тестов с множественным выбором, наличие здравого смысла и некоторые научные сведения

Система ARISTO имеет некоторые ограничения. Она работает только с вопросами с несколькими вариантами ответов, а не с произвольными вопросами, и не может ни читать, ни генерировать диаграммы.[1]

Система T5 (*Text-To-Text Transfer Transformer*) предназначена для выдачи текстовых ответов на различные виды текстового ввода. Она включает в себя стандартную модель архитектуры "трансформер" "кодер-декодер", предобученную на 35 млрд слов из корпуса Colossal Clean Crawled Corpus (C4) объемом 750 Гбайт. Этот обучающий набор без маркировки был разработан для предоставления модели таких обобщаемых лингвистических знаний, которые будут полезны для решения многих конкретных задач. Затем для каждой отдельной задачи система T5 обучалась на входах, состоящих из имени задачи, отделенного двоеточием от некоторого контента. Например, при получении входа "translate English to German: That is good" (*перевести с английского на немецкий: Это хорошо*) система должна была предоставить выход "Das ist gut". Для некоторых задач вход мог быть размечен, — например, в задании по применению схемы Винограда во входных данных выделяется местоимение с неоднозначной связью. Так, для входа "referent: The city councilmen refused the demonstrators a permit because *they* feared violence" (*ссылка: "Члены городского совета отказали демонстрантам в разрешении, потому что* они *боялись насилия"*) правильным ответом будет "The city councilmen" (*члены городского совета*), а не "the demonstrators" (*демонстранты*).

Многое еще предстоит сделать для улучшения систем обработки естественного языка. Одна проблема заключается в том, что модели архитектуры "трансформер" опираются только на узкий контекст, ограниченный несколькими сотнями слов. В некоторых экспериментальных подходах предпринимаются попытки расширить этот контекст: система Reformer (Китаев и др. [1232], 2020) способна обрабатывать контекст до миллиона слов.

Недавно достигнутые результаты показывают, что использование большего количества обучающих данных приводит к получению лучших моделей, — например, система ROBERTA достигла современного уровня результатов после обучения на 2,2 трлн слов. Если использование большего количества текстовых данных позволяет достичь лучших результатов, то что произойдет, если подключить к обучению и другие типы данных: структурированные базы данных, числовые данные, изображения и видео? Для обучения на большом корпусе видеоданных необходим прорыв в скорости обработки таких данных аппаратным обеспечением, а также может потребоваться несколько новых прорывов в области теории ИИ.

Внимательный читатель может задаться вопросом, зачем в предыдущей главе рассматривались такие аспекты языка, как грамматика, синтаксический анализ и семантическая интерпретация, если в этой главе предлагается отбросить их в

[1] Было отмечено, что на некоторых экзаменах с предоставленными вариантами ответов можно получить хороший балл, даже не глядя на вопросы, поскольку у неправильных ответов есть явные признаки (Гуруранган и др. [938], 2018). Похоже, что это замечание верно и для вопросов с визуальными ответами (Чао и др. [391], 2018).

пользу моделей, управляемых исключительно данными? В настоящее время ответ на этот вопрос прост: модели, управляемые данными, легче разрабатывать и поддерживать, и на стандартных тестах их показатели лучше, чем показатели систем ручной сборки, которые можно построить с использованием разумного количества человеческих усилий на основании подходов, обсуждавшихся в главе 23. Может оказаться так, что модели архитектуры "трансформер" и родственные им изучают скрытые представления, отражающие те же самые основные идеи, что и грамматики и семантическая информация, или же внутри этих огромных моделей происходит нечто совершенно иное, — мы просто этого не знаем. Нам точно известно, что систему, обученную на текстовых данных, легче поддерживать и адаптировать к новым предметным областям и новым естественным языкам, чем такую систему, которая опирается на созданные вручную признаки.

Может случиться и так, что будущие прорывы в явном грамматическом и семантическом моделировании заставят маятник качнуться назад. Возможно, более вероятно появление гибридных подходов, объединяющих лучшие концепции из обоих направлений. Например, Китаев и Кляйн ([1233], 2018) использовали механизм внимания для улучшения традиционного синтаксического анализатора, получив наилучшие результаты среди когда-либо зарегистрированных для тестового набора Penn Treebank. Аналогичным образом Ринггаард и соавт. ([1884], 2017) показали, как можно улучшить анализатор зависимостей за счет применения встраиваний слов и рекуррентной нейронной сети. Их система SLING помещает результаты анализа непосредственно в представление семантической структуры, смягчая этим проблему накопления ошибок, неизбежную в случае традиционных конвейерных систем.

Безусловно, есть достаточно места для улучшений: системы обработки естественных языков не только по-прежнему отстают от человека при решении многих задач. Гораздо важнее то, что они, к сожалению, показывают такие результаты после обработки в тысячи раз большего объема текста, чем любой человек сможет прочитать за всю свою жизнь. Это указывает на то, что имеется еще много возможностей для новых идей от лингвистов, психологов и исследователей в области обработки естественных языков.

Резюме

Вот перечень основных тем, рассмотренных в этой главе.

- Непрерывные представления слов с использованием метода встраивания более устойчивы, чем дискретные атомарные представления, и могут быть предобучены с использованием непомеченых текстовых данных.
- Рекуррентные нейронные сети позволяют эффективно моделировать локальный и удаленный контексты, сохраняя соответствующую информацию в своих векторах скрытого состояния.

- Модели класса от последовательности к последовательности могут использоваться для задач машинного перевода и генерации текста.
- Модели архитектуры "трансформер" используют механизм самовнимания, благодаря чему они способны моделировать дальний контекст так же хорошо, как и локальный. Эти модели позволяют эффективно использовать аппаратные средства матричного умножения.
- Трансферное обучение, включающее в себя предобученные контекстные встраивания слов, позволяет разрабатывать модели на основе очень больших непомеченых корпусов текста, которые затем могут успешно применяться для решения ряда задач. Модели, предобученные для прогнозирования пропущенных слов, также могут применяться при решении других задач — включая предоставление ответов на вопросы и распознавание семантического следования — после соответствующей точной настройки для целевой предметной области.

Библиографические и исторические заметки

Распределение слов и фраз на естественном языке следует **закону Ципфа** (Ципф [2449], 1935; [2450], 1949): частота n-го наиболее употребляемого слова приблизительно обратно пропорциональна n. Это означает, что во всех системах имеет место проблема разреженности данных: даже при миллиардах слов обучающих данных неизбежно будут постоянно появляться новые слова и фразы, которые ранее системе не встречались.

Обобщению вновь появившихся слов и фраз способствуют представления, отражающие следующую основную идею: слова со схожим значением появляются в сходных контекстах. Дирвестер и соавт. ([591], 1990) предложили проецировать слова на низкоразмерные векторы посредством декомпозиции матрицы совместного появления, образованной словами и документами, в которых эти слова встречаются. Еще одна возможность заключается в обработке окружающих слов как контекста — скажем, при выборе окна размером в 5 слов. Браун и соавт. ([323], 1992) группировали слова в иерархические кластеры в соответствии с их биграмным контекстом. Подобный подход доказал свою эффективность для таких задач, как распознавание именованных объектов (Туриан и др. [2232], 2010). Систему WORD2VEC (Миколов и др. [1568], 2013) можно считать первой заметной демонстрацией преимуществ метода встраивания слов, полученных за счет обучения нейронных сетей. Векторы встраивания слов GloVe (Пеннингтон и др. [1772], 2014) были получены при непосредственной обработке матрицы совместного появления слов, построенной на основе миллиардов слов текста. Леви и Гольдберг в работе [1398] (2014) подробно объясняют, почему и как эти встраивания слов способны улавливать языковые закономерности.

Бенжио и др. ([173], 2003) впервые применили нейронные сети для построения моделей языка, предложив объединить "*а*) распределенное представление для каждого слова вместе с *б*) функцией вероятности для последовательностей слов, выраженных в терминах этих представлений". Миколов и соавт. в работе [1569] (2010) продемонстрировали использование рекуррентных нейронных сетей для моделирования локального контекста в моделях языка. Юзефович и соавт. ([1155], 2016) показали, что рекуррентная нейронная сеть (RNN), обученная на миллиарде слов, может превзойти по производительности тщательно разработанные вручную *n*-граммные модели. Важность контекстных представлений для слов была подчеркнута Петерсом и соавт. ([1780], 2018), предложившими для этих представлений особое название: ELMo (Embeddings from Language Models — *встраивания из моделей языка*).

Обратите внимание, что некоторые авторы сравнивают качество моделей языка, измеряя их ▶**перплексию** (*perplexity*). Перплексия распределения вероятностей равна 2^H, где H — энтропия этого распределения (см. раздел 19.3.3). Языковая модель с меньшей перплексией будет лучшей моделью при равных прочих характеристиках. Но на практике все остальные характеристики редко бывают равными. Поэтому более информативно измерять производительность на реальных задачах, а не полагаться на перплексию.

Говард и Рудер ([1072], 2018) описывают среду разработки ULMFiT (*Universal Language Model Fine-tuning*), упрощающую тонкую настройку предобученной модели языка без необходимости использования обширного корпуса документов из целевой проблемной области. Позднее Рудер и соавт. выпустили учебное пособие ([1931], 2019) по трансферному обучению для задач обработки естественного языка.

Миколов и соавт. ([1569], 2010) выдвинули идею использования рекуррентных нейронных сетей для задач обработки естественного языка, а Суцкевер и соавт. ([2154], 2015) предложили идею глубокого обучения сетевой модели класса от последовательности к последовательности. Чжу и соавт. ([2442], 2017) и Лю и соавт. ([1429], 2018) показали, что подход с использованием метода обучения без учителя действительно работает, что существенно упрощает сбор необходимых данных. Вскоре было установлено, что такие модели могут на удивление хорошо справляться с различными задачами, например с генерацией подписей к изображениям (Карпати и Фей-Фей [1191], 2015; Виньялс и др. [2275], 2017).

Девлин и соавт. ([614], 2018) показали, что модели архитектуры "трансформер", предобученные на задачах с применением маскированной модели языка, можно непосредственно использовать для решения многих задач. Эти модели получили название "BERT" (*Bidirectional Encoder Representations from Transformers*). Предобученные модели BERT можно подвергнуть точной настройке для использования в определенных предметных областях и решения конкретных задач, включая предоставление ответов на вопросы, именование распознаваемых объектов, классификацию текстов, анализ тональности текста и логический вывод на естественном языке.

Система XLNet (Янг и др. [2399], 2019) является улучшенной моделью BERT, в которой устранены расхождения между предобучением и тонкой настройкой. Среда разработки Ernie 2.0 (Сан и др. [2148], 2019) позволяет извлечь из обучающих данных больше за счет учета порядка предложений и наличия именованных объектов, а не просто анализа совместного появления слов, — она продемонстрировала лучшую производительность в сравнении с моделями BERT и XLNet. В ответ исследователи пересмотрели и усовершенствовали модели BERT: система RoBERTa (Лю и др. [1431], 2019) использует больше данных, различных гиперпараметров и процедур обучения, что позволило ей достичь соответствия показателям XLNet. В системе Reformer (Китаев и др. [1232], 2020) диапазон контекста существенно расширен и может охватывать вплоть до миллиона слов. В то же время в системе ALBERT (*A Lite BERT*) было выбрано иное направление развития — количество параметров было сокращено со 108 до 12 млн при сохранении высокой точности, что позволило использовать ее на мобильных устройствах.

Система XLM (Лампле и Конне [1348], 2019) представляет собой модель архитектуры "трансформер" с обучающими данными на нескольких языках. Такой подход полезен для машинного перевода, но также обеспечивает получение более надежных представлений для задач на одном языке. Две другие важные системы, GPT-2 (Редфорд и др. [1842], 2019) и T5 (Раффел и др. [1843], 2019), были описаны выше в этой главе. В последней статье также был представлен корпус из 35 млрд слов, получивший название *Colossal Clean Crawled Corpus* (C4).

В алгоритмах предобучения также были предложены различные многообещающие улучшения (Янг и др. [2399], 2019; Лю и др. [1431], 2019). Предобученные контекстные модели описаны Петерсом и соавт. в работе [1780] (2018), а также в статье Дай и Ле [517] (2016).

Тестовый набор GLUE (*General Language Understanding Evaluation*), включающий подборку задач и инструментов для оценки систем обработки естественного языка, был представлен Венгом и соавт. ([2293], 2018). В набор тестовых задач входят задания на предоставление ответов на вопросы, определение тональности текста, анализ семантического следования, перевод и выполнение синтаксического анализа. Модели архитектуры "трансформер" настолько доминировали в таблице лидеров (уровень показателей человека оказался в этой таблице далеко внизу, на девятом месте), что новая версия тестового набора, SuperGLUE (Венг и др. [2292], 2019), была дополнена задачами, разработанными таким образом, чтобы быть более трудными для компьютеров, но все же относительно простыми для человека.

В конце 2019 года система T5 стала абсолютным лидером с результатом 89,3, всего на пол-очка отставая от базового уровня человека, равного 89,8. По трем из десяти задач система T5 в действительности превосходит показатели производительности человека: предоставление ответа "да/нет" на вопрос (например, такой: "Находится ли Франция в том же часовом поясе, что и Великобритания?") и две задачи на понимание прочитанного, заключающиеся в предоставлении ответов на вопросы после прочтения параграфа или новостной статьи.

Машинный перевод является основной областью применения моделей языка. В 1933 году Петр Троянский получил патент на "переводящую машину", но в то время еще не существовало компьютеров, пригодных для реализации его идеи. В 1947 году Уоррен Уивер, опираясь на работы в области криптографии и теории информации, писал Норберту Винеру: "Когда я смотрю на статью на русском языке, я говорю: «В действительности это написано на английском, но закодировано странными символами. Сейчас я приступлю к декодированию»". Сообщество предпринимало попытки подобного декодирования, но в то время еще не было достаточного количества данных и вычислительных ресурсов, чтобы реализовать этот подход на практике.

В 1970-х годах ситуация начала изменяться и система SYSTRAN (Тома [2219], 1977) стала первой коммерчески успешной системой машинного перевода. Работа системы SYSTRAN опиралась на лексические и грамматические правила, подготовленные лингвистами вручную, а также на обучающие данные. В 1980-х годах сообщество занималось чисто статистическими моделями, основанными на частоте слов и фраз (Браун и др. [318], 1988; Коэн [1253], 2009). Но когда размер обучающих наборов достиг миллиардов и даже триллионов токенов (Брантс и др. [292], 2007), это привело к появлению систем, которые давали понятные результаты, но все же не достигающие уровня беглого языка (Оч и Ней [1704], 2004; Золлман и др. [2452], 2008). Оч и Ней в работе [1705] (2002) показывают, как дискриминативное обучение позволило достичь прогресса в машинном переводе в начале 2000-х годов.

Суцкевер и соавт. в работе [2154] (2015) впервые показали, что можно обучить машинному переводу нейронную сквозную модель класса от последовательности к последовательности. Баданау и соавт. ([109], 2015) продемонстрировали преимущества модели, которая совместно обучается выравниванию предложений в исходном и целевом языках и переводу между этими языками. Вашвани и соавт. ([2266], 2018) показали, что нейронные системы машинного перевода могут быть дополнительно улучшены путем замены сетей LSTM архитектурой "трансформер", в которой для захвата контекста используется механизм внимания. Такие нейронные системы перевода быстро обогнали системы на базе статистических методов анализа фраз, а применение архитектуры "трансформер" очень скоро распространилось и на системы решения других задач обработки естественного языка.

Исследованиям в области предоставления **ответов на вопросы** способствовало создание SQuAD, первого крупномасштабного набора данных для обучения и тестирования систем предоставления ответов на вопросы (Раджпуркар и др. [1847], 2016). С тех пор для таких задач был разработан ряд моделей глубокого обучения (Сео и др. [2027], 2017; Кескар и др. [1219], 2019). В системе ARISTO (Кларк и др. [447], 2019) глубокое обучение используется в сочетании с ансамблем других методов. С 2018 года в большинстве моделей систем предоставления ответов на вопросы используются предобученные языковые представления, что привело к заметному улучшению их показателей по сравнению с более ранними системами.

Логический вывод на естественном языке — это задача оценки, следует ли гипотеза (*собаки должны есть*) из некоторой предпосылки (*все животные должны есть*). Этот класс задач был популяризирован появлением тестового набора PASCAL Challenge (Даган и др. [513], 2005). Сейчас для подобных систем доступны крупномасштабные наборы данных (Боумен и др. [274], 2015; Вильямс и др. [2348], 2018). Системы на основе предобученных моделей, таких как ELMO и BERT, на текущий момент демонстрируют наилучшую производительность при решении задач логического вывода на естественном языке.

Конференция *Conference on Computational Natural Language Learning* (CoNLL) фокусируется на применении обучения в области обработки естественного языка. Все конференции и журналы, упомянутые в главе 23, в настоящее время включают работы и статьи по глубокому обучению, занимающему сейчас ведущее положение в области обработки естественного языка.

ГЛАВА 25

Компьютерное зрение

В данной главе речь идет о том, как глазами видеокамеры соединить компьютер с первозданным, неприкрашенным миром.

У большинства животных есть глаза, что часто обходится им недешево: глаза занимают много места, используют энергию и являются весьма хрупкими. Однако все эти издержки оправдываются огромной пользой, которую они приносят. Агент, способный видеть, может предсказывать будущее — он вовремя заметит то, во что может врезаться; получит возможность принять правильное решение: напасть, убежать или завязать знакомство; сможет догадаться, что перед ним — болото или твердый грунт, а также правильно оценить расстояние до желанного плода. В этой главе обсуждается, как выделить полезную информацию из потока данных, поступающих через глаза или видеокамеры.

25.1. Введение

Зрение — это канал восприятия, принимающий внешние **стимулы** в виде световых сигналов и формирующий некоторое представление о мире. Большинство агентов, обладающих зрением, используют **пассивное зондирование** — им не нужно излучать свет, чтобы видеть. И напротив, **активное зондирование** предполагает отправку некоторого сигнала, такого как радиоволны в радарах или ультразвук в сонарах, чтобы получать и анализировать отраженные сигналы. Примерами агентов, использующих активное зондирование, являются летучие мыши (ультразвук), дельфины (звук), глубоководные рыбы (свет) и некоторые роботы (свет, звук, радиоволны). Чтобы понимать информацию, поступающую из канала восприятия, необходимо изучить как физические, так и статистические явления, которые имеют место в процессе восприятия, а также, что именно должно быть результатом восприятия. В этой главе обсуждение концентрируется на зрении, но в реальном мире роботы используют различные датчики: для восприятия звука и прикосновения, для определения расстояния, температуры, ускорения и глобальных координат.

► **Характеристика,** или **признак** (*feature*), — это числовое значение, полученное путем применения к изображению некоторых простых вычислений. Очень полезная информация может быть получена непосредственно из значений

характеристик. Агент в мире вампуса имеет пять датчиков, каждый из которых предоставляет один бит информации. Эти биты, которые являются характеристиками, могут интерпретироваться непосредственно программой. В качестве другого примера можно привести тот факт, что многие летающие животные вычисляют простую функцию, предоставляющую хорошую оценку времени до контакта с близлежащим объектом, — эта характеристика может передаваться непосредственно мышцам, управляющим полетом, или крыльями, что позволяет очень быстро менять направление. Подобный подход **извлечения характеристик** предполагает выполнение простых вычислений, применяемых непосредственно к показаниям датчиков.

В подходе, строящемся на **моделировании** зрения, используется два вида моделей. **Объектная модель** может представлять собой тот или иной вид точной геометрической модели, создаваемой системами автоматизированного проектирования. Либо это может быть некоторое расплывчатое заключение об общих свойствах объектов, например утверждение, что все лица при низком разрешении выглядят примерно одинаково. **Модель рендеринга** описывает физические, геометрические и статистические процессы, которые производят стимулы, поступающие в датчики из внешнего мира. Хотя модели рендеринга сейчас достаточно сложны и точны, стимулы обычно можно трактовать неоднозначно. Белый объект при слабом освещении вполне может выглядеть, как черный объект при интенсивном освещении. Маленький близлежащий объект может выглядеть так же, как большой, но удаленный объект. Без дополнительных свидетельств невозможно с уверенностью утверждать, чем является то, что мы видим: игрушечным Годзиллой, разрывающим игрушечное здание, или настоящим монстром, разрушающим реальное здание.

Есть два основных способа справиться с подобной неопределенностью. Во-первых, одни интерпретации могут быть более вероятными, чем другие. Например, можно с уверенностью утверждать, что в фильме снимался не настоящий Годзилла, разрушающий настоящие здания, поскольку настоящих Годзилл просто не существует. Во-вторых, некоторые неоднозначности на текущий момент не имеют большого значения. Например, отдаленный пейзаж может представлять собой реальные деревья или быть нарисованным на плоской, соответствующим образом окрашенной поверхности. Для большинства приложений это отличие не имеет значения, поскольку эти объекты далеко и столкновение или какое-то иное взаимодействие с ними в ближайшее время не предполагается.

Двумя основными задачами в области компьютерного зрения являются ▶**реконструкция**, когда агент строит модель мира на основе одного изображения или множества изображений, и ▶**распознавание**, когда агент на основе визуальной и другой информации выявляет различия между объектами, с которыми он сталкивается. Обе проблемы следует интерпретировать очень широко. Построение геометрической модели на основе изображений, безусловно, является задачей реконструкции (и ее решения очень ценны), но иногда может потребоваться

построить карту различных текстур на поверхности, и это тоже задача реконструкции. Определение названий объектов, присутствующих на изображении, без сомнения, является задачей распознавания. Но иногда требуется получить ответ на такие вопросы, как "Она спит?", "Он питается мясом?" или "С какой стороны здесь вход?" Получение ответов на подобные вопросы также относится к задачам распознавания.

За последние тридцать лет исследований были созданы мощные инструменты и предложены различные методы решения этих основных задач. Для понимания этих методов необходимо, прежде всего, понимать процессы образования изображений.

25.2. Формирование изображения

На изображении внешний вид объектов искажается. Взглянув на фотоснимок уходящих вдаль железнодорожных путей, можно увидеть, как рельсы постепенно сближаются и в конечном счете сходятся в одной точке. Если подержать руку перед глазами, то можно закрыть ладонью луну, даже если на самом деле луна больше вашей руки (то же самое можно проделать и с солнцем, но, проверяя это утверждение, можно повредить себе глаза). Если расположить книгу перед лицом, а затем наклонить ее назад, после чего вернуть в прежнее положение, то *на воспринимаемом глазами изображении* книга сначала будет уменьшаться, а затем вновь возрастать в размере по вертикали. Этот эффект, известный как **ракурс** (*foreshortening*), наглядно представлен на рис. 25.1. Моделирование этих эффектов очень важно для построения эффективных систем распознавания объектов и обязательно должно учитываться при реконструкции геометрии.

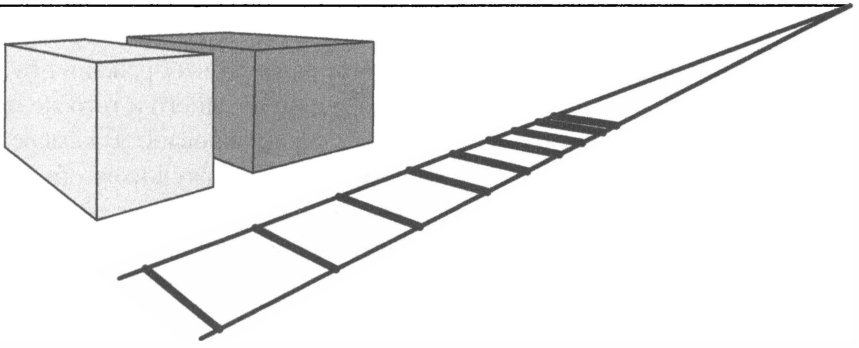


Рис. 25.1. Геометрия сцены на изображениях выглядит искаженной. Параллельные линии кажутся сходящимися, как уходящие вдаль железнодорожные пути. Здания, имеющие прямые углы в реальном мире, на изображении будут представлены с искаженными углами

25.2.1. Получение изображения без линз: камера-обскура

Датчики изображения собирают свет, рассеянный объектами в ▶**сцене**, и создают двухмерное (2D) ▶**изображение**. В наших глазах эти датчики состоят из двух типов клеток: около 100 млн палочек, чувствительных к свету в широком диапазоне длин волн, и 5 млн колбочек. Колбочки, обеспечивающие цветное зрение, бывают трех основных типов, каждый из которых чувствителен к определенному диапазону длин волн. В фотокамерах изображение формируется на плоскости изображения. В пленочных камерах в плоскости изображения находится пленка, покрытая галогенидами серебра, а в цифровых камерах плоскость изображения разделена на миллионы светочувствительных ▶**пикселей**, образующих прямоугольную сетку.

Всю плоскость изображения называют ▶**сенсором**, а каждый пиксель представляет собой отдельный крошечный датчик — обычно элемент с зарядовой связью (ПЗС) или комплементарный металл-оксидный полупроводник (CMOS). Каждый фотон, попадающий на такой датчик, производит электрический заряд, величина которого зависит от длины волны фотона. Выходной сигнал датчика является суммой таких эффектов, накопленных за некоторый интервал времени, и это означает, что каждый датчик в плоскости изображения выдает средневзвешенное значение интенсивности поступающего на него света. Значение усредняется по длине волны, по направлению, по которому могли прибыть фотоны, по времени и по площади датчика.

Чтобы увидеть сфокусированное изображение, следует убедиться, что все фотоны, приходящие на датчик, поступают примерно из одного и того же места на объекте во внешнем мире. Простейший способ сформировать сфокусированное изображение заключается в наблюдении стационарных объектов с помощью ▶**камеры-обскуры**, состоящей из небольшого отверстия O на передней стенке коробки и плоскости изображения, расположенной на противоположной стенке коробки (рис. 25.2). Если отверстие, которое иногда называют ▶**апертурой**, достаточно мало, на каждый крошечный датчик в плоскости изображения будут поступать только те фотоны, которые приходят примерно из одного и того же места на объекте, и поэтому изображение в целом будет сфокусировано. В камере-обскуре можно получить сфокусированные изображения и движущихся объектов, но только если они перемещаются лишь на относительно небольшое расстояние за временно*е* окно датчика. В противном случае изображение движущегося объекта будет размытым, расфокусированным, — эффект, известный как ▶**размытие в движении**. Один из способов манипулирования временны*м* окном — открывать и закрывать отверстие.

Камеры-обскуры упрощают понимание геометрической модели поведения фотокамер (которое более сложное, но схожее с поведением большинства других устройств обработки изображений). Используя трехмерную (3D) систему координат с началом координат в точке O, рассмотрим в сцене точку P с координатами

(X, Y, Z). Точка P в камере-обскуре проецируется на точку P' в ее плоскости изображения с координатами (x, y, z). Если f — это ►**фокусное расстояние**, т.е. расстояние от отверстия камеры до плоскости изображения, то из теоремы подобия треугольников можно получить следующие уравнения:

$$\frac{-x}{f} = \frac{X}{Z}, \quad \frac{-y}{f} = \frac{Y}{Z}, \quad \Rightarrow \quad x = \frac{-fX}{Z}, \quad y = \frac{-fY}{Z}.$$

Эти уравнения определяют процесс формирования изображения, называемый ►**перспективной проекцией**. Заслуживает внимания то, что значение координаты Z находится в знаменателе, а это означает, что чем дальше объект, тем меньше его изображение. Кроме того, наличие знака "минус" означает, что изображение *инвертировано*, т.е. повернуто на 180° по сравнению с самой сценой.

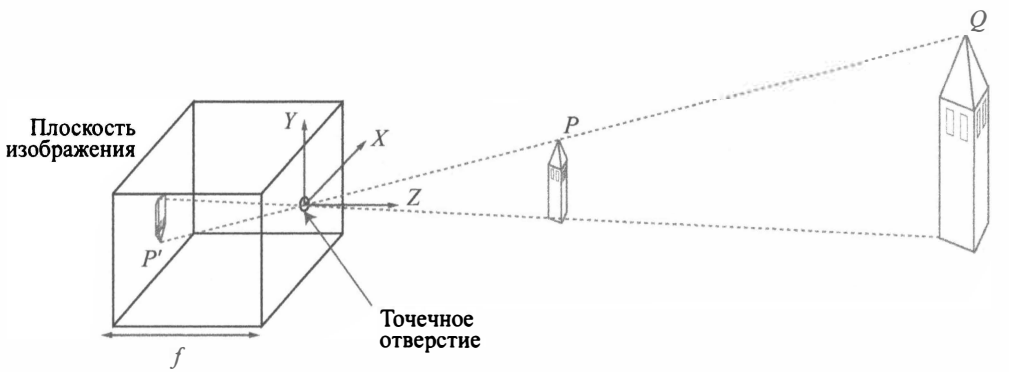


Рис. 25.2. Каждый светочувствительный элемент на задней плоскости камеры-обскуры получает свет, проходящий через отверстие в передней плоскости из небольшого диапазона направлений. Если отверстие достаточно мало, в результате будет получено изображение, сфокусированное на плоскости позади отверстия. Такая процедура получения проекции приводит к тому, что большие удаленные объекты выглядят на изображении так же, как более мелкие, но расположенные ближе: свет в точку P' на плоскости изображения может прийти как из точки P, находящейся вблизи на игрушечной башне, так и из точки Q на далекой реальной башне

Перспективное изображение обладает рядом геометрических эффектов. Удаленные объекты выглядят маленькими, а параллельные линии сходятся в одну точку на горизонте (вспомните о железнодорожных путях на рис. 25.1). Почему это происходит? Линия в сцене, проходящая через точку (X_0, Y_0, Z_0) в направлении (U, V, W), может быть описана как множество точек $(X_0 + \lambda U, Y_0 + \lambda V, Z_0 + \lambda W)$, где λ изменяется в пределах от $-\infty$ до $+\infty$. Различные варианты точек (X_0, Y_0, Z_0)

дают разные линии, параллельные друг другу. Проекция точки P_λ на этой линии до плоскости изображения определяется следующей формулой:

$$P_\lambda = \left(f\frac{X_0 + \lambda U}{Z_0 + \lambda W}, f\frac{Y_0 + \lambda V}{Z_0 + \lambda W} \right).$$

При $\lambda \to -\infty$ или $\lambda \to +\infty$ эта формула принимает вид $P_\infty = (fU/W, fV/W)$, если $W \neq 0$. А это означает, что две параллельные линии, проходящие через разные точки в пространстве, на изображении будут сходиться — при больших значениях λ точки на изображениях будут почти одинаковыми независимо от значений (X_0, Y_0, Z_0) (опять же, вспомните железнодорожные пути на рис. 25.1). Точка P_∞ называется ▶**точкой схода**, связанной с семейством прямых линий с ориентацией (U, V, W). Все линии, имеющие одну и ту же ориентацию, имеют одинаковую точку схода.

25.2.2. Системы линз

Камеры-обскуры хорошо фокусируют свет, но из-за того, что отверстие в них очень мало, внутрь попадает совсем немного света и изображение получается темным. В течение короткого периода времени лишь несколько фотонов столкнется с каждой точкой на сенсоре, поэтому в выходном сигнале от каждой точки будут преобладать случайные флуктуации; говорят, что изображение на темной фотопленке зернистое, а темное цифровое изображение — зашумленное. В любом случае такое изображение будет низкого качества.

Увеличение отверстия (апертуры) камеры-обскуры сделает изображение в ней ярче, поскольку свет будет собираться из более широкого диапазона направлений. Однако при большей апертуре свет, попадающий в конкретную точку на плоскости изображения, будет приходить уже не из одной, а из нескольких точек на сцене реального мира, в результате чего изображение окажется расфокусированным. Следовательно, нужно найти какой-то способ заново сфокусировать изображение.

Глаза позвоночных и современные камеры используют систему ▶**линз**: это один фрагмент прозрачной ткани в глазу и система из нескольких стеклянных линз — в камерах. На рис. 25.3 можно видеть, что свет от кончика пламени свечи распространяется во все стороны. Камера (или глаз) с линзами захватывает весь свет, попадающий в любое место на ее линзе — а это гораздо бо́льшая площадь, чем у крошечного отверстия в камере-обскуре, — и фокусирует весь этот свет в одну точку на плоскости изображения. Свет из других частей пламени свечи будет аналогичным образом собран и сфокусирован на других точках на плоскости изображения. В результате будет получено более яркое, менее зашумленное и хорошо сфокусированное изображение.

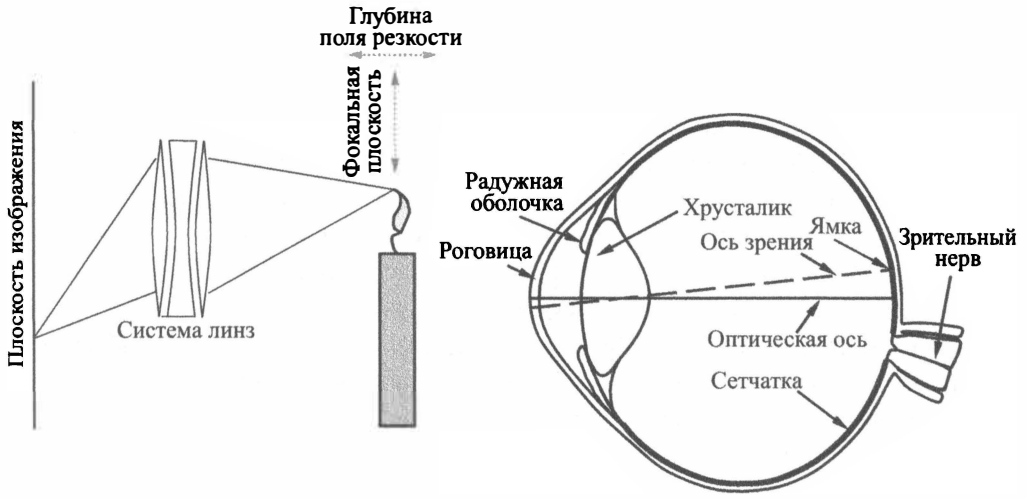


Рис. 25.3. Линзы собирают свет, исходящий из одной точки в сцене (здесь это кончик пламени свечи) в некотором диапазоне направлений, и направляют его таким образом, чтобы он попал в одну точку на плоскости изображения. Точки в сцене, расположенные вблизи фокальной плоскости — в пределах глубины поля резкости, — будут сфокусированы должным образом. В камерах для изменения расположения фокальной плоскости перемещаются определенные элементы системы линз, тогда как в глазу человека для этого изменяется *форма* линзы (хрусталика) благодаря работе специальных мышц

Системы линз не позволяют одновременно фокусировать весь свет, поступающий отовсюду в реальном мире. Конструкция системы линз ограничивает ее возможности: правильно фокусируется только свет, исходящий из тех точек, которые находятся от линзы на расстоянии Z, ограниченном определенным диапазоном. Центр этого диапазона — место, где фокус будет наиболее четким — называется ►**фокальной плоскостью**, а диапазон глубин, в пределах которого фокус остается достаточно острым, называется ►**глубиной поля резкости**. Чем больше апертура системы линз (входное отверстие), тем меньше глубина резкости.

А что делать, если потребуется переместить фокус на что-то, находящееся на ином расстоянии? В камерах для перемещения фокальной плоскости применяется перемещение элементов системы линз вперед и назад, тогда как в глазу человека эта же цель достигается изменением формы линзы, т.е. хрусталика. Однако с возрастом хрусталик глаза имеет тенденцию к отвердеванию, что снижает его способность регулировать фокусные расстояния, и это вынуждает многих людей улучшать свое зрение с помощью дополнительной внешней линзы — очков.

25.2.3. Масштабированная ортогональная проекция

На изображении с перспективой геометрические соотношения часто нарушаются. Например, окно в здании через улицу выглядит намного меньшим, чем расположенное по соседству, но если эти два окна поместить рядом одно с другим, они будут примерно одинакового размера, даже если одно будет чуть дальше, чем другое. Для обработки таких объектов, как окна, вместо перспективной проекции можно использовать упрощенную модель под названием ▶**масштабированная ортогональная проекция**. Если глубина Z всех точек на объекте попадает в диапазон $Z_0 \pm \Delta Z$ при $\Delta Z \ll Z_0$, то коэффициент масштабирования перспективной проекции f/Z можно аппроксимировать константой $s = f/Z_0$. В этом случае уравнения для проекции точки с координатами сцены (X, Y, Z) на плоскость изображения принимают вид $x = sX$ и $y = sY$. Ракурс в модели масштабированной ортогональной проекции по-прежнему сохраняется, поскольку этот эффект вызывается наклоном объекта относительно видовой плоскости.

25.2.4. Свет и тень

Яркость пикселя на изображении является функцией яркости того пятна поверхности в сцене, которое проецируется на этот пиксель. Для современных камер эта функция является линейной для средней интенсивности света, но характеризуется ярко выраженной нелинейностью в случае более темного или более яркого освещения. Здесь мы будет использовать линейную модель. Яркость изображения — сильный, хотя и неоднозначный сигнал в отношении как формы объекта, так и его идентичности. Неоднозначность возникает по той причине, что существует три фактора, определяющие количество света, исходящего от точки на объекте к изображению: общая интенсивность ▶**внешнего освещения** (*ambient light*); точка находится на освещенном источником света участке или в тени; а также количество света, ▶**отражаемого** этой точкой.

Человек удивительно хорошо справляется с устранением такой неоднозначности в яркости — обычно мы легко отличаем черный объект в ярком свете от белого объекта в тени, даже если фотометрически они оба имеют одну и ту же яркость. Тем не менее люди иногда смешивают затенение и маркировку — полоса темного макияжа под скулой часто воспринимается как падающая тень, что делает лицо на вид более худым.

Большинство поверхностей отражает свет посредством ▶**диффузного отражения**. При диффузном (или рассеянном) отражении свет отражается от поверхности равномерно по всем направлениям, поэтому яркость такой поверхности не зависит от направления, с которого на нее смотрят. Такими свойствами обладает большинство тканей, большинство обычных красок, а также грубо обработанные деревянные поверхности, бо́льшая часть растительности и грубо обработанный камень или бетон.

При ▶**зеркальном отражении** падающий свет покидает поверхность в некотором лепестке направлений, определяемых тем направлением, с которого поступает свет. Одним из примеров является зеркало — то, что в нем видно, зависит от направления, с которого в него смотрят. В этом случае лепесток направлений очень узок, поэтому в зеркале легко различать разные объекты.

Однако для многих других поверхностей лепесток направлений отражения будет шире. Такие поверхности отбрасывают небольшие яркие пятна, обычно называемые ▶**бликами**. При движении отражающей поверхности или источника света блики тоже перемещаются. На удалении от таких зеркально отражающих пятен поверхность будет иметь диффузное отражение. Участки зеркального отражения часто имеются на металлических поверхностях, на гладко окрашенных поверхностях, на поверхности изделий из пластика или мокрых поверхностях. Их легко идентифицировать, потому что обычно они яркие и невелики по размеру (рис. 25.4). Практически для любых целей достаточно моделировать все поверхности как диффузные с зеркальными участками.

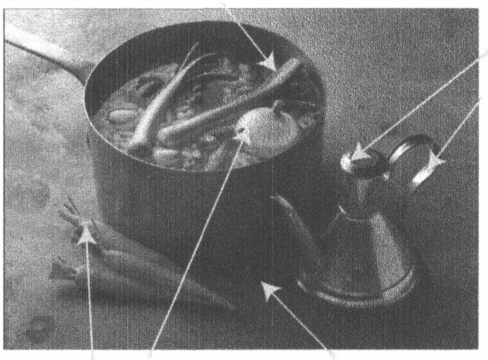


Рис. 25.4. На этой фотографии показаны различные эффекты освещения. Здесь есть участки зеркального отражения на графинчике из нержавеющей стали. Лук и морковь — это яркие диффузные поверхности, потому что свет падает прямо на них. Тени появляются в тех точках поверхностей, которые полностью закрыты от источника света. Внутри ковша есть несколько темных рассеянных поверхностей, на которые свет падает под большим углом. (Внутри ковша также есть тени.)

Основным источником освещения вне помещений является солнце, все лучи которого проходят параллельно друг другу в известном направлении, поскольку оно находится очень далеко. Это поведение представляет собой модель с ▶**отдаленным точечным источником света**. Это самая важная модель освещения,

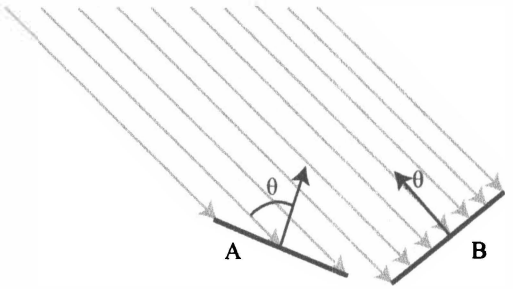


Рис. 25.5. Два пятна на поверхности освещаются удаленным точечным источником света, лучи которого показаны серыми стрелками. Пятно A расположено к лучам света под углом θ, близким к 90°, и собирает меньше энергии, поскольку пересекает меньше световых лучей на единицу площади своей поверхности. Поверхность пятна B направлена почти точно на источник света (угол θ близок к 0°) и собирает значительно больше световой энергии

и она достаточно эффективна для сцен как внутри помещений, так и на открытом воздухе. Количество света, собранного пятном поверхности в этой модели, зависит от угла θ между направлением освещения и нормалью (перпендикуляром) к поверхности (рис. 25.5).

Пятно на поверхности с диффузным отражением, освещаемое по этой модели, будет отражать некоторую часть падающего на нее света в соответствии с ее ▶**диффузным альбедо**. Для реальных поверхностей этот показатель находится в диапазоне 0,05–0,95. Согласно ▶**закону косинуса Ламберта**, яркость диффузного пятна определяется выражением

$$I = \rho I_0 \cos\theta,$$

где I_0 — интенсивность источника света, θ — угол между направлением на источник света и нормалью к поверхности, а ρ — диффузное альбедо. Этот закон указывает, что яркие пиксели в изображении соответствуют тем пятнам на поверхности, которые обращены непосредственно к свету, а темные пиксели — тем пятнам, на которые свет падает под значительным углом, так что затенение поверхности предоставляет некоторую информацию о ее форме. Если источник света с поверхности не виден, то она находится в ▶**тени**. Тени очень редко бывают однородного черного цвета, потому что на затененную поверхность обычно попадает свет от каких-то других источников. На открытом воздухе самым важным источником света, после солнца, является небо, которое также достаточно ярко. В помещении затененные участки освещает свет, отраженный от других поверхностей. Эти ▶**взаимные отражения** также могут оказывать существенное влияние на яркость отдельных поверхностей. Подобные эффекты иногда моделируются путем

добавления к прогнозируемой освещенности некоторого постоянного члена, представляющего ▶ **рассеянное освещение**.

25.2.5. Цвет

Фрукт — это взятка, которую дерево предлагает животным за перенос своих семян. Деревья, способные подать сигнал, когда эта взятка уже готова к переносу, имеют преимущество, как и те животные, которые способны прочесть эти сигналы. В результате большинство фруктов начинают свой рост с зеленого цвета, а при созревании становятся красными или желтыми, и большинство поедающих фрукты животных способно увидеть и правильно истолковать такое изменение цвета. В общем случае свет, попадающий в глаз, обладает разным количеством энергии в зависимости от длины волны, что может быть представлено как спектральная плотность энергии.

Обычные камеры и система человеческого зрения реагируют на свет в диапазоне длин волн от примерно 380 нм (фиолетовый) до примерно 750 нм (красный). В системах цветного изображения существуют различные типы рецепторов, которые более или менее сильно реагируют на световые волны различной длины. У человека ощущение цвета возникает, когда система зрения сравнивает реакцию рецепторов, расположенных на сетчатке рядом друг с другом. Системы цветного зрения животных обычно имеют относительно немного типов рецепторов и поэтому представляют относительно мало деталей в функции распределения спектральной плотности энергии (одни животные имеют только один тип рецепторов, тогда как у других их может быть до шести типов). Цветное зрение человека обеспечивается тремя типами рецепторов. Большинство систем цветных камер также использует только три типа рецепторов, поскольку изображения в них создаются для людей, однако некоторые специализированные системы способны выполнять очень детализированные измерения спектральной плотности энергии.

Поскольку у большинства людей имеется лишь три типа рецепторов, чувствительных к цвету, используется ▶ **принцип трех основных цветов**. Эта идея впервые была предложена Томасом Янгом в 1802 году. Он утверждал, что любая смесь световых волн с разными значениями длины волны, вне зависимости от ее сложности, может быть представлена в виде смеси, состоящей лишь из трех ▶ **основных цветов**. Основные цвета источников света выбираются так, чтобы ни одна возможная смесь из двух цветов не соответствовала третьему. Наиболее распространенный вариант основных цветов — это красный (*red*), зеленый (*green*) и синий (*blue*), сокращенно — ▶ **RGB**. Хотя цвет некоторого объекта может определяться многими компонентами с разной частотой излучения, всегда можно подобрать конкретный оттенок, смешав в той или иной пропорции только три основных цвета, и при этом большинство людей согласятся с выбранными в данной смеси пропорциями. А это означает, что можно представить цветные изображения всего тремя числовыми значениями для одного пикселя — RGB-значениями.

Для большинства приложений компьютерного зрения можно достаточно точно смоделировать поверхности как имеющие три различных (RGB) диффузных альбедо и источники света — как имеющие три излучателя основных цветов (RGB) различной интенсивности. Далее к лучу каждого из них можно применить закон косинуса Ламберта и получить числовые значения красной, зеленой и синей компонент окраски освещенного ими пикселя. Такая модель правильно предсказывает, что одна и та же поверхность создает различные цветные изображения одного и того же пятна при освещении ее лучами разного цвета. В действительности наблюдатель-человек обладает способностью довольно хорошо игнорировать эффекты освещения поверхности разными цветами и правильно давать оценку того цвета, которую она имела бы при освещении белым светом, — этот эффект известен как ▶ **постоянство цвета**.

25.3. Простые свойства изображения

Свет, отражаясь от объектов в сцене, формирует изображение, состоящее, скажем, из двенадцати миллионов пикселей, каждый из которых представлен тремя байтами. Как и в случае любых иных датчиков, на изображении будет шум, но в данном случае важнее то, что объем этих данных очень велик. Чтобы начать анализ этих данных, необходимо прежде получить его некоторое упрощенное представление, выявляющее то, что в нем важно, и уменьшающее излишнюю детализацию. В большинстве современных практических методов эти представления обучаются на основании данных. Однако есть четыре свойства изображений и видео, которые являются наиболее общими: края, текстура, оптический поток и сегментация изображения на участки со схожими признаками.

Края встречаются там, где имеет место большая разница в интенсивности пикселей в пределах определенной части изображения. Построение представлений краев включает локальные операции над изображением: достаточно сопоставить значение данного пикселя со значениями нескольких находящихся рядом пикселей, и при этом не требуется никаких знаний о том, что представляет собой изображение в целом. Следовательно, обнаружение краев может выполняться на ранних этапах общего конвейера обработки и рассматриваться как операция "предварительной обработки" или "низкого уровня".

Другие операции требуют обработки большей области изображения. Например, описание текстуры применяется к некоторой совокупности пикселей: чтобы сказать "полосатый", нужно увидеть несколько полос. Оптический поток представляет, куда смещаются пиксели при переходе от одного изображения в последовательности к следующему, и может охватывать значительную область в изображении. При сегментации изображение разделяется на участки пикселей, которые естественным образом связаны друг с другом и образуют единую группу, но чтобы выполнить это, необходимо проанализировать весь участок. Операции, подобные этим, иногда называют операциями "среднего уровня".

25.3.1. Края

►**Края** (или **границы**) — это прямые или кривые *линии* в плоскости изображения, разделяющие участки со "значимыми" изменениями уровня яркости. Целью обнаружения краев является переход от перегруженного подробностями многомегабайтного представления изображения к его более компактному, абстрактному представлению, как показано на рис. 25.6. Различные эффекты в сцене очень часто приводят к большим изменениям интенсивности освещенности соседних участков и как результат создают на изображении края. Изменения по глубине (на рисунке обозначены цифрой 1) могут приводить к образованию краев, поскольку при пересечении таких границ цвет обычно меняется. При изменении ориентации поверхности (на рисунке обозначены цифрой 2) обычно меняется и интенсивность ее освещенности. Интенсивность освещенности изображения также часто меняется при изменении отражательной способности поверхности, связанной со свойствами материала (на рисунке обозначены цифрой 3). Наконец, тени (на рисунке обозначены цифрой 4) представляют различия в освещении, вызванные краями в изображении, даже если объект не имеет краев. Краевые детекторы не способны установить причину неоднородности в освещении, — решение этого вопроса оставляется для дальнейшей обработки.

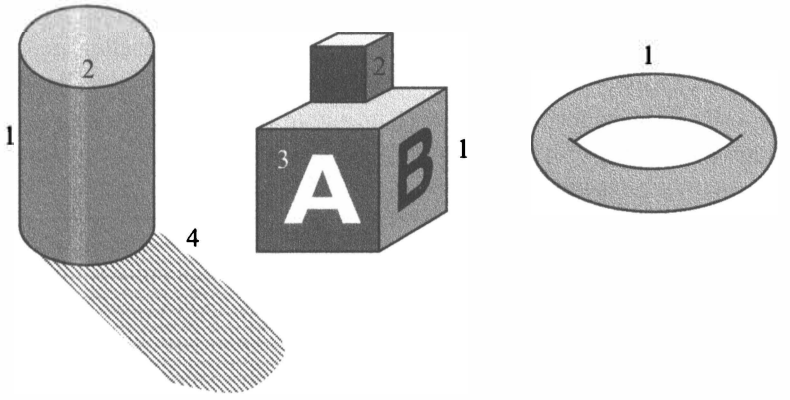


Рис. 25.6. Виды краев: 1 — изменения (разрывы) по глубине; 2 — изменения в ориентации поверхностей; 3 — изменения в отражательной способности; 4 — тени, представляющие собой неоднородности (разрывы) в освещенности

Поиск краев требует определенной осторожности. На рис. 25.7, *а* приведен график изменения интенсивности $I(x)$ вдоль одномерного сечения изображения, перпендикулярного краю, расположенному в точке $x = 50$.

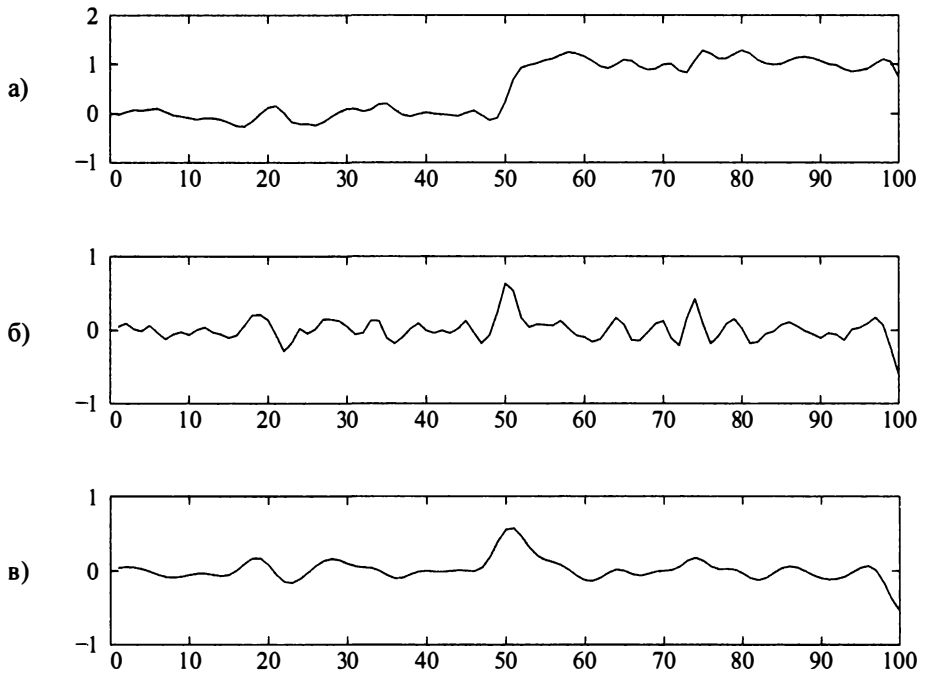


Рис. 25.7. а) Профиль интенсивности $I(x)$ вдоль одномерного сечения, перпендикулярного краю на изображении (точка $x = 50$). **б)** График производной функции интенсивности $I'(x)$. Большие значения этой функции соответствуют краям, но функция сильно зашумлена. **в)** Производная сглаженной версии функции интенсивности. Создаваемый шумом кандидат края в точке $x = 75$ исчез

Можно продифференцировать интенсивности в изображении и искать те места, где величина производной $I'(x)$ будет большой. Такой подход работает, но подвержен ошибкам; на рис. 25.7, *б* можно видеть, что, хотя истинный край отмечен пиком при $x = 50$, есть и дополнительные пики в других местах (например, $x = 75$), которые могут быть ошибочно приняты за истинные края. Они возникают из-за наличия "шума" на изображении. ►**Шум** в данном случае означает изменение значения пикселя, не связанное с каким-либо краем. Например, в камере может присутствовать тепловой шум; на поверхности объекта могут быть царапины, изменяющие направление нормали к поверхности в небольшом регионе; на поверхности могут существовать незначительные изменения альбедо и т.д. В каждом из этих случаев градиент может оказаться достаточно большим, но это вовсе не значит, что здесь присутствует какая-либо граница. Если сначала "сгладить" изображение, паразитные пики уменьшатся, как показано на рис. 25.7, *в*.

Сглаживание предполагает использование окружающих пикселей для подавления шума. "Истинное" значение пикселя прогнозируется как взвешенная сумма значений соседних пикселей, с бо́льшим весом для ближайших соседей. Естественным выбором для присвоения весов является ▶**фильтр Гаусса**. Напомним, что функция Гаусса с нулевым средним и стандартным отклонением σ имеет вид

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \quad \text{в одном измерении или}$$

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad \text{в двух измерениях.}$$

Применение фильтра Гаусса означает замену интенсивности $I(x_0, y_0)$ суммой по всем пикселям (x, y) значений $I(x, y)G_\sigma(d)$, где d — расстояние от (x_0, y_0) до (x, y). Взвешенная сумма такого вида используется так часто, что для нее предусмотрены особое название и обозначение. Говорят, что функция h является ▶**сверткой** (*convolution*) двух функций, f и g (обозначается как $h = f * g$), если имеют место следующие соотношения:

$$h(x) = \sum_{u=-\infty}^{+\infty} f(u)g(x-u) \quad \text{в одном измерении или}$$

$$h(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v)g(x-u, y-v) \quad \text{в двух измерениях.}$$

Таким образом, операция сглаживания осуществляется путем свертки функции интенсивности изображения с функцией Гаусса: $I * G_\sigma$. Значения σ в 1 пиксель будет достаточно, чтобы сгладить небольшое количество шума, тогда как при 2 пикселях будет сглажено большее его количество, но с потерей некоторых деталей. Поскольку влияние функции Гаусса на результат быстро ослабевает с расстоянием, на практике можно в суммах заменить пределы $\pm\infty$ чем-то вроде $\pm 3\sigma$.

В результате появляется возможность провести некоторую оптимизацию: операции сглаживания и поиска краев теперь можно объединить в одну операцию. Существует теорема, согласно которой для любых функций f и g производная их свертки, $(f * g)'$, равна свертке с производной, $f * (g')$. Поэтому вместо сглаживания изображения с последующим дифференцированием можно просто выполнить свертку изображения с производной гауссовой функцией сглаживания, G'_σ. Далее в полученном результате как края отмечаются те пики, которые превышают некоторый порог, выбранный для устранения ложных пиков, появившихся из-за шума.

Существует естественное обобщение этого алгоритма от одномерных сечений к обобщенному двухмерному изображению. В двух измерениях ребра могут проходить под любым углом θ. Если рассматривать яркость изображения как

скалярную функцию переменных x и y, то ее градиент будет представлять собой вектор

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix}.$$

Края соответствуют тем местам на изображениях, где яркость резко меняется, а следовательно, величина градиента, $\|\nabla I\|$ в краевой точке должна быть больше. Когда изображение становится ярче или темнее, вектор градиента в каждой точке становится длиннее или короче, но направление градиента

$$\frac{\nabla I}{\|\nabla I\|} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

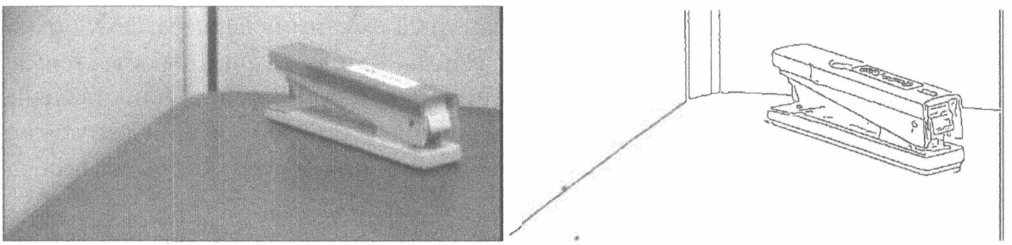
не меняется. Это дает нам значение $\theta = \theta(x, y)$ для каждого пикселя, определяющее ►**ориентацию** края в этом пикселе. Данный признак часто бывает полезным, потому что не зависит от интенсивности изображения.

Как и следовало ожидать после обсуждения возможностей обнаружения краев в одномерных массивах, для формирования градиента фактически вычисляется не ∇I, а скорее $\nabla(I * G_\sigma)$ — градиент результата сглаживания изображения посредством его свертки с гауссианом. Свойство сверток состоит в том, что они эквивалентны свертке исходного изображения с частными производными гауссиана. Как только градиент будет вычислен, можно будет выявить края, отыскав граничные точки и связав их друг с другом. Чтобы установить, является ли точка краевой, следует рассмотреть другие точки на небольшом расстоянии вперед и назад вдоль направления градиента. Если величина градиента в одной из таких точек больше, можно получить лучшую краевую точку, слегка сместив кривую края. Кроме того, если величина градиента некоторой точки слишком мала, данная точка не может быть краевой. Поэтому для любой краевой точки величина градиента должна быть локальным максимумом вдоль направления градиента, а само значение градиента должно быть выше подходящего случаю порога.

После того как пиксели краев будут выявлены с помощью приведенного выше алгоритма, на следующем этапе нужно будет связать друг с другом те пиксели, которые принадлежат к одним и тем же кривым краев. Эту операцию можно выполнить, приняв предположение, что два любых соседних пикселя, являющихся краевыми пикселями с совместимыми ориентациями, должны принадлежать к одной и той же кривой края.

На рис. 25.8, *а* показано изображение сцены, представляющей лежащий на столе степлер, а на рис. 25.8, *б* — результаты применения к этому изображению алгоритма обнаружения краев. Как видите, эти результаты не идеальны: есть участки, на которых края пропадают и образуются пропуски, а также есть края,

появившиеся под воздействием шума, не соответствующие каким-либо значимым деталям в этой сцене. Все эти ошибки должны быть исправлены на последующих этапах обработки изображения.



а) б)

Рис. 25.8. а) Фотография степлера. **б)** Края, вычисленные в результате обработки изображения *а*

25.3.2. Текстура

В повседневной речи под ▶**текстурой** подразумевается свойство поверхностей, связанное с осязательными ощущениями, возникающими, если провести по ней пальцем (слова "текстура" "текстиль" и "текст" происходят от одного и того же латинского корня "texere", означающего "ткать"). В области компьютерного зрения под этим словом понимается некий узор на поверхности, многократно скопированный шаблон, который можно обнаружить визуально, — обычно он повторяется приблизительно регулярно. К примерам текстуры можно отнести шаблонное расположение окон на здании, узор из петель на свитере, пятна на шкуре леопарда, множество травинок на газоне, гальку на пляже или толпу людей на стадионе.

В одних случаях расположение узора будет периодическим, как узор на свитере, а в других, например галька на пляже, закономерность будет существовать только в статистическом смысле: плотность гальки будет примерно одинаковой на разных участках пляжа. Обычная грубая модель текстуры представляет собой повторяющуюся структуру из элементов, иногда называемых ▶**текселями**. Такая модель очень полезна, потому что на удивление трудно создать или найти настоящие текстуры, которые никогда не повторяются.

Текстура — это свойство участка изображения, а не отдельного пикселя. Хорошее описание текстуры участка должно обобщенно представлять, как он выглядит, и оно не должно меняться при изменении освещения. Последнее требование исключает использование краевых точек: если текстура ярко освещена, множество мест в пределах участка будет иметь высокую контрастность, что приведет к генерации соответствующих краевых точек. Но если на ту же текстуру посмотреть при

менее ярком свете, яркость большинства этих краевых точек не превысит порогового значения. Также описание должно разумным образом измениться, если участок будет повернут. Очень важно сохранять различие между вертикальными и горизонтальными полосами, *но* не в том случае, когда после поворота вертикальные полосы превратятся в горизонтальные.

Было показано, что представления текстур со схожими свойствами будут полезны при решении двух ключевых задач. Первая — идентификация объектов: зебра и лошадь имеют схожие формы, но разную текстуру. Вторая — сопоставление участков на одном изображении с участками на другом изображении; это ключевой этап восстановления трехмерной информации на основе нескольких изображений (раздел 25.6.1).

Базовая конструкция для представления текстуры такова. Для заданного участка изображения вычисляется ориентация градиента каждого его пикселя, а затем этот участок характеризуется гистограммой обнаруженных на нем ориентаций. Градиентные ориентации в значительной степени инвариантны к изменениям освещения (вектор градиента изменит свою длину, но его направление останется неизменным). Создается впечатление, что гистограмма ориентаций способна захватить важные аспекты текстуры. Например, наличие вертикальных полос на гистограмме будет отмечено двумя пиками (один — для левой стороны каждой полосы, другой — для правой), тогда как леопардовым пятнам будет соответствовать более равномерное распределение ориентаций.

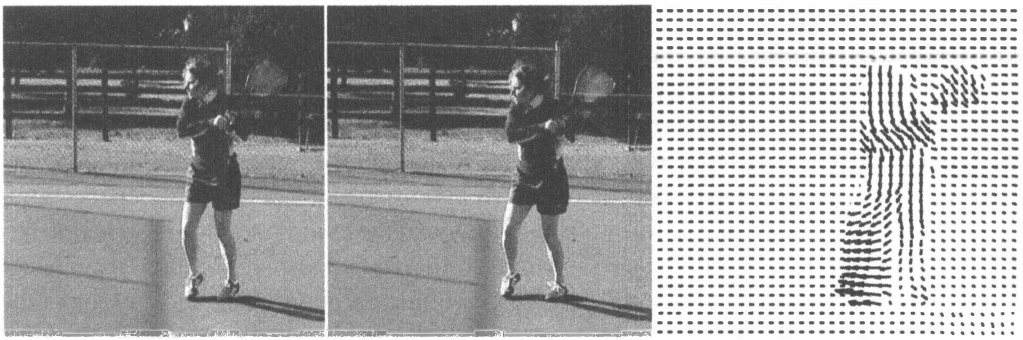
Но при таком подходе остается неизвестным, насколько большой участок следует описывать. Здесь возможны две стратегии. В специализированных приложениях информация об изображении сама указывает, насколько большим должен быть участок (например, можно расширять участок с полосками, пока он не покроет всю зебру). Альтернативой является описание участка с центром в каждом пикселе для некоторого диапазона масштабов. Такой диапазон обычно начинается с нескольких пикселей и закачивается размерами всего изображения. Далее участок делится на сегменты и для каждого сегмента создается гистограмма ориентаций, после чего шаблон полученных гистограмм обобщается по всем сегментам. Понятно, что подобные описания невозможно построить вручную, поэтому для создания таких представлений текстур используются сверточные нейронные сети. Однако построенные сетями представления, похоже, отражают эту конструкцию очень грубо.

25.3.3. Оптический поток

Теперь давайте рассмотрим, что происходит, когда обрабатывается видеопоследовательность, а не одно статическое изображение. Всякий раз, когда имеет место относительное перемещение между видеокамерой и одним или несколькими объектами в сцене, в изображении возникает кажущееся движение, называемое ►**оптическим потоком**. Оптический поток описывает направление и скорость

движения признаков *в изображении* как результат относительного движения между наблюдателем и сценой. Например, удаленные объекты, видимые из движущегося автомобиля, будут иметь гораздо более медленное кажущееся движение, чем близлежащие объекты, поэтому скорость кажущегося движения позволяет получить определенную информацию о расстоянии.

На рис. 25.9, *а* и *б* показаны два кадра из видеозаписи игры в теннис. На рис. 25.9, *в* показаны векторы оптического потока, вычисленные на основе этих двух изображений. В оптическом потоке зашифрована полезная информация о структуре сцены — теннисист движется, а фон (в основном) нет. Более того, векторы потока сообщают что-то и о том, что именно делает игрок — одна его рука и одна нога перемещаются быстро, а другие части тела — значительно медленнее.



а) б) в)

Рис. 25.9. а), б). Два последовательных кадра видеозаписи игры в теннис. **в)** Поле оптического потока, соответствующее кажущемуся смещению объектов сцены от одного кадра к другому. Обратите внимание, как движение теннисной ракетки и передней ноги захвачено направлениями стрелок векторов потока

Векторное поле оптического потока может быть представлено его компонентами $v_x(x, y)$ в направлении x и $v_y(x, y)$ — в направлении y. Чтобы измерить оптический поток, необходимо найти соответствующие точки между одним временным кадром и следующим. Очень простой способ основан на том факте, что участки изображения, сосредоточенные вокруг соответствующих точек, имеют сходные шаблоны интенсивности. Рассмотрим блок пикселей с центром в пикселе p, (x_0, y_0), в момент времени t. Этот блок пикселей необходимо сравнить с блоками пикселей, центрами которых являются различные пиксели-кандидаты q_i с координатами $(x_0 + D_x, y_0 + D_y)$ во время $t + D_t$. Одним из возможных критериев сходства является ▶**сумма квадратов разностей** (*Sum of Squared Differences* — ▶**SSD**):

$$\mathrm{SSD}(D_x, D_y) = \sum_{(x,y)} (I(x, y, t) - I(x + D_x, y + D_y, t + D_t))^2.$$

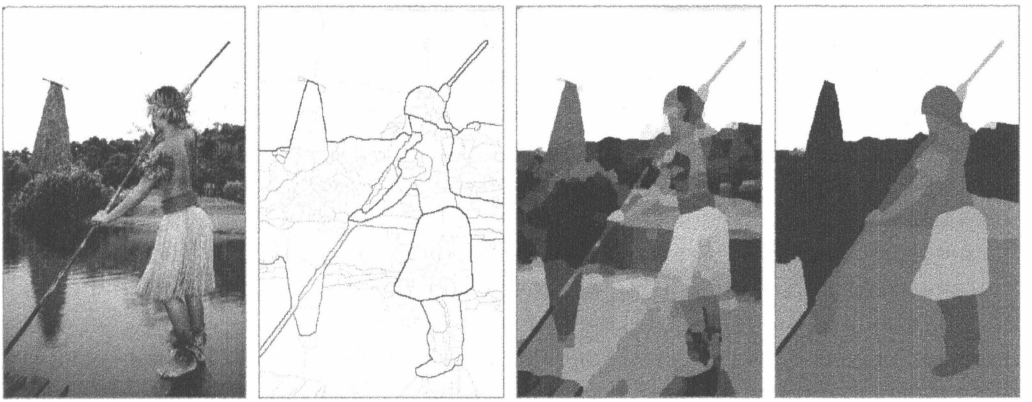
Здесь координаты (x, y) принимают значения среди пикселей в блоке с центром в точке (x_0, y_0). Найдем значения (D_x, D_y), минимизирующие выражение для SSD. В таком случае оптический поток в точке (x_0, y_0) принимает значение $(v_x, v_y) = (D_x/D_t, D_y/D_t)$. Обратите внимание, что для того, чтобы этот подход работал, в сцене должна быть какая-то текстура, в результате чего в изображении появляются сегменты, содержащие значительное изменение яркости среди пикселей. Если смотреть на равномерно белую стену, то значение SSD будет почти одинаковым для различных пикселей-кандидатов q и работа алгоритма сведется к выдвижению слепого предположения. Самые эффективные алгоритмы измерения оптического потока зависят от множества дополнительных ограничений, которые необходимы, чтобы справиться с ситуациями, когда общее поле сцены текстурировано только частично.

25.3.4. Сегментация изображений

►**Сегментация** — это процесс разбиения изображения на группы пикселей, имеющих схожие характеристики. Основная идея состоит в том, что каждый пиксель изображения может быть связан с определенными визуальными свойствами, такими как яркость, цвет и текстура. В пределах одного объекта или одной части объекта эти атрибуты изменяются относительно мало, тогда как при пересечении границы между двумя объектами обычно происходит существенное изменение одного или более этих атрибутов. Необходимо найти вариант разбиения изображения на такие множества пикселей, чтобы указанные ограничения удовлетворялись как можно лучше. Обратите внимание, что недостаточно просто найти края, поскольку многие края не являются границами объектов. Так, например, для изображения тигра в траве края могут быть созданы для обеих сторон каждой полосы на шкуре тигра и каждой травинки. Среди всех этих запутанных данных о краях можно и не заметить тигра за полосками.

Есть два способа изучения данной проблемы, один из которых направлен на выявление границ таких групп, а другой — на обнаружение самих этих групп, называемых ►**сегментами** (*region*). Оба варианта проиллюстрированы на рис. 25.10, где *а* — это исходное изображение, *б* — пример определения границ сегментов, а *в* и *г* — два варианта выделения самих сегментов.

Одним из средств формализации проблемы обнаружения граничных кривых является задача классификации, поддающаяся методам машинного обучения. Пусть граничная кривая, проходящая через пиксель с расположением (x, y), имеет ориентацию θ. Тогда на изображении окрестность с центром в (x, y) будет выглядеть примерно как диск, разрезанный на две половины диаметром с ориентацией по θ. Можно вычислить вероятность $P_b(x, y, \theta)$ того, что через этот пиксель проходит граничная кривая с ориентацией θ, путем сравнения признаков в двух половинах окрестности. Естественный способ предсказать эту вероятность — обучить машину задаче классификации с использованием обучающего набора в виде естественных изображений, на которых человек пометил истинные границы — назначение



а) б) в) г)

Рис. 25.10. а) Исходное изображение. **б)** Контуры границ сегментов; здесь чем выше значение P_b, тем темнее линия контура. **в)** Выделение в изображении сегментов, соответствующее тонкому разбиению изображения. Сегменты отображаются в их средних цветах (на черно-белом варианте — средней яркости). **г)** Выделение в изображении сегментов, соответствующее более грубому разбиению изображения, в результате чего получено меньше сегментов

такого классификатора состоит в том, чтобы отмечать именно те границы, которые были помечены человеком, и никакие другие.

Обнаружение границ с использованием этого метода работает лучше, чем обнаружение границ с помощью простой методики выявления краев, описанной выше. Но и в этом случае все еще есть два ограничения. Первое — граничные пиксели, определенные по пороговому значению $P_b(x, y, \theta)$, необязательно образуют замкнутые кривые, поэтому такой подход не обеспечивает выделение сегментов. Второе — при принятии решений используется только локальный контекст и не используются глобальные ограничения согласованности.

Альтернативный подход строится на попытке "кластеризации" пикселей на сегменты на основе их яркости, цвета и текстурных свойств. Существует много различных способов, которыми эта идея может быть формализована математически. Например, Ши и Малик ([2048], 2000) определили ее как задачу сегментации графа. Узлы графа соответствуют пикселям, а ребра — соединениям между пикселями. Ребрам, соединяющим пары пикселей i и j, присваиваются веса W_{ij} в соответствии с тем, насколько близки значения яркости, цвета, текстуры и так далее этих двух пикселей. Затем осуществляется поиск разбиений, минимизирующих критерий *нормализованного отсечения*. Грубо говоря, критерием сегментации графа является критерий минимизации суммы весов соединений между группами и максимизации суммы весов соединений в пределах групп.

Как оказалось, подходы, основанные на поиске границ и нахождении сегментов, могут быть связаны, но здесь эти возможности рассматриваться не будут. Сегментация строится исключительно на локальных атрибутах низкого уровня, таких как яркость и цвет, и поэтому от нее не следует ожидать получения безусловно правильных границ всех объектов в сцене. Чтобы надежно отыскать границы, связанные с объектами, необходимо также применять знания высокого уровня о тех типах объектов, присутствия которых можно ожидать в сцене. В настоящее время популярной стратегией является выполнение чрезмерной сегментации изображения, когда гарантированно не будет пропущена пометка любых истинных границ, но при этом, возможно, также будут дополнительно помечены многие ложные границы. Полученные сегменты, называемые ▶**суперпикселями**, обеспечивают значительное снижение вычислительной сложности для различных алгоритмов, поскольку количество суперпикселей может исчисляться сотнями, тогда как необработанных пикселей в изображении миллионы. Использование знаний об объектах высокого уровня подробно обсуждается в следующем разделе, а фактическое обнаружение объектов на изображениях — тема раздела 25.5.

25.4. Классификация изображений

Классификация изображений используется в двух основных вариантах. В первом случае изображения являются *объектами*, взятыми из заданной таксономии классов, и на картинке больше нет ничего значимого, например каталог одежды или образцов мебели, для которых фон не имеет значения, а выход классификатора — это "кашемировый свитер" или "рабочий стул".

Во втором случае каждое изображение представляет некоторую *сцену*, содержащую много объектов. Так, на просторах саванны можно увидеть жирафа и льва, а в гостиной можно увидеть диван и лампу, но едва ли следует ожидать присутствия в гостиной жирафа или подводной лодки. В результате появляется возможность крупномасштабной классификации изображений, результатом которой должен быть точный ответ: "саванна" или "гостиная".

Современные системы классификации изображений опираются на ▶**внешний вид** (*appearance*), т.е. цвет и текстуру, а не геометрию объектов. Но здесь есть две трудности. Во-первых, разные экземпляры одного и того же класса могут выглядеть по-разному: одни кошки черные, а другие — рыжие. Во-вторых, одна и та же кошка в разное время может выглядеть по-разному в зависимости от тех или иных эффектов, указанных ниже, — как показано на рис. 25.11.

- **Освещение.** Меняет яркость и цвет изображения.
- **Ракурс**. Изменение ракурса приводит к искажению структуры, видимой под углом.

- **Аспект**. Объекты могут выглядеть по-разному, если смотреть на них с разных сторон. Если смотреть сбоку, бублик выглядит как сплющенный овал, а при взгляде сверху это кольцо.
- **Окклюзия**. Некоторые части объекта могут быть скрыты по той или иной причине. Объекты часто частично закрывают друг друга, или одни части объекта могут закрывать другие его части, — эффект, известный как **самоокклюзия**.
- **Деформация**. Многие объекты способны менять свою форму. Так, теннисист, отбивая мяч, меняет положение своих рук и ног.

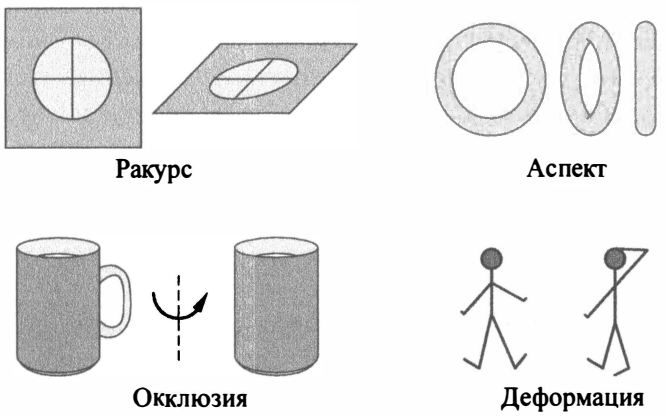


Рис. 25.11. Главные причины изменения внешнего вида, из-за которых разные изображения одного и того же объекта могут выглядеть по-разному. Во-первых, может изменяться ракурс, под которым видны элементы изображения, — как в случае круга в верхнем левом углу рисунка, который при наклоне объекта будет выглядеть уже как эллипс. Во-вторых, объекты, при взгляде на них с разных сторон, могут довольно резко менять свою форму, — явление, известное как аспект. Соответствующий пример в верхнем правом углу рисунка включает три различных аспекта бублика. Два изображения кружки внизу слева иллюстрируют явление окклюзии: ручка кружки исчезает из виду при повороте кружки. Поскольку ручка является частью кружки, в данном случае имеет место самоокклюзия. Наконец, в нижнем правом углу рисунка приведен пример объекта, форма которого может деформироваться во времени

Современные методы позволяют решить эти проблемы посредством обучения сверточной нейронной сети на основе представлений и классификаций из очень больших наборов обучающих данных. При достаточно разнообразном обучающем наборе система классификации много раз встретится с любым важным эффектом в процессе обучения и на этой основе сможет настроиться на правильное истолкование данного эффекта.

25.4.1. Классификация изображений с помощью сверточных нейронных сетей

Системы на основе ▶**сверточных нейронных сетей** (*Convolutional neural networks* — ▶**CNN**) показали себя как чрезвычайно успешные системы классификации изображений. При наличии достаточного объема обучающих данных и достаточной изобретательности в процессе обучения сети CNN демонстрируют очень успешную классификацию, — достигнутые ими результаты гораздо лучше, чем когда-либо достигнутые с использованием других методов.

В разработке систем классификации изображений важнейшую роль сыграл набор данных ImageNet, включающий более 14 млн предназначенных для обучения изображений, классифицированных по более чем 30 тыс. тщательно продуманных тончайших категорий. Появление набора ImageNet также стимулировало прогресс в проведении ежегодных конкурсов. На них системы оцениваются как по точности классификации в одном наилучшем предположении, так и по точности на базе пяти лучших ответов. В последнем случае системы выдают по пять прогнозов, например *маламут*, *хаски*, *акита*, *самоед*, *эскимосская собака*. В наборе ImageNet выделяется 189 подкатегорий в *породах собак*, так что даже очень любящим собак людям иногда сложно правильно пометить изображение каким-то одним предположением.

В первом конкурсе ImageNet в 2010 году системы смогли достичь точности по пяти лучшим прогнозам на уровне не лучше 70%. Появление сверточных нейронных сетей в 2012 году и их последующее усовершенствование позволили к 2019 году достичь точности 98% по пяти лучшим прогнозам (что превосходит этот показатель для человека) и точности 87% по одному лучшему прогнозу. Создается впечатление, что основной причиной этого успеха является тот факт, что признаки, используемые в системах классификации на базе сетей CNN, были получены ими из обучающих данных, а не заранее подготовлены исследователями вручную, — это гарантировало, что отобранные признаки будут действительно полезны для задач классификации.

Прогресс в классификации изображений оказался столь быстрым по причине доступности больших наборов данных, таких как ImageNet, вследствие проведения соревнований на основе этих наборов данных, которые открыты и находятся в свободном доступе, а также благодаря широкому распространению успешных моделей. Победители соревнований публикуют программный код, а часто и предобученные параметры своих моделей, тем самым облегчая другим разработчикам работу с успешными архитектурами в попытке сделать их еще лучше.

25.4.2. Почему сверточные нейронные сети хорошо классифицируют изображения

Задача классификации изображений во всей своей полноте лучше всего понимается при взгляде на используемые при обучении наборы данных, но ImageNet слишком велик, чтобы разобраться в деталях. Другой популярный набор данных, MNIST, включает 70 тыс. рукописных изображений цифр 0–9 и часто используется в качестве стандартного набора данных для "прогрева". Анализ этого набора данных (несколько примеров из него показаны на рис. 25.12) позволяет раскрыть некоторые важные и довольно общие свойства. Можно взять изображение цифры и внести в него небольшие изменения, не меняя ее идентичности: цифру можно сжать, наклонить, сделать ярче или темнее, меньше или больше. Все это означает, что значения отдельных пикселей в данном случае не особенно информативны: известно, что цифра 8 должна иметь несколько темных пикселей в центре, а цифра 0 — нет, но эти темные пиксели будут находиться в немного различных местоположениях в каждом из вариантов цифры 8.

Другое важное свойство изображений заключается в том, что локальные шаблоны могут оказаться довольно информативными. Цифры 0, 6, 8 и 9 имеют петли; цифры 4 и 8 имеют пересечения; цифры 1, 2, 3, 5 и 7 имеют окончания линий, но не имеют петель или пересечений; цифры 6 и 9 имеют петлю и окончание линии. Кроме того, информативны и пространственные отношения между локальными шаблонами. Цифра 1 имеет два окончания линии, находящихся одно над другим; цифра 6 имеет окончание линии над петлей. Из этих наблюдений можно вывести стратегию, которая в настоящее время является центральным принципом компьютерного зрения: необходимо построить функции, соответствующие шаблонам в малых, локализованных соседствах, после чего другие функции будут отыскивать шаблоны из *этих* функций; затем следующие отыскивают шаблоны из предыдущих и т.д.

Это именно то, что сверточные нейронные сети делают очень хорошо. Можно рассматривать отдельный слой — **свертку**, за которой следует слой с функцией активации ReLU — как локальный детектор шаблонов (рис. 25.12). Свертка измеряет, насколько точно каждое локальное окно в изображении совпадает с шаблоном ядра, после чего функция ReLU сбрасывает в нуль показатели для окон с низким совпадением и подчеркивает окна с высокими показателями. В результате свертка с несколькими ядрами обнаружит несколько шаблонов, после чего можно будет перейти к обнаружению композитных шаблонов, применив другой сверточный слой к выходу первого слоя.

Приглядимся к выходу первого сверточного слоя. Каждое местоположение получает входные данные об этом местоположении от пикселей в окне. Выход функции ReLU, как уже было показано, образует простой детектор шаблонов. Теперь, если поместить второй слой поверх этого, каждое местоположение во втором слое получит входные данные об этом местоположении как выходные значения первого

слоя в окне. Это означает, что местоположения во втором слое находятся под влиянием пикселей в окне большего размера, чем в первом слое. Это можно представить себе как поиск "шаблона из шаблонов". Если поверх второго слоя поместить третий слой, местоположения в этом третьем слое будут находиться под влиянием пикселей в еще большем окне; четвертый слой будет под влиянием еще большего окна и т.д. Сеть создает шаблоны на нескольких уровнях и делает это, *изучая* данные, вместо использования готовых шаблонов, изначально заданных ей программистом.

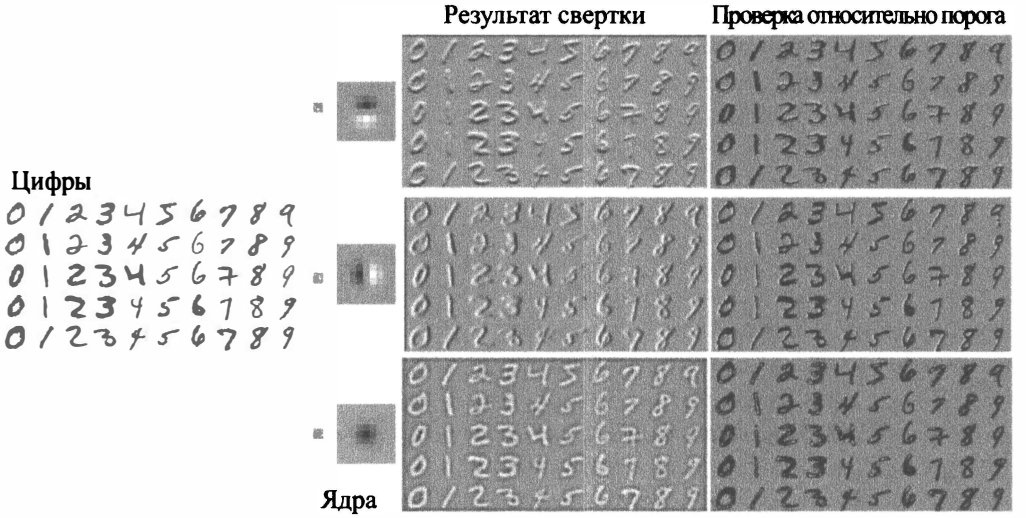


Рис. 25.12. Слева представлены некоторые изображения цифр из набора данных MNIST. Ближе к центру приведены три варианта ядер. Слева они показаны в реальном размере (маленькие блоки), а справа увеличены, чтобы подробнее представить их содержание: нейтральный серый цвет представляет нулевые, более светлый — положительные, а более темный — отрицательные значения. В правой части рисунка показаны результаты применения этих ядер к изображениям. В последнем столбце отмечены пиксели, где отклик больше, чем порог (светло-серый), или меньше, чем порог (темно-серый). Можно заметить, что было получено в результате (сверху вниз): детектор горизонтальной полосы; детектор вертикальной полосы и (труднее заметить) детектор окончания линии. Эти детекторы обращают внимание на контрастность полосы, поэтому (например) светлая сверху и темная снизу горизонтальная полоса вызывают положительный (более светлый) отклик, а темная сверху и светлая снизу дают отрицательный (более темный) отклик. Полученные детекторы относительно эффективны, но не идеальны

Хотя обучение сети CNN "из коробки" иногда приводит к нужному результату, будет полезно познакомиться с несколькими практическими приемами. Одним из наиболее важных является ▶**расширение набора данных**, когда отдельные

обучающие примеры многократно копируются и при этом слегка модифицируются. Например, можно случайным образом немного сместить, повернуть или растянуть изображение или, опять же случайным образом, несколько изменить цветовой оттенок части пикселей. Применение таких искусственных изменений точки обзора или характеристик освещения к набору данных позволяет увеличить его размер, хотя, безусловно, такие новые примеры будут сильно коррелировать с оригиналами. Также можно использовать прием расширения и в процессе тестирования сети, а не только ее обучения. При таком подходе изображение копируется и модифицируется несколько раз (например, посредством случайной обрезки) и процедура классификации выполняется для каждого измененного изображения. Выходы классификатора по каждому экземпляру затем используются для голосования в отношении окончательного решения по общему классу.

При классификации изображений сцен полезным может оказаться каждый пиксель. Но когда классифицируются изображения объектов, некоторые из пикселей не являются частью объекта и поэтому могут лишь отвлекать внимание. Например, если кошка лежит на собачьей лежанке, требуется, чтобы классификатор сконцентрировался на пикселях, относящихся к изображению кошки, а не лежанки. Современные классификаторы изображений хорошо с этим справляются, точно классифицируя изображение как "кошка", даже если саму кошку на изображении представляет фактически лишь несколько пикселей. Для этого есть две причины. Во-первых, системы классификации, построенные на базе сетей CNN, способны игнорировать шаблоны, которые не являются отличительными признаками. Во-вторых, шаблоны, которые лежат вне объекта, также могут быть отличительными признаками (например, игрушка для кошки, ошейник с маленьким колокольчиком или миска кошачьей еды может помочь нам понять, что мы смотрим на кошку). Этот эффект известен как ▶**контекст**. Контекст может как помочь, так и помешать, что довольно сильно зависит от конкретного набора данных и приложения.

25.5. Обнаружение объектов

Классификаторы изображений выдают прогноз по поводу того, *что* находится на изображении, они классифицируют все изображение как принадлежащее одному классу. Детекторы объектов отыскивают на изображении несколько объектов и сообщают, к *какому классу* относится каждый из них, а также *где* находится каждый объект, отображая вокруг объекта ▶**ограничивающую рамку**.[1] Множество возможных классов определяется заранее, поэтому можно попытаться обнаружить все лица, все машины или всех кошек.

[1] Мы будем использовать термин "рамка" для обозначения любой выровненной по осям прямоугольной области изображения, а термин "окно" — главным образом как синоним для "рамки", но с тем уточнением, что термин "окно" применяется ко входным данным и в нем предполагается что-то обнаружить, тогда как термин "рамка" — к выходным данным, когда что-то уже было обнаружено.

Детектор объектов можно построить, рассматривая большое изображение через небольшое прямоугольное ▶**скользящее окно**. В каждом местоположении окна то, что в нем присутствует, классифицируется с использованием классификатора на базе сети CNN. Затем классификации с высокими оценками (здесь — кошка, там — собака) принимаются, а содержимое всех остальных окон игнорируется. После некоторой работы по разрешению возможных конфликтов получаем окончательный набор объектов с их местоположением. Однако имеется еще несколько деталей, которые следует проработать.

- **Определить форму окна**. На данный момент самый простой выбор — использовать прямоугольники, выровненные по осям. (Альтернативный вариант — некоторая форма маски, вырезающей объект в изображении — едва ли когда-либо использовался по причине сложности его представления и связанных с этим вычислений.) Однако все еще требуется выбрать ширину и высоту этих прямоугольников.
- **Создать классификатор для окон.** Как это сделать с помощью сети CNN, нам уже известно.
- **Принять решение, какие окна будут анализироваться.** Из всех возможных окон желательно выбрать только те, которые, вероятно, содержат интересные объекты.
- **Выбрать окна, которые будут представлены в отчете.** Окна будут перекрываться, и нежелательно несколько раз упоминать в отчете один и тот же объект в разных, лишь немного различающихся окнах. Иногда некоторые объекты вообще не стоит упоминать, — представьте себе количество стульев и людей на изображении большого заполненного слушателями лекционного зала. Следует ли каждого из них рассматривать как отдельный объект? Может, следует поместить в отчет сведения только о тех объектах, которые кажутся большими на изображении — скажем, сидят в первом ряду. Правильный выбор зависит от предполагаемого назначения детектора объектов.
- **Используя эти окна, указать точное местоположение объектов.** Как только будет установлено, что объект находится где-то в данном окне, можно провести дополнительные вычисления, чтобы выяснить его точное местоположение в окне.

Давайте подробнее рассмотрим задачу принятия решения, какие окна будут анализироваться (рис. 25.13). Поиск среди всех возможных окон неэффективен — в изображении с $n \times n$ пикселями можно выделить $O(n^4)$ различных прямоугольных окон. Однако мы знаем, что окна, содержащие объекты, скорее всего, будут иметь пиксели с довольно схожими цветом и текстурой. С другой стороны, окна, отрезающие лишь часть объекта, будут включать области или края, выходящие за пределы этого окна с той или иной стороны. Следовательно, имеет смысл создать

механизм, оценивающий "объектность" — возможность наличия в ограничивающей рамке объекта, независимо от того, что это за предмет. Можно выполнить поиск ограничивающих рамок, которые будут выглядеть так, как будто объект в них присутствует, а затем выполнить классификацию объектов только для тех рамок, которые прошли тест на "объектность".

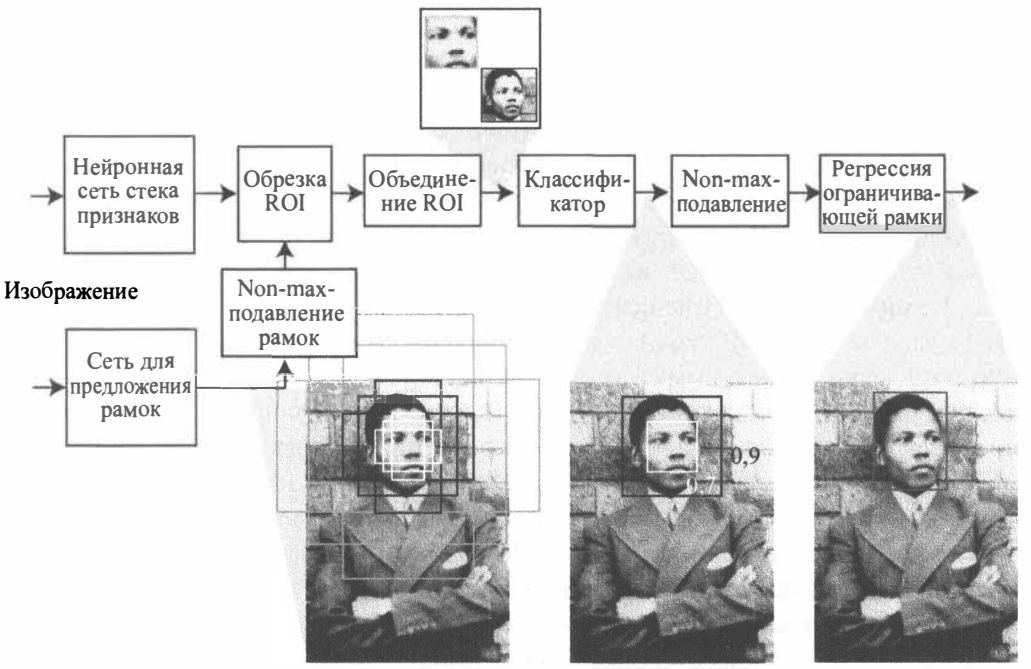


Рис. 25.13. В детекторе объектов класса Faster RCNN используются две сети. В детектор объектов введена фотография молодого Нельсона Манделы. Первая сеть вычисляет баллы "объектности" "кандидатных" ограничивающих рамок, называемых якорными рамками, центрированных в некоторой точке сетки. В каждой точке сетки применяется девять типов якорных рамок (три размера, три пропорции сторон) Для приведенного изображения тест на предметность прошли только внутренняя белая рамка и внешняя черная рамка. Вторая сеть представляет собой стек признаков и вычисляет представление изображения, подходящее для классификации. Рамки с самыми высокими оценками объектности вырезаются из этой карты признаков, стандартизируются в размерах посредством процедуры объединения ROI и передаются классификатору. Черная рамка имеет более высокую оценку, чем белая, и перекрывает ее, поэтому белая рамка отклоняется механизмом немаксимального подавления. Наконец, на этапе регрессии ограничивающей рамки черная рамка изменяется так, чтобы точно соответствовать лицу. Это необходимо для того, чтобы относительно грубый исходный выбор местоположения, размера и пропорции рамки не оказывал влияния на точность

Сеть, предназначенная для нахождения сегментов с объектами, называется ▶**сетью региональных предложений** (*regional proposal network* — ▶**RPN**). Создание детектора объектов по методу, известному как Faster RCNN, начинается с кодирования большого набора ограничительных рамок в виде карт фиксированного размера. Затем создается сеть, способная прогнозировать оценку для каждой рамки, которая обучается таким образом, чтобы давать высокую оценку, когда в рамке присутствует объект, и малую — в противном случае. Закодировать рамку в виде карты несложно. Рассматриваются рамки, центрированные в точках, расположенных в разных местах изображения. Нет необходимости рассматривать каждую возможную точку (поскольку перемещение на один пиксель не может изменить классификацию), хороший вариант — выбор центральных точек с **шагом** (смещением между центральными точками) 16 пикселей. Для каждой центральной точки рассматривается несколько возможных рамок, называемых **якорными рамками**. В методе Faster RCNN используется девять типов рамок: маленького, среднего и большого размеров, высокие, широкие и квадратные по пропорциям сторон.

В терминах архитектуры нейронных сетей строится трехмерный блок, в котором каждое пространственное местоположение имеет два измерения для центральной точки и одно измерение — для типа рамки. В результате любая рамка с достаточно хорошим показателем "объектности" классифицируется как **область интереса** (*region of interest* — ROI) и должна быть подвергнута проверке классификатором. Однако классификаторы CNN предпочитают изображения фиксированного размера, а рамки, направляемые на прохождение теста на "объектность", будут различаться по размеру и форме. Невозможно сделать так, чтобы все рамки включали одинаковое количество пикселей, но можно сделать так, чтобы они имели одинаковое количество признаков за счет выборки пикселей для извлечения признаков, — этот процесс называют "**объединение ROI**" (*ROI pooling*). Эта карта признаков фиксированного размера затем передается в классификатор.

Теперь пришло время принять решение о том, какие окна должны быть представлены в отчете. Предположим, что анализируются окна размером 32×32 пикселя с шагом 1: каждое окно смещено на один пиксель относительно предыдущего. В этом случае будет много похожих окон, которым, вероятно, будет присвоена одинаковая оценка. Если все они получат оценку, превышающую установленный порог, совсем не нужно представлять в отчете их все, потому что они, скорее всего, ссылаются на немного различающиеся виды одного и того же объекта. С другой стороны, если шаг будет слишком велик, может оказаться, что объект не присутствует ни в одном из окон и будет потерян. Чтобы избежать всего этого, можно использовать жадный алгоритм, получивший название ▶**немаксимального подавления** (*non-maximum suppression* — *NMS*). Сначала создается отсортированный список всех окон с оценкой, превышающей порог, а затем, пока в списке есть окна, выбирается окно с наибольшей оценкой, которое принимается как содержащее объект, после чего из списка исключаются все остальные окна, в значительной степени с ним перекрывающиеся.

Теперь осталось решить последнюю проблему — необходимо указать точное местоположение объектов. Предположим, что имеется окно с высокой оценкой, успешно прошедшее через механизм немаксимального подавления. Однако это окно едва ли окажется точно в нужном месте (напомним, что анализировалось сравнительно небольшое количество окон с ограниченным числом возможных размеров). Воспользуемся представлением признаков, вычисленным классификатором, для прогнозирования возможных улучшений, позволяющих обрезать окно до ограничивающей рамки правильных размеров, — этот этап называют ▶**регрессией ограничивающей рамки**.

Оценка детекторов объектов требует определенных усилий. Прежде всего необходимо иметь тестовый набор: коллекцию изображений, на которых каждый объект помечен меткой его истинной категории и ограничительной рамкой. Обычно рамки и метки в таких наборах проставляются людьми. Затем каждое изображение подается в детектор объектов и полученный отчет сравнивается с исходными пометками. Безусловно, следует принимать ограничивающие рамки, сдвинутые на несколько пикселей, — рамки в пометках также не являются идеальными. Значение выставляемой оценки должно сбалансировано учитывать воспроизведение (нахождение всех объектов, которые там имеются) и точность (отсутствие объектов, которых там нет).

25.6. Трехмерный мир

Изображения являются двухмерным представлением трехмерного мира. Но эти двухмерные изображения богаты подсказками о трехмерности мира. Один вид таких подсказок появляется, когда имеется несколько изображений одного и того же мира и можно сопоставить точки в разных изображениях. Другой вид сигналов о трехмерности мира может быть доступен в пределах одного изображения.

25.6.1. 3D-сигналы из нескольких видов

Два изображения объектов в трехмерном мире лучше, чем одно, — по нескольким причинам.

- Если есть два фото одной и той же сцены, снятые с разных точек зрения, и имеется необходимая информация о двух использованных для съемки камерах, можно построить 3D-модель — набор точек с указанием их координат в трех измерениях, — выяснив, какая точка на первом фото совпадает с соответствующей точкой на втором фото, и выполнив необходимые геометрические расчеты. Это утверждение справедливо почти для всех пар точек зрения и практически для всех видов камер.
- Если имеется два фото с достаточным количеством точек, о которых известно, какая точка на первом снимке совпадает с соответствующей точкой на втором снимке, для построения 3D-модели не потребуется много знать об

использованных камерах. Два вида двух точек дают четыре пары координат x, y, а чтобы определить точку в трехмерном пространстве, достаточно только трех координат. Дополнительная точка поможет понять то, что необходимо знать о фотокамерах. Это утверждение справедливо почти для всех пар направлений просмотра и практически для всех видов камер.

Ключевая проблема состоит в том, как установить, какая из точек на первом снимке отвечает соответствующей точке на втором снимке. Для сопоставления точек часто бывает достаточно детального описания локального внешнего вида точки на основе простых признаков текстур (таких, как в разделе 25.3.2). Например, в сцене уличного движения, в которой зеленый свет светофора на двух снимках может быть виден только в одном месте сцены, можно предположить, что эти элементы соответствуют друг другу. Геометрия серии фотоизображений очень точно проработана и детализирована (но, к сожалению, слишком сложна, чтобы привести ее здесь). Теория дает строгие геометрические ограничения на то, какая точка на одном изображении может соответствовать определенной точке на другом изображении. Другие ограничения можно получить, рассуждая о гладкости реконструированных поверхностей.

Существует два способа получить несколько видов одной сцены. При использовании первого способа необходимо иметь две камеры или два глаза (раздел 25.6.2). Второй способ — перемещение (раздел 25.6.3). Если имеется более чем два вида одной сцены, можно восстановить геометрию мира и все видимые детали очень точно. В разделе 25.7.3 обсуждаются некоторые практические приложения этой технологии.

25.6.2. Бинокулярные стереоданные

Большинство позвоночных имеют два глаза. Это не только удобно в качестве резерва на случай потери одного глаза, но и дает некоторые другие преимущества. У большинства травоядных глаза расположены по обе стороны головы, что обеспечивает им более широкий обзор. У хищников оба глаза направлены вперед, благодаря чему они получают ▶**бинокулярные стереоданные**. Закройте один глаз и держите оба направленных вверх указательных пальца перед лицом таким образом, чтобы один палец закрывал другой *в поле зрения открытого глаза*. Теперь проделайте то же самое с другим глазом: вы заметите, что пальцы оказались смещенными относительно друг друга. Такое смещение объекта при переходе от точки зрения слева к точке зрения справа называют ▶**рассогласованием**. При правильном выборе системы координат, если наложить друг на друга вид слева и вид справа на один и тот же объект, находящийся на некотором расстоянии, изображения этого объекта окажутся смещенным по горизонтали, причем величина этого смещения будет обратно пропорциональна расстоянию до объекта. Наглядно этот эффект представлен на рис. 25.14, где ближайшая точка пирамиды смещена влево на правом изображении и вправо — на левом изображении.

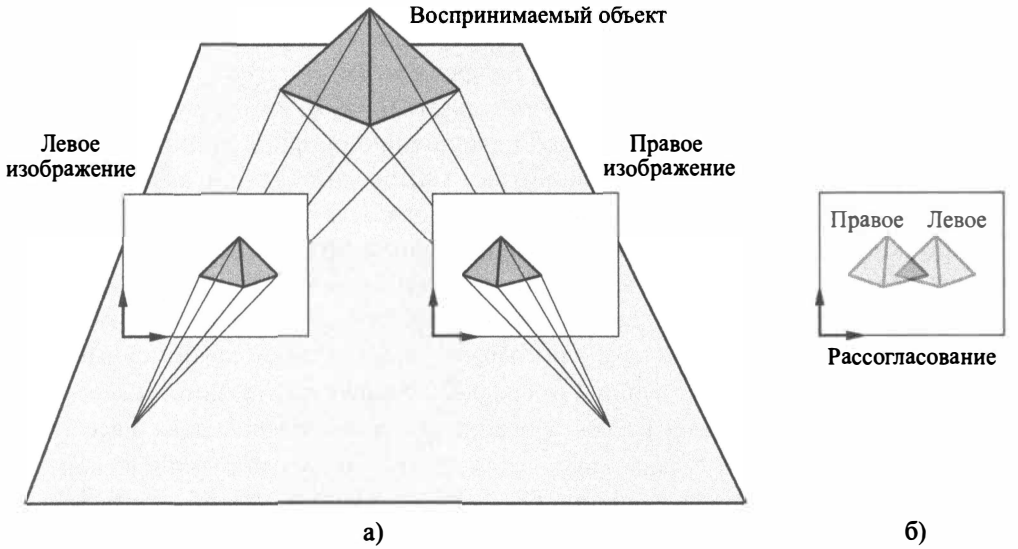


Рис. 25.14. а) Перемещение камеры параллельно плоскости изображения приводит к перемещению элементов изображения в плоскости камеры. Возникающее в результате несоответствие в позициях является признаком глубины. **б)** Рассогласование четко видно, если наложить друг на друга изображения, полученные с левой и правой точек зрения

Для измерения рассогласования сначала необходимо решить задачу соответствия — для точки на левом изображении определить ее "напарницу" на правом изображении, т.е. найти на нем точку, которая является результатом проекции той же точки сцены. Эта задача аналогична той, которая решается при измерении оптического потока, и большинство простых подходов к ее решению будут довольно схожими. Это методы поиска блоков соответствующих пикселей на левом и правом изображениях с использованием суммы квадратов различий (как в разделе 25.3.3). В более сложных методах применяется более детальное представление текстур блоков пикселей (как в разделе 25.3.2). На практике обычно используются гораздо более сложные алгоритмы, в которых вводятся дополнительные ограничения.

Если допустить, что есть возможность измерить рассогласование, то как из этого можно извлечь информацию о глубине в сцене? Прежде всего потребуется установить геометрические отношения между рассогласованием и глубиной. Сначала рассмотрим случай, когда оба глаза (или обе камеры) смотрят вперед и их оптические оси параллельны. Тогда геометрическое отношение между правой и левой камерами можно представить как простое смещение по оси x на величину b, так называемую ►**опорную линию**. Можно воспользоваться уравнениями для

оптического потока из раздела 25.3.3, если рассматривать ситуацию как результат переноса вектора **T**, выполненного за время δt, тогда $T_x = b/\delta t$ и $T_y = T_z = 0$. Величина рассогласования по горизонтали и по вертикали определяется составляющими оптического потока, умноженными на время δt, т.е. $H = v_x\delta_t$, $V = v_y\delta_t$. Выполнив подстановку, получим окончательный результат: $H = b/Z$, $V = 0$. Другими словами, горизонтальное рассогласование равно отношению опорной линии к глубине, а вертикальное рассогласование равно нулю. Теперь можно вычислить глубину Z, поскольку значение b известно, а значение H можно измерить.

При нормальных условиях люди обычно ▶**фиксируют** свой взгляд, — это означает, что они выбирают в качестве объекта наблюдения некоторую точку в сцене и оптические оси обоих глаз пересекаются в этой точке. На рис. 25.15 показана ситуация, в которой взгляд человека, смотрящего двумя глазами, зафиксирован в точке P_0, находящейся на расстоянии Z от средней точки между глазами. Для удобства мы будем вычислять *угловое* рассогласование, измеряемое в радианах. Рассогласование в точке фиксации P_0 равно нулю. Для некоторой другой точки P в сцене, которая находится дальше на величину δZ, можно вычислить угловые смещения левого и правого изображений точки P, которые обозначим соответственно как P_L и P_R. Если каждое из этих изображений смещено на угол $\delta\theta/2$ относительно P_0, то смещение между P_L и P_R, которое представляет собой рассогласование в точке P, выражается просто как $\delta\theta$. Из рис. 25.15 следует, что $\operatorname{tg}\theta = \dfrac{b/2}{Z}$ и $\operatorname{tg}(\theta - \delta\theta/2) = \dfrac{b/2}{Z+\delta Z}$, но для малых углов $\operatorname{tg}\theta \approx \theta$, поэтому

$$\delta\theta/2 = \frac{b/2}{Z} - \frac{b/2}{Z+\delta Z} \approx \frac{b\delta Z}{2Z^2},$$

и поскольку фактическое рассогласование составляет $\delta\theta$, получим

$$\text{Рассогласование} = \frac{b\delta Z}{Z^2}.$$

У людей величина базисной линии b составляет примерно 6 см. Предположим, что расстояние Z равно приблизительно 100 см. Тогда наименьшее различимое значение $\delta\theta$ (соответствующее размеру светочувствительного элемента глаза, т.е. его одному пикселю) составит около 5 угловых секунд, что дает разность расстояний δZ примерно 0,4 мм. При $Z = 30$ см получим на удивление малое значение: $Z = 0{,}036$ мм. Это означает, что на расстоянии 30 см люди способны различать значения глубины, которые отличаются друг от друга на столь малую величину, как 0,036 мм, и это позволяет им продевать нитку через игольное ушко и выполнять другие тонкие операции.

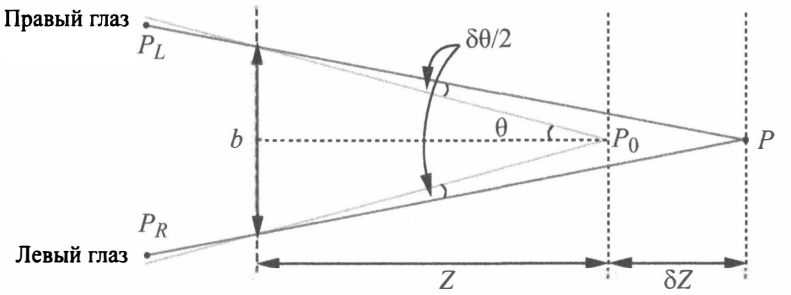


Рис. 25.15. Соотношение между рассогласованием и глубиной в стереоданных. Центры проекций двух глаз находятся на расстоянии b друг от друга, а оптические оси пересекаются в точке фиксации взгляда P_0. Точка P в сцене проецируется в глазах на точки P_L и P_R. Представленное в виде углов рассогласование между ними составляет $\delta\theta$ (на диаграмме показаны два угла по $\delta\theta/2$)

25.6.3. 3D-сигналы от движущейся камеры

Предположим, что имеется камера, движущаяся в сцене. Вернемся к рис. 25.14 и пометим левое изображение как "время t", а правое изображение — как "время $t+1$". Геометрия не изменилась, поэтому все рассуждения в ходе анализа стереоданных можно полностью распространить и на случай, когда камера движется. То, что выше называлось рассогласованием, в этом разделе воспринимается как видимое движение на изображении, называемое оптическим потоком. Этот оптический поток является источником информации как о движении камеры, так и о геометрии сцены. Чтобы понять это утверждение, запишем (без доказательства) уравнение, связывающее оптический поток с поступательной скоростью наблюдателя **T** и глубиной в сцене.

Поле оптического потока представляет собой векторное поле скоростей в изображении $(v_x(x,y), v_y(x,y))$. Выражения для этих компонент в системе координат, центрированной на камере, и в предположении, что ее фокусное расстояние $f=1$, будут иметь вид

$$v(x,y)=\frac{-T_x+xT_z}{Z(x,y)} \quad \text{и} \quad v_y(x,y)=\frac{-T_y+yT_z}{Z(x,y)},$$

где $Z(x,y)$ — координата z (т.е. глубина) точки в сцене, соответствующей точке на изображении в (x,y).

Обратите внимание, что обе компоненты оптического потока, $v_x(x,y)$ и $v_y(x,y)$, равны нулю в точке $x=T_x/T_z$, $y=T_y/T_z$. Эта точка называется ▶**фокусом расширения** поля оптического потока. Предположим, что начало координат в плоскости

x–y было смещено так, чтобы оно оказалось в фокусе расширения. В этом случае выражения для оптического потока принимают особенно простую форму. Пусть (x', y') — новые координаты, определенные как $x' = x - T_x/T_z$, $y' = y - T_y/T_z$. Тогда

$$v_x(x', y') = \frac{x'T_z}{Z(x', y')}, \qquad v_y(x', y') = \frac{y'T_z}{Z(x', y')}.$$

Обратите внимание, что здесь имеет место неоднозначность масштабного коэффициента (вот почему было принято предположение, что фокусное расстояние $f = 1$ — это безопасно). Если камера будет двигаться вдвое быстрее, а каждый объект на сцене будет в два раза больше и вдвое дальше от камеры, то поле оптического потока останется точно таким же. Однако у нас все еще остается возможность извлечь достаточно полезную информацию.

1. Предположим, что вы — муха, пытающаяся сесть на стену, и вам нужно получить полезную информацию от поля оптического потока. Однако поле оптического потока не может указать величину расстояния до стены или скорость вашего движения к стене из-за неоднозначности шкалы. Но если разделить расстояние на скорость, неопределенность шкалы уже не будет иметь значения: результатом будет время до контакта со стеной, определяемое как Z/T_z, и это действительно полезная информация при совершении посадки. Существуют надежные экспериментальные доказательства, что множество разных видов животных действительно использует этот сигнал.
2. Рассмотрим две точки на глубине Z_1 и Z_2 соответственно. Можно не знать абсолютного значения любой из этих величин, но, принимая во внимание обратное отношение величины оптического потока в этих точках, можно определить отношение глубин Z_1/Z_2. Это сигнал параллакса движения, который мы используем, когда смотрим из бокового окна движущейся автомашины или поезда и делаем вывод, что более медленно перемещающиеся элементы ландшафта находятся дальше от нас.

25.6.4. 3D-сигналы от единственного вида

Даже единственное изображение предоставляет немало информации о трехмерном мире. Это так, даже если изображение представляет собой просто карандашный набросок. Такие рисунки, выполненные простыми линиями, очаровали изучающих зрение ученых, потому что у людей есть способность создавать ощущение трехмерности форм и сцены, хотя сам их рисунок, казалось бы, содержит лишь совсем небольшую часть информации, доступной в обширном множестве сцен, которые могут быть представлены одним и тем же рисунком. Одним из ключевых источников информации является окклюзия: если на изображении есть доказательства того, что один объект закрывает другой, то закрывающий объект находится ближе к глазам.

В изображениях реальных сцен сильным признаком трехмерности структуры является текстура. В разделе 25.3.2 объяснялось, что текстура — это повторяющийся узор из текселей. И хотя распределение текселей на объектах в сцене может быть равномерным — например, галька на пляже, — на изображении они могут казаться неоднородными; галька в более удаленной части пляжа будет казаться мельче, чем галька на его ближних участках. Другим хорошим примером может быть кусок ткани с узором в горошек. На самой ткани все точки имеют одинаковую форму и размер, но на изображении за счет перспективы некоторые точки, видимые под другим ракурсом, будут представлены как эллипсы. В современных методах подобные сигналы используются посредством *обучения* системы отображению изображений в трехмерную структуру (раздел 25.7.4) вместо проведения непосредственных рассуждений о базовой математике текстуры.

Затенение, т.е. изменение интенсивности света, поступающего от разных участков поверхности в сцене, определяется геометрией сцены и отражательными свойствами поверхности. Есть очень убедительные доказательства того, что затенение также является признаком для трехмерных форм. Физический аргумент прост. Из физической модели, обсуждавшейся в разделе 25.2.4, следует, что, если нормаль к поверхности направлена на источник света, поверхность будет выглядеть ярче, а если она отклоняется от него, то темнее. К сожалению, эти рассуждения усложняются, когда коэффициент отражения поверхности неизвестен, а поле освещения неровное, но человек, кажется, способен извлекать полезные сведения из восприятия затенения. К сожалению, пока еще слишком мало известно об алгоритмах того, как это работает.

Если на изображении есть знакомый объект, его вид очень сильно зависит от его видимого ▶**положения** (*pose*), т.е. от его позиции и ориентации по отношению к зрителю. Существуют прямые алгоритмы восстановления положения объекта в пространстве на основании соответствий между точками на изображении объекта и соответствующими точками на модели объекта. Восстановление положения известного объекта имеет множество применений на практике. Например, в задаче управления манипуляциями промышленных роботов манипулятор робота не сможет захватить требуемый предмет, пока не будет установлено его положение в пространстве. Применение роботизированной хирургии зависит от точности вычислений изменений в положении камеры, хирургического инструмента и пациента (для определения трансформаций в расположении инструмента относительно положения пациента).

Пространственные отношения между объектами являются еще одним важным сигналом. Приведем такой пример. Все пешеходы примерно одинаковой высоты, и они, как правило, стоят на земле. Тогда, если известно, где на изображении находится линия горизонта, можно ранжировать пешеходов по расстоянию до камеры. Идея состоит в том, что когда известно, где находятся ноги пешеходов, то те, чьи ноги на изображении ближе к линии горизонта, будут находиться дальше от камеры, и поэтому их изображения должны быть меньше по размеру. Эти рассуждения

означают, что есть возможность исключить некоторые сигналы от соответствующего детектора: если на изображении будет обнаружен большой по размеру пешеход, ноги которого находятся близко к линии горизонта, то это указывает на обнаружение огромного пешехода, которого просто не может существовать, а значит, детектор в чем-то ошибся. В свою очередь, достаточно надежные сигналы об обнаружении пешеходов позволяют оценить расположение линии горизонта, если в сцене они находятся на разных расстояниях от камеры, поскольку относительное масштабирование изображений пешеходов — это надежный сигнал в отношении того, где проходит линия горизонта. Следовательно, можно извлечь оценку текущего положения линии горизонта на основании данных детектора пешеходов, а затем использовать эту оценку для отбрасывания ошибочных свидетельств этого детектора.

25.7. Использование компьютерного зрения

В этом разделе обсуждается ряд практических приложений компьютерного зрения. В настоящее время в области компьютерного зрения уже существует много надежных готовых инструментов и соответствующего инструментария для разработчиков, поэтому диапазон успешных и полезных приложений чрезвычайно широк. Многие из этих приложений были разработаны энтузиастами в домашних условиях для специальных целей, что является свидетельством того, насколько полезны эти методы и какое значение они имеют. (Например, один энтузиаст создал великолепную дверь для животных, работа которой основана на обнаружении объектов: можно настроить ее так, что она не позволит войти кошке, если та несет в зубах мертвую мышь, — веб-поиск поможет вам найти подходящие изображения.)

25.7.1. Понимание того, что делают люди

Если бы удалось создать систему, понимающую, что делают люди посредством анализа видео, можно было бы создать интерфейс "человек-компьютер", обеспечивающий системам наблюдение за людьми и соответствующую реакцию на их поведение. При наличии таких интерфейсов можно было бы реализовать очень многое: улучшить проекты зданий и общественных мест за счет сбора и использования информации о том, что люди делают на публике; построить более точные и менее навязчивые системы наблюдения за безопасностью; создать автоматизированных спортивных комментаторов; обеспечить дополнительную безопасность на строительных площадках и рабочих местах за счет выдачи предупреждений в тех случаях, когда люди и машины находятся в опасной близости; разработать компьютерные игры, заставляющие игрока вставать и двигаться; добиться экономии энергии, за счет управления обогревом и освещением здании с учетом того, где находятся их обитатели и что они делают.

Уровень достижений в решении некоторых проблем из этой области сейчас чрезвычайно высок. Существуют методы, обеспечивающие очень точное определение положения суставов человека на изображении, что позволяет довольно хорошо оценить реальную трехмерную конфигурацию его тела (рис. 25.16). Эти методы построены на том факте, что изображения тела человека включают слабые эффекты перспективы, а длина самих сегментов тела при их перемещении изменяется незначительно, поэтому ракурс сегмента тела на изображении является хорошим признаком для определения угла между ним и плоскостью камеры. При наличии датчика глубины эти оценки можно делать достаточно быстро, чтобы имело смысл включить их в интерфейсы компьютерных игр.

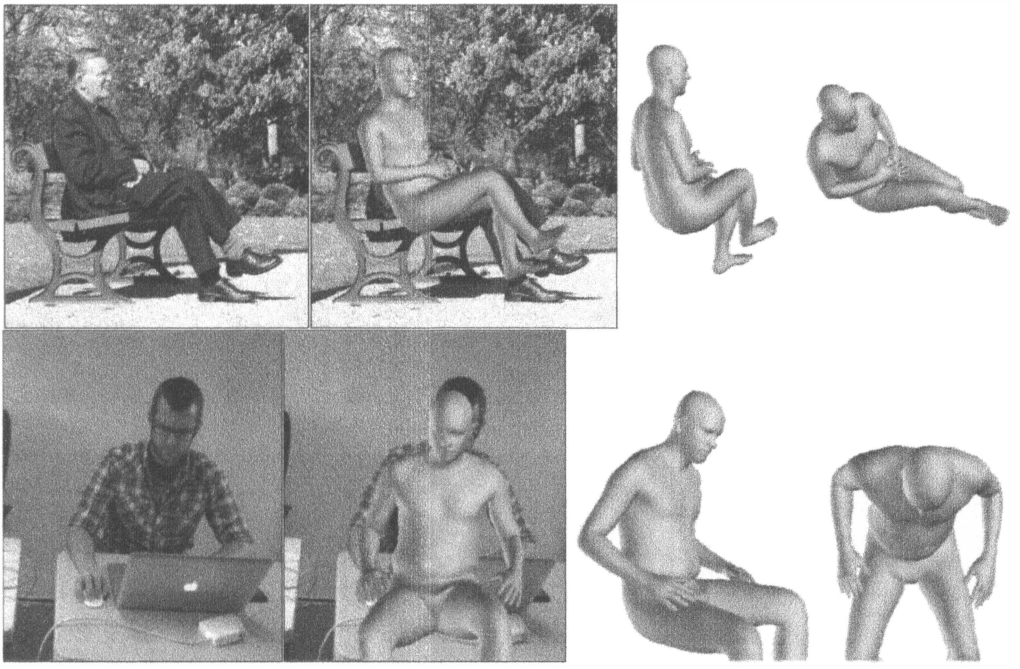
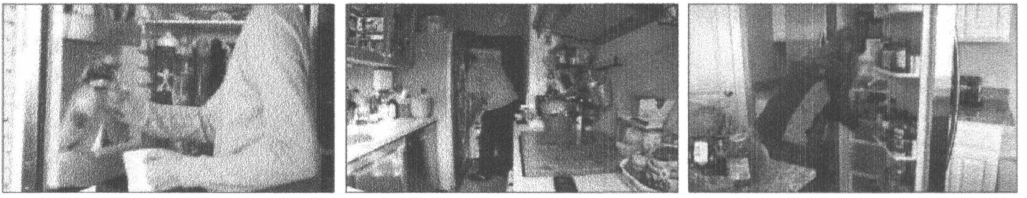


Рис. 25.16. Реконструкция положения тела человека по одному изображению теперь вполне доступна — на этом рисунке представлены две реконструкции трехмерных фигур, полученные с использованием единственного изображения. Это стало возможным, потому что используемые методы позволяют оценить местоположение суставов, углы сочленения суставов в трехмерном пространстве, форму тела и его положение по отношению к плоскости изображения. **Слева** показаны исходные изображения, в **центре** — они же, но с наложенными изображениями восстановленного тела, а **справа** — по две другие проекции реконструированного тела. Демонстрация разных проекций затрудняет сокрытие ошибок в реконструкции

Классифицировать то, что делают люди, сложнее. Видео, на котором представлено достаточно структурированное поведение, такое как балет или гимнастика тай-чи, в которой есть довольно специфические "словари", ссылающиеся на очень точно очерченные действия на простом фоне, распознать достаточно просто. Хорошие результаты можно получить при доступности большого количества помеченных данных и соответствующей сверточной нейронной сети. Однако в конечном счете может оказаться трудно доказать, что методы действительно работают, поскольку они очень сильно полагаются на контекст. Например, классификатор, очень хорошо отмечающий "плавательные" последовательности движений, в действительности может оказаться просто детектором плавательного бассейна, и совсем не будет работать, скажем, для пловца в реке.

Более общие задачи все еще остаются открытыми, например как связать наблюдения за телом и расположенные рядом объекты с целями и намерениями движущихся людей. Один из источников затруднений состоит в том, что схожее по цели поведение может выглядеть по-разному, а разные линии поведения могут быть очень схожими, как на рис. 25.17.

Открытие холодильника



Извлечение чего-либо из холодильника

Рис. 25.17. Одно и то же действие может выглядеть по-разному, а разные действия могут выглядеть почти одинаково. Приведенные здесь примеры показывают действия, взятые из набора изображений о естественном поведении, причем метки к ним были выбраны создателями набора данных, а не предложены алгоритмом классификации. Для изображений **верхнего ряда** метка — "открытие холодильника", при этом некоторые из них сделаны крупным планом, а другие — издалека. Для изображений **нижнего ряда** метка — "извлечение чего-либо из холодильника". Обратите внимание, что на изображениях в обоих рядах рука субъекта всегда находится близко к двери холодильника, — для того, чтобы понять разницу между выполняемыми действиями, необходимо достаточно тонкое суждение о том, где расположена рука и где расположена дверь

Другая сложность связана с масштабом времени. То, что некто делает на изображении, довольно сильно зависит от шкалы времени, как показано на рис. 25.18. Еще одним важным эффектом, представленным на этом рисунке, является то, что поведение является составным — несколько распознанных вариантов поведения может быть объединено в единое поведение более высокого уровня, такое как "легкий перекус".

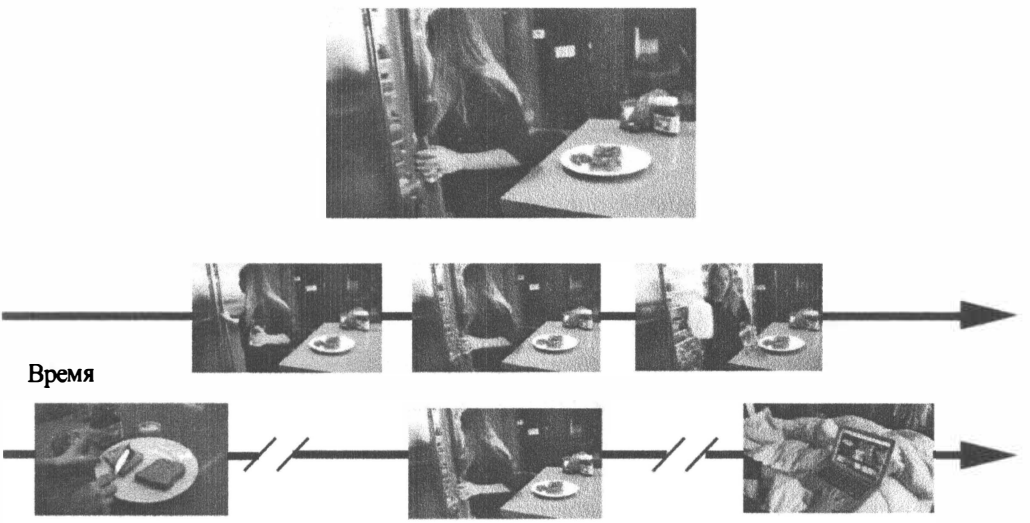


Рис. 25.18. То, что мы называем действием, зависит от масштаба времени. Единственный кадр **вверху** рисунка лучше всего описать как "открытие холодильника" (закрывая холодильник, мы обычно не смотрим на его содержимое). Но если обратиться к короткому видеоклипу (три кадра **в центре** рисунка), то выполняемое действие лучше всего можно будет определить как извлечение молока из холодильника. Что касается более продолжительного видеоклипа (три кадра **внизу** рисунка), то действие в нем лучше всего описать как легкий перекус. Обратите внимание, что все это иллюстрирует одну из особенностей формирования составного поведения: извлечение молока из холодильника *иногда* является частью легкого перекуса, а открытие холодильника *обычно* является частью извлечения молока из холодильника

Может также иметь место ситуация, когда в одно и то же время выполняется два несвязанных варианта поведения, например напевать песенку, доставая молоко для перекуса. Проблема в том, что не существует общепринятого "словаря" для элементов поведения. Люди склонны думать, что они знают много названий разных видов поведения, однако оказываются неспособными составить длинный список таких названий, если попросить их об этом. Все это затрудняет подготовку наборов данных с согласованными пометками образцов поведения.

Обученные системы классификации гарантированно будут вести себя хорошо только в том случае, если обучающий и тестовый наборы будут иметь одно и то же распределение. У нас нет возможности проверить выполнение этого ограничения применительно к изображениям, но опытным путем было установлено, что классификаторы изображений и детекторы объектов работают очень хорошо. Однако в отношении данных о деятельности человека связь между обучающими наборами и тестовыми наборами данных менее надежна, потому что люди способны делать очень много разных вещей во многих контекстах. Например, предположим, что у нас есть детектор пешеходов, хорошо работающий на большом наборе данных. Вполне могут существовать редкие ситуации (например, люди, сидящие на одноколесных велосипедах), которые отсутствуют в обучающем наборе, поэтому нельзя сказать наверняка, как детектор поведет себя в таких случаях. Проблема состоит в том, что необходимо доказать, что этот детектор безопасен, что бы ни делали, как бы ни вели себя пешеходы, что очень сложно для современных теорий машинного обучения.

25.7.2. Связывание картинок и слов

Многие люди создают фотографии и видеоролики, а затем размещают их в Интернете и обмениваются ими. Сложность заключается в отыскании того, что хотелось бы увидеть. Как правило, люди склонны искать что-либо, используя словесные описания (а не, скажем, эскизы). Поскольку большинство изображений выкладывается без словесных описаний их содержания, вполне естественно попытаться создать ▶**системы присвоения тегов**, предназначенные для пометки изображений соответствующими словесными маркерами. Выбор базового механизма вполне очевиден: использование систем классификации изображений и распознавания объектов с последующим присвоением изображению соответствующих словесных характеристик. Однако присвоенные таким образом теги нельзя считать исчерпывающим описанием того, что происходит на изображении. Важно указать, кто и что делает, но теги не фиксируют этого. Например, пометка изображения кота на улице тегами категорий объектов “кошка”, “улица”, “мусорный бак” и “рыбьи кости” не дает информации, что на изображении кошка вытаскивает рыбьи кости из открытого мусорного бака, стоящего на улице.

В качестве альтернативы методу присвоения тегов можно предложить разработать ▶**систему генерации подписей** (*captioning systems*) к изображениям, т.е. такую систему, которая создаст заголовок (или подпись) из одного или нескольких предложений, описывающих изображение. Выбор основного механизма здесь также вполне очевиден — связать сверточную нейронную сеть (для представления изображения) с рекуррентной нейронной сетью или сетью с архитектурой “трансформер” (для генерации предложений) и обучить полученную в результате систему на основе обучающего набора данных из подписанных изображений. В Интернете доступно достаточно много изображений с подписями, — создатели таких

наборов данных используют человеческий труд для дополнения каждого изображения соответствующими подписями, позволяющими захватить вариации в естественном языке. Например, набор данных COCO (*Common Objects in Context*) — это обширная коллекция более чем из 200 тыс. изображений с пятью заголовками (подписями) к каждому изображению.

Современные методы генерации подписей предполагают использование детекторов объектов для поиска набора слов, описывающих изображение, и передачу этого набора в модель класса от последовательности к последовательности, обученную генерации предложений. Наиболее точные методы дополнительно включают поиск среди предложений, которые модель способна сгенерировать, для отыскания наилучших, но чем строже применяемые критерии, тем медленнее будет выполняться поиск. Предложения оцениваются по сумме баллов, при простановке которых учитывается, во-первых, используются ли в сгенерированном предложении фразы, чаще всего встречающиеся в истинных аннотациях, и во-вторых, не используются ли в нем другие фразы. Эти оценки едва ли можно рассматривать непосредственно как функцию потерь, однако для обучения сетей, получающих очень хорошие оценки, могут использоваться методы обучения с подкреплением. Часто в обучающем наборе присутствует изображение, описание которого включает тот же набор слов, что и изображение в тестовом наборе, и в этом случае система генерации подписей может просто извлекать допустимую подпись, а не создавать новую. Системы генерации подписей способны одновременно показывать отличные результаты и совершать нелепые ошибки (рис. 25.19).

Ребенок ест кускок еды во рту

Маленький мальчик ест кусок торта

Маленькая птичка, сидящая на ветке

Маленький коричневый медведь сидит в траве

Рис. 25.19. Автоматические системы генерации подписей к изображениям могут давать как хорошие результаты, так и ошибочные. Две подписи слева правильно описывают соответствующие изображения, хотя сочетание "ест..." и "во рту" — это довольно типичная ошибка, характерная для языка моделей CNN, использовавшихся в ранних системах генерации подписей. Судя по подписям для двух изображений справа, системе, видимо, ничего неизвестно о существовании белок, поэтому она пытается угадать изображенное животное по окружающему контексту, а также не в состоянии распознать, что обе белки едят

Системы генерации подписей могут скрывать свое невежество, не упоминая детали, которые не могут правильно распознать, или используя контекстные подсказки, чтобы попробовать угадать. Например, системы генерации подписей склонны к ошибкам в идентификации пола людей на изображениях и часто пытаются правильно его угадать, следуя статистическим показателям из обучающих данных. Но это часто может приводить к ошибкам — мужчинам тоже нравится делать покупки, а женщинам — кататься на сноуборде. Один из способов выяснить, имеет ли система правильное представление о том, что показано на изображении, — заставить ее давать ответы на вопросы об изображении. Такой вариант называется системой ▶**ответа на визуальный вопрос** (*visual question answering* — ▶**VQA**). Альтернативой может быть система ▶ **визуального диалога**, в которой задаются изображение, подпись к нему и некоторый диалог. Получив эти данные, система должна ответить на последний вопрос в предоставленном диалоге. Как показано на рис. 25.20, задача зрительного анализа остается чрезвычайно сложной, и системы VQA часто допускают ошибки.

Рис. 25.20. Системы ответа на визуальные вопросы выдают ответы (обычно выбираемые из предоставленного множества вариантов) на вопросы об изображениях, заданные на естественном языке. Для изображений в **верхнем** ряду система предоставила разумные ответы на довольно сложные вопросы об изображениях. Для изображений в **нижнем** ряду были предоставлены не столь удовлетворительные ответы. Так, система пытается угадать количество отверстий в пицце, потому что не понимает, что следует считать отверстием, и испытывает затруднения при подсчете. Аналогично система выбирает коричневый цвет для лапы кошки, потому что фон коричневый, а ей не удалось правильно локализовать правую переднюю лапу

25.7.3. Реконструкция из нескольких видов

Задача восстановления набора точек из нескольких видов, которые могут быть получены из видео или из подборки туристических фотографий, похожа на реконструкцию точек из двух точек зрения, но здесь есть некоторые важные различия. Потребуется проделать гораздо больше работы для установления соответствия между точками в разных видах, при этом точки могут входить в поле зрения и выходить из него, что делает процесс согласования и восстановления еще сложнее. Но большее количество видов означает увеличение количества ограничений на реконструкцию и восстановленные параметры просмотра, поэтому, как правило, удается выполнить очень точную оценку как положения точек, так и параметров их наблюдения. Грубо говоря, реконструкция выполняется путем сопоставления точек по парам изображений с расширением этих совпадений до групп изображений, приводя в конечном итоге к приблизительному решению в отношении как геометрии, так и параметров наблюдения с последующим уточнением этого решения. Уточнение означает минимизацию ошибки между расположением точек, предсказанным моделями (геометрии и параметров просмотра), и местом расположения признаков на изображении. Подробные процедуры слишком сложны, чтобы рассмотреть их здесь полностью, но в настоящее время они уже очень хорошо проработаны и достаточно надежны.

Все геометрические ограничения на соответствия известны для любой потенциально полезной формы камеры. Процедуры могут быть обобщены для работы с представлениями, которые не являются ортографическими, для работы с точками, которые доступны для наблюдения только в некоторых видах, и для работы с неизвестными параметрами камеры (например, фокусным расстоянием), а также допускают применение различных сложных методов поиска возможных соответствий. Вполне реально точно реконструировать модель всего города на основании большого числа его различных изображений. Вот некоторые варианты применения этих методов.

- **Построение модели.** Например, можно построить систему моделирования, получающую на входе множество описывающих объект изображений и создающую максимально детальную трехмерную сетку из текстурированных многоугольников, предназначенную для применения в компьютерной графике и приложениях виртуальной реальности. Обычной практикой является построение подобных моделей на базе видеозаписей, но теперь такие модели могут быть построены и на основе совершенно случайных наборов картинок. Например, можно построить 3D-модель Статуи Свободы на базе изображений, найденных в Интернете.
- **Смешение анимации с реальными актерами в видео.** Для вставки персонажей компьютерной графики в реальное видео необходимо знать, как двигалась камера при съемке реального видео, — это позволит правильно рендерить персонаж, меняя вид в соответствии с движением камеры.

- **Восстановление пути.** Мобильным роботам необходимо знать, где они находились ранее. Если робот имеет камеру, можно построить модель пути камеры в мире, что и будет служить представлением пути робота.
- **Управление строительством.** Здания представляют собой чрезвычайно сложные объекты, и отслеживать все происходящее в процессе строительства сложно и дорого. Один из способов такого слежения состоит в том, чтобы раз в неделю совершать полет над стройплощадкой, снимая текущее состояние дел. Затем на основании этих данных строится трехмерная модель реального состояния объекта и анализируются различия, существующие между планами и достигнутыми результатами, с использованием различных методов визуализации. На рис. 25.21 иллюстрируется работа этого приложения.

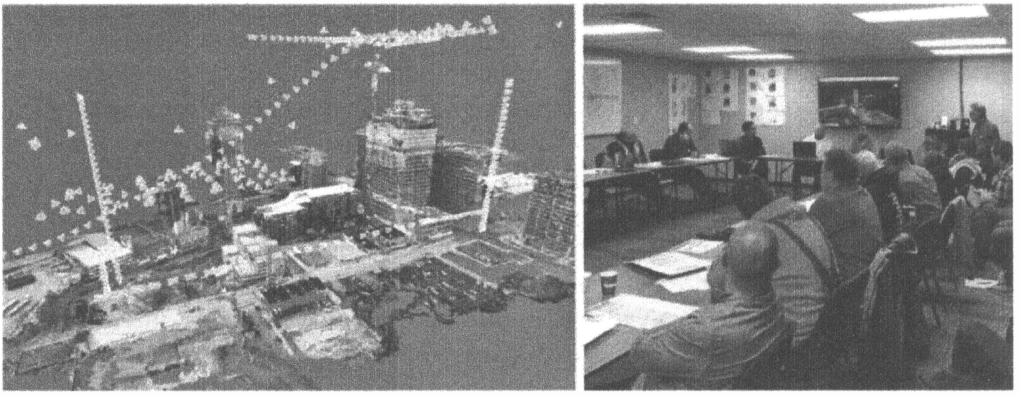


Рис. 25.21. Трехмерные модели строительных площадок создаются на основе изображений с использованием алгоритмов построения структуры на основе движения и многовидового стерео. Такие модели помогают строительным компаниям координировать работы при строительстве крупных объектов, сравнивая трехмерную модель фактического состояния объекта на текущий момент с планами его строительства. **Слева**: визуализация геометрической модели, построенной на основании съемки объекта дронами. Реконструированные 3D-точки отображаются в цвете, благодаря чему последние достигнутые результаты представляются как прогресс на текущий момент (обратите внимание на частично построенное здание с краном). Маленькие светлые пирамиды показывают положение дрона при выполнении съемки в этой точке и позволяют визуализировать траекторию его полета. **Справа**: такие системы уже реально используются строительными организациями: эта группа анализирует текущую модель строительной площадки и сравнивает ее с планами строительства в рамках координационного совещания

25.7.4. Геометрические представления из единственного вида

Геометрические представления особенно полезны, если предполагается движение, поскольку они позволяют выяснить, где агент находится, куда он может переместиться и с чем он может столкнуться. Но не всегда есть возможность использовать несколько видов для построения геометрической модели. Например, когда вы открываете дверь и делаете шаг в комнату, ваши глаза находятся слишком близко друг к другу, чтобы составить хорошее представление о расстоянии до отдаленных объектов в противоположном конце комнаты. Можно, конечно, подвигать головой взад и вперед, но это потребует некоторого времени да и просто неудобно.

Альтернативой является прогнозирование ▶ **карты глубины** — массива, приписывающего глубину каждому пикселю в изображении, номинально исходя из данных с камеры, т.е. из одного изображения. Для многих типов сцен можно на удивление легко выполнить это точно, поскольку карта глубины имеет довольно простую структуру. Это замечание особенно верно для комнат и вообще различных внутренних сцен. Механика очень проста. Создается набор данных из изображений и карт глубины, а затем сеть обучается прогнозировать карту глубины на основе изображений. Так можно решить множество различных интересных вариантов этой задачи. Основная проблема, связанная с использованием карты глубины, заключается в том, что эта карта не дает никакой информации об обратных сторонах объектов и находящемся за ними пространстве. Но есть методы, позволяющие предсказать, какие ▶ **воксели** (3D-пиксели) заняты известными объектами (т.е. объектами, геометрия которых известна) и как будет выглядеть карта глубины, если некоторый объект удалить. Эти методы работают, потому что формы объектов довольно сильно стилизованы.

Как было показано в разделе 25.6.4, восстановление положения известного объекта с помощью трехмерной модели не представляет сложности. А теперь представим, что имеется только одно изображение — скажем, воробья. Если вы уже видели много изображений птиц, подобных воробью, то вполне сможете как дать разумную оценку положения воробья, так и построить его геометрическую модель, исходя из этого единственного изображения. Опираясь на виденные ранее изображения, вы построите небольшое параметрическое семейство геометрических моделей похожих на воробья птиц, а затем примените процедуру оптимизации, чтобы найти наилучшее множество параметров и точек зрения для объяснения изображения, которое видите. Этот подход работает и в отношении предоставления текстуры для этой модели, и в отношении даже тех ее частей, которые на изображении не видны (рис. 25.22).

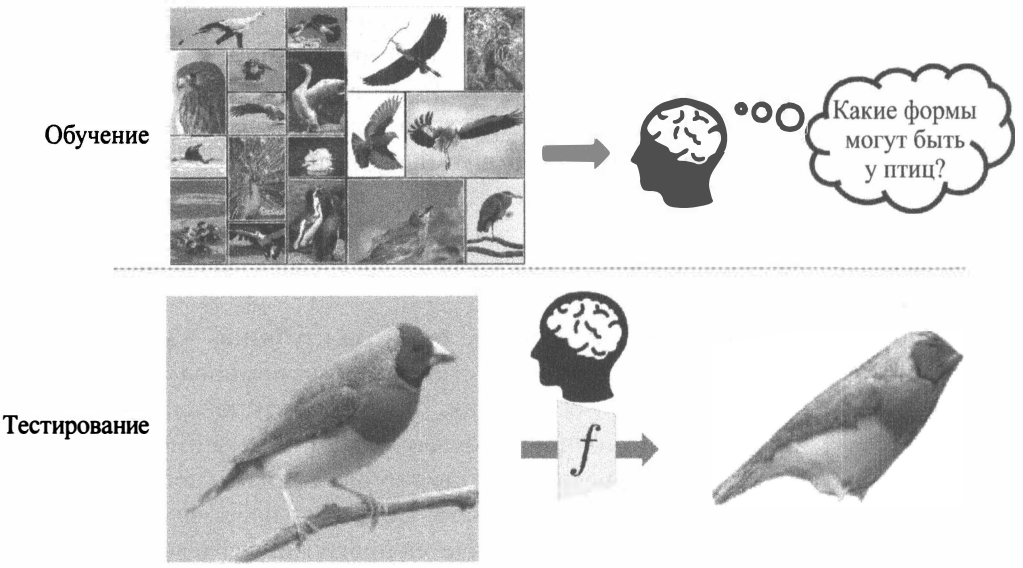


Рис. 25.22. Если вы видели много картинок, относящихся к некоторой категории — скажем, птиц (**вверху**), — то сможете использовать их для создания трехмерной реконструкции на основе единственного нового изображения (**внизу**). Сначала необходимо убедиться в том, что все объекты имеют довольно схожую геометрию (поэтому изображение страуса не пригодится, если анализируется изображение воробья), и методы классификации помогут в этом разобраться. На основе многих изображений можно оценить, как элементы текстуры в изображении распределены по объекту, и, таким образом, подобрать текстуру даже для тех частей птицы, которые не видны (внизу)

25.7.5. Создание изображений

В настоящее время можно довольно легко вставлять модели компьютерной графики в фотографии, причем весьма убедительным образом, — как показано на рис. 25.23, на котором изображение статуэтки было вставлено в фотографию комнаты. Прежде всего оцениваются карта глубины и альбедо изображения, а затем — освещение на изображении (посредством его сопоставления с известным освещением на других изображениях). Далее объект помещается в карту глубины изображения и полученный мир визуализируется программой физического рендеринга — стандартным инструментом в компьютерной графике. В завершение измененное изображение смешивается с исходным изображением.

Нейронные сети также могут быть обучены для выполнения ▶ **преобразования изображений**: отображения изображений типа X (например, размытого изображения, аэрофотоснимка города или рисунка нового продукта) в изображения типа Y

(например, в четкую версию изображения, в дорожную карту города или в фотографию продукта). Проще всего эта цель достигается, когда обучающие данные состоят из пар (X, Y) изображений; на рис. 25.24, *слева* каждая пара обучающих примеров включает аэрофотоснимок и соответствующий участок дорожной карты. При обучении для определения ошибки выход сети сравнивается с требуемым выходом, а также подключается компонент потерь из порождающей состязательной сети (*generative adversarial network* — GAN), назначение которой — обеспечить, чтобы вывод имел правильные варианты признаков для изображений типа Y. Как можно видеть на рис. 25.24, *справа*, системы такого класса работают очень хорошо.

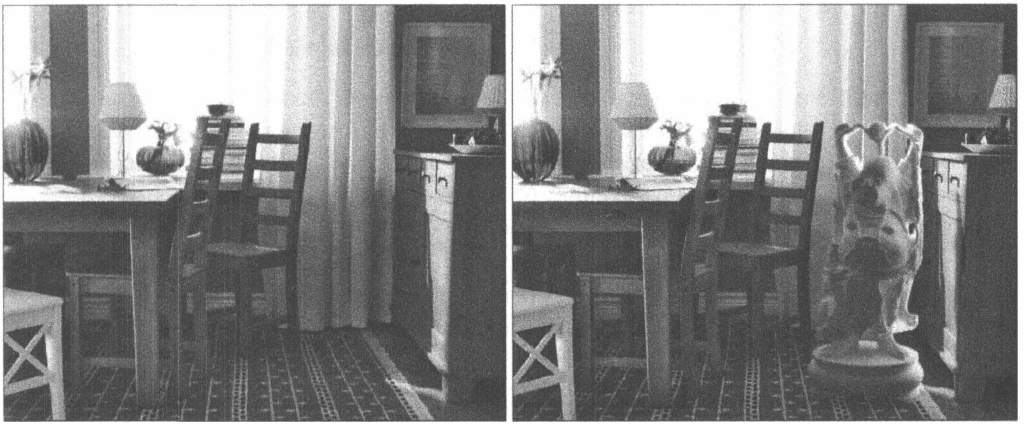


Рис. 25.23. Слева — изображение реальной сцены. **Справа** — эта же сцена со вставленным объектом компьютерной графики. Создается впечатление, что свет падает на объект в правильном направлении и что объект отбрасывает соответствующие тени. Сгенерированное изображение выглядит убедительно, даже если на самом деле имеют место небольшие ошибки в освещении и тенях, — просто потому, что люди не являются экспертами в выявлении подобных ошибок

В некоторых случаях может не существовать парных изображений типов (X, Y), но могут иметься большие независимые наборы изображений типа X (скажем, изображений лошадей) и типа Y (скажем, изображений зебр). Представим художника, которому поручено создать образ зебры, работающей в поле. Художник был бы рад возможности выбрать подходящее случаю изображение лошади, после чего компьютер автоматически превратил бы его в изображение зебры (рис. 25.25). Для достижения подобной цели можно обучить две сети преобразования изображений, но с дополнительным ограничением, называемым *циклическим ограничением*. Первая сеть будет отображать лошадей на зебр, а вторая — зебр на лошадей, но при этом циклическое ограничение требует, чтобы при отображении X в Y и вновь в X (или Y в X и вновь в Y) было получено то, что присутствовало в начале.

И вновь, потери в сети GAN гарантируют, что лошадь (или зебра) на изображениях, которые выводят сети, будет "подобна" реальной лошади (или зебре).

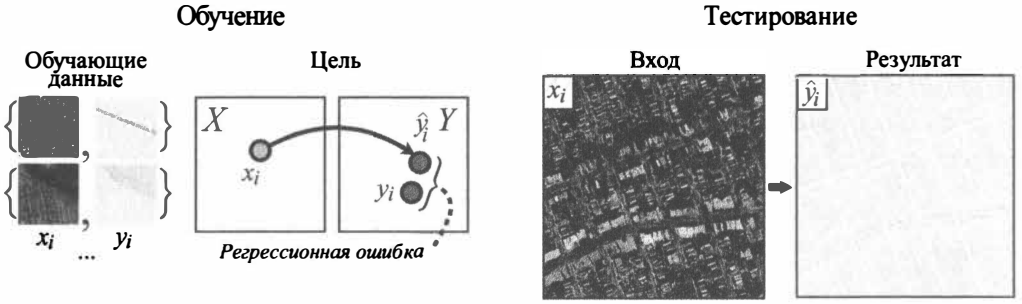


Рис. 25.24. Пары изображений в задаче преобразования, где вход является аэрофотоснимками, а выход — соответствующими фрагментами дорожной карты. **Слева** — этап обучения, цель которого состоит в том, чтобы обучить сеть создавать фрагмент дорожной карты из заданного на входе аэрофотоснимка. (Система также может научиться создавать аэрофотоснимки из фрагментов карты.) Сеть обучается путем сравнения $\hat{y}_i$ (выход для примера x_i типа X) с правильным выходным значением y_i типа Y. **Справа** — этап тестирования, во время которого сеть должна создавать новые изображения типа Y из предоставленных ей входных изображений типа X

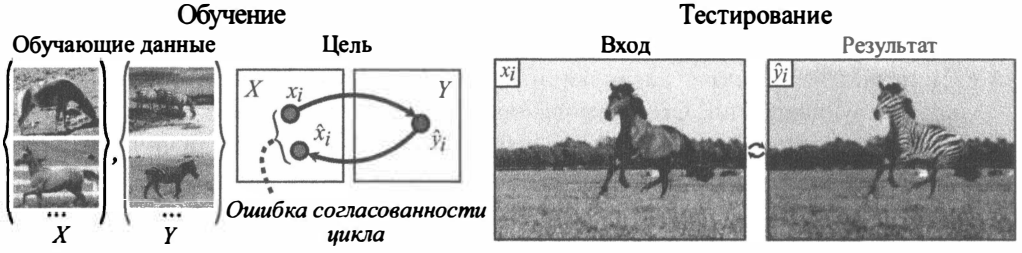


Рис. 25.25. Обучение преобразованию на непарных изображениях. Даны два набора изображений (здесь тип X — лошади, а тип Y — зебры), но соответствующие пары отсутствуют и система обучается преобразованию любых лошадей в зебр. Метод предусматривает обучение двух предсказывающих элементов: один отображает тип X на тип Y, а другой — тип Y на тип X. Если первая сеть отображает лошадь x_i на зебру $\hat{y}_i$, то вторая сеть должна отобразить $\hat{y}_i$ вновь на оригинал x_i. Разница между x_i и $\hat{x}_i$ — это как раз то, на чем обучаются обе сети. Цикл от Y до X и обратно должен быть замкнут. Такие сети могут успешно выполнять разнообразные преобразования изображений

Другой художественный эффект называется ▶ **передачей стиля**: вход состоит из двух изображений — *содержания* (например, фотография кота) и *стиля* (например, абстрактный рисунок). Выходные данные должны быть версией кота, представленной в абстрактном стиле (рис. 25.26). Ключ к пониманию того, как решить эту задачу, состоит в том, что если проанализировать глубокую сверточную нейронную сеть (CNN), обученную распознаванию объектов (скажем, на наборе данных ImageNet), то можно обнаружить, что первым слоям сети свойственна тенденция представлять стиль изображения, а последним — представлять контент. Пусть p будет изображением содержания, а s — изображением стиля, и пусть $E(x)$ будет вектором активаций первых слоев для изображения x, а $L(x)$ — вектором активаций последних слоев для изображения x. Если необходимо сгенерировать какое-то изображение x, которое будет иметь содержание, похожее на дом, то следует минимизировать $|L(x) - L(p)|$, а если оно должно быть в стиле, аналогичном живописи импрессионистов, то следует минимизировать $|E(x) - E(s)|$. Теперь для поиска изображения x, минимизирующего потери, можно использовать метод градиентного спуска с функцией потерь, представляющей собой линейную комбинацию этих двух факторов.

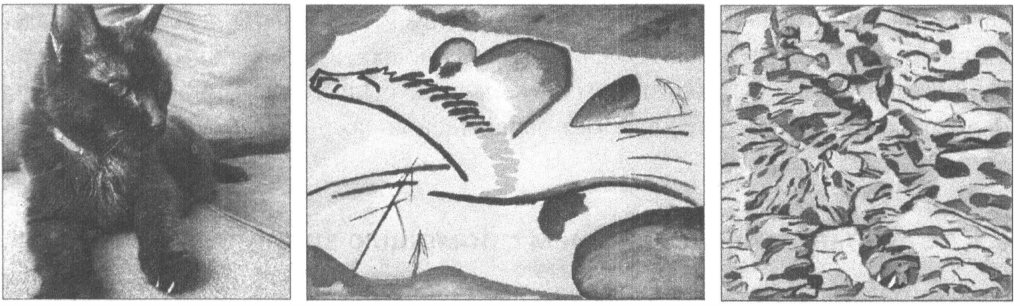


Рис. 25.26. Передача стиля. *Содержание* фотографии кота комбинируется со *стилем*, характерным для абстрактного рисунка, в результате чего получается новое изображение кота, выполненное в схожем абстрактном стиле

Порождающие состязательные сети (GAN) могут создавать новые фотореалистичные изображения, чаще всего успешно обманывая большинство людей. Один тип таких изображений называется ▶ **дипфейк** (*deepfake*); это изображение или видео, которое выглядит, как конкретный человек, но генерируется моделью. Например, когда Кэрри Фишер было 60 лет, сгенерированная моделью копия ее лица в возрасте 19 лет была наложена на тело другого актера при съемках фильма "Изгой-один". С одной стороны, киноиндустрия создает все более и более совершенные дипфейки для достижения художественных целей, а с другой — исследователи работают над разработкой контрмер, направленных на обнаружение любых

дипфейков с целью смягчения разрушительных последствий от публикации поддельных новостей.

Созданные изображения также могут быть использованы для обеспечения конфиденциальности. Например, существуют наборы данных из рентгеновских изображений, полученных в результате рентгенологической практики, которые были бы полезны для исследователей, но которые не могут быть опубликованы по причине соблюдения конфиденциальности пациентов. Порождающие модели изображений могут считать такой частный набор данных рентгеновских изображений и создать искусственный набор данных, который можно будет передать исследователям. Такой набор данных должен быть, во-первых, похожим на набор обучающих данных, во-вторых, отличаться от него и, в-третьих, быть контролируемым. Обратимся к рентгеновским снимкам грудной клетки. Искусственный набор данных должен быть похожим на набор обучающих данных в том смысле, что каждое отдельное изображение не должно вызвать у рентгенолога сомнений в его истинности, а частота появления каждого эффекта должна соответствовать истинной, чтобы рентгенолог не мог удивиться тому, как часто, скажем, имеет место пневмония. Новый набор данных должен быть другим в том смысле, что он не должен раскрывать личной информации. Новый набор данных должен быть контролируемым в том смысле, что должна существовать возможность регулировать частоту эффектов для отражения существующих реалий. Например, пневмония у людей старшего возраста встречаются чаще, чем у молодых людей. Каждая из этих целей технически труднодостижима, тем не менее уже были созданы подобные наборы данных, которые способны ввести в заблуждение даже практикующих рентгенологов (рис. 25.27).

25.7.6. Управление движением с помощью зрения

Одним из наиболее важных направлений применения систем компьютерного зрения является предоставление информации как для манипулирования объектами (их обнаружение, захват, вращение и т.д.), так и для управления передвижением, избегая столкновений с препятствиями. Способность использовать зрение для этих целей присуща системам зрения даже самых примитивных животных. Во многих случаях зрительная система зрения минимальна в том смысле, что из доступного светового поля извлекается только такая информация, которая требуется животному для организации его поведения. Вполне вероятно, что системы зрения современных животных эволюционировали от самых ранних, примитивных организмов, которые имели единственное светочувствительное пятно на одном конце тела, необходимое им для ориентации в направлении к свету (или от него). Как было указано в разделе 25.6, в нервной системе мухи имеется очень простая система распознавания оптического потока, позволяющая мухе садиться на стены.

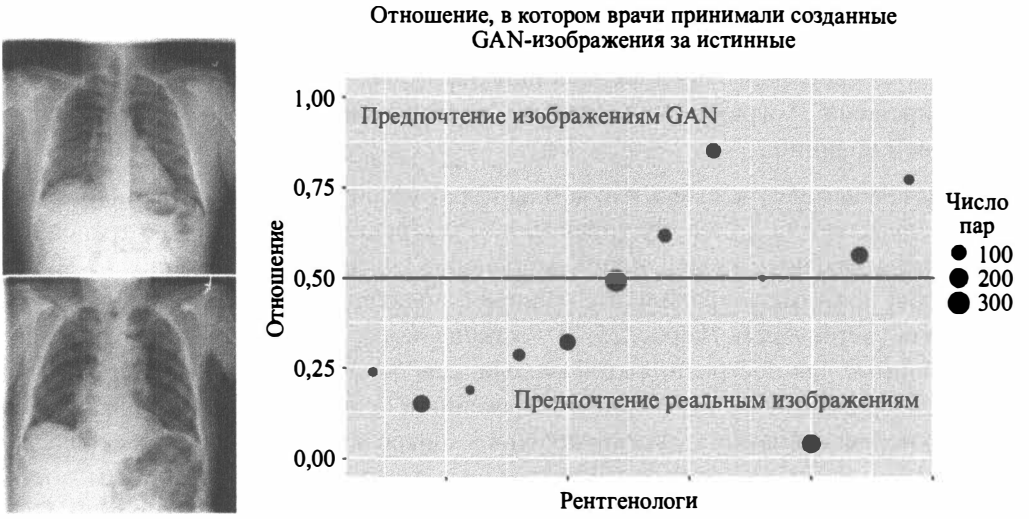


Рис. 25.27. Порождающие состязательные сети (GAN) способны генерировать рентгеновские изображения легких. Слева приведены два рентгеновских снимка, один из которых является реальным, а другой сгенерирован сетью GAN. Справа приведены результаты тестового опроса рентгенологов, в котором им предлагалось определить, какой из снимков в каждой паре, подобной приведенной слева, является настоящим. В среднем выбор был сделан правильно в 61% случаев, — это немного лучше, чем результат простого угадывания. Однако результаты отдельных участников значительно различались по точности: диаграмма справа показывает частоту ошибок для 12 разных рентгенологов. У одного из них уровень ошибок был около 0%, а у другого ошибочными оказались 80% ответов. Размер каждой точки соответствует числу изображений, просмотренных данным рентгенологом

Предположим, что вместо совершения посадки на стену требуется построить беспилотный автомобиль с автоматическим управлением. Такой проект предъявляет гораздо более высокие требования к системе восприятия. Восприятие в беспилотном автомобиле должно обеспечивать решение следующих задач.

- **Боковой контроль: управление движением в поперечном направлении.** Обеспечение того, чтобы транспортное средство надежно придерживалось своей полосы движения или плавно переходило на другую полосу движения в случае необходимости.
- **Продольный контроль: управление движением в продольном направлении.** Обеспечение безопасной дистанции до движущегося впереди транспортного средства.
- **Предотвращение столкновений с препятствиями.** Слежение за транспортными средствами на соседних полосах движения и готовность

к маневрам уклонения при необходимости. Обнаружение пешеходов и предоставление им возможности безопасно пересекать дорогу.

- **Следование сигналам управления дорожным движением.** К ним относятся светофоры, предписывающие дорожные знаки ("стоп", "остановка запрещена", "ограничение скорости" и пр.), а также ручные сигналы дорожной полиции.

Задача водителя (человека или компьютера) заключается в выполнении необходимых действий по рулевому управлению, ускорению и торможению транспортного средства с целью наилучшего выполнения поставленных заданий.

Чтобы принимать правильные решения, водитель должен построить модель мира и объектов в нем. На рис. 25.28 показаны некоторые визуальные заключения, необходимые для построения этой модели. Для бокового контроля водитель должен поддерживать представление о положении и ориентации автомобиля относительно полосы движения. Для продольного контроля водитель должен соблюдать безопасную дистанцию до транспортных средств впереди (что может быть достаточно сложно реализовать, скажем, на извилистых многополосных дорогах). Предотвращение столкновений с препятствиями и следование сигналам управления дорожным движением требуют дополнительных визуальных заключений.

Дороги были спроектированы для людей, которые для ориентации используют зрение, поэтому, в принципе, можно управлять автомобилем, пользуясь только зрением. Однако на практике коммерческие беспилотные автомобили используют различные датчики, включая камеры, лидары, радары и микрофоны. Лидар или радар позволяет непосредственно измерять дистанцию, причем с большей точностью, чем может быть достигнуто при использовании методов, опирающихся только на зрение, как описывается в разделе 25.6. Наличие нескольких типов датчиков повышает общую производительность системы, что особенно важно в условиях плохой видимости, например работа радара не зависит от наличия тумана, блокирующего работу камеры и лидары. Микрофоны позволяют обнаруживать приближающиеся транспортные средства (особенно снабженные сиренами), прежде чем они станут видимыми.

Было проведено множество исследований в отношении мобильных роботов, перемещающихся как в помещении, так и в открытом пространстве. Разработано множество приложений, таких как последняя миля посылки или доставка пиццы. В традиционных подходах эта задача разбивается на два этапа, как показано на рис. 25.29.

- **Построение карты**. Задача одновременной локализации и построения карты (*Simultaneous Localization and Mapping* — SLAM; см. раздел 26.4.1) представляет собой задачу построения трехмерной модели мира, включая местоположение робота в этом мире (или, более точно, местоположение каждой из камер робота). Эта модель (обычно представляющая собой облако точек препятствий) может быть построена на базе серии изображений, полученных при разных позициях камеры.

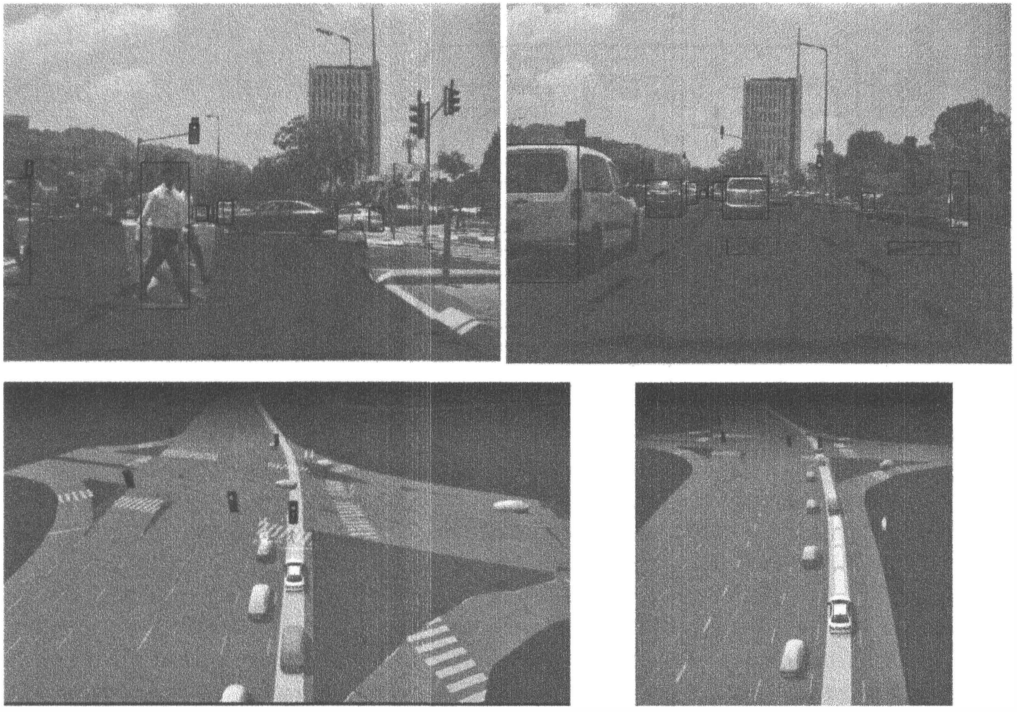


Рис. 25.28. Датчик Mobileye на базе видеокамеры для беспилотных транспортных средств. **Верхний ряд:** два изображения с фронтальной камеры, сделанные с интервалом в несколько секунд. Затемненная часть дороги (в цвете залита зеленым) представляет свободное пространство — ту область, в которой транспортное средство сможет физически двигаться в ближайшем будущем. Объекты представлены с помощью трехмерных ограничивающих рамок, определяющих их стороны (красная рамка с задней стороны, синяя — с правой стороны, желтая — с левой стороны и зеленая — с передней стороны). К объектам относятся транспортные средства, пешеходы, внутренний край дорожных знаков (необходимо для бокового контроля), элементы дорожной разметки и знаки пешеходного перехода, дорожные знаки и указатели, а также светофоры. Не показаны животные, палки и шишки, тротуары, перила и прочие предметы (например, кушетка, упавшая с кузова грузовика). Каждый объект затем помечается данными о его положении в трехмерном пространстве и скорости. **Нижний ряд:** полная физическая модель окружающей среды, визуализированная на базе обнаруженных объектов. (Изображения демонстрируют лишь результаты системного видения датчика Mobileye.)

- **Планирование пути.** Как только робот получит доступ к трехмерной карте и сможет локализовать свое местоположение в ней, его целью станет поиск траектории, исключающей какие-либо столкновения и ведущей из его текущей позиции к месту нахождения цели (см. раздел 26.6).

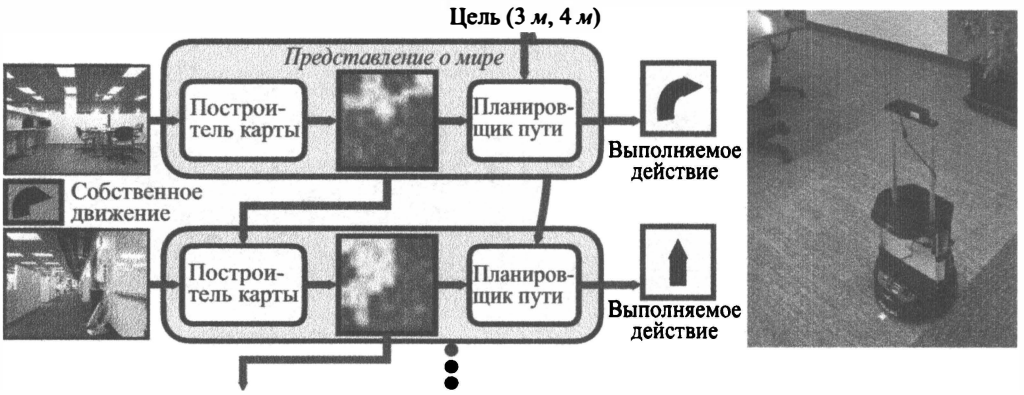


Рис. 25.29. Задача навигации решается путем ее разбиения на два этапа: построение карты и планирование пути. На каждом очередном этапе времени информация от датчиков используется для постепенного построения изменяющейся модели мира. Эта модель вместе со спецификацией цели передается планировщику пути, определяющему действие, которое робот должен предпринять следующим для достижения цели. Модель мира может быть чисто геометрической (как в классической задаче SLAM), семантической (как полученная в результате обучения) и даже топологической (построенной на базе ориентиров). В представленном случае робот повернул направо

Было изучено много вариантов этого общего подхода. Например, при когнитивном подходе к составлению карты и планированию пути два этапа построения карты и планирования пути представлены двумя модулями в нейронной сети, прошедшими совместное сквозное обучение для минимизации функции потерь. Такая система не должна создавать полную карту, которая часто является избыточной и невостребованной, когда все, что требуется, — это достаточное количество информации, чтобы переместиться из точки А в точку Б, не сталкиваясь по пути с препятствиями.

Резюме

Хотя на первый взгляд восприятие кажется легким делом для человека, в области ИИ оно связано со значительным объемом сложных вычислений. Задача зрения состоит в извлечении информации, необходимой, например, для манипулирования, навигации и распознавания объектов.

- Геометрические и физические аспекты формирования изображения хорошо изучены. Если дано описание трехмерной сцены, можно легко построить ее изображение таким, каким оно будет видимо из любой произвольно

выбранной позиции видеокамеры (это задача компьютерной графики). Задача выполнения обратного процесса, т.е. перехода от изображения к описанию сцены (это задача компьютерного зрения), является более сложной.

- Представление изображений фиксирует целый ряд признаков: края, текстуру, оптический поток, участки и т.д. Эти признаки дают необходимые подсказки для определения границ объектов и соответствия между изображениями.
- Сверточные нейронные сети применяются для построения точных классификаторов изображений, работающих на основе изученных ими признаков. В первом приближении можно считать, что эти признаки являются шаблонами шаблонов. Трудно предсказать, будут ли подобные классификаторы хорошо работать всегда, потому что те или иные тестовые данные могут отличаться от обучающих данных некоторым важным для системы образом. Тем не менее опыт показывает, что такие системы часто оказываются достаточно точными для использования на практике.
- Классификаторы изображений можно превратить в детекторы объектов. Сначала один классификатор оценивает содержимое отдельных ограничивающих рамок в изображении на "объектность", а затем другой классификатор принимает решение, находится ли в данной рамке объект и какой именно это объект. Методы обнаружения объектов пока еще не идеальны, но вполне могут использоваться в широком диапазоне практических приложений.
- При наличии более одного вида сцены можно восстановить ее трехмерную структуру и отношения между отдельными видами. Восстановить трехмерную геометрию сцены во многих случаях оказывается возможным даже на основе единственного изображения.
- Сегодня методы компьютерного зрения уже очень широко применяются на практике.

Библиографические и исторические заметки

В этой главе обсуждение было сосредоточено на зрении, но другие каналы восприятия также изучались и нашли применение в робототехнике. В отношении слухового восприятия (слуха) выше уже обсуждалась задача распознавания речи, но значительная работа также была проведена и по восприятию музыки (Коэлш и Зибель [1254], 2005) и по машинному обучению музыке (Энгел и др. [689], 2017), а также по машинному обучению восприятию звуков в целом (Шаран и Моир [2043], 2016).

Тактильное восприятие или осязание (Люо и др. [1468], 2017) также имеет большое значение в робототехнике и обсуждается в главе 26. Автоматическое восприятие запахов (обоняние) изучалось в меньшем масштабе, тем не менее было показано, что модели глубокого обучения могут быть обучены прогнозировать запахи, исходя из структуры молекул (Санчес-Ленгелинг и др. [1971], 2019).

Настойчивые попытки понять, как функционирует зрение человека, предпринимались с самых древних времен. Евклид (ок. 300 г. до н.э.) в своих трудах писал о естественной перспективе — отображении, связывающем с каждой точкой P в трехмерном мире направление луча OP, соединяющего центр проекции O с точкой P. Он также был хорошо знаком с понятием параллакса движения. На древнеримских картинах, подобных тем, которые были засыпаны пеплом при извержении Везувия в 79 году и сохранились до наших дней, присутствует неформальная перспектива более чем с одной линией горизонта.

Следующий важный шаг в развитии математической трактовки перспективной проекции, на этот раз в контексте проекции на плоские поверхности, был сделан в XV веке в Италии, в период Возрождения. Создателем первых картин, выполненных с учетом геометрически правильной проекции трехмерной сцены, принято считать Брунеллески (1413). В 1435 году Альберти составил свод правил построения перспективной проекции, ставший источником вдохновения для многих поколений художников. Особенно весомый вклад в развитие науки о перспективе (как она называлась в те времена) внесли Леонардо да Винчи и Альбрехт Дюрер. Составленные Леонардо в конце XV века описания игры света и тени (светотени), теневых и полутеневых областей затенения, а также воздушной перспективы до сих пор не потеряли своего значения (Кемп [1215], 1989).

Хотя знаниями о перспективе владели уже древние греки, в их представлениях присутствовала забавная путаница в отношении того, какую роль играют глаза в процессе зрения. Аристотель полагал, что глаза — это устройства, испускающие лучи, подобно современным лазерным дальномерам. Эти ошибочные взгляды были отвергнуты лишь благодаря работам арабских ученых X века, в частности Альхазена.

За этими исследованиями последовала разработка различных видов камер. Они представляли собой комнаты (*camera* на латыни — "темная комната"), в которые свет попадал через маленькое отверстие в одной стене, при этом на противоположной стене появлялось изображение сцены, происходящей снаружи. Безусловно, во всех этих камерах изображение было перевернутым, что постоянно вызывало смущение исследователей. Ведь если глаз рассматривать как аналогичное устройство формирования изображения, то почему мы видим предметы в правильном положении? Эта загадка не давала покоя величайшим умам той эпохи (включая Леонардо). Окончательно решить проблему удалось лишь благодаря работам Кеплера и Декарта. Декарт поместил анатомический препарат глаза, с задней стенки которого была удалена непрозрачная оболочка, в отверстие оконного ставня. В результате было получено перевернутое изображение, сформировавшееся на куске бумаги, заменившем сетчатку. Хотя изображение на сетчатке глаза действительно перевернуто, это не вызывает у нас проблем, поскольку мозг интерпретирует полученное изображение правильно. Говоря современным языком, достаточно лишь обеспечить правильный доступ к структуре данных.

Следующие значительные успехи в изучении зрения были достигнуты в XIX веке. Благодаря трудам Гельмгольца и Вундта, описанным в главе 1, методика

проведения психофизических экспериментов стала строгой научной дисциплиной. А труды Юнга, Максвелла и Гельмгольца привели к созданию трехкомпонентной теории цветоощущения. Стереоскоп, изобретенный Уитстоуном ([2329], 1838), убедительно продемонстрировал, что люди способны определять глубину изображения лишь благодаря тому, что на левый и правый глаза поступают немного разные изображения. Этот прибор быстро завоевал популярность в гостиных и салонах по всей Европе.

Основная концепция бинокулярных стереоданных о том, что два изображения сцены, полученные с немного различающихся точек зрения, несут информацию, достаточную для получения трехмерной реконструкции сцены, нашла применение в новой области — фотограмметрии. Позднее в этой области были получены ключевые математические результаты; например, Круппа в работе [1316] (1913) доказал, что если даны два изображения пяти различных точек одного и того же объекта, то можно реконструировать данные о повороте и переносе камеры с одной позиции в другую, а также о глубине сцены (с точностью до коэффициента масштабирования).

Хотя геометрия стереоскопического зрения была известна уже давно, оставалось неясным, как человек решает основную задачу фотограмметрии, автоматически согласуя соответствующие точки изображений. Удивительные способности людей решать задачу нахождения соответствия были продемонстрированы с помощью стереограмм случайных точек Юлеша ([1158], 1971). В области компьютерного зрении на решение проблемы соответствия были потрачены значительные усилия.

В первой половине XX века наиболее значительные результаты исследований в области зрения были получены представителями школы гештальт-психологии, возглавляемой Максом Вертхеймером. Они указывали на важность организации восприятия: для человека образ не является коллекцией точечных данных, предоставляемых фоторецепторами (пикселей), скорее образ организован как их связные группы. В компьютерном зрении такому пониманию соответствует задача нахождения сегментов и кривых. Гештальтисты также обратили внимание на феномен "фигура и фон": контур, разделяющей две области изображения, которые в реальном мире лежат на разной глубине, кажется принадлежащим только ближнему региону, "фигуре", а не более отдаленной области, "фону".

После Второй мировой войны работа в области гештальт-психологии была продолжена Дж. Дж. Гибсоном ([851], 1950; [852], 1979), который указал на важность градиентов оптического потока и текстуры при оценке таких переменных среды, как поворот и наклон поверхности. Он еще раз подчеркнул значимость стимулов и их разнообразие. В работе [853] (1955) Гибсон, Олум и Розенблатт показали, что поле оптического потока содержит достаточно информации для определения движения наблюдателя относительно окружающей среды. Гибсон особенно подчеркивал важность роли активного наблюдателя, собственное движение которого упрощает сбор информации о внешней среде.

Первые работы в области компьютерного зрения восходят к 1960-м годам. Опубликованный в Массачусетсском технологическом институте тезис Робертса ([1895], 1963) о восприятии кубов и других объектов в мире блоков стал одной из самых ранних публикаций в этой области. Робертс представил несколько ключевых идей, в том числе обнаружение краев и сопоставление на основе моделей.

В 1960-х и 1970-х годах прогресс был довольно медленным из-за отсутствия необходимых вычислительных ресурсов и средств хранения данных. Много внимания уделялось низкоуровневой визуальной обработке с использованием методов, заимствованных из связанных областей, таких как обработка сигналов, распознавание образов и кластеризация данных.

Обнаружение краев рассматривалось как важный первый шаг в обработке изображений, поскольку это уменьшало количество данных, подлежащих обработке. Широко используемая техника обнаружения краев Кэнни была предложена Джоном Кэнни в работе [364] (1986). Мартин, Фолкес и Малик в работе [1508] (2004) показали, как в рамках машинного обучения можно объединять несколько подсказок, таких как яркость, текстура и цвет, с целью улучшения поиска граничных кривых.

Тесно связанная проблема поиска сегментов изображений из пикселей со схожей яркостью, цветом и текстурой естественным образом поддается формулировке, при которой поиск наилучшего разбиения сводится к задаче оптимизации. Три ведущих подхода строятся на использовании марковских случайных полей (Геман и Геман [833], 1984), вариационной формулировки (Мамфорд и Шах [1634], 1989) и нормализованных срезов (Ши и Малик [2048], 2000).

На протяжении большей части 1960-х, 1970-х и 1980-х годов в области визуального распознавания господствовали две различные парадигмы, различавшиеся взглядами на то, что следует считать основной проблемой в этой области. В области компьютерного зрения исследования по распознаванию объектов преимущественно фокусировались на вопросах, возникающих при проецировании трехмерных объектов на двухмерные изображения. Идея выравнивания, также впервые представленная Робертсом, вновь всплыла в 1980-х годах в работах Лоу ([1449], 1987), Хаттенлохера и Уллмана ([1104], 1990).

Сообщество по распознаванию образов выбрало другой подход, рассматривая аспект 3D–2D-проецирования для этой задачи как несущественный. Их основные интересы сосредоточивались в таких областях, как оптическое распознавание символов и распознавание рукописных почтовых индексов, в которых основная проблема заключается в изучении типичных вариаций характеристик объектов некоторого класса и в их выделении из других классов. Развитие архитектур нейронных сетей для анализа изображений можно проследить, начиная с работ Хьюбела и Визеля ([1087], 1962; [1088], 1968), занимавшихся исследованиями зрительной коры у кошек и обезьян. Они разработали иерархическую модель зрительного пути с нейронами в нижних областях мозга (преимущественно в области под названием "V1"), реагирующими на такие признаки, как ориентированные края и

полосы, и нейронами в более высоких областях, реагирующими на более специфические стимулы.

Фукусима в работе [798] (1980) предложил архитектуру нейронной сети для распознавания шаблонов, явно мотивированную иерархией Хьюбела и Визеля. Его модель имеет чередующиеся слои простых клеток и сложных клеток, включая, таким образом, субдискретизацию, а также имеет инвариантность по сдвигу, включая, таким образом, сверточную структуру. ЛеКун и соавт. ([1372], 1989) предприняли дополнительный шаг, используя обратное распространение для обучения весов этой сети, и родилось то, что сегодня мы называем сверточной нейронной сетью. Сравнение подходов приведено ЛеКуном и соавт. в работе [1373] (1995).

С конца 1990-х годов на фоне значительного возрастания роли вероятностного моделирования и статистического машинного обучения в области искусственного интеллекта в целом наметилось определенное сближение этих двух традиций. Исследования, проведенные в обоих этих направлениях, внесли значительный вклад в общее дело. Одним из них было исследование по распознаванию лиц (Роули и др. [1924], 1998; Виола и Джонс [2276], 2004), результаты которого продемонстрировали всю мощь методов распознавания образов при решении важных и полезных задач.

Другим была разработка точечных дескрипторов, позволяющих построить векторы признаков из частей объектов (Шмид и Мор [1995], 1996). Существует три ключевые стратегии построения хороших локальных дескрипторов точек: в одной используется ориентация для достижения инвариантности освещения, в другой необходимо подробно описать структуру изображения только вблизи точки, а дальше — только приблизительно, а в третьей необходимо использовать пространственные гистограммы для подавления изменений, вызванных небольшими ошибками в определении местоположения точки. В предложенном Лоу ([1450], 2004) дескрипторе SIFT эти идеи использовались очень эффективно. Другим популярным вариантом был дескриптор HOG, разработанный Далалом и Триггсом ([518], 2005).

В 1990-е и 2000-е годы наблюдалось продолжение дебатов между теми, кто предпочитал искусное построение признаков, подобно дескрипторам SIFT и HOG, и поклонниками нейронных сетей, считавших, что хорошие признаки должны появляться автоматически в результате их сквозного обучения. Как возможность смягчить это противостояние рассматривалось создание эталонных стандартных наборов данных, предназначенных для измерения производительности систем, и в 2000-х годах результаты тестирования на базе стандартного набора данных для распознавания объектов, PASCAL VOC, свидетельствовали в пользу разработанных вручную признаков. Однако эта ситуация резко изменилось, когда Крижевский и соавт. ([1313], 2013) показали, что в тестовой задаче классификации изображений на основе набора данных ImageNet их нейронная сеть (под названием "AlexNet") продемонстрировала значительно более низкий уровень ошибок в сравнении с

любыми системами, использующими общепринятые на тот момент методы компьютерного зрения.

В чем заключался секрет успеха сети AlexNet? Помимо технических инноваций (например, использования элементов с функцией активации ReLU), следует отдать должное **большим данным** и **большим вычислениям**. Под большими данными здесь подразумевается наличие больших наборов данных с метками категорий, таких как ImageNet, который использовался в качестве обучающих данных для подобных крупных, глубоких сетей с миллионами параметров. Предыдущие наборы данных, подобные Caltech-101 или PASCAL VOC, не позволяли получить достаточного количества обучающих данных, а наборы MNIST и CIFAR вообще рассматривались сообществом компьютерного зрения как "игрушечные". Новая ветвь маркированных наборов данных, предназначенных для сравнительного анализа и для извлечения статистики по изображениям, была построена на использовании желания людей загружать свои коллекции фотографий в Интернет на таких сайтах, как Flickr. В свою очередь, способом, обеспечившим возможность большим вычислениям доказать свою полезность, стало использование графических процессоров (GPU) — результата достижений в разработках аппаратного обеспечения, изначально движимых лишь все возрастающими потребностями индустрии видеоигр.

В течение года или двух ситуация окончательно прояснилась. Например, появились регионально-ориентированные сверточные нейронные сети (RCNN) — в работе Гиршика и соавт. [864] (2016) было показано, что архитектура сети AlexNet может быть изменена за счет использования таких идей компьютерного зрения, как региональное предложение, что делает возможным обнаружение объектов на современном уровне при обучении сети даже на наборе PASCAL VOC. Стало понятно, что, как правило, более глубокие сети работают лучше и что опасения в отношении переобучения на сверхбольших наборах данных были преувеличены. Также появились новые методы, такие как **пакетная нормализация**, позволяющие работать с регуляризацией.

Реконструкция трехмерной структуры с нескольких точек зрения берет свое начало в литературе по фотограмметрии. Уже в эпоху компьютерного зрения ранними влиятельными работами стали публикации Ульмана [2242] (1979) и Лонге-Хиггинса [1441] (1981). Опасения по поводу устойчивости структуры при движении были значительно смягчены Томаси и Канаде в работе [2220] (1992), которые показали, что при использовании нескольких кадров и получающейся в результате широкой базовой линии форму можно восстановить достаточно точно.

В 1990-х годах концептуальным новшеством стало изучение проективной структуры при движении. В этом случае калибровка камеры не требуется, как было показано Фаугерасом в работе [710] (1992). Это открытие было связано с введением использования геометрических инвариантов при распознавании объектов, как это следует из обзора Манди и Циссермана [1635] (1992) и разработки аффинных структур при движении Коендериком и ван Дорном ([1256], 1991).

В 1990-х годах, благодаря огромному увеличению скорости работы компьютеров и объемов хранилищ данных, а также широкому распространению доступного цифрового видео, методы анализа движений нашли много новых приложений. Построение геометрических моделей сцен из реального мира, предназначенных для рендеринга методами компьютерной графики, оказалось особенно популярным направлением во главе с алгоритмами реконструкции, подобными алгоритму, разработанному Дебевеком и соавт. ([575], 1996). Учебники Хартли и Циссермана ([975], 2000), а также Фаугераса и соавт. ([711], 2001) предоставляют исчерпывающее изложение геометрии многих видов.

Люди способны воспринимать форму и пространственное расположение объектов, исходя из единственного изображения, однако моделирование таких действий оказалось довольно сложной задачей для исследователей в области компьютерного зрения. Вывод формы на основе затенения впервые изучался Бертольдом Хорном ([1060], 1970), а в публикации Хорна и Брукса [1061], (1989) предоставлен обширный обзор основных работ, опубликованных в тот период, когда эта проблема интенсивно изучалась. Гибсон ([851], 1950) был первым, кто предложил градиенты текстуры как сигналы для определения формы. Математика затенения контуров и более общее понимание визуальных явлений в проекции на гладкие изогнутые объекты во многом обязаны работам Коендерика и ван Дорна, результаты которых детально излагаются в книге Коендерика *Solid Shape* ([1255], 1990).

В последнее время больше внимания обращается на решение задачи восстановления формы и поверхности на основании единственного изображения как задачи вероятностного вывода, в котором геометрические сигналы не смоделированы явно, а используются неявно в рамках машинного обучения. Хорошим примером является исследование Хоима и соавт. [1049] (2007); недавно оно было переработано с использованием глубоких нейронных сетей.

Если обратиться к приложениям компьютерного зрения для руководства действиями, то можно заметить, что пионерами в этой области были Дикманнс и Зэпп, которые в работе [618] (1987) впервые продемонстрировали беспилотный автомобиль, способный ездить по автостраде на высоких скоростях. Позднее Померлау ([1806], 1993) также сообщил о достижении аналогичных результатов, но уже с использованием нейронной сети. Сегодня создание беспилотных автомобилей — это большой бизнес, в котором известные автомобильные компании конкурируют с новыми участниками, такими как Baidu, Cruise, Didi, Google Waymo, Lyft, Mobileye, Nuro, Nvidia, SamSung, Tata, Tesla, Uber или Voyage, на рынке систем, предоставляющих любые возможности, начиная от оказания помощи водителю до полностью автономных беспилотных автомобилей.

Для читателей, интересующихся зрением человека, в качестве лучшего комплексного изложения этой темы можно предложить книгу *Vision Science: Photons to Phenomenology* Стивена Палмера ([1725], 1999). Книга *Visual Perception: Physiology, Psychology and Ecology* Вики Брюса, Патрика Грина и Марка Джорджсона ([326], 2003) представляет собой более краткий учебник. Книги *Eye, Brain and*

Vision Дэвида Хьюбела ([1089], 1988) и *Perception* Ирвина Рока ([1903], 1984) — это дружественные введения, посвященные нейрофизиологии и восприятию соответственно. Книга Дэвида Марра *Vision* ([1499], 1982) сыграла историческую роль в установлении связи между компьютерным зрением и традиционными областями биологического зрения — психофизикой и нейробиологией. Хотя многие из предложенных автором конкретных моделей для таких задач, как обнаружение краев и обнаружение объектов не выдержали испытания временем, теоретический аспект рассмотрения, при котором каждая задача проанализирована на уровнях информационном, вычислительном и методологии реализации, по-прежнему можно считать блестящим.

В области компьютерного зрения наиболее полными учебниками, доступными на сегодняшний день, являются *Computer Vision: A Modern Approach* (Форсит и Понс [762], 2002) и *Computer Vision: Algorithms andApplications* (Желески [2167], 2011). Геометрические проблемы в компьютерном зрении подробно рассмотрены в книге *Multiple View Geometry in Computer Vision* (Хартли и Циссерман [975], 2000). Все упомянутые выше книги были написаны до "революции" глубокого обучения, поэтому для ознакомления с последними результатами обратитесь к соответствующим первоисточникам.

Двумя основными журналами по компьютерному зрению являются журнал IEEE *Transactions on Pattern Analysis and Machine Intelligence* и *International Journal of Computer Vision*. К числу конференций по компьютерному зрению относятся ICCV (*International Conference on Computer Vision*), CVPR (*Computer Vision and Pattern Recognition*) и ECCV (*European Conference on Computer Vision*). Результаты исследований со значительным компонентом машинного обучения публикуются также в NeurIPS (*Neural Information Processing Systems*), а работы в области интерфейса с компьютерной графикой часто представляются на конференции ACM SIGGRAPH (*Special Interest Group in Graphics*). Много статей о компьютерном зрении в виде препринтов появляются на сервере arXiv, а предварительные отчеты о новых полученных результатах публикуются в блогах крупных исследовательских лабораторий.

Упражнения

25.1. В тени дерева с плотной, густой кроной можно обнаружить множество пятен света. На удивление все эти пятна кажутся круглыми. Почему? С чем это связано? Ведь в конечном итоге просветы между листьями, через которые проникают лучи солнца, вряд ли имеют круглую форму.

25.2. Рассмотрим изображение белой сферы, плавающей перед черным фоном. Кривую на изображении, отделяющую белые пиксели от черных, иногда называют "контуром" сферы. Покажите, что контур сферы, видимый в камере с перспективой, может оказаться эллипсом. Почему в наших глазах сферы не принимают вид эллипсов?

25.3. Рассмотрим цилиндр бесконечной длины с радиусом r, ось которого направлена вдоль оси y. Цилиндр имеет ламбертову поверхность и фотографируется камерой, оптическая ось которой направлена вдоль положительной полуоси z. Что можно ожидать увидеть на полученном изображении, если цилиндр освещается точечным источником света, находящимся на бесконечно большом расстоянии со стороны положительной полуоси x? Поясните свой ответ, нарисовав контуры равной яркости на полученном изображении. Будут ли контуры равной яркости расположены через равномерные интервалы?

25.4. Края на изображении могут соответствовать различным визуальным эффектам, возникающим в сцене. Рассмотрим рис. 25.30 в предположении, что это изображение реальной трехмерной сцены. Отыщите на этом изображении десять краев, отделяющих участки с разной яркостью, и для каждого из них укажите, соответствует ли оно разрыву:

а) по глубине;
б) по ориентации поверхности;
в) по отражательной способности;
г) по освещенности (тени).

Рис. 25.30. Изображение реальной трехмерной сцены

25.5. Какие из приведенных ниже утверждений являются истинными, а какие ложными?

а) Обнаружение соответствующих друг другу точек в стереоскопических изображениях — самая простая стадия процесса определения глубины на стереоизображении.
б) Извлечение формы из текстуры можно выполнить, проектируя на сцену сетку световых полос.
в) Линии равной длины в сцене всегда проецируются в изображении линиями равной длины.
г) Прямые линии в изображении обязательно соответствуют прямым линиям в сцене.

25.6. Какие из приведенных ниже утверждений являются истинными, а какие ложными?

а) Обнаружение соответствующих друг другу точек в стереоскопических изображениях — самая простая стадия процесса определения глубины на стереоизображении.
б) Для стереоскопических изображений одной и той же сцены бо́льшая точность вычисления глубины достигается, если две камеры расположены на большем расстоянии одна от другой.
в) Линии равной длины в сцене всегда проецируются в изображении линиями равной длины.
г) Прямые линии в изображении обязательно соответствуют прямым линиям в сцене.

25.7. На рис. 25.31 показаны две видеокамеры в точках X и Y, с помощью которых ведется наблюдение за сценой. Нарисуйте изображение, поступающее на каждую видеокамеру, в предположении, что все обозначенные на рисунке точки находятся в одной и той же горизонтальной плоскости. Исходя из этих двух изображений, какие выводы можно сделать по поводу относительных расстояний до точек A, B, C, D и E от базисной линии видеокамер? На каком основании сделаны эти выводы?

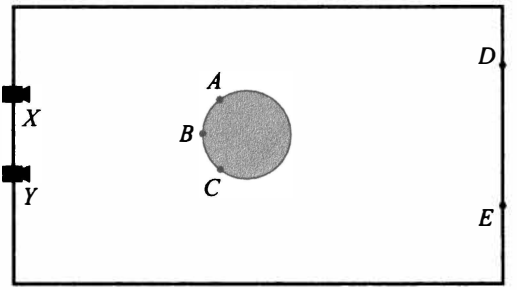


Рис. 25.31. Вид сверху на систему компьютерного зрения с двумя видеокамерами, в которой наблюдается бутылка, расположенная на некотором расстоянии от стены

ГЛАВА 26

Робототехника

В этой главе обсуждается, как оснастить агентов датчиками и физическими исполнительными механизмами, чтобы они могли передвигаться и немного пошалить в реальном мире.

26.1. Роботы

► **Роботы** — это физические агенты, выполняющие поставленные задачи с помощью манипуляций в физическом мире. Для этой цели роботов оснащают ► **исполнительными механизмами**, такими как ноги, колеса, шарниры или захваты. Исполнительные механизмы разрабатываются для предоставления роботам возможности приложения физически усилий к среде. Когда они это делают, происходят некоторые события: может измениться состояние робота (например, беспилотный автомобиль вращает колеса и тем самым достигает перемещения по дороге), может измениться состояние окружающей среды (например, рука робота использует свой захват, чтобы передвинуть стакан на стойке), может измениться даже состояние людей, находящихся вокруг робота (например, экзоскелет выполняет движение, и это меняет конфигурацию ноги человека; или мобильный робот направляется к дверям лифта, а человек это замечает и вежливо отодвигается, чтобы освободить ему дорогу, или даже нажимает для робота кнопку лифта).

Роботы также оснащены ► **датчиками**, позволяющими им воспринимать окружающую среду. В современной робототехнике используется разнообразный набор датчиков, включая видеокамеры, радары, лазеры и микрофоны для измерения состояния окружающей среды и окружающих людей, а также гироскопы, акселерометры, датчики деформации и крутящего момента для измерения собственного состояния робота.

Для робота максимизация ожидаемой полезности означает выбор того, как задействовать свои эффекторы для приложения *необходимых* физических сил — тех сил, которые приведут к таким изменениям в состоянии, что можно будет получить настолько много ожидаемого вознаграждения, насколько это возможно. В конечном счете роботы пытаются выполнить в физическом мире какое-то поставленное перед ними задание.

Роботам приходится действовать в среде, которая является частично наблюдаемой и стохастической: видеокамера не может видеть, что находится за углом, а исполнительные механизмы могут проскальзывать. Более того, люди, действующие в той же окружающей среде, непредсказуемы, поэтому роботу требуется прогнозировать их поведение.

Роботы обычно моделируют окружающую среду с помощью непрерывного пространства состояний (положение робота описывается непрерывными координатами) и непрерывного пространства действий (например, величина тока, который робот направляет в свой двигатель, также измеряется в непрерывных единицах). Некоторые роботы оперируют в многомерных пространствах: беспилотные автомобили должны знать как свои местоположение, ориентацию и скорость, так и близлежащих участников движения; руки робота имеют шесть или семь суставов, каждый из которых может выполнять движения независимо, а роботы, имитирующие тело человека, имеют сотни суставов.

Возможности обучения роботов ограничены, поскольку реальный мир упорно отказывается действовать быстрее, чем в реальном времени. В моделируемой среде можно использовать алгоритмы обучения (такие, как алгоритм Q-обучения, обсуждавшийся в главе 22), позволяющие всего за несколько часов провести обучение на миллионах примеров. В реальных условиях на обучение в таких объемах могут уйти годы, к тому же роботом нельзя рисковать — он не может подвергаться испытаниям (а значит, и учиться на них), которые могут причинить ему вред. Поэтому передача реальному роботу в реальном мире того, что было изучено в условиях моделирования — задача **симуляция–реальность**, — является активной областью исследований. Практические робототехнические системы должны включать предварительные знания о роботе, физической среде и задачах, которые требуется выполнять, чтобы робот мог быстро обучаться и действовать безопасно.

Робототехника объединяет многие концепции, которые обсуждались ранее в этой книге, включая вероятностную оценку состояния, восприятие, планирование, обучение без учителя, обучение с подкреплением и теорию игр. Для одних концепций робототехника является сложным примером применения. А для других концепций в этой главе открываются новые возможности применения, например, за счет введения непрерывных версий методов, которые ранее рассматривались только для дискретного случая.

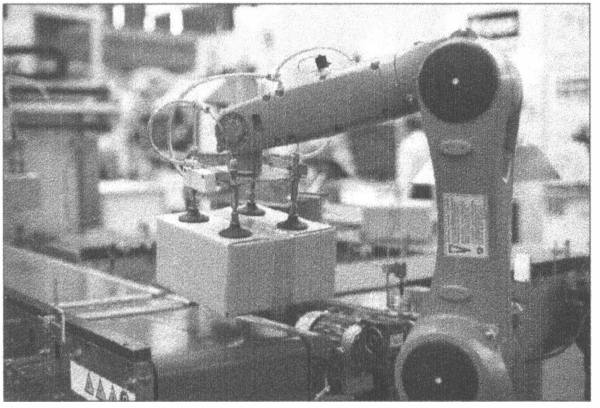
26.2. Аппаратное обеспечение роботов

До сих пор в этой книге предполагалось, что архитектура агентов — датчики, исполнительные механизмы и процессоры — уже определена и осталось заняться лишь разработкой программы агента. Но успехи в создании реальных роботов не в меньшей степени зависят от того, насколько удачно будут спроектированы датчики и исполнительные механизмы, подходящие для выполнения поставленной задачи.

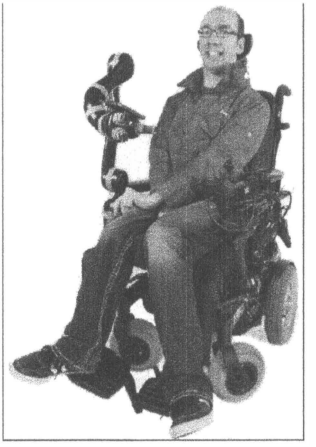
26.2.1. Типы роботов с аппаратной точки зрения

Говоря о роботах, мы обычно представляем себе нечто с головой и двумя руками, перемещающееся в окружающем пространстве на ногах или колесах. Такие ►**антропоморфные роботы** были широко популяризированы в фантастике всех видов — книгах (*Кибериада* Станислава Лема), кинофильмах (*Терминатор*) или мультфильмах (*Волли, Лило и Стич*). Но настоящие роботы существуют во множестве форм и размеров.

Роботы ►**манипуляторы** — это просто роботизированные руки. Они необязательно должны быть прикреплены к телу робота какого-либо вида, они могут быть прикреплены просто к своему рабочему месту или полу, как в заводских цехах (рис. 26.1, *а*). Одни из них способны манипулировать большими грузами, как те, которые работают на производственных линиях автозаводов, тогда как другие — механические руки, монтируемые на инвалидных колясках для оказания помощи людям с ограниченными двигательными возможностями (рис. 26.1, *б*) — создают меньшие усилия, но безопаснее для окружающих.



а)



б)

Рис. 26.1. а) Промышленный роботизированный манипулятор со специализированным конечным исполнительным механизмом. **б)** Роботизированный манипулятор-рука марки Kinova, установленный на инвалидной коляске

Другой тип — ►**мобильные роботы**. Роботы такого типа передвигаются в пределах своей среды, пользуясь колесами, ногами или иными типами движущих механизмов. ►**Дроны**, или ►**квадрокоптеры** (рис. 26.2, *б*), — это тип беспилотного летательного аппарата (*Unmanned Aerial Vehicle* — (►**UAV**)), тогда как автономные подводные аппараты (*Autonomous Underwater Vehicles* — ►**AUV**)

предназначены для исследования глубин океанов. С другой стороны, многие мобильные роботы всегда находятся в помещениях и перемещаются на колесах, как роботы-пылесосы или роботы для доставки полотенец в отелях. К их аналогам вне помещений относятся ▶**беспилотные автомобили**, или ▶**роверы**, предназначенные для исследования новых местностей, включая даже поверхность Марса (рис. 26.2, *а*). Наконец, ▶**роботы на ногах** предназначены для перемещения по пересеченной местности, которая недоступна роботам с колесами. Недостатком в этом случае можно считать то, что управлять ногами в процессе перемещения — более сложная задача, чем управление колесными движителями.

а) б)

Рис. 26.2. а) Марсоход НАСА *Кьюриосити* сделал селфи на поверхности Марса. **б)** Дрон *Skydio*, сопровождающий семью на велосипедной прогулке

К другим типам роботов относятся протезы, экзоскелеты, роботы с крыльями, многотельные системы — рои или стаи небольших механизмов, а также интеллектуальные среды, в которых роботизировано все помещение.

26.2.2. Восприятие мира

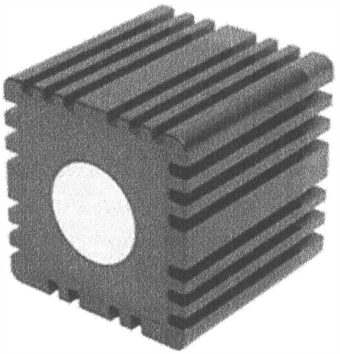
Датчики представляют собой интерфейс восприятия между роботом и окружающей средой. В ▶**пассивных датчиках**, таких как видеокамеры, реализуется непосредственное наблюдение за окружающей средой: они фиксируют сигналы, генерируемые другими источниками в окружающей среде. ▶**Активные датчики**, такие как сонар, посылают собственную энергию в окружающую среду, полагаясь на предположение, что хотя бы часть этой энергии будет отражена обратно

на датчик. Активные датчики, как правило, предоставляют больше информации, чем пассивные датчики, но за счет увеличения потребляемой ими мощности и повышения опасности возникновения помех при использовании нескольких активных датчиков одновременно. Также различают, направлен ли датчик на восприятие окружающей среды, определение местоположения робота или уточнение его внутренней конфигурации.

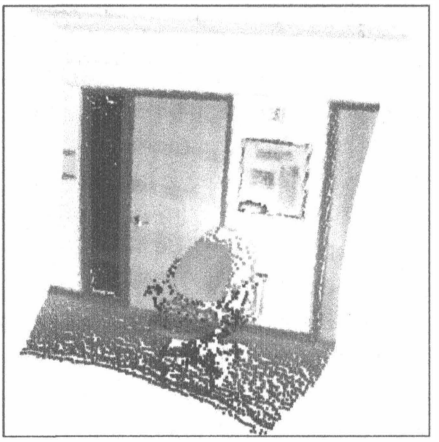
►**Дальномеры** представляют собой датчики, измеряющие расстояние до ближайших объектов. ►**Сонар** (или звуковой локатор) является активным датчиком, излучающим направленные звуковые волны, отражаемые объектами, и часть этого звука поступает назад в датчик. Время поступления и интенсивность отраженного сигнала определяют расстояние до ближайших объектов. Сонар — это технология, широко используемая в подводных аппаратах, но она также была популярна в роботах первого поколения, предназначенных для помещений. ►**Стереозрение** (см. раздел 25.6) построено на использовании нескольких видеокамер для наблюдения окружающей среды с немного различающихся точек зрения, с последующим анализом параллакса, обнаруженного на этих изображениях для вычисления расстояний до окружающих предметов.

В мобильных наземных роботах сонар и стереозрение в настоящее время используются редко, поскольку точность их недостаточно надежна. *Kinect* — это популярный тип недорогого датчика, сочетающего в себе видеокамеру и проектор ►**структурированного света**, который проецирует на сцену узор из линий сетки. Камера воспринимает, как изгибаются линии этой сетки на изображении объектов, и это предоставляет датчику требуемую информацию о форме объектов в сцене. При необходимости проецирование может осуществляться инфракрасным светом, чтобы не мешать другим датчики (таким, как глаз человека).

Большинство наземных роботов сейчас оснащены активными оптическими дальномерами. Как и сонар, эти датчики оптического диапазона излучают активные сигналы (свет) и измеряют время до того момента, когда отражение этого сигнала возвращается на датчик. На рис. 26.3, *а* показана ►**дальностная камера** такого типа, которая формирует *дальностные* изображения (на которых пиксели содержат оценку расстояний до наблюдаемых точек), подобные показанному на рис. 26.3, *б*, с частотой до 60 кадров в секунду. В беспилотных автомобилях часто используются ►**сканирующие лидары** (сокращение от *light detection and ranging*) — активные датчики, излучающие лазерные лучи и воспринимающие отраженный луч, обеспечивая измерение дальности с точностью до сантиметра на расстоянии 100 метров. В них сложные механические устройства в виде вращающихся зеркал или микроэлектромеханических систем используются для того, чтобы просканировать световым лучом окружающую среду и построить карту. На больших расстояниях и при ярком дневном свете сканирующие лидары, как правило, работают лучше, чем дальностные камеры.



а)



б)

Рис. 26.3. а) Дальностная камера. **б)** Трехмерное дальностное изображение, полученное с помощью этой камеры. Дальностное изображение позволяет обнаруживать препятствия и объекты в непосредственной близости от робота

►**Радар** часто выбирается в качестве датчика расстояния для воздушных транспортных средств (автономных или нет). Радарные датчики могут измерять расстояния до километров и имеют преимущество перед оптическими датчиками в том, что способны "видеть" сквозь туман. Для измерения расстояний на другом конце диапазона предназначены ►**тактильные датчики**, такие как контактные усики, контактные панели и чувствительные к прикосновениям покрытия. Эти датчики измеряют расстояние на основании физического контакта и могут устанавливаться только для обнаружения объектов, находящихся очень близко к роботу.

Вторым важным классом являются ►**датчики местоположения**. В большинстве датчиков местоположения используется измерение дальности в качестве основного компонента для определения местоположения. Вне помещений наиболее распространенным решением проблемы локализации является использование ►**глобальной системы позиционирования** (*Global Positioning System* — GPS). В системе GPS измеряется расстояние до спутников, излучающих импульсные сигналы. В настоящее время на орбите находятся 31 работающий спутник системы GPS и 24 спутника ее российского аналога, системы GLONASS. Приемники GPS-сигналов позволяют определить расстояние до спутника путем анализа фазовых сдвигов. Методом триангуляции сигналов от нескольких спутников GPS-приемники способны определить свое абсолютное местоположение на Земле с точностью до нескольких метров. ►В системе **дифференциального GPS** используется

второй наземный приемник с известными координатами, что обеспечивает миллиметровую точность при идеальных условиях.

К сожалению, система GPS не работает в помещении или под водой. В помещениях локализация часто обеспечивается путем подключения маяков, расположенных в известных местах в окружающей среде. Многие внутренние помещения содержат большое число базовых станций беспроводной связи, что может помочь роботам определять свое местоположение посредством анализа беспроводных сигналов. Активные подводные сонарные маяки также способны обеспечить определение местоположения, используя звуковые сигналы для информирования автономных подводных аппаратов об их относительном расстоянии от этих маяков.

К третьему важному классу относятся ▶**проприоцептивные датчики**, информирующие робота о его собственном движении и состоянии. Для измерения точной конфигурации робототехнического шарнира приводящие его в действие электродвигатели часто оснащаются ▶**датчиками угла поворота** вала (*shaft decoder*), точно измеряющими угловое движение вала. В манипуляторах роботов датчики угла поворота вала помогают отслеживать положение суставов. В мобильных роботах датчики угла поворота вала предоставляют данные о количестве оборотов колеса, которые могут использоваться для ▶**одометрии** — измерения пройденного расстояния. К сожалению, колеса имеют тенденцию к боковому скольжению и проскальзыванию, поэтому результаты одометрии будут точными только для коротких дистанций. Внешние силы, такие как ветер или океанские течения, также увеличивают неопределенность в позиционировании. Уменьшить эту неопределенность можно с использованием ▶**инерционных датчиков**, таких как гироскопы, использующих в своей работе сопротивление массы к изменению скорости.

Другие важные аспекты состояния робота измеряются ▶**датчиками силы** и ▶**датчиками крутящего момента**. Это необходимо, когда роботы работают с хрупкими объектами или любыми объектами, точный размер и форма которых неизвестна. Представьте себе робот-манипулятор весом в тонну, вкручивающий лампочку. Очень просто приложить слишком много силы для ее удержания и просто раздавить лампочку. Датчики силы позволяют роботу ощутить, насколько сильно он сжимает колбу лампочки, а датчики крутящего момента — почувствовать, каких усилий достаточно для ее вкручивания. Высококачественные датчики позволяют измерять силу в трех поступательных и трех вращательных направлениях и делают замеры с частотой несколько сот раз в секунду, чтобы робот смог вовремя обнаружить неожиданные изменения силы и исправить ситуацию, прежде чем раздавит лампочку. Однако оснащение робота высококачественными датчиками и необходимой вычислительной мощностью для отслеживания их показаний может оказаться серьезной проблемой.

26.2.3. Выполнение движений

Механизм, инициирующий движение исполнительных устройств, называется ▶**приводом** или актуатором — в качестве примера можно привести трансмиссии, зубчатые передачи, кабели, трубчатые соединения. Наиболее распространенный тип привода — ▶**электропривод**, в котором для раскручивания мотора используется электричество. Электропривод чаще всего используются в таких системах, которые нуждаются во вращательном движении, — как суставы на руке робота. Для получения механического движения в ▶**гидравлических приводах** используется гидравлическая жидкость (например, масло или вода), а в ▶**пневматических приводах** — сжатый воздух, подаваемые под давлением.

Приводы часто используются для приведения в движение соединений, скрепляющих твердые тела (звенья). Такие соединения (или шарниры) имеются в руках и ногах. В шарнирных вращающихся или ▶**поворотных соединениях** одно звено вращается относительно другого, а в шарнирных скользящих или ▶**призматических соединениях** одно звено скользит по другому. Оба эти типа — одноосные соединения (движение возможно только относительно одной оси). Другие виды соединений включают в себя сферические, цилиндрические и плоские соединения, которые являются многоосевыми.

Для взаимодействия с объектами в окружающей среде роботы используют захваты. Основной тип захвата — это ▶**захват с параллельными губками**, с двумя "пальцами" и одним приводом, сдвигающим эти пальцы друг к другу, чтобы схватить предмет. Этот исполнительный механизм одновременно и любят, и ненавидят за его простоту. Трехпалые захваты предоставляют немного больше гибкости при сохранении простоты. На другом конце спектра — гуманоидные (антропоморфные) руки. Например, захват *Shadow Dexterous Hand* имеет в общей сложности 20 приводов. В результате это устройство предлагает гораздо больше гибкости для выполнения сложных манипуляций, в том числе копирования движений руки манипулятора (представьте, что вы поднимаете свой мобильный и вращаете его в руке, чтобы повернуть правой стороной вверх), но эта гибкость имеет свою цену — обучение робота управлению столь универсальными захватами намного сложнее.

26.3. Решение задач в робототехнике

Теперь, когда уже известно, каким может быть аппаратное обеспечение робота, пришло время рассмотреть агентское программное обеспечение, управляющее оборудованием, предназначенным для достижения поставленных целей. Сначала нужно определиться с вычислительной средой этого агента. Ранее в этой книге обсуждались поиск в детерминированных средах, марковские сети принятия решения (MDP) для стохастических, но полностью наблюдаемых сред, частично наблюдаемые марковские процессы принятия решения (POMDP) для частично

наблюдаемых сред и теория игр для ситуаций, в которых агент действует не в полной изоляции. С учетом этой вычислительной среды необходимо создать экземпляры ее компонентов: функции вознаграждения или полезности, состояния, действия, наблюдаемые пространства и т.д.

Ранее отмечалось, что задачи робототехники являются недетерминированными, частично наблюдаемыми и мультиагентными. Используя понятия теории игр из главы 18, можно отметить, что иногда агенты кооперируются для достижения своих целей, а иногда конкурируют друг с другом. В узком коридоре, где только один агент может пойти первым, робот и человек будут сотрудничать, потому что оба не хотят столкнуться друг с другом. Но в иных ситуациях они могут в той или иной степени конкурировать, чтобы быстрее добраться до места назначения. Если робот будет слишком вежлив и всегда будет уступать дорогу, в людных местах он может просто застрять и никогда не достичь своей цели.

Следовательно, когда роботы действуют изолированно и окружающая среда им известна, задачи, которые они решают, могут быть сформулированы в виде марковских процессов принятия решения (MDP); когда информации у них недостаточно, их задачи превращаются в POMDP, а когда они действуют в окружении людей, достижение их целей часто можно сформулировать как игру.

Какова будет функция вознаграждения робота при такой формулировке? Обычно робот действует с целью обслуживания тех или иных потребностей человека; например, доставка еды больному пациенту является вознаграждением для пациента, а не для него. В большинстве случаев применения робототехники разработчики роботов могут попытаться определить достаточно хорошую замещающую функцию вознаграждения, однако истинная функция вознаграждения всегда определяется пользователем, которому робот должен помогать. Поэтому роботу необходимо либо самому расшифровать потребности пользователя, либо положиться на инженера, который определит для него некоторое приближение к истинным потребностям пользователя.

Что касается пространств действий, состояний и наблюдения робота, то наиболее общая форма их представления будет такой, в которой наблюдения являются необработанными сигналами датчиков (например, изображения, поступающие с камер, или лазерные отсчеты, поступающие из лидара), действия представляют собой необработанные величины электрических токов, направляемых в двигатели, а состояния — это все то, что роботу нужно знать для принятия решения. Сказанное означает, что существует огромный разрыв между восприятием низкого уровня или управлением двигателями, и планами высокого уровня, которые робот должен выполнять. Чтобы преодолеть этот разрыв, робототехники разделяют задачу на отдельные аспекты с целью ее упрощения.

Например, мы знаем, что при правильном решении задачи POMDP восприятие и действие должны взаимодействовать: восприятие информирует, какие действия имеют смысл, но и действие информирует восприятие, если агенты предпринимают действия для сбора информации, когда эта информация имеет ценность для

будущих временных этапов. Тем не менее в роботах восприятие часто отделяется от действия и результаты восприятия используются таким образом, как если бы получить больше информации в будущем было невозможно. Чтобы продвинуться дальше, необходимо применять иерархическое планирование, поскольку цель высокого уровня, такая как "добраться до кафетерия", очень далека от команды, направляемой двигателю, — скажем, "повернуть главную ось на 1°".

В робототехнике часто используется трехуровневая иерархия. На уровне ▶**планирования задачи** формируется план или стратегия действий высокого уровня, иногда называемых примитивами действий или подцелями: переместиться к двери, открыть ее, переместиться к лифту, нажать кнопку и т.д. На следующем уровне ▶**планирования движения** требуется найти путь, по которому робот сможет добраться из одной точки в другую, последовательно достигая каждую из заданных подцелей. Наконец на уровне ▶**управления** реализуется выполнение запланированного движения за счет применения исполнительных механизмов робота. Поскольку задачи уровня планирования обычно определяются в терминах отдельных состояний и действий, в этой главе обсуждение будет сосредоточено в первую очередь на планировании движения и управлении.

Независимо от прочих аспектов ▶**изучение предпочтений** отвечает за оценку цели конечного пользователя, а ▶**прогнозирование поведения человека** применяется для предсказания действий других людей, присутствующих в окружающей среде робота. Все это в совокупности и определяет поведение робота.

Всякий раз, разделяя задачу на отдельные части, мы уменьшаем ее сложность, но при этом отказываемся от возможностей отдельных частей в оказании помощи друг другу. Действие может способствовать улучшению восприятия или же определять, какой тип восприятия будет полезен. Аналогичным образом решения на уровне планирования движения могут оказаться не лучшими, если принять во внимание, как будет отслеживаться это движение, а решения на уровне планирования задач могут привести к выработке плана, реализовать который на уровне движения будет просто невозможно. Поэтому прогресс в этих отдельных областях дает толчок в направлении их интеграции: к совместному выполнению планирования движения и контроля, к совместному выполнению планирования задач и планирования движения, а также реинтеграции восприятия, прогнозирования и действия — замыканию петли обратной связи. Сегодня в робототехнике отмечается постоянный прогресс в каждой из этих областей и одновременно этот прогресс является опорой для достижения лучшей интеграции.

26.4. Восприятие в робототехнике

Восприятие в робототехнике — это процесс, в ходе которого роботы отображают результаты измерений датчиков на внутренние структуры представления среды. Во многих случаях для этого используются методы компьютерного зрения,

рассмотренные в предыдущей главе. Однако в робототехнике восприятие часто связано и с другими типами датчиков, такими как лидар или тактильные сенсоры.

Задача восприятия является сложной, поскольку датчики, как правило, в той или иной степени зашумлены, а среда является частично наблюдаемой, непредсказуемой и часто динамической. Другими словами, роботы сталкиваются со всеми проблемами **оценки состояний** (или **фильтрации**), которые обсуждались в разделе 14.2. В качестве эмпирического правила хорошее внутреннее представление для роботов должно обладать следующими тремя свойствами.

1. Содержать достаточно информации, чтобы робот мог принимать правильные решения.
2. Иметь структуру, обеспечивающую их эффективное обновление.
3. Быть естественным в том смысле, что внутренние переменные должны соответствовать естественным переменным состояния в физическом мире.

В главе 14 было показано, что модели перехода и восприятия для частично наблюдаемой среды могут быть представлены с помощью фильтров Калмана, скрытых марковских моделей (HMM) и динамических байесовских сетей. Кроме того, в упомянутой выше главе были описаны как точные, так и приближенные алгоритмы обновления **доверительного состояния**, представленного как распределение апостериорных вероятностей по переменным состояния среды. Также в главе 14 было приведено несколько динамических моделей байесовских сетей для этого процесса. При решении робототехнических задач в модель в качестве наблюдаемых переменных обычно включают собственные прежние действия робота. На рис. 26.4 приведена система обозначений, используемая в данной главе: $\mathbf{X}_t$ — состояние среды (включая робота) во время t, $\mathbf{Z}_t$ — результаты наблюдений, полученные во время t, а A_t — действие, предпринятое после получения этих результатов наблюдения.

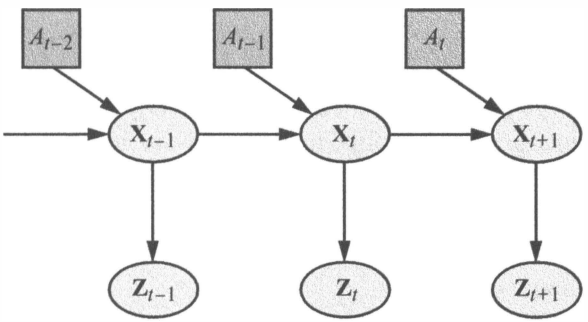


Рис. 26.4. Восприятие робота можно рассматривать как временной вывод из последовательностей действий и измерений, как показано на примере этой динамической сети принятия решений

Требуется вычислить новое доверительное состояние, $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t})$, на основании текущего доверительного состояния $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ и нового наблюдения $\mathbf{z}_{t+1}$. Как это сделать, было показано в разделе 14.2, но в данном случае есть два различия: результат обусловлен не только действиями, но и наблюдениями, и приходится иметь дело с *непрерывными*, а не *дискретными* переменными. Таким образом, необходимо откорректировать рекурсивное уравнение фильтрации (уравнение (14.5) в разделе 14.2.1) для использования в нем интегрирования, а не суммирования:

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t}) = \\ = \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t) P(\mathbf{x}_t \mid \mathbf{z}_{1:t}, a_{1:t-1}) d\mathbf{x}_t . \tag{26.1}$$

Это уравнение показывает, что апостериорное распределение вероятностей по переменным состояния $\mathbf{X}$ во время $t+1$ вычисляется рекурсивно на основании соответствующей оценки, полученной на один временной шаг раньше. В эти вычисления включены данные о предыдущем действии a_t и о текущих показаниях датчиков $\mathbf{z}_{t+1}$. Например, если цель заключается в разработке робота, играющего в футбол, то $\mathbf{X}_{t+1}$ может представлять местонахождение футбольного мяча относительно робота. Распределение апостериорных вероятностей $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ — это распределение вероятностей по всем состояниям, отражающее все, что известно о прежних показаниях датчиков и управляющих воздействиях. Уравнение (26.1) показывает, как рекурсивно оценить это местонахождение, инкрементно развертывая вычисления и включая в этот процесс данные о показаниях датчиков (например, изображения с видеокамеры) и команды управления движением робота. Вероятность $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)$ называется **моделью перехода** или ►**моделью движения**, а вероятность $\mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1})$ является **моделью восприятия**.

26.4.1. Локализация и составление карты

►**Локализация** — это задача определения, где что находится, включая самого робота. Для упрощения рассмотрим мобильного робота, который медленно движется в плоском двухмерном мире. Также предположим, что робот имеет точную карту окружающей среды. (Пример подобной карты приведен на рис. 26.7.) Положение такого мобильного робота определяется двумя декартовыми координатами со значениями x и y, а его ориентация (направление движения) определяется углом θ, как показано на рис. 26.5, *а*. Если поместить эти три значения в вектор, то любое конкретное состояние робота будет определяться вектором $\mathbf{X}_t = (x_t, y_t, \theta_t)^\top$. Пока все идет хорошо.

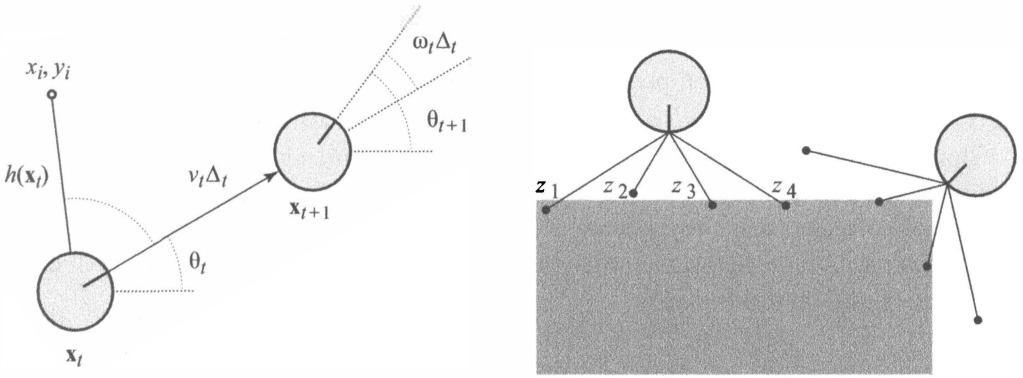
В этой кинематической аппроксимации каждое действие состоит из "мгновенной" спецификации двух скоростей — скорости поступательного движения v_t и скорости вращательного движения ω_t. Для небольших временных интервалов Δt грубая детерминированная модель движения таких роботов определяется как

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta t \end{pmatrix}.$$

Обозначение $\hat{\mathbf{X}}$ относится к детерминированному предсказанию состояния. Безусловно, поведение физических роботов является довольно непредсказуемым, поэтому обычно оно моделируется гауссовым распределением со средним $f(\mathbf{X}_t, v_t, \omega_t)$ и ковариацией Σ_x (математическое определение приведено в приложении А).

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t,\ v_t,\ \omega_t) = \mathcal{N}(\hat{\mathbf{X}}_{t+1}, \Sigma_x).$$

Это распределение вероятностей является моделью движения робота. Оно моделирует эффекты движения a_t на расположение робота.



а) б)

Рис. 26.5. а) Упрощенная кинематическая модель мобильного робота. Робот представлен в виде окружности с линией радиуса внутри, указывающей направление его движения. Состояние $\mathbf{x}_t$ состоит из координат расположения (x_t, y_t) (показано неявно) и угла ориентации θ_t. Новое состояние $\mathbf{x}_{t+1}$ получается посредством обновления координат расположения на величину $v_t\Delta t$ и угла ориентации на величину $\omega_t\Delta t$. На рисунке также показана отметка в точке (x_i, y_i), которая наблюдается в момент времени t. **б)** Модель датчика расстояния. Представлены два возможных расположения робота при заданном диапазоне сканирования (z_1, z_2, z_3, z_4). Гораздо более вероятно, что эти результаты измерения расстояний были получены в расположении слева, а не в расположении справа

Далее необходимо разработать модель восприятия. Рассмотрим модели восприятия двух типов. В первой предполагается, что датчики обнаруживают *стабильные*, *различимые* характеристики среды, называемые ▶**отметками** (*landmark*).

Для каждой отметки они сообщают дальность и азимут. Предположим, что состояние робота определяется выражением $\mathbf{x}_t = (x_t, y_t, \theta_t)^\top$ и он воспринимает наличие отметки, местонахождение которой, как известно, определяется координатами $(x_i, y_i)^\top$. При отсутствии шума дальность и азимут можно вычислить с помощью простого геометрического соотношения (см. рис. 26.5, *а*):

$$\hat{\mathbf{z}} = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \operatorname{arctg}\dfrac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}.$$

Еще раз отметим, что полученные результаты измерений искажаются шумом. Для простоты предположим, что это гауссов шум с ковариацией Σ_x, что дает модель восприятия

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \mathcal{N}(\hat{\mathbf{z}}_t, \Sigma_z).$$

Несколько иная модель восприятия используется для ▶**матрицы датчиков расстояния**, каждый из которых имеет фиксированный азимут относительно робота. Такие матрицы датчиков выдают вектор значений расстояний $\mathbf{z}_t = (z_1, \ldots, z_M)^\top$.

При заданном расположении $\mathbf{x}_t$ пусть $\hat{z}_j$ — это вычисленное расстояние вдоль направления j-го луча от $\mathbf{x}_t$ до ближайшего препятствия. Как и ранее, эти результаты могут быть искажены гауссовым шумом. Обычно предполагается, что погрешности для различных направлений лучей независимы и заданы в виде идентичных распределений, поэтому имеем

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \alpha \prod_{j=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2}.$$

На рис. 26.5, *б* приведен пример четырехлучевого дальномера и двух возможных расположений робота, одно из которых на полном основании можно рассматривать как расположение, в котором были получены рассматриваемые результаты измерения дальностей, а другое — нет. Сравнивая модель измерения дальностей с моделью отметок, можно убедиться, что модель измерения дальностей обладает преимуществом в том, что не требует *идентификации* отметки для получения возможности интерпретировать результаты измерения дальностей. И действительно, как показано на рис. 26.5, *б*, робот направляется в сторону стены, не имеющей характерных особенностей. С другой стороны, если бы перед ним *была* видимая, четко идентифицируемая отметка, то робот мог бы выполнить немедленную локализацию.

В разделе 14.4 описан фильтр Калмана, позволяющий представить доверительное состояние в виде единственного многомерного гауссова распределения, и фильтр частиц, который представляет доверительное состояние в виде коллекций частиц, соответствующих состояниям. В большинстве современных алгоритмов

локализации используется одно из этих двух представлений доверительного состояния робота $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$.

Локализация с использованием фильтрации частиц называется ▶**локализацией Монте-Карло**, (*Monte Carlo Localization* — MCL). Алгоритм MCL идентичен алгоритму фильтрации частиц, приведенному на рис. 14.17 в разделе 14.5.3, — достаточно лишь предоставить подходящую модель движения и модель восприятия. Одна из версий алгоритма, в которой используется модель восприятия результатов измерения дальности, приведена на рис. 26.6. Работа этого алгоритма продемонстрирована на рис. 26.7, где показано, как робот определяет свое местонахождение в офисном здании. На рис. 26.7, *а* частицы распределены равномерно согласно распределению априорных вероятностей, указывающему на наличие глобальной неопределенности в отношении положения робота. На рис. 26.7, *б* показано, что поступил первый ряд результатов измерений и частицы формируют кластеры в областях с высоким распределением апостериорных доверительных состояний. На рис. 26.7, *в* показано, что поступило достаточное количество результатов измерений, чтобы переместить все частицы в одно место.

---

function MONTE-CARLO-LOCALIZATION(a, z, N, $P(X' \mid X, v, \omega)$, $P(z \mid z^*)$, *map*)
  returns множество выборок S для следующего временного этапа
  inputs: a, скорости робота v и ω
    z, вектор результатов измерения дальностей M
    $P(X' \mid X, v, \omega)$, модель движения
    $P(z \mid z^*)$, модель шума для датчика расстояний
    map, двухмерная карта окружающей среды
  persistent: S, вектор выборок размером N
  local variables: W, вектор весов размером N
    S', временный вектор выборок размером N

  if S пусто **then**
    for $i = 1$ **to** N **do** // фаза инициализации
      $S[i] \leftarrow$ выборка из $P(X_0)$
  for $i = 1$ **to** N **do** // цикл обновления
    $S'[i] \leftarrow$ выборка из $P(X' \mid X = S[i], v, \omega)$
    $W[i] \leftarrow 1$
    for $j = 1$ **to** M **do**
      $z^* \leftarrow$ RAYCAST($j, X = S'[i]$, *map*)
      $W[i] \leftarrow W[i] \cdot P(z_j \mid z^*)$
    $S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S', W)
  return S

Рис. 26.6. Алгоритм локализации Монте-Карло, в котором используется модель восприятия результатов измерения дальностей при наличии независимого шума

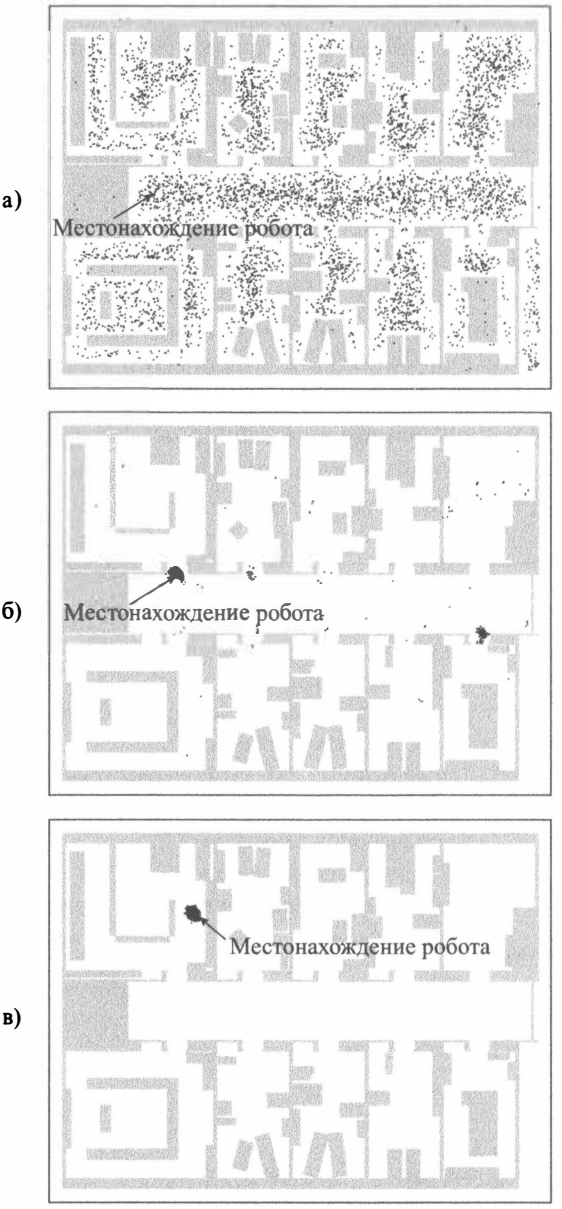


Рис. 26.7. Пример локализации по методу Монте-Карло, построенной на применении алгоритма фильтрации частиц, для определения местонахождения мобильного робота. **а)** Первоначальное состояние, характеризующееся глобальной неопределенностью. **б)** Приблизительно бимодальное состояние неопределенности после прохождения по (симметричному) коридору. **в)** Унимодальное состояние неопределенности после перехода в помещение, отличимое от других помещений

Еще один важный способ локализации основан на применении фильтра Калмана. В фильтре Калмана апостериорная вероятность $\mathbf{P}(\mathbf{X}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})$ представляется с помощью гауссова распределения. Обозначим среднее этого гауссова распределения как μ_t, а его ковариацию — как Σ_t. Основным недостатком использования гауссовых доверительных состояний является то, что они замкнуты только при использовании линейных моделей движения f и линейных моделей измерения h. Для нелинейных моделей f и h результат обновления фильтра в общем случае уже не будет гауссовым. Следовательно, алгоритмы локализации, в которых используется фильтр Калмана, ▶ **линеаризуют** модели движения и восприятия. Линеаризацией называется локальная аппроксимация нелинейной функции с помощью линейной. На рис. 26.8 показано применение концепции линеаризации к (одномерной) модели движения робота. На рис. 26.8, *а* показана нелинейная модель движения $f(\mathbf{x}_t, a_t)$ (параметр a_t на этом графике не показан, поскольку он не играет никакой роли при линеаризации). На рис. 26.8, *б* эта функция аппроксимируется линейной функцией $\tilde{f}(\mathbf{x}_t, a_t)$. График этой линейной функции проходит по касательной к кривой f в точке μ_t, определяющей среднее значение оценки состояния во время t. Такая линеаризация называется ▶ **разложением в ряд Тейлора** первой степени. Фильтр Калмана, линеаризующий функции f и h с помощью разложения в ряд Тейлора, называется ▶ **расширенным фильтром Калмана** (*Extended Kalman Filter* — EKF). На рис. 26.9 показана последовательность оценок, полученных роботом, действующим под управлением алгоритма локализации на основе расширенного фильтра Калмана.

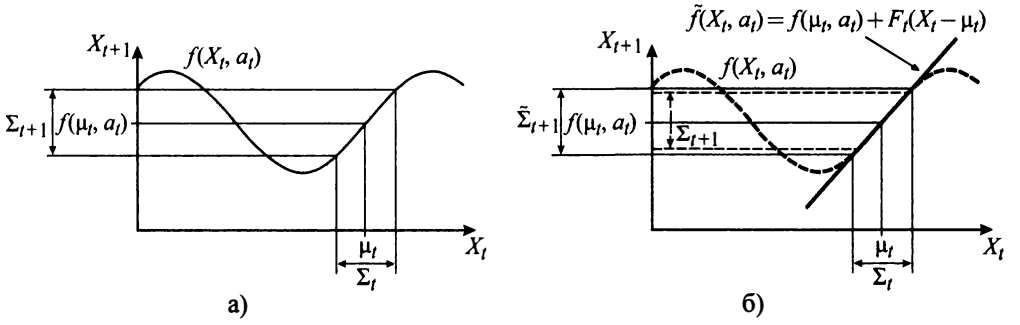


Рис. 26.8. Одномерная иллюстрация линеаризованной модели движения. **а)** Функция f, проекция среднего μ_t и интервал ковариации (основанный на Σ_t) во время $t+1$. **б)** Линеаризованная версия представляет собой касательную к кривой функции f при значении μ_t. Проекция среднего μ_t определена правильно. Однако проекция ковариации $\tilde{\Sigma}_{t+1}$ отличается от Σ_{t+1}

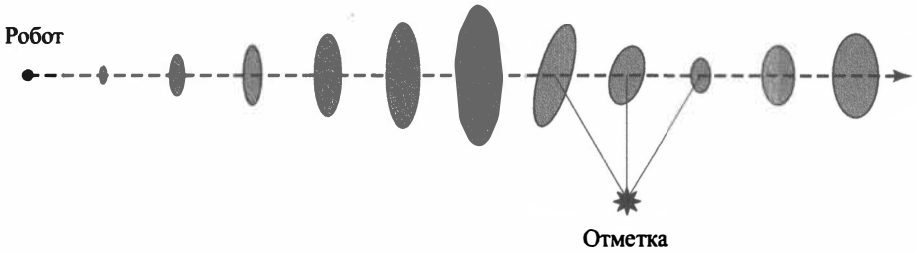


Рис. 26.9. Пример локализации с использованием расширенного фильтра Калмана. Робот движется по прямой. По мере его продвижения неопределенность постепенно возрастает, как показано с помощью эллипсов погрешностей. Однако после обнаружения роботом отметки с известной позицией неопределенность в его местоположении уменьшается

По мере передвижения робота неопределенность в оценке его местонахождения возрастает, как показано с помощью эллипсов погрешностей. Когда робот начинает получать данные о дальности и азимуте до отметки с известным местонахождением, его погрешность уменьшается, но затем снова начинает возрастать, как только робот теряет отметку из виду. Алгоритмы EKF действуют успешно, когда отметки легко идентифицируются. В противном случае распределение апостериорных вероятностей может оказаться мультимодальным, как показано на рис. 26.7, *б*. Задача, для решения которой требуется знать идентификацию отметок, представляет собой пример задачи **ассоциации данных**, которая обсуждалась в разделе 15.1.1 (рис. 15.3).

В некоторых ситуациях карта окружающей среды недоступна и тогда роботу приходится строить эту карту самому. Данная задача слегка напоминает проблему курицы и яйца: навигационному роботу необходимо определить свое местоположение на карте, которая ему абсолютно неизвестна, и в то же время он должен создать эту карту, хотя даже не знает, где он на самом деле находится. Эта задача очень важна для многих случаев применения роботов и широко изучалась под названием ►**одновременная локализация и построение карты** (*simultaneous localization and mapping* — ►**SLAM**).

Задачи SLAM можно решать с использованием множества различных вероятностных методов, включая расширенный фильтр Калмана, рассмотренный выше. Использовать метод EKF несложно: просто увеличиваем вектор состояния для включения расположения отметок в окружающей среде. К счастью, объем обновлений в EKF масштабируется квадратично, поэтому для небольших карт (т.е. до нескольких сотен отметок) необходимые вычисления вполне осуществимы. Более сложные карты часто получают с помощью методов ослабления графа, схожих с методами байесовского логического вывода, обсуждавшимися в главе 13. Решать задачи SLAM можно также с помощью метода ожидания-максимизации.

26.4.2. Другие типы восприятия

Не все инструменты восприятия роботов предназначены для локализации и составления карт. Роботов наделяют также способностями воспринимать температуру, запахи, звуки и т.д. Многие из этих измеряемых величин могут оцениваться с использованием различных вариантов динамических байесовских сетей. Все, что требуется для такой оценки, — это распределения условных вероятностей, характеризующих эволюцию переменных состояния во времени, и модели восприятия, описывающие связь между результатами измерений и переменными состояния.

Также можно запрограммировать робота в качестве реактивного агента, работающего без явных рассуждений о распределении вероятностей по состояниям. Этот подход будет рассмотрен в разделе 26.9.1.

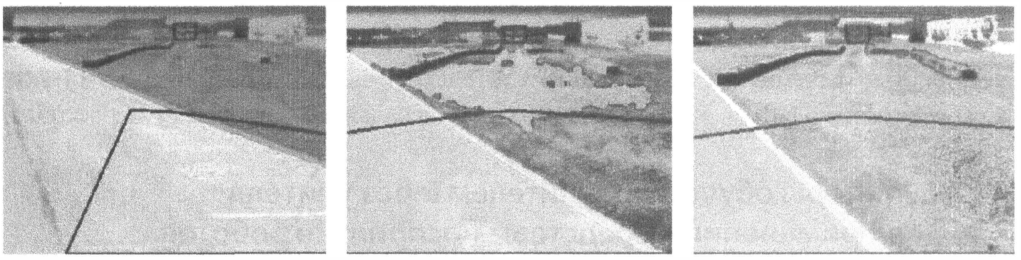
В робототехнике явно прослеживается тенденция к использованию представлений с полностью определенной семантикой. Вероятностные методы превосходят другие подходы во многих сложных задачах восприятия, таких как локализация и картирование. Тем не менее статистические методы иногда оказываются слишком громоздкими, поэтому на практике столь же эффективными могут оказаться и более простые решения. Самым лучшим учителем, позволяющим понять, какой подход будет оптимальным, является опыт работы с реальными физическими роботами.

26.4.3. Методы обучения с учителем и без учителя в применении к средствам восприятия роботов

В восприятии робота важную роль играет машинное обучение. Это утверждение особенно верно, когда лучшее внутреннее представление неизвестно. Один из популярных подходов заключается в отображении многомерных потоков восприятия в пространства меньшего размера с использованием методов машинного обучения без учителя (см. главу 19). Такой подход называется ▶**низкоразмерным встраиванием** (*low-dimensional embedding*). Машинное обучение позволяет обучать датчики и модели движения на основании входных данных, одновременно находя для них подходящее внутреннее представление.

Еще один метод машинного обучения позволяет роботам постоянно адаптироваться к большим изменениям в показаниях датчика. Представьте, что из освещенного солнцем пространства вы заходите в темную комнату, освещенную неоновой лампой. Понятно, что внутри все покажется значительно темнее. Но смена источника света повлияет и на все цвета: свет неоновой лампы имеет гораздо более сильную составляющую в диапазоне зеленого цвета в сравнении с солнечным светом. Однако каким-то образом мы, люди, кажется, вовсе не замечаем таких изменений. Когда мы заходим с улицы в освещенную неоновым светом комнату вместе с другими людьми, нам вовсе не кажется, что их лица вдруг позеленели. Наше восприятие быстро адаптируется к новым условиям освещения, и наш мозг просто игнорирует подобные различия.

Адаптивные методы восприятия позволяют роботам приспосабливаться к подобным изменениям. Типичный пример приведен на рис. 26.10, взятом из области управления беспилотным транспортом. Здесь беспилотный наземный автомобиль адаптирует свой классификатор к понятию "поверхность, подходящая для проезда". Как это работает? Сначала робот использует лазер для классификации небольшого участка поверхности непосредственно перед ним. Если лазерный дальномер воспринимает эту область как плоскую, она используется как положительный обучающий пример для концепции "поверхность, подходящая для проезда". Далее, метод смешения гауссианов, аналогичный EM-алгоритму, обсуждавшемуся в главе 20, применяется для обучения классификатора распознаванию определенного цвета и коэффициентов текстур на основании небольших пакетов выборок. Изображения на рис. 26.10 являются результатом применения подобного классификатора к полному изображению.



а) б) в)

Рис. 26.10. Последовательность классификаций "поверхность, подходящая для проезда" с использованием адаптивного видения. **а)** Как поверхность, подходящая для проезда, сейчас классифицируется только дорога (светло-серая область). Здесь Λ-образная темная линия показывает направление, в котором движется автомобиль. **б)** Транспортному средству предписано съехать с дороги, и классификатор начинает классифицировать некоторую часть бокового травяного покрытия как подходящую для проезда. **в)** Транспортное средство обновило свою модель "поверхности, подходящей для проезда", чтобы наряду с дорогой включить в нее и травяное покрытие

Подобный подход, заставляющий роботов собирать собственные обучающие данные (с метками!), называют ►**самообучением**. В данном случае робот использует машинное обучение, чтобы превратить датчик ближнего действия, хорошо подходящий для классификации местности, в датчик, способный видеть намного дальше. Это позволит роботу двигаться быстрее, замедляясь только в тех случаях, когда модель восприятия указывает на такие изменения на поверхности земли, которые следует более тщательно исследовать датчиками ближнего действия.

26.5. Планирование и управление

Размышления робота в конечном итоге сводятся к принятию решения, как следующее действие будет выполняться — от абстрактного уровня задачи и до величины управляющих токов, которые необходимо подать на двигатели. В этом разделе столь сложная задача будет упрощена за счет предположения, что данные восприятия (и, где это необходимо, прогноз) уже получены, поэтому мир наблюдаем. Также предположим, что переходы в мире (динамика) детерминированы.

Начнем с отделения движения от управления. Определим ▶ **путь** как последовательность точек в геометрическом пространстве, по которым будет следовать робот (или часть робота, например рука). Это определение связано с понятием пути, использовавшимся в главе 3, но здесь имеется в виду именно последовательность точек в пространстве, а не последовательность дискретных действий. Задача поиска хорошего пути называется ▶ **планированием движения** (*motion planning*).

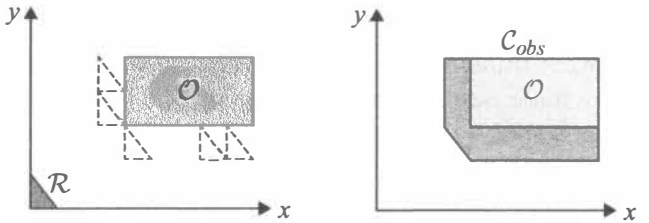
Когда путь определен, остается решить задачу выполнения последовательности действий для следования по этому пути — задачу ▶ **управления прохождением траектории** (*trajectory tracking control*). ▶ **Траектория** — это путь, с каждой точкой которого связана временн*ая* отметка. Путь — всего лишь описание последовательности действий: "Идти от точки A к точке B, затем — к точке C и т.д.", тогда как траектория — это цепочка указаний о выполняемых действиях во времени: "Начать из точки A, потратить 1 секунду, чтобы добраться до точки B, затем — еще 1,5 секунды, чтобы добраться до точки C, и т.д."

26.5.1. Пространство конфигураций

Представим простого робота $\mathcal{R}$, имеющего форму прямоугольного треугольника, — на рис. 26.11, *а* он изображен как темно-серый треугольник в нижнем левом углу. Робот должен выработать план такого пути, который не проходит через прямоугольник $\mathcal{O}$. Физическое пространство, в котором движется робот, называется ▶ **рабочим пространством**. Этот конкретный робот может двигаться в любом направлении в плоскости x–y, но не может поворачиваться вокруг своей оси. На рис. 26.11, *а* пунктиром показано пять других возможных положений робота, каждое из которых находится настолько близко к препятствию $\mathcal{O}$, насколько это допустимо.

Тело робота можно представить в виде множества точек (x, y) (или точек (x, y, z) для трехмерного робота), как и препятствие $\mathcal{O}$. При таком представлении избегать препятствия означает, что ни одна точка робота не должна перекрывать какую-либо точку препятствия. Планирование движения в подобном случае требует вычислений на основе множеств точек, которые могут быть сложными и трудоемкими.

Вычисления можно упростить, если использовать такую схему представления, в которой все точки, входящие в состав робота, представлены как единственная точка в абстрактном многомерном пространстве, которое называется ▶ **пространством конфигураций** или ▶ **C-пространством**. Идея состоит в том, что все



а) б)

Рис. 26.11. Простой робот в форме треугольника, способный свободно перемещаться в пространстве, но при этом он должен избегать прямоугольного препятствия. **а)** Рабочее пространство. **б)** Пространство конфигурации

множество точек, образующих тело робота, может быть вычислено, если известны, во-первых, основные измерения робота (для нашего треугольного робота это могут быть длины трех его сторон) и, во-вторых, текущее **положение** (*pose*) робота, т.е. его местоположение в пространстве и ориентация.

Для нашего простого треугольного робота достаточно С-пространства с двумя измерениями: если известны координаты (x, y) конкретной точки робота — в данном случае будем использовать вершину прямого угла, — то можно вычислить, где находится любая другая точка треугольника (поскольку размер и форма треугольника известны и треугольник не может вращаться). На рис 26.11, *а* темно-серый треугольник можно представить конфигурацией (0, 0).

Если изменить правила и позволить роботу вращаться, то, чтобы можно было рассчитать, где находится каждая его точка, понадобится уже три измерения: (x, y, θ). Здесь θ — это угол поворота робота вокруг его оси в плоскости x–y. Если роботу дополнительно предоставить способность растягиваться и сжиматься, равномерно изменяя свои размеры в соответствии с коэффициентом масштабирования s, то пространство конфигураций будет иметь уже четыре измерения (x, y, θ, s).

Пока будем придерживаться простого двухмерного пространства конфигураций для не вращающегося треугольного робота с постоянными размерами. Следующая задача — выяснить, где в С-пространстве находятся точки препятствия $\mathcal{O}$. Рассмотрим пять пунктирных треугольников, приведенных на рис. 26.11, *а*; обратите внимание, где у каждого из них находится вершина прямого угла. Теперь представим, что треугольник может скользить по плоскости x–y. Понятно, что вершина прямого угла не может войти в область препятствия, — более того, она не может подойти к нему ближе, чем показано для любого из пяти пунктирных треугольников. Следовательно, можно отметить, что область, в которой не может оказаться вершина прямого угла — представление ▶**препятствия в С-пространстве**, — это пятисторонний многоугольник C_{obs}, приведенный на рис. 26.11, *б*.

В повседневной жизни можно говорить о наличии множества препятствий для робота — стол, стул, стены и т.д. Однако математическое представление будет немного проще, если понимать все это как единственное объединенное "препятствие", которое может включать и разрозненные компоненты. В общем случае препятствием в C-пространстве является набор всех точек q в C, таких, что если поместить робота в эту конфигурацию, то в рабочем пространстве его геометрия будет пересекаться с геометрией препятствия.

Пусть препятствиями в рабочем пространстве будут множество точек O, и пусть множество всех точек в конфигурации робота q будет $\mathcal{A}(q)$. Тогда препятствие в C-пространстве можно определить как

$$C_{obs} = \{q : q \in C \text{ и } \mathcal{A}(q) \cap O \neq \{\}\},$$

а ▶**свободное пространство** определяется как $C_{free} = C - C_{obs}$.

Пространство конфигураций становится более интересным для роботов с движущимися частями. Рассмотрим роботизированную руку с двумя шарнирами, представленную на рис. 26.12, *а*. В целом этот робот прикреплен к столу так, чтобы его основание не двигалось, но рука имеет два шарнира, движения которых выполняются независимо друг от друга, — это называют ▶**степенями свободы**

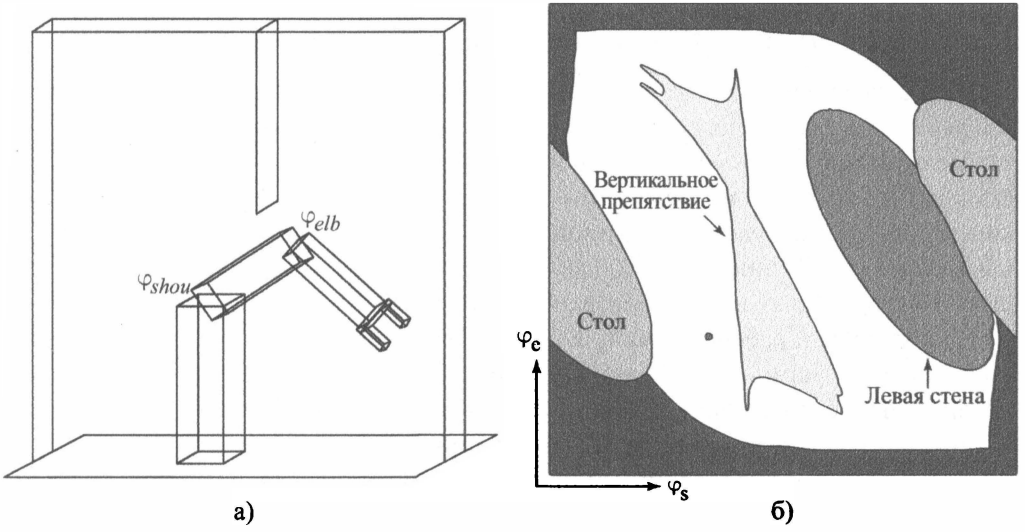


Рис. 26.12. а) Представление рабочего пространства роботизированной руки с двумя степенями свободы. Рабочее пространство представляет собой узкий бокс с плоским препятствием, свисающим с потолка. **б)** Пространство конфигурации для того же робота. В этом пространстве только белые области представляют конфигурации, в которых нет столкновений руки с препятствиями. На этой диаграмме точка в белой области соответствует конфигурации робота, показанной слева

(*degrees of freedom* — ▶**DOF**). Движение в шарнирах вызывает изменение координат (x, y) локтя, захвата и каждой точки руки. Пространство конфигурации этой роботизированной руки является двухмерным: $(\theta_{shou}, \theta_{elb})$, где θ_{shou} — угол плечевого шарнира, а θ_{elb} — угол локтевого шарнира.

При известной конфигурации этой двухзвенной роботизированной руки с помощью простой тригонометрии можно определить, где каждая ее точка находится в рабочем пространстве. В общем случае под ▶**прямой кинематикой** понимают функцию вида

$$\phi_b : C \rightarrow W,$$

которая на входе принимает конфигурацию и выводит местоположение конкретной точки b на роботе, когда робот находится в этой конфигурации. Особенно полезным прямое кинематическое отображение будет для конечного исполнительного элемента робота, ϕ_{EE}. Множество всех точек робота в конкретной конфигурации q обозначается как $\mathcal{A}(q) \subset W$:

$$\mathcal{A}(q) = \bigcup_b \{\phi_b(q)\}.$$

Обратная задача, состоящая в отображении желаемого местоположения точки на роботе на конфигурацию (конфигурации), в которой робот должен находиться, чтобы это произошло, называется ▶**обратной кинематикой**:

$$IK_b : x \in W \mapsto \{q \in C \text{ при условии, что } \phi_b(q) = x\}.$$

Иногда функция отображения обратной кинематики может принимать в качестве входа не только позицию, но и желаемую ориентацию. Например, если необходимо, чтобы манипулятор захватил объект, можно вычислить желаемое положение и ориентацию его захвата для достижения этой цели, а затем воспользоваться обратной кинематикой для определения целевой конфигурации робота. После этого планировщику нужно будет лишь найти способ перевести робота из его текущей конфигурации в конфигурацию достижения цели без столкновения с препятствиями.

Препятствия в рабочем пространстве часто изображаются в виде простых геометрических форм, — особенно в учебниках по робототехнике, в которых, как правило, преобладает ориентация на полигональные препятствия. Но как выглядят препятствия в пространстве конфигурации?

Для рычага с двумя шарнирами простые препятствия в рабочей области, такие как вертикальная линия, будут иметь очень сложное представление в пространстве конфигураций, как показано на рис. 26.12, *б*. Отдельные области пространства конфигураций с заливкой разными оттенками серого цвета соответствуют разным объектам в рабочем пространстве робота: темная область, окружающая все свободное пространство, соответствует конфигурациям, в которых робот сталкивается с самим собой. Легко видеть, что такое столкновение вызывается экстремальными значениями углов его плечевого и локтевого шарниров. Два участка

овальной формы по обе стороны от робота соответствуют столу, на котором смонтирован робот. Третий овальный участок с более темной заливкой соответствует левой стене.

Наконец, наиболее интересным объектом в пространстве конфигураций является простое вертикальное препятствие, свисающее с потолка и мешающее движениям робота. Этот объект имеет любопытную форму: он чрезвычайно нелинеен, а в некоторых местах является даже вогнутым. Приложив немного воображения, читатель легко узнает форму захвата на верхнем левом конце манипулятора.

Рекомендуем читателю немного задержаться и внимательно изучить эту важную схему. Форма данного препятствия в пространстве конфигураций не так уж очевидна! Точка внутри свободного пространства на рис. 26.12, *б* отмечает конфигурацию робота, представленную на рис. 26.12, *а*. На рис. 26.13 приведены три дополнительные конфигурации как в рабочем пространстве, так и в пространстве конфигураций. В конфигурации "Конф.-1" захват окружает вертикальное препятствие.

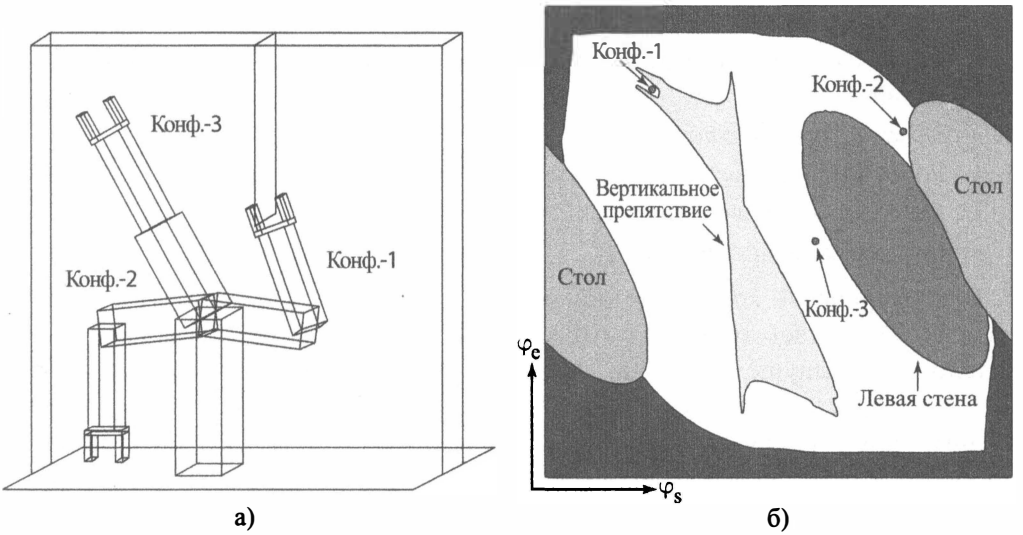


Рис. 26.13. Три конфигурации робота, представленные в рабочем пространстве и в пространстве конфигураций

Как видите, даже если рабочее пространство робота представлено плоскими многоугольниками, в пространстве конфигураций форма свободного пространства может оказаться очень сложной. Поэтому на практике обычно применяется *ощупывание* пространства конфигураций вместо явного его построения. Планировщик может вырабатывать конфигурацию, а затем проверять ее для определения того, находится ли она в свободном пространстве, применяя кинематику робота и определяя наличие столкновений в различных координатах рабочего пространства.

26.5.2. Планирование движений

Задача ▶**планирования движения** — это разработка плана, следуя которому, робот сможет перейти от одной конфигурации к другой без столкновения с препятствиями. Это основной конструкционный элемент при реализации движения и манипуляции. В разделе 26.5.4 рассматривается, как это можно делать в случае задач со сложной динамикой, подобной рулевому управлению автомобилем, который может сойти с запланированной траектории, если слишком быстро поворачивать руль. Ну а пока сосредоточимся на простой задаче планирования движения, заключающейся в нахождении геометрии пути, свободного от столкновений с препятствиями. Планирование движения — это типичная **задача поиска** в непрерывном пространстве состояний, но часто это пространство можно дискретизировать, а затем применить любой из алгоритмов поиска, обсуждавшихся в главе 3.

Задачу планирования движения иногда называют ▶**задачей переноса рояля**. Это название указывает на ее сходство с проблемой, стоящей перед грузчиками при переносе большого тяжелого рояля неправильной формы из одной комнаты в другую так, чтобы не ударить его обо что-нибудь. Нам известно следующее.

- *Мир* рабочего пространства W в $\mathbb{R}^2$ для плоскости или в $\mathbb{R}^3$ для трех измерений.
- *Область препятствия* $O \subset W$.
- Робот с пространством конфигураций C и множеством точек $\mathcal{A}(q)$ для $q \in C$.
- Исходная конфигурация $q_s \in C$.
- Целевая конфигурация $q_g \in C$.

В пространстве конфигураций область препятствия индуцирует препятствие C_{obs} и соответствующее ему свободное пространство C_{free}, как было определено в предыдущем разделе. Необходимо найти непрерывный **путь** через свободное пространство. Для представления этого пути будем использовать параметризованную кривую $\tau(t)$, где $\tau(0) = q_s$ и $\tau(1) = q_g$, а $\tau(t)$ для каждого t между 0 и 1 является некоторой точкой в пространстве C_{free}. Таким образом, переменная t параметризует, насколько далеко находится точка на пути от исходного состояния до целевого. Обратите внимание, что переменная t чем-то напоминает время, поскольку значение t увеличивается с увеличением расстояния, пройденного вдоль пути, но при этом t всегда остается точкой в интервале [0, 1] и не измеряется в секундах.

Задачу планирования движения можно усложнить разными способами: определением цели как множества возможных конфигураций, а не единственной конфигурацией, определением цели в рабочем пространстве, а не в пространстве конфигураций, определением функции стоимости (например, длины пути), которая должна быть минимизирована, и также требованием удовлетворения ограничений (например, если путь определяет перенос чашки кофе, гарантировать, что чашка всегда будет ориентирована вертикально, чтобы кофе не пролился).

Пространства планирования движения. Сделаем шаг назад и убедимся, что пространства, используемые в процессе планировании движения, понимаются правильно. Прежде всего, есть рабочее пространство или мир W. Точки в пространстве W представляют собой точки в нашем обычном трехмерном мире. Далее, есть пространство конфигураций C. Точки q в пространстве C являются d-мерными, где d — число степеней свободы робота, и отображаются на множество точек $\mathcal{A}(q)$ в W. Наконец, существует пространство путей. Пространство путей является пространством функций. Каждая точка в этом пространстве отображается на всю кривую пути в пространстве конфигураций. Это пространство ∞-мерно! Интуитивно понятно, что необходимо d измерений для каждой конфигурации вдоль пути, а на пути имеется столько конфигураций, сколько точек в интервале [0, 1] числовой оси. Теперь давайте рассмотрим некоторые способы решения задачи планирования движения.

Графы видимости

Для упрощенного случая двухмерных пространств конфигурации и полигонального пространства конфигураций препятствий построение ▶**графов видимости** — удобный способ решения задачи планирования движения с решением в виде гарантировано кратчайшего пути. Пусть $V_{obs} \subset C$ — это множество вершин многоугольников, образующих препятствие C_{obs}, и пусть $V = V_{obs} \cup \{q_s, q_g\}$.

Построим граф $G = (V, E)$ на множестве вершин V с ребрами $e_{ij} \in E$, соединяющими вершину v_i с другой вершиной v_j, если линия, соединяющая две вершины, свободна от столкновений, т.е. если $\{\lambda v_i + (1 - \lambda)v_j : \lambda \in [0, 1]\} \cap C_{obs} = \{\}$. Если это так, то говорят, что данные две вершины "могут видеть друг друга", — именно по этой причине данный вид графов получил свое название.

Чтобы решить задачу планирования движения, все, что в данном случае нужно сделать, — это выполнить поиск в дискретном графе (например, по методу жадного поиска по первому наилучшему совпадению) по графу G с начальным состоянием q_s и целью q_g. На рис. 26.14 приведен пример графа видимости и найденное оптимальное трехэтапное решение (утолщенная серая линия). Оптимальный поиск по графу видимости в результате всегда будет давать оптимальный путь (если он существует) или сообщать об ошибке, если такого пути не существует.

Диаграммы Вороного

Графы видимости склоняют к выбору путей, проходящих в непосредственной близости от препятствия, — если необходимо обойти стол, чтобы добраться до двери, кратчайшим путем будет двигаться настолько близко к столу, насколько это возможно. Однако, когда движение или восприятие недетерминировано, такое решение будет связано с риском наткнуться на стол. Один из способов решения этой проблемы — принять, что тело робота немного больше, чем на самом деле, что

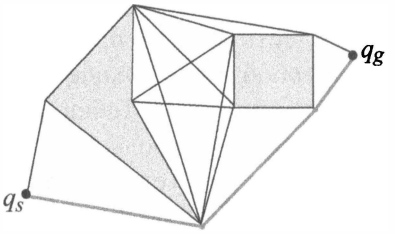


Рис. 26.14. Граф видимости. Линии соединяют каждую пару вершин, которые могут "видеть" друг друга; это линии, которые не проходят через препятствие. Кратчайший путь должен лежать на этих линиях

обеспечит необходимую буферную зону. Другой способ состоит в принятии решения, что длина пути не является единственным показателем, который требуется оптимизировать. В разделе 26.8.2 показано, как можно подобрать хорошую метрику, исходя из примеров поведения человека.

Третий способ — воспользоваться иным приемом, располагая пути как можно дальше от препятствий, вместо того чтобы прижиматься к ним. ►**Диаграмма Вороного** является именно таким представлением, позволяющим решить эту задачу. Чтобы понять, что представляет собой диаграмма Вороного, рассмотрим такое рабочее пространство, в котором препятствиями являются, скажем, дюжина маленьких точек, разбросанных по плоскости. Теперь окружим каждую из этих точек препятствий некоторой областью — ►**ячейкой Вороного**, — состоящей из всех точек на плоскости, которые ближе к данной точке препятствия, чем к любой другой из точек препятствий. В результате вся плоскость будет поделена между дюжиной ячеек, связанных с отдельными точками препятствий. Диаграмма Вороного состоит из множества таких ячеек, а ►**граф Вороного** состоит из ребер и вершин этих ячеек.

Когда препятствия являются областями, а не точками, все остается практически неизменным. Каждая ячейка по-прежнему содержит все точки, находящиеся ближе всех к одному препятствию, чем к любому другому, при этом расстояние измеряется до ближайшей точки на препятствии. Границы между ячейками по-прежнему соответствуют точкам, которые находятся на одинаковом расстоянии между двумя препятствиями, но теперь граница будет представлена некоторой кривой, а не прямой линией. В многомерных пространствах вычисление таких границ может оказаться чрезмерно дорогостоящим. На рис. 26.15 приведен пример диаграммы Вороного для пространства конфигураций робота, приведенного на рис. 26.12, *б*. Она представляет собой геометрическое место всех точек, равноудаленных от двух или более препятствий.

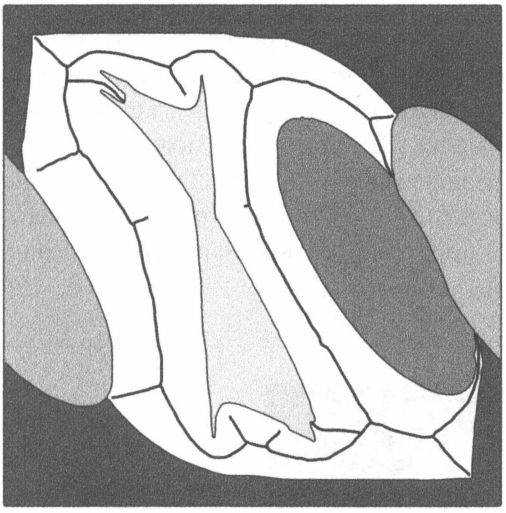


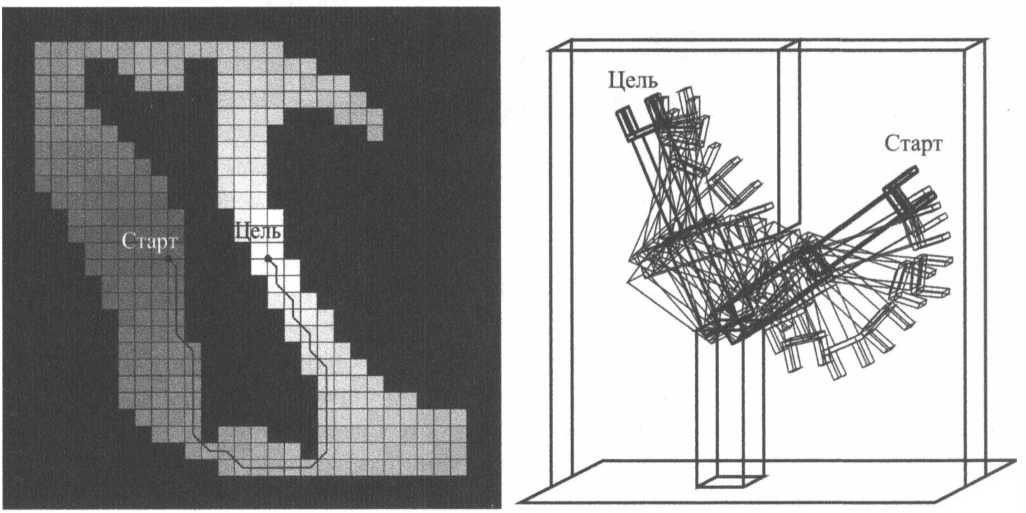
Рис. 26.15. Диаграмма Вороного — это геометрическое место точек (черная линия), равноудаленных от двух или более препятствий в пространстве конфигураций

Чтобы решить задачу планирования движения, соединяем прямой линией начальную точку q_s с ближайшей точкой на графе Вороного, а затем делаем то же самое для точки цели q_g. Далее, для нахождения кратчайшего пути по графу Вороного используется обычный поиск в дискретном графе. Для таких задач, как навигация по коридорам в помещении, этот метод обеспечивает хороший путь, проходящий по центру коридоров. Однако в уличных условиях этот метод может привести к выбору неэффективных путей, например предлагать излишний 100-метровый обходной маршрут, чтобы при движении строго придерживаться середины широко открытого 200-метрового пространства.

Клеточная декомпозиция

Альтернативный подход к планированию движения заключается в дискретизации пространства конфигураций. Методы ▶ **клеточной декомпозиции** позволяют разложить свободное пространство на конечное число непрерывных областей, называемых ячейками (или клетками). Эти ячейки проектируются так, чтобы задачу планирования пути в пределах одной ячейки можно было решить простыми средствами (например, передвижением по прямой линии). Тогда задача планирования пути сводится к задаче поиска в дискретном графе (как и в случае графов видимости или графов Вороного) с целью нахождения пути через последовательность ячеек.

Простейшая клеточная декомпозиция представляет собой сетку с равномерным шагом. На рис. 26.16, *а* приведены декомпозиция свободного пространства робота с помощью квадратной сетки и путь решения, оптимальный для сетки с данными размерами. На этом рисунке затенение в оттенках серого указывает значение каждой клетки сетки свободного пространства, т.е. *стоимость* самого короткого пути от этой ячейки к цели. (Эти стоимости можно вычислить с помощью детерминированной формы алгоритма VALUE-ITERATION, приведенного на рис. 17.6 в разделе 17.2.1.) На рис. 26.16, *б* показана соответствующая траектория манипулятора робота в рабочем пространстве. Конечно, для поиска кратчайшего пути также можно использовать алгоритм A*.



а) б)

Рис. 26.16. а) Функция стоимости и путь, найденный с помощью аппроксимации пространства конфигураций в виде сетки квадратных ячеек. **б)** Тот же путь, визуально представленный с помощью координат рабочего пространства. Обратите внимание на то, как робот сгибает руку в локтевом шарнире для предотвращения столкновения с вертикальным препятствием

Преимущество подобной клеточной декомпозиции в том, что она проста в реализации, однако ей свойственны три ограничения. Во-первых, она применима только для пространств конфигураций с малым количеством измерений, поскольку число ячеек в сетке возрастает экспоненциально при увеличении количества измерений d. (Звучит знакомо? Это проклятие размерности.) Во-вторых, путь через дискретизированное пространство состояний не всегда будет гладким. На рис. 26.16, *а* видно, что диагональные части пути зубчатые и, следовательно,

роботу будет трудно точно им следовать. Робот может попытаться сгладить путь, найденный в качестве решения, но это не так уж просто.

В-третьих, имеет место особая проблема, связанная с тем, что некоторые клетки являются "смешанными", т.е. не принадлежащими полностью ни к свободному, ни к занятому препятствиями пространству. Найденный в качестве решения путь, включающий такую ячейку, может не соответствовать действительному решению, поскольку не будет существовать способа пересечения данной клетки в желаемом направлении по прямой линии. В результате процедура планирования пути становится *противоречивой*. С другой стороны, если при планировании разрешить использовать только полностью свободные клетки, то процедура планирования станет *неполной* по той причине, что могут возникать случаи, в которых единственно возможные пути к цели проходят через смешанные клетки. При этом может оказаться, что доступный проход в действительности достаточно широк, чтобы робот мог через него пройти, но в сетке этот проход представлен только смешанными клетками.

Первый подход к устранению указанных проблем — *дальнейшее разделение* смешанных клеток, возможно, с использованием ячеек, вдвое меньших по сравнению с первоначальным размером. Такая операция может продолжаться рекурсивно до тех пор, пока не будет найден путь, проходящий только по свободным клеткам. Однако этот метод может применяться только в том случае, если есть возможность определить, является ли данная конкретная клетка смешанной, а эта операция будет простой, только когда границы в пространстве конфигураций будут определены с помощью относительно простых математических описаний.

Важно отметить, что метод декомпозиции ячеек не обязательно требует явного представления пространства препятствия C_{obs}. Принять решение, включить клетку в путь или нет, можно с помощью ▶ **контроллера коллизий** (*collision checker*). Это важнейшее понятие в задаче планирования движения. Контроллер коллизий представляет собой функцию $\gamma(q)$, принимающую значение 1, если конфигурация указывает на столкновение с препятствием, и значение 0 — в противном случае. Намного проще проверить, есть ли в конкретной конфигурации столкновение с препятствием, чем явно строить все пространство препятствия C_{obs}.

Рассматривая путь решения, показанный на рис. 26.16, *а*, можно заметить дополнительную сложность, которую необходимо преодолеть. Этот путь содержит произвольно острые углы, но физический робот обладает импульсом и не способен изменять направление движения мгновенно. Данную проблему можно решить путем сохранения для каждой ячейки сетки точного непрерывного состояния (положения и скорости), которое имело место, когда ячейка была достигнута при поиске. Теперь предположим, что при распространении информации к соседним ячейкам сетки это непрерывное состояние используется в качестве основы, к которой применяется модель движения непрерывного робота для перехода в соседние клетки. В результате выполняется не мгновенный поворот на 90°, а округлый поворот, отвечающий всем физическим законам движения. Теперь можно

гарантировать, что расчетная траектория будет гладкой и робот действительно сможет ей следовать. Единственным алгоритмом, позволяющим реализовать все эти требования, является ▶ **гибридный A***.

Рандомизированное планирование движений

При рандомизированном планировании движения поиск выполняется по графу в *случайной* декомпозиции пространства конфигураций вместо его регулярной клеточной декомпозиции. Ключевой идеей является выборка случайного множества точек и создания между ними ребер, если есть очень простой способ добраться от одной точки до другой (например, по прямой линии) без столкновения с препятствием, после чего в полученном графе выполняется поиск.

Одним из вариантов использования этой идеи является алгоритм ▶ **вероятностной дорожной карты (▶ PRM)**. Предполагается наличие доступа к контроллеру коллизий γ (его определение приведено в предыдущем разделе) и ▶ **простому планировщику** $B(q_1, q_2)$, возвращающему путь от точки q_1 до q_2 (или извещение о неудаче) и делающему это *быстро*. Этот простой планировщик не обязан быть совершенным — он может вернуть извещение о неудаче, даже если в действительности решение существует. Его назначение состоит в том, чтобы попытаться быстро соединить точки q_1 и q_2, а затем дать основному алгоритму знать, удалось ли ему это. Он будет использоваться лишь для определения, существует ли ребро между двумя вершинами.

Работа алгоритма начинается с формирования выборки M ▶ **опорных точек** (*milestones*) в свободной области пространства конфигураций в дополнение к точкам q_s и q_g. Применяется метод выборки с отклонением, при котором точки в пространстве конфигурации выбираются случайным образом и подвергаются проверке с использованием контроллера коллизий γ до тех пор, пока не будет найдено в общей сложности M опорных точек. Далее, чтобы попытаться соединить пары опорных точек, алгоритм обращается к простому планировщику. Если простой планировщик возвращает сообщение об успехе, то ребро между данной парой точек добавляется к графу, в противном случае граф остается без изменений. Предпринимаются попытки соединить каждую опорную точку либо с k ближайшими соседями (этот вариант алгоритма называется k-PRM), либо со всеми опорными точками в сфере радиуса r. По завершении построения графа алгоритм выполняет поиск пути в нем от точки q_s до q_g. Если путь не найден, то отбираются еще M опорных точек, добавляемых к графу, и процесс повторяется.

На рис. 26.17 представлена дорожная карта с путем, найденным между двумя конфигурациями. Алгоритмы класса PRM не являются полными, но являются тем, что называется ▶ **вероятностно полными**, — в конечном итоге они найдут путь, если таковой существует. Очевидно, это можно объяснить тем, что они продолжают отбирать все больше опорных точек. Алгоритмы PRM хорошо работают даже в многомерных конфигурационных пространствах.

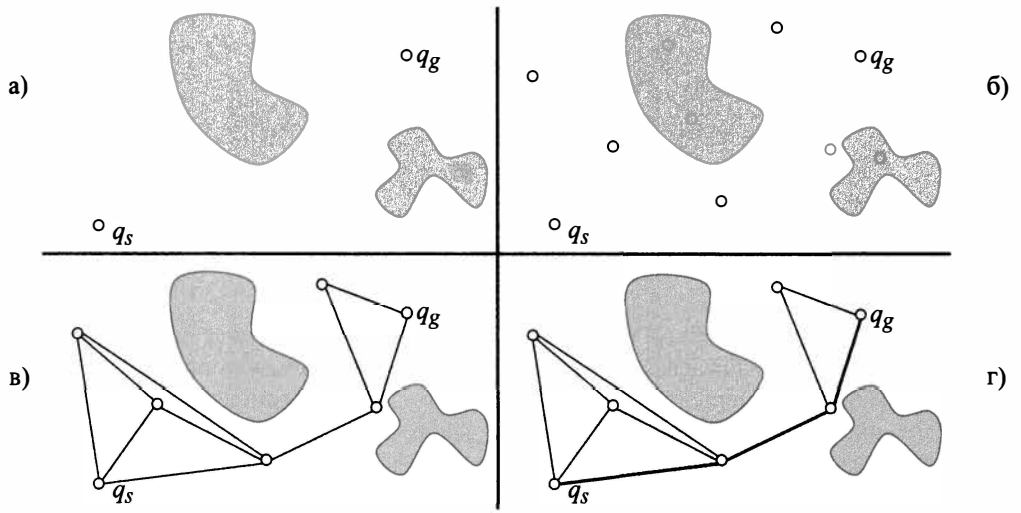


Рис. 26.17. Алгоритм вероятностной дорожной карты (PRM). **а)** Исходная и целевая конфигурации. **б)** Выборка M опорных точек без коллизий (здесь $M = 5$). **в)** Соединение каждой из опорных точек с k ближайшими соседями (здесь $k = 3$). **г)** Поиск кратчайшего пути от исходной точки до целевой в полученном графе

Алгоритмы PRM также популярны при ►**планировании с несколькими запросами**, когда имеется несколько задач планирования движения в одном и том же пространстве конфигураций. Часто, когда робот достигает цели, ему поручают достичь другой цели в том же рабочем пространстве. В этом случае алгоритмы PRM действительно полезны, поскольку робот может заранее выделить время на разработку своей дорожной карты, а затем компенсировать затраты, используя эту дорожную карту для выполнения многих заданий.

Быстро исследующие случайные деревья

Расширение метода PRM с названием ►**быстро исследующие случайные деревья** (*rapidly exploring random trees* — ►**RRT**) получило большую популярность при планировании одиночных запросов. Согласно этому методу, инкрементно строятся одновременно два дерева: одно — с корнем в точке q_s, а другое — с корнем в точке q_g. Выбираются случайные опорные точки, и для каждой предпринимается попытка подключить ее к уже существующим деревьям. Если опорную точку удается соединить с обоими деревьями, то это означает, что решение уже найдено, как показано на рис. 26.18. Если нет, алгоритм находит ближайшую точку в каждом дереве и добавляет к нему новое ребро, которое проходит от этой точки на расстояние δ до опорной точки. Такой подход обеспечивает дереву тенденцию роста в направлении ранее неисследованных участков пространства конфигураций.

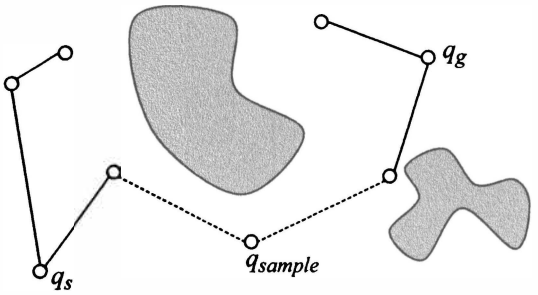
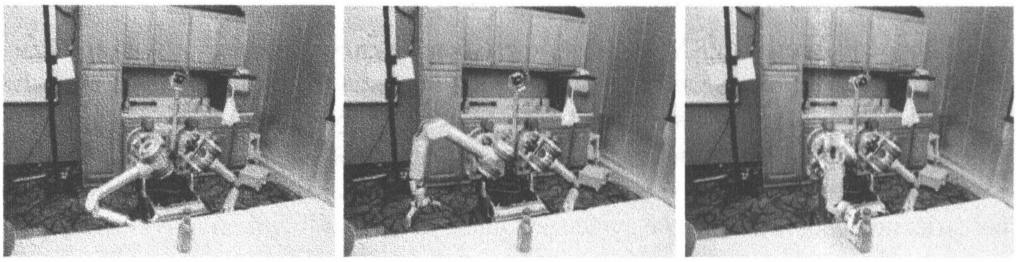


Рис. 26.18. Двунаправленный алгоритм RRT строит два дерева (одно — из исходной точки, а другое — из целевой) путем последовательного подключения каждой случайно выбранной опорной точки к ближайшему узлу в каждом дереве, если такое соединение возможно. Если выбранная опорная точка может быть соединена с обоими деревьями, то это означает, что требуемый путь найден

Создатели роботов любят алгоритм RRT за простоту использования. Однако найденные этим алгоритмом решения, как правило, не являются оптимальными и не обеспечивают гладкости траектории. Поэтому при использовании алгоритма RRT к полученным результатам часто применяются дополнительные процедуры постобработки. Наиболее распространенным вариантом является метод сокращения пути, по которому на пути решения случайным образом выбирается одна из вершин и предпринимается попытка удалить ее за счет соединения соседних с ней вершин друг с другом (с помощью простого планировщика). Подобные попытки повторяются вновь и вновь в пределах отпущенного на это времени. И даже после такой обработки найденные траектории все еще могут выглядеть довольно неестественно из-за случайного положения опорных точек, выбранных при работе алгоритма, как показано на рис. 26.19.



а) б) в)

Рис. 26.19. Этапы прохождения траектории, созданной алгоритмом RRT и дополнительно обработанной по методу сокращения пути

Алгоритм ▶**RRT*** представляет собой модификацию алгоритма RRT, позволяющую сделать его асимптотически оптимальным: решение сходится к оптимальному решению, поскольку отбирается все больше и больше опорных точек. Основная идея состоит в том, чтобы выбрать ближайшего соседа, исходя из понятия *стоимости* пути, а не расстояния только до опорной точки, и перестроить дерево, поменяв местами родительские вершины, если добраться до них будет дешевле через новую опорную точку.

Оптимизация траектории для кинематического планирования

Алгоритмы рандомизированной выборки, как правило, сначала строят сложный, но выполнимый путь, а затем оптимизируют его. Оптимизация траектории выполняется в обратном порядке: она начинается с простого, но неосуществимого пути, а затем предпринимаются попытки устранить из него все столкновения с препятствием. Цель состоит в том, чтобы найти путь, оптимизирующий функцию стоимости[1] путей. Иначе говоря, требуется минимизировать функцию стоимости $J(\tau)$, где $\tau(0) = q_s$ и $\tau(1) = q_g$.

Функция стоимости J называется **функционалом**, поскольку это функция, аргументами которой являются также функции: аргумент функции J — это τ, сам по себе являющийся функцией $\tau(t)$, принимающей в качестве входных данных точку в интервале [0, 1] и отображающей ее в конфигурацию. Стандартный функционал стоимости представляет собой компромисс между двумя важными аспектами движения робота: предотвращением столкновений и обеспечением эффективности:

$$J = J_{obs} + \lambda J_{eff},$$

где эффективность J_{eff} является оценкой длины пути и может также включать оценку его гладкости. Очень удобный способ определения эффективности — квадратичный, когда интегрируется квадрат первой производной функции τ (скоро будет показано, почему этот подход действительно способствует построению коротких путей):

$$J_{eff} = \int_0^1 \frac{1}{2} \|\dot{\tau}(s)\|^2 \, ds.$$

Для терма препятствия J_{eff} предположим, что есть возможность вычислить расстояние $d(x)$ от любой точки $x \in W$ до ближайшего края препятствия. Это расстояние будет положительным вне препятствия, равно нулю — на его границе и отрицательным — внутри препятствия. Такое решение называется ▶**полем расстояния со знаком** (*signed distance field*). Теперь можно определить поле стоимости в рабочем пространстве (обозначим его как c); оно будет иметь высокую

[1] У создателей роботов принято минимизировать функцию стоимости J, тогда как в других областях ИИ обычно ставится задача максимизировать функцию полезности U или вознаграждение R.

стоимость внутри препятствий и небольшую стоимость — снаружи от него. Воспользовавшись этой стоимостью, можно сделать точки в рабочей области абсолютно неприемлемыми, если они находятся внутри препятствия, и нежелательными — если рядом с ним (это позволит избежать появления проблемы, характерной для графов видимости, когда траектории постоянно прижимаются к краям препятствия). Безусловно, робот не является точкой в рабочей области, поэтому потребуется еще немного работы — необходимо рассмотреть все точки b на теле робота:

$$J_{obs} = \int_0^1 \int_b c(\underbrace{\phi_b(\tau(s))}_{\in W}) \left\| \frac{d}{ds} \underbrace{\phi_b(\tau(s))}_{\in W} \right\| db\, ds.$$

Это выражение называется ▶**интегралом пути**, — в нем не просто интегрируется значение c вдоль пути для каждой точки тела робота, оно еще и умножается на производную, чтобы сделать стоимость инвариантной относительно *скорости* прохождения пути. Представим, что робот перемещается по полю стоимости, накапливая стоимость по мере продвижения. Независимо от того, насколько быстро или медленно он перемещается по выбранной траектории, он должен накапливать одну и ту же стоимость.

Простейшим способом решить описанную выше задачу оптимизации и найти путь является использование *градиентного спуска*. Если вам интересно, как получить градиенты функционалов по отношению к функциям, то обратитесь к разделу математики под названием *вариационное исчисление*. Сделать это особенно просто для функционалов вида

$$J[\tau] = \int_0^1 F(s, \tau(s), \dot{\tau}(s))ds\,,$$

которые представляют собой интегралы функций, зависящих только от параметра s, значения функции в точке s и производной функции в точке s. В этом случае ▶**уравнение Эйлера–Лагранжа** говорит, что градиент будет равен

$$\nabla_\tau J(s) = \frac{\partial F}{\partial \tau(s)}(s) - \frac{d}{dt}\frac{\partial F}{\partial \dot{\tau}(s)}(s).$$

Если внимательнее присмотреться к выражениям для J_{eff} и J_{obs}, то можно заметить, что они оба соответствуют этой схеме. В частности, для J_{eff} имеем $F(s, \tau(s), \dot{\tau}(s)) = \| \dot{\tau}(s) \|^2$. Чтобы было немного удобнее, будем вычислять градиент только для J_{eff}. Можно видеть, что F не зависит напрямую от $\tau(s)$, поэтому первый член формулы будет равен нулю. Выражение для градиента сокращается до

$$\nabla_\tau J(s) = 0 - \frac{d}{dt}\dot{\tau}(s),$$

поскольку частной производной от F относительно $\dot{\tau}(s)$ является $\dot{\tau}(s)$.

Обратите внимание, насколько упрощает задачу принятое выше определение J_{eff}, — это просто квадрат производной (и перед ним даже поставлена 1/2, так что 2 сокращается). Практика показывает, что подобный трюк часто имеет место при оптимизации — искусство состоит не только в том, как оптимизировать функцию стоимости, но и в том, чтобы выбрать такую функцию стоимости, которая будет хорошо сочетаться с тем, как ее будут оптимизировать. Упростив формулу градиента, получим

$$\nabla_{\tau} J(s) = -\ddot{\tau}(s).$$

Теперь, поскольку выражение для J_{eff} является квадратичным, установка этого градиента в нуль дает решение для τ, если отсутствуют столкновения с препятствиями. Интегрируя один раз, получаем, что первая производная должна быть константой; интегрируя еще раз, получаем, что $\tau(s) = a \cdot s + b$, причем a и b определяются ограничениями конечной точки для $\tau(0)$ и $\tau(1)$. Таким образом, оптимальным путем по отношению к J_{eff} является прямая линия от исходной точки до целевой! Это действительно самый эффективный способ перехода от одной точки к другой, если, конечно, отсутствуют препятствия, о которых следует беспокоиться.

Безусловно, добавление в расчеты J_{obs} сделает вычисления сложнее, но здесь мы не будем усложнять ситуацию и рассматривать вывод его градиента. Робот обычно инициализирует свой путь в виде прямой линии, которая может проходить сквозь некоторые препятствия. Далее рассчитывается градиент стоимости относительно текущего выбранного пути, который будет "отталкивать" траекторию пути от препятствий (рис. 26.20). Имейте в виду, что метод градиентного спуска позволяет находить лишь *локально оптимальные* решения, — так же, как и метод

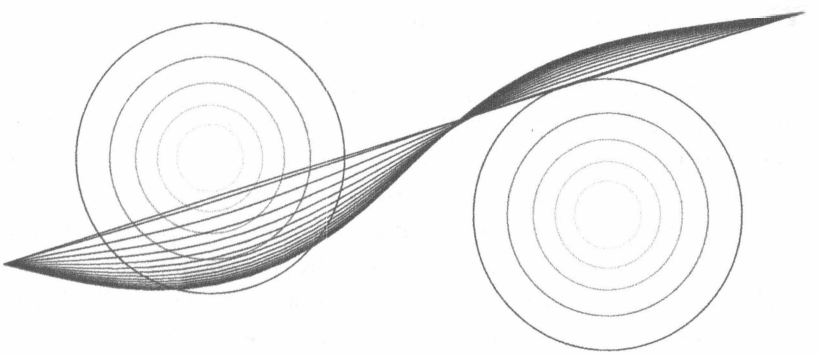
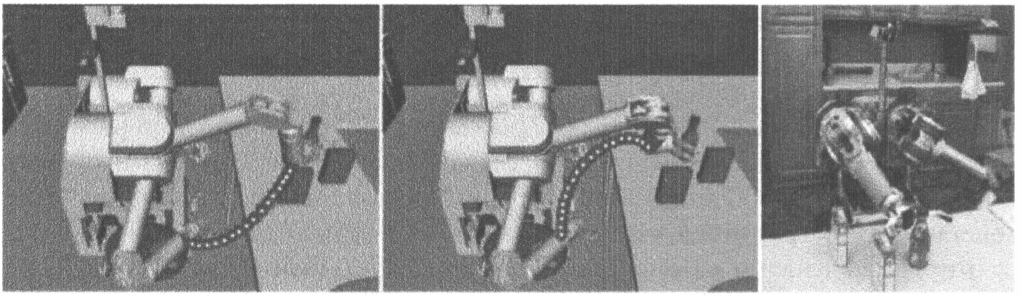


Рис. 26.20. Оптимизация траектории при планировании движения. Два точечных препятствия с окружающими их круговыми полосами снижения стоимости. Оптимизатор начинает работу с прямой траектории и позволяет препятствиям изгибать траекторию пути для защиты от столкновений с препятствиями, отыскивая в поле стоимости путь с минимальной стоимостью

восхождения к вершине. Такие методы, как имитация отжига (раздел 4.1.2), могут использоваться для разведки с целью повысить вероятность того, что этот местный оптимум является достаточно хорошим. Реальный пример применения метода оптимизации траектории при планировании движения приведен на рис. 26.21.



а) б) в)

Рис. 26.21. Задача достижения точки, удобной для захвата бутылки, была решена методом оптимизации траектории. **а)** Исходная траектория, ведущая к требуемому конечному положению исполнительного механизма без учета препятствий. **б)** Конечная траектория безопасного пути после оптимизации. **в)** Манипулятор робота достиг конечной точки оптимизированной траектории

26.5.3. Управление прохождением траектории

Выше обсуждалось, как *планировать* движения, но не как их *осуществлять* — подавать ток на моторы, чтобы они создавали крутящий момент, необходимый для передвижения робота. Это сфера ▶ **теории управления**, поля исследований в области искусственного интеллекта, значение которой неуклонно возрастает. Здесь есть два основных вопроса, на которые необходимо получить ответы: как превратить математическое описание пути в последовательность действий в реальном мире (управление без обратной связи) и как убедиться, что движение происходит по правильному пути (управление с обратной связью).

От конфигураций до крутящих моментов для управления без обратной связи. Выбранный путь $\tau(t)$ определяет необходимую последовательность конфигураций. Робот начинает движение из состояния покоя при $q_s = \tau(0)$. Из этой точки двигатели робота должны преобразовать подаваемый на них ток в крутящие моменты, приводящие робота в движение. Но к каким крутящим моментам должен стремиться робот, чтобы закончить движение в точке $q_g = \tau(1)$?

Здесь на первый план выходит понятие ▶ **динамической модели** (или модели перехода). Можно предоставить роботу функцию f, вычисляющую эффект от воздействия крутящих моментов на конфигурацию. Помните формулу $F = ma$ из школьного курса физики? Что-то подобное существует и для крутящих моментов:

$u = f^{-1}(q, \dot{q}, \ddot{q})$, где u — крутящий момент, $\dot{q}$ — скорость, а $\ddot{q}$ — ускорение.[2] Если робот, находящийся в конфигурации q и имеющий скорость $\dot{q}$, применит крутящий момент u, это приведет к ускорению $\ddot{q} = f(q, \dot{q}, u)$. Здесь кортеж $(q, \dot{q})$ является ▶**динамическим состоянием**, поскольку включает в себя скорость, тогда как q является ▶**кинематическим состоянием** и одного его недостаточно для точного вычисления того, какой крутящий момент следует применить. Функция f является детерминированной динамической моделью в марковском процессе принятия решения (MDP) в пространстве динамических состояний с крутящими моментами в качестве действий. Функция f^{-1} — это ▶**инверсная динамика**, говорящая о том, какой крутящий момент следует применить, если необходимо получить определенное ускорение, что приведет к изменению скорости и, следовательно, к изменению динамического состояния.

Теперь, по наивности, переменную $t \in [0, 1]$ можно рассматривать как "время", изменяющееся в пределах от 0 до 1, и выбрать крутящий момент с использованием инверсной динамики:

$$u(t) = f^{-1}(\tau(t), \dot{\tau}(t), \ddot{\tau}(t)), \tag{26.2}$$

в предположении, что робот начинает движение при динамическом состоянии $(\tau(0), \dot{\tau}(0))$. Однако в действительности все не так просто.

Путь τ создавался как последовательность точек, без учета скоростей и ускорений. По этой причине путь может не удовлетворять соотношению $\dot{\tau}(0) = 0$ (робот начинает движение с нулевой скоростью) или может оказаться, что функция τ вообще не дифференцируема (не говоря уже о дифференцируемости дважды). Кроме того, значение конечной точки "1" также неясно: какому количеству секунд это значение отвечает?

На практике, прежде чем даже подумать о следовании по пути τ, он обычно ▶**синхронизуется** (*to retime*), т.е. выполняется преобразование пути в траекторию $\xi(t)$, отображающую интервал $[0, T]$ для некоторого периода времени T на точки в пространстве конфигураций C. (Символ ξ — это греческая буква "кси".) Подобная синхронизация сложнее, чем можно было бы подумать, но есть приблизительные способы ее выполнения, например путем выбора максимальной скорости и ускорения и использования профиля, в котором скорость возрастает до максимального значения и фиксируется на этом значении как можно дольше, а затем уменьшается обратно до 0. Если предположить, что у нас есть возможность это реализовать, то приведенное выше уравнение (26.2) можно переписать как

$$u(t) = f^{-1}(\xi(t), \dot{\xi}(t), \ddot{\xi}(t)). \tag{26.3}$$

---

[2] Здесь мы опускаем детали, представленные в формуле как f^{-1}, которые включают массу, инерцию, гравитацию, а также кориолисовы и центробежные силы.

Но даже при замене пути τ фактической траекторией ξ использование приведенного выше уравнения для крутящих моментов (называемое ▶**законом управления**) на практике связано с проблемой. Вспомнив материал из раздела об обучении с подкреплением, можно догадаться, почему это так. Это уравнение прекрасно работает в ситуации, когда функция f точна, но докучливая реальность неизбежно вмешивается и путает все карты: в реальных системах нельзя точно измерить массу и инерцию, а в функции f невозможно правильно учесть такие физические явления, как ▶"**залипание**" в двигателях (трение, препятствующее приведению неподвижных поверхностей в движение и заставляющее их "прилипать" друг к другу). Поэтому, когда робот начинает прикладывать к манипулятору вычисленные крутящие моменты, а функция f не точна, погрешность постепенно накапливается и истинная траектория все больше отклоняется от расчетной.

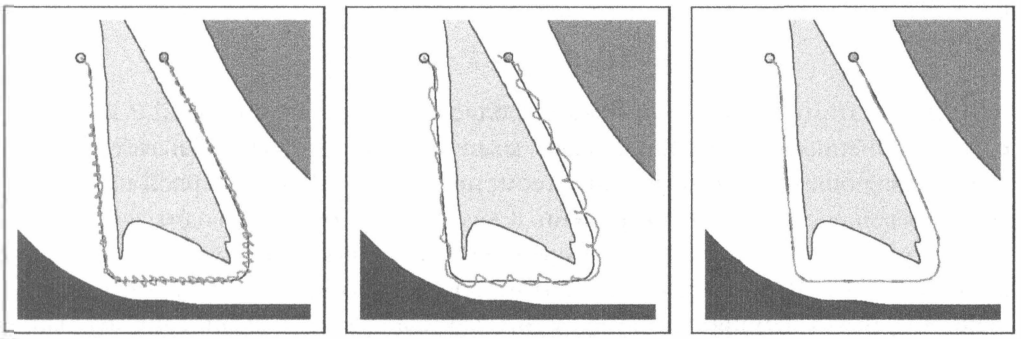
Однако вместо того, чтобы просто позволить этим погрешностям неуклонно накапливаться, робот может использовать процесс управления, в котором отслеживается, где он, по его мнению, сейчас находится, и эта конфигурация сравнивается с той, где он хотел бы быть, после чего для минимизации погрешности применяются соответствующие крутящие моменты.

Контроллеры, прикладывающие усилия, обратно пропорциональные наблюдаемой погрешности, называются пропорциональными контроллерами или ▶**P-контроллерами**. Управляющее воздействие, выработанное P-контроллером, определяется по формуле

$$u(t) = K_P(\xi(t) - q_t),$$

где q_t — текущая конфигурация, а K_P — постоянная, представляющая ▶**коэффициент усиления** контроллера. От него зависит, какое усилие будет прилагать контроллер, компенсируя отклонения между фактическим состоянием q_t и требуемым состоянием $\xi(t)$.

На рис. 26.22, *а* показано, что может пойти не так при использовании пропорционального контроллера. Всякий раз, когда из-за шума или ограничений на силы, которые может применить робот, происходит отклонение от требуемого пути, вырабатывается противодействующая сила, величина которой пропорциональна этому отклонению. На первый взгляд это может показаться подходящим решением, поскольку отклонения должны компенсироваться противодействующей силой, чтобы удерживать робота на правильном пути. Однако, как видно на рис. 26.22, *а*, пропорциональный контроллер может потребовать от робота приложения слишком большого усилия, приводящего к уклонению от нужной траектории в противоположную сторону, в результате чего движение робота становится зигзагообразным, пересекающим траекторию то в одном, то в другом направлении. Это результат естественной инерции робота: вернувшись на правильную траекторию, робот будет иметь некоторую скорость, которую невозможно погасить мгновенно.



а) б) в)

Рис. 26.22. Управление манипулятором робота. **а)** С использованием пропорционального контроллера с коэффициентом усиления 1,0. **б)** С использованием пропорционального контроллера с коэффициентом усиления 0,1. **в)** С использованием PD-контроллера (пропорциональный дифференциальный) с коэффициентом усиления 0,3 для пропорционального компонента и 0,8 — для дифференциального компонента. Во всех трех случаях робот пытается перемещать манипулятор по гладкой траектории, но в случаях *а* и *б* имеют место существенные отклонения от нее

На рис. 26.22, *а* пропорциональный контроллер имеет коэффициент усиления $K_P = 1$. На первый взгляд можно решить, что выбор меньшего значения K_P позволит устранить проблему, обеспечив роботу более плавный подход к желаемой траектории. К сожалению, все обстоит иначе. На рис. 26.22, *б* показана траектория при $K_P = 0{,}1$, — в движении робота все так же проявляется колебательное поведение. Уменьшение коэффициента усиления способствует лишь уменьшению интенсивности колебаний, но не устраняет проблему. В действительности при отсутствии трения Р-контроллер работает в соответствии с законом пружины, поэтому он до бесконечности будет совершать колебания вокруг заданной целевой точки.

Разработан ряд контроллеров, намного превосходящих обсуждавшийся выше Р-контроллер, действующий на основании простого закона управления. Контроллер называется ▶ **стабильным**, если небольшие возмущения приводят к возникновению ограниченной погрешности, связывающей сигнал робота и опорный сигнал. Контроллер называется ▶ **строго стабильным**, если позволяет вернуть робота на опорный путь после воздействия подобных возмущений. Понятно, что обсуждавшийся выше Р-контроллер можно считать стабильным, но он не является строго стабильным, поскольку не способен обеспечить возвращение робота на его опорную траекторию.

Простейший контроллер, позволяющий добиться строгой стабильности в данной проблемной области, называется ▶ **PD-контроллером**, — здесь буква P представляет слово *proportional* (пропорциональный), а буква D — слово *derivative*

(дифференциальный). Для описания PD-контроллеров применяется следующее уравнение:

$$u(t) = K_P(\xi(t) - q_t) + K_D(\dot{\xi}(t) - \dot{q}_t). \quad (26.4)$$

Согласно этому уравнению, PD-контроллеры представляют собой P-контроллеры, дополненные дифференциальным компонентом, который добавляет к значению управляющего воздействия $u(t)$-терм, пропорциональный первой производной от погрешности $\xi(t) - q_t$ по времени. К какому результату приводит добавление такого терма? В общем случае дифференциальный терм гасит колебания в управляемой системе. Чтобы убедиться в этом, рассмотрим ситуацию, в которой погрешность резко изменяется во времени, как было в случае описанного выше P-контроллера. При этом производная такой погрешности прикладывается в направлении, противоположном пропорциональному терму, что приводит к уменьшению общего отклика на возмущение. Однако если та же погрешность продолжит свое присутствие и не изменится, то производная уменьшится до нуля и при выборе управляющего воздействия будет доминировать пропорциональный терм.

Результаты применения такого PD-контроллера с коэффициентами усиления $K_P = 0{,}3$ и $K_D = 0{,}8$ при управлении манипулятором робота показаны на рис. 26.19, *в*. Очевидно, что в конечном итоге траектория манипулятора стала гораздо более гладкой и на ней внешне незаметны какие-либо колебания.

Однако PD-контроллеры на самом деле могут давать сбои. В частности, PD-контроллеры могут оказаться неспособными отрегулировать погрешность до нуля даже при отсутствии внешних возмущений. Чаще всего подобная ситуация является результатом систематической внешней силы, которая не является частью модели. Например, беспилотный автомобиль, двигающийся по наклонной поверхности, может систематически уходить в ту или иную сторону. Износ в шарнирах манипулятора робота также вызывает аналогичные систематические погрешности. В таких ситуациях необходима обратная связь с пропорциональным перерегулированием, позволяющая приблизить погрешность к нулю. Решение этой проблемы заключается в том, что к закону управления нужно добавить третий терм, основанный на результатах *интегрирования* погрешности по времени:

$$u(t) = K_P(\xi(t) - q_t) + K_I \int_0^t (\xi(s) - q_s)ds + K_D(\dot{\xi}(t) - \dot{q}_t). \quad (26.5)$$

Здесь K_I — третий параметр усиления. В терме $\int_0^t (\xi(s) - q_s)ds$ вычисляется интеграл погрешности по времени. Под влиянием этого терма корректируется длительные отклонения между опорным сигналом и фактическим состоянием. Таким образом, интегральные термы гарантируют отсутствие систематических погрешностей в действиях контроллера за счет повышения опасности колебательного поведения.

Контроллер, закон управления которого включает все три терма, называется ►**PID-контроллером** (от *proportional, integral, derivative* — пропорциональный,

интегральный, дифференциальный). PID-контроллеры широко используются в промышленности для решения самых разных задач управления. Эти три терма можно понимать следующим образом: *пропорциональный*: воздействие тем сильнее, чем больше погрешность; *дифференциальный*: воздействие еще сильнее, если погрешность возрастает; *интегральный*: воздействие должно увеличиваться, если успех не достигнут в течение длительного времени.

Промежуточный уровень между управлением без обратной связи на основе инверсной динамики и PID-управлением с обратной связью называют ▶ **расчетным управлением крутящим моментом** (*computed torque control*). Рассчитывается крутящий момент, необходимый в соответствии с имеющейся моделью, а неточности модели компенсируются с помощью термов пропорциональных погрешностей:

$$u(t) = \underbrace{f^{-1}(\xi(t), \dot{\xi}(t), \ddot{\xi}(t))}_{\text{прямая связь}} + \underbrace{m(\xi(t))(K_P(\xi(t) - q_t) + K_D(\dot{\xi}(t) - \dot{q}_t))}_{\text{обратная связь}}. \tag{26.6}$$

Первый терм называется ▶ **компонентом прямой связи**, поскольку он прогнозирует, где робот должен оказаться, и вычисляет, какой крутящий момент может для этого потребоваться. Второй терм является ▶ **компонентом обратной связи**, поскольку он вводит в закон управления информацию о текущей погрешности в динамическом состоянии. Элемент $m(q)$ является матрицей инерции в конфигурации q — в отличие от обычного PD-управления, здесь коэффициент усиления изменяется с изменением конфигурации системы.

Планы или стратегии

Давайте сделаем шаг назад, чтобы углубить понимание аналогии между тем, что говорилось в этой главе до сих пор, и тем, что вы узнали из глав, посвященных поиску, марковским процессам принятия решений (MDP) и методам машинного обучения с подкреплением. В робототехнике в ходе анализа движения действительно анализируется лежащий в основе MDP, где состояниями являются динамические состояния (конфигурация и скорость), а действиями являются управляющие сигналы, обычно в виде крутящих моментов. Если еще раз взглянуть на приведенные выше законы управления, то можно понять, что эти законы являются *стратегиями*, а не *планами*, — они указывают роботу, какие действия ему следует предпринять по достижении им *любого* состояния из числа тех, которых он может достичь. Тем не менее эти стратегии обычно далеки от *оптимальных* стратегий. Поскольку динамическое состояние является непрерывным и многомерным (как и пространство действий), оптимальные политики очень трудно выделить в вычислительном отношении.

Вместо этого проблема решается иным путем. Сначала вырабатывается план — в упрощенном пространстве состояний и действий: для этого используется только кинематическое состояние и предполагается, что состояния достижимы друг для друга без учета подлежащей динамики. Это планирование движения, и его

результатом является опорный путь. Если бы динамика была точно известна, его можно было бы превратить в план движения из исходного состояния в пространстве действий с помощью уравнения (26.3).

Но поскольку динамическая модель обычно неточна, опорный план следует превратить в стратегию, которая будет стремиться ему следовать — возвращать систему к опорному плану, когда она от него отклоняется. При такой схеме действий субоптимальность вводится двумя способами: сначала — путем планирования без учета динамики, а затем — принятием предположения, что при отклонении от плана оптимальным решением будет возврат к исходному плану. Ниже будут описаны методы, позволяющие вычислять стратегии непосредственно на основании динамических состояний, полностью избегая разделения.

26.5.4. Оптимальное управление

Вместо использования планировщика для создания кинематического пути, а затем беспокойства только о динамике системы при отслеживании фактического состояния дел в этом разделе обсуждается, как все это можно делать одновременно. Возьмем задачу оптимизации траектории для кинематических путей и превратим ее в задачу истинной оптимизации траектории с учетом динамики: будем оптимизировать непосредственно сами действия, принимая во внимание динамику (или переходы).

Такой подход гораздо ближе к тому, который обсуждался в главах, посвященных поиску и MDP. Если динамика системы известна, то можно найти последовательность действий, которые следует выполнить для достижения цели, как это делалось в главе 3. А когда нет полной уверенности, то можно попробовать разработать подходящую стратегию, как в главе 17.

В этом разделе более подробно рассматривается марковский процесс принятия решений (MDP), лежащий в основе действий робота. Потребуется переключиться с привычных дискретных MDP на непрерывные. Обозначим динамическое состояние мира через x, как это принято в обычной практике, — эквивалент s в дискретном MDP. Пусть x_s и x_g — исходное и целевое состояния.

Необходимо найти последовательность действий, которые при выполнении роботом приводят к парам "состояние-действие" с низкой совокупной стоимостью. Действия — это крутящие моменты, которые мы обозначим через $u(t)$ для t, начиная в 0 и заканчивая в T. Формально требуется найти последовательность крутящих моментов u, которая минимизирует совокупную стоимость J

$$\min_u \int_0^T J(x(t), u(t))dt \tag{26.7}$$

с учетом ограничений

$$\forall t,\ \dot{x}(t) = f(x(t), u(t))$$
$$x(0) = x_s, x(T) = x_g.$$

Как это связано с планированием движения и отслеживанием траектории? Давайте представим, что категории эффективности и отсутствия препятствий включены в функцию стоимости J, как это делалось при оптимизации траектории по кинематическому состоянию. Динамическое состояние — это конфигурация и скорость, а крутящие моменты u меняют его через динамику f из задачи отслеживания траектории с прямой связью. Отличие состоит в том, что в данном случае конфигурации и крутящие моменты рассматриваются одновременно. Иногда может потребоваться также рассматривать предотвращение столкновений как жесткое ограничение, что уже упоминалось ранее, когда рассматривалась оптимизация траектории только для кинематического состояния.

Чтобы решить такую задачу оптимизации, можно воспользоваться градиентами функции стоимости J, но уже не в отношении последовательности конфигураций τ, а непосредственно по отношению к управляющим сигналам u. Иногда бывает полезно также включить последовательность состояний x в качестве переменной решения и использовать ограничения динамики для гарантии, что x и u будут согласованы. Существуют различные методы оптимизации траектории, в которых используется подобный подход, — два из них известны под названиями ▶**многоточечная пристрелка** (*multiple shooting*) и ▶**прямая коллокация** (*direct collocation*). Ни один из этих методов не позволяет найти глобальное оптимальное решение, но на практике они способны эффективно обеспечивать хождение для гуманоидных роботов и управление движением для беспилотных автомобилей.

Чудеса начинаются в том случае, когда в приведенной выше задаче функция стоимости J квадратична, а функция управления f — линейна относительно x и u. Требуется свести к минимуму

$$\min \int_0^\infty x^T Q x + u^T R u dt \quad \text{при условии, что} \quad \forall t,\ \dot{x}(t) = Ax(t) + Bu(t).$$

Появляется возможность оптимизировать на бесконечном горизонте, вместо конечного и получать стратегию из любого состояния, а не просто последовательность управляющих воздействий. Для достижения этого параметры Q и R должны быть положительно определенными матрицами. В результате получаем ▶**линейно-квадратичный регулятор** (*linear quadratic regulator* — ▶**LQR**). В случае LQR функция оптимальной стоимости (называемая **стоимостью прохождения**) является квадратичной, а оптимальная стратегия — линейной. Стратегия имеет вид $u = -Kx$, где нахождение матрицы K требует решения алгебраического ▶**уравнения Риккати** — никакой локальной оптимизации, никакой итерации по значениям, никакой итерации по стратегиям не требуется!

Из-за простоты поиска оптимальной стратегии метод LQR находит много применений на практике, — даже несмотря на тот факт, что реальные задачи редко имеют квадратичные стоимости и линейную динамику. Действительно полезный метод называется ▶**итерационным LQR** (▶**ILQR**), — в этом случае работа

начинается с нахождения решения, а затем, чтобы прийти к новому решению, вокруг него итеративно вычисляются линейное приближение динамики и квадратичное приближение стоимости с последующим решением полученной системы LQR. Различные варианты метода LQR также часто используются для отслеживания траектории.

26.6. Планирование движений в условиях неопределенности

В робототехнике неопределенность возникает из-за частичной наблюдаемости среды, а также под влиянием стохастических (или не предусмотренных моделью) результатов действий робота. Также могут проявляться погрешности, обусловленные использованием приближенных алгоритмов, таких как фильтрация частиц, в результате чего робот не будет получать точных данных о текущем доверительном состоянии, даже несмотря на то, что для описания среды применяется идеальная модель.

В большинстве современных роботов для принятия решений используются детерминированные алгоритмы, такие как алгоритмы планирования пути, рассматривавшиеся в предыдущих разделах, или алгоритмы поиска, представленные в главе 3. Эти детерминированные алгоритмы адаптируются двумя способами: во-первых, они применяются в непрерывном пространстве состояния, которое преобразуется в дискретное пространство (например, с помощью графов видимости или клеточной декомпозиции). Во-вторых, они справляются с неопределенностью в текущем состояния посредством выбора ►**наиболее вероятного состояния** из распределения вероятностей, сформированного алгоритмом оценки состояния. Такой подход ускоряет вычисления и лучше подходит для детерминированных алгоритмов поиска. В этом разделе обсуждаются методы, позволяющие справляться с неопределенностью, аналогичные более сложным алгоритмам поиска, описанным в главе 4.

Во-первых, вместо детерминистических планов неопределенность требует использования стратегий. Выше уже обсуждалось, как управление отслеживанием траекторий превращает план в стратегию компенсации ошибок в динамике. Однако иногда, когда наиболее вероятная гипотеза изменяется достаточно сильно, отслеживание плана, разработанного для другой гипотезы, будет уже слишком неоптимальным. Именно здесь начинается ►**динамическая перепланировка** (*online replanning*): можно рассчитать новый план на основе нового доверительного состояния. Сегодня многие роботы используют метод под названием ►**управление с прогнозирующими моделями** (*model predictive control* — MPC), по которому планы строятся на более короткий временной горизонт и пересчитываются на каждом этапе времени. (По этой причине метод MPC тесно связан с поиском в реальном времени и игровыми алгоритмами.) Такое решение эффективно приводит

к следующей стратегии: на каждом этапе запускается планировщик и принимается первое действие во вновь построенном плане. Если поступит новая информация или робот окажется не там, где ожидалось, то это нормально, поскольку в любом случае будет выполнено перепланирование и его результат укажет, что делать дальше.

Во-вторых, неопределенность требует выполнения действий по ▶**сбору информации**. Когда рассматривается только та информация, которая уже имеется, и план составляется на ее основе (это называется отделением оценки от управления), на каждом временном этапе эффективно решается (приблизительно) новая задача MDP, соответствующая текущему доверительному состоянию о том, где робот находится или как устроен мир. Но на самом деле неопределенность лучше охватывается в структуре POMDP (частично наблюдаемой MDP): существует что-то, что непосредственно не наблюдается, — это может быть местоположение или конфигурация робота, местоположение объектов в мире или параметры самой динамической модели: например, где именно находится центр масс второго звена на этом манипуляторе?

То, что утрачивается, когда решаемая задача не является POMDP, — это способность рассуждать о *новой информации*, которую робот получит в *будущем*: в задаче MDP планируется только то, что известно, а не то, что *можно* узнать в конечном итоге. Помните о ценности информации? Роботы, которые строят планы на основе их текущего доверительного состояния так, как будто они никогда не узнают чего-то еще, не в состоянии принять во внимание ценность информации. Они никогда не предпримут действий, которые в данный момент кажутся неоптимальными на основании того, что им известно, но на самом деле приведут к получению дополнительной информации, которая позволит роботу преуспеть.

Как выглядит подобное действие для движущегося робота? Робот может переместиться ближе к опорной точке, чтобы получить более точную оценку того, где он находится, даже если эта опорная точка не лежит на его пути в соответствии с тем, что он знает в настоящий момент. Такое действие будет оптимальным только в том случае, если робот примет во внимание новые наблюдения, которые получит в результате его выполнения, в отличие от рассмотрения только той информации, которая у него уже имеется.

Чтобы обойти это условие, в методах робототехники действия по сбору информации иногда определяются явно — например, перемещение манипулятора, пока он не коснется поверхности, так называемые ▶**осторожные движения** (*guarded movements*) — и гарантируется, что робот выполнит его, прежде чем формировать план достижения его реальной цели. Каждое осторожное движение состоит из команды движения и условия завершения, представляющего собой предикат, заданный на значениях сенсора робота и определяющий, когда следует остановиться.

Иногда самой цели можно достичь с помощью последовательности осторожных движений, гарантированно успешных независимо от неопределенности.

В качестве примера на рис. 26.23 показано двухмерное пространство конфигураций с узким вертикальным отверстием. Это может быть пространство конфигурации для задачи вставки прямоугольного штифта в указанное отверстие или ключа от автомобиля — в замок зажигания. Команды движения задают постоянную скорость, а условия завершения определены как контакт с поверхностью. Чтобы смоделировать неопределенность в управлении, предположим, что вместо движения в заданном направлении фактическое движение робота лежит в охватывающем его конусе C_v.

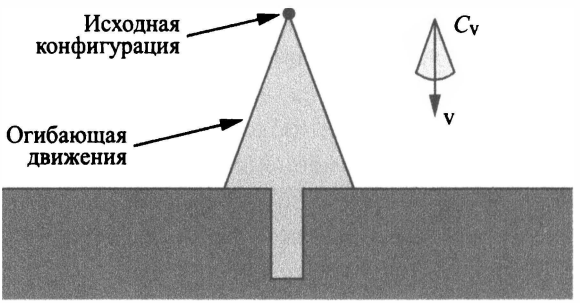


Рис. 26.23. Двухмерное пространство конфигураций, конус неопределенности скорости и огибающая возможных движений робота. Намеченная скорость равна v, но из-за неопределенности фактическая скорость может находиться где угодно в пределах конуса C_v, а это приведет к тому, что конечная конфигурация может оказаться в любой точке внутри огибающей движения. В результате возникает неопределенность в отношении того, удастся попасть в отверстие или нет

На рис. 26.23 показано, что произойдет, если робот попытается двигаться из исходной конфигурации прямо вниз. Из-за неопределенности в скорости движение может происходить в любом направлении в пределах конической огибающей, — возможно, что это приведет к попаданию в отверстие, но с большей вероятностью движение завершится на горизонтальной поверхности с той или другой стороны от него. Поскольку робот не будет иметь информации о том, с какой стороны от отверстия движение завершилось, он не будет знать и куда двигаться дальше.

Более разумная стратегия представлена на рис. 26.24 и 26.25. На рис. 26.24 робот сознательно выбирает движение в одну из сторон от отверстия. Выполняемая команда движения показана на рисунке, а условие ее завершения — контакт с поверхностью в любом месте. На рис. 26.25 команда движения задает скольжение по поверхности в направлении отверстия. Поскольку все возможные скорости в огибающей движения направлены вправо, робот будет скользить вправо всякий раз, когда вступит в контакт с горизонтальной поверхностью.

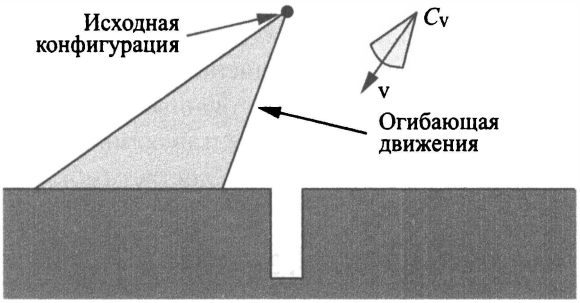


Рис. 26.24. Первая команда движения и результирующая огибающая возможных движений робота. Независимо от любых погрешностей, роботу известно, что в конечной конфигурации он будет находиться слева от отверстия

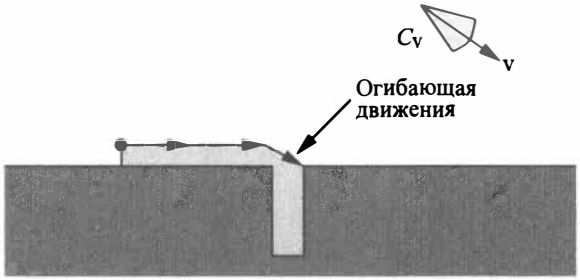


Рис. 26.25. Вторая команда движения и результирующая огибающая возможных движений робота. Даже при максимальной погрешности в направлении движения попадание в отверстие обеспечено

Достигнув отверстия, робот будет скользить вниз по его правому вертикальному краю, потому что все возможные скорости будут направлены вниз относительно вертикальной поверхности. Движение будет продолжаться, пока не будет достигнуто дно отверстия, поскольку это условие его завершения. Несмотря на неопределенность в управлении, все возможные траектории движения заканчиваются контактом с дном отверстия, — конечно, при условии, что возможные неровности на горизонтальной поверхности не вынудят робота застрять на одном месте.

В других методах, помимо осторожных движений, применяется изменение функции стоимости с целью стимулировать действия, о которых известно, что они приведут к получению информации. Примером является эвристика ►**"прибрежного плавания"** (*coastal navigation*), которая требует, чтобы робот оставался рядом с известными ему опорными точками. В более общем случае методы могут

включать ожидаемое **получение информации** (уменьшение энтропии убеждения) как терм в функции стоимости, что требует от робота явных рассуждений о том, сколько информации может принести каждое действие, когда он решает, что ему предпринять. Хотя в вычислительном отношении такие подходы сложнее, их преимущество в том, что робот изобретает собственные действия по сбору информации, вместо того чтобы полагаться на предоставляемые человеком эвристики и сценарии выбора стратегии, которым часто не достает гибкости.

26.7. Обучение с подкреплением в робототехнике

До сих пор рассматривались задачи, в которых робот имеет доступ к динамической модели мира. Однако во многих задачах записать такую модель очень сложно, что вынуждает нас обратиться к методам обучения с подкреплением.

Одной из проблем в обучении с подкреплением применительно к робототехнике является непрерывный характер пространств состояний и действий. С этой проблемой можно справиться либо посредством дискретизации, либо, чаще, посредством использования функции аппроксимации. Стратегии или функции значений представляются как комбинации известных полезных признаков или как глубокие нейронные сети. Нейронные сети способны отображать необработанные входы непосредственно в выходы и, таким образом, позволяют в значительной степени избежать необходимости разработки признаков, но они требуют значительно больше данных.

Более сложная проблема заключается в том, что роботы действуют в реальном мире. Ранее было показано, как обучение с подкреплением можно использовать для обучения игре в шахматы или го, проводя обучающие игры в симуляторе. Но когда настоящий робот движется в реальном мире, необходимо убедиться, что его действия безопасны (вещи могут ломаться!), и быть готовыми к тому, что прогресс будет не таким быстрым, как при симуляции, поскольку реальный мир отказывается двигаться быстрее, чем одна секунда за секунду. Многое из того, что нас интересует в отношении использования обучения с подкреплением в робототехнике, сводится к тому, как можно уменьшить сложность образца реального мира — сократить количество взаимодействий с физическим миром, в которых нуждается робот, прежде чем он обучится решать поставленную задачу.

26.7.1. Использование моделей

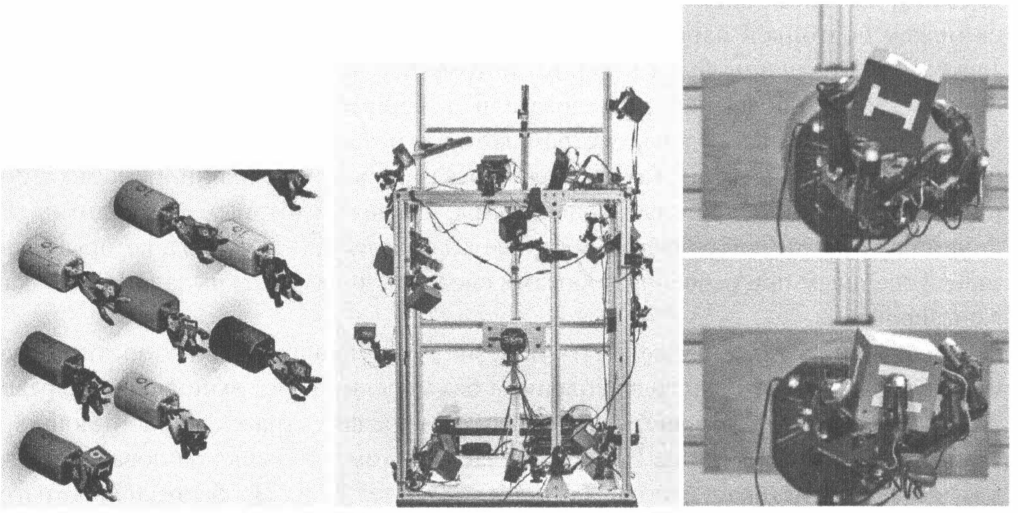
Естественный способ избежать необходимости во многих образцах из реального мира — использовать как можно больше знаний о его динамике. Например, можно не иметь точных сведений о коэффициенте трения или массе некоторого объекта, но можно знать уравнения, описывающие динамику как функцию этих параметров.

В подобном случае привлекательным вариантом является **обучение с подкреплением на основе модели** (раздел 22.1), при котором робот может переключаться между подгонкой параметров динамики и вычислением лучшей стратегии. Даже когда уравнения были неверны, потому что не обеспечивали моделирования каждой детали физики, исследователи экспериментировали с изучением терма ошибки в дополнение к параметрам, которые позволяли компенсировать неточность физической модели. Так же можно отказаться от уравнений и вместо этого подгонять локально линейные модели мира, каждая из которых аппроксимирует динамику в некоторой области пространства состояний, — подход, который оказался успешным при освоении роботами сложных динамических задач, таких как жонглирование.

Модель мира также может быть полезна для уменьшения сложности образцов в методах обучения с подкреплением без модели за счет выполнения перевода ▶**"симуляция–реальность"** (*sim-to-real*): передача стратегий, работающих в симуляции, в реальный мир. Идея заключается в том, чтобы использовать модель в качестве симулятора для поиска стратегии (раздел 22.5). Чтобы изучить стратегию, которая хорошо переносится, во время обучения можно добавить к модели шум, тем самым делая стратегию более надежной. Или можно изучать стратегии, которые будут работать с *различными* моделями путем выборки различных параметров в симуляции, — такой подход иногда называют ▶**рандомизацией предметной области**. На рис. 26.26 приведен пример, в котором задача выполнения тонких манипуляций изучается на симуляциях различных визуальных атрибутов вместе с физическими атрибутами, такими как трение или затухание.

Наконец, гибридные подходы, заимствующие идеи как из алгоритмов на основе модели, так и из безмодельных алгоритмов, направлены на то, чтобы взять лучшее из обоих. Гибридный подход впервые был использован в архитектуре Dyna, основная идея которой заключалась в переключениях между действием и улучшением стратегии, но при этом улучшение стратегии осуществлялось двумя дополняющими друг друга способами: стандартным безмодельным способом использования опыта для непосредственного обновления стратегии и основанным на модели способом использования опыта для подгонки модели, а затем способом планирования на ее основе для выработки стратегии.

В более современных методах экспериментировали с подбором локальных моделей, планированием на их основе для выработки действий и использованием этих действий в качестве руководства для подгонки стратегии с последующими итерациями для получения все лучших и лучших моделей вокруг областей, в которых нуждается стратегия. Такой подход успешно применялся в **сквозном обучении** (*end-to-end learning*), при котором стратегия принимает пиксели в качестве входных данных и выдает на выход непосредственно крутящие моменты как действия, — это позволило впервые наглядно продемонстрировать результаты применения глубокого обучения с подкреплением к физическим роботам.



а) б) в)

Рис. 26.26. Изучение надежной стратегии. **а)** Несколько имитаций передаются роботизированной руке, обучающейся тонкой манипуляции объектами с различными случайными параметрами физических размеров и освещенности. **б)** Реальная среда с одной роботизированной рукой в центре клетки, в окружении камер и дальномеров. **в)** Симуляция и обучение на реальных объектах способствовали изучению нескольких различных стратегий захватывания объектов — здесь показаны захват тремя пальцами и захват четырьмя пальцами

Модели также могут использоваться с целью обеспечения **безопасной разведки**. Обучение идет медленно, но безопасно, и это может быть лучшим вариантом, чем учиться быстро, но попасть в аварию и сгореть, пройдя лишь половину пути. Поэтому, возможно, более важным, чем сокращение реальных образцов, является сокращение образцов реального мира в *опасных* состояниях — мы не хотим, чтобы роботы падали со скал, разбивали наши любимые кружки или, что еще хуже, на полном ходу сталкивались с предметами и людьми. Аппроксимирующая модель со связанной с ней неопределенностью (например, с учетом диапазона значений ее параметров) позволяет направлять исследования и налагать ограничения на действия, которые роботу разрешено предпринимать, во избежание подобных опасных состояний. В настоящее время это одно из наиболее активных направлений исследований в области робототехники и управления.

26.7.2. Использование другой информации

Модели полезны, но чтобы дополнительно снизить сложность выборки, можно сделать и больше.

При постановке задачи обучения с подкреплением необходимо выбрать пространства состояний и действий, представление функции стратегии или стоимости и используемую функцию вознаграждения. Принятые решения оказывают большое влияние на то, насколько легко или сложно будет решить поставленную задачу.

Один из подходов заключается в использовании высокоуровневых ▶ **примитивов движения** вместо низкоуровневых действий, таких как команды задания крутящего момента. Примитив движения — это параметризованный навык, которым обладает робот. Например, у робота-футболиста может быть навык "передать мяч игроку в точке (x, y)". Все, что стратегия должна установить, — как эти примитивы объединять и как задавать их параметры, а не изобретать их заново. При таком подходе обучение часто проходит гораздо быстрее, чем при низкоуровневом подходе, однако при этом ограничивается пространство возможного поведения, которое робот может изучить.

Другой способ уменьшить количество образцов реального мира, необходимых для обучения, — повторно использовать информацию из предыдущих эпизодов обучения в других задачах вместо того, чтобы начинать все с нуля. Такой подход подпадает под эгиду **метаобучения** или **трансферного** обучения.

Наконец, люди являются отличным источником информации. В следующем разделе речь пойдет о том, как роботы могут взаимодействовать с людьми, в том числе как использовать их действия, чтобы направлять обучение робота.

26.8. Люди и роботы

До сих пор обсуждение фокусировалось на планировании и обучении роботов тому, как им следует действовать в условиях *изоляции*. Это полезно для некоторых типов роботов, таких как роверы, которые отправляют исследовать далекие планеты от нашего имени. Но по большей части люди не создают роботов для работы в полной изоляции. Их строят, чтобы помогать нам и работать в среде, где есть люди, — вокруг нас и с нами.

В результате возникают две дополнительные проблемы. Первая состоит в оптимизации вознаграждения, когда люди действуют в той же среде, что и робот. Это называют **проблемой координации** (см. раздел 18.1). Когда вознаграждение робота зависит не только от его действий, но и от действий людей, робот должен выбирать свои действия таким образом, чтобы они хорошо сочетались с действиями людей. Когда человек и робот находятся в одной команде, их взаимодействие превращается в **сотрудничество**.

Вторая проблема — это задача оптимизации того, чего люди хотят на самом деле. Если робот должен помогать людям, его функция вознаграждения должна

стимулировать действия, которых от него ожидают люди. Выявление правильной функции вознаграждения (или стратегии) для робота само по себе является задачей взаимодействия. Ниже эти две проблемы рассматриваются по очереди.

26.8.1. Координация

На данный момент будем предполагать, что у робота есть доступ к четко определенной функции вознаграждения. Но вместо того чтобы оптимизировать ее в изоляции, теперь робот должен оптимизировать ее в окружении людей, которые также действует. Например, когда беспилотный автомобиль выезжает на шоссе, робот-водитель должен согласовать свой маневр с водителем-человеком, едущим по целевой полосе, — должен ли он ускориться и выехать на шоссе перед ним или притормозить и выехать после него? Позднее, подъезжая к знаку останова и готовясь повернуть направо, он должен будет учитывать действия велосипедиста на велосипедной дорожке и следить за пешеходом, собирающимся ступить на пешеходный переход.

Или рассмотрим ситуацию, когда мобильный робот движется в коридоре. Кто-то, двигающийся прямо навстречу роботу, слегка уклоняется вправо, демонстрируя, с какой стороны от робота он хочет пройти. Робот должен ответить аналогичным образом, уточнив этим свои намерения.

Люди как приблизительно рациональные агенты

Один из способов сформулировать координацию с человеком — смоделировать ее как игру между роботом и человеком (см. раздел 18.2). При таком подходе делается явное предположение, что люди являются агентами, стимулируемыми определенными целями. Из этого вовсе не следует автоматически, что они являются совершенно рациональными агентами (т.е. находят оптимальные решения в игре), но это означает, что робот может структурировать то, что он думает о человеке, воспользовавшись понятием возможных целей, которые человек может иметь. Подобная игра характеризуется следующими элементами.

- Состояние окружающей среды фиксирует конфигурации обоих агентов — как робота, так и человека. Обозначим его как $x = (x_R, x_H)$.
- Каждый агент может предпринимать действия — u_R и u_H соответственно.
- У каждого агента есть цель, которую можно представить в виде стоимости, J_R и J_H. Каждый агент стремится достичь своей цели безопасно и эффективно.
- Как и в любой игре, каждая цель зависит от состояния и действий *обоих* агентов: $J_R(x, u_R, u_H)$ и $J_H(x, u_H, u_R)$. Вспомним о взаимодействии между автомобилем и пешеходом: машине следует остановиться, если пешеход ступает на переход, и можно двигаться вперед, если пешеход ждет на тротуаре.

Три важных аспекта усложняют эту игру. Во-первых, человек и робот необязательно знают цели друг друга. Это делает игру ▶ **игрой с неполной информацией**.

Во-вторых, пространство состояний и действий *непрерывно*, как это было на протяжении всего обсуждения в данной главе. В главе 5 объяснялось, как выполнить поиск по дереву для нахождения решения в играх с дискретными пространствами, но как это сделать в случае игр с непрерывными пространствами?

В-третьих, хотя на высоком уровне игровая модель имеет смысл — люди предпринимают действия и у них есть цели, — поведение человека не всегда можно охарактеризовать как решение игры. Игра связана с вычислительными трудностями не только для робота, но и для нас, людей. Необходимо думать о том, что робот будет делать в ответ на то, что делает человек, а это зависит от того, что робот подумает о том, что будет делать человек, и довольно скоро мы доберемся до "что ты думаешь, я думаю, ты думаешь, что я думаю" — и так далее до бесконечности! Люди не могут справиться со всем этим и проявляют определенные субоптимальности. А это означает, что робот должен учитывать подобные субоптимальности.

Итак, что же делать беспилотному автомобилю, если проблема координации настолько сложна? Предпримем нечто похожее на то, что уже делалось ранее в этой главе. В задаче планирования пути и управления его прохождением базовая задача марковского принятия решения была разбита на две: планирование траектории и последующее отслеживание ее прохождения с помощью контроллера. В данном случае мы также возьмем игру и разделим ее на два этапа: прогнозирование возможных действий человека и принятие решения о том, что робот должен делать, учитывая эти прогнозы.

Прогнозирование действий человека

Предсказывать действия человека сложно, потому что они будут зависеть от действий робота, и наоборот. Одна из хитростей, которые можно использовать в робототехнике, — притвориться, что человек игнорирует робота. Робот предполагает, что люди оптимальны (с учетом некоторого шума) в отношении своей цели, которая роботу неизвестна и моделируется им как не зависящая от действий робота: $J_H(x, u_H)$. В частности, чем выше ценность действия для достижения цели (чем ниже стоимость его выполнения), тем больше вероятность, что человек его предпримет. Робот может создать модель для $P(u_H \mid x, J_H)$, например, с использованием функции softmax (раздел 22.5, уравнение (22.14)):

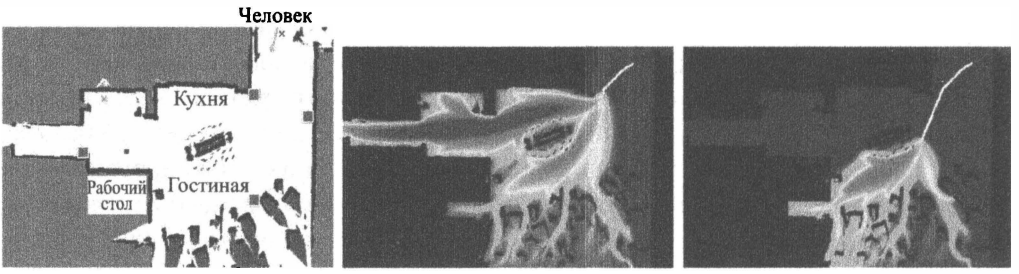
$$P(u_H \mid x, J_H) \propto e^{-Q(x, u_H; J_H)}, \tag{26.8}$$

где $Q(x, u_H; J_H)$ — функция Q-значения, соответствующая J_H (отрицательный знак используется здесь потому, что в робототехнике принято минимизировать стоимость (затраты), а не максимизировать вознаграждение). Обратите внимание, что робот вообще не предполагает выбор действительно оптимальных действий, как не предполагает и того, что действия выбираются на основе рассуждений о нем.

Вооруженный этой моделью робот использует текущие действия человека как свидетельство в отношении J_H. Если есть модель наблюдения за тем, как человеческие действия зависят от цели человека, то каждое действие человека может быть включено в нее с целью обновления убеждений робота в отношении того, что является целью человека:

$$b'(J_H) \propto b(J_H)P(u_H \mid x, J_H). \tag{26.9}$$

Соответствующий пример приведен на рис. 26.27: робот отслеживает местоположение человека и по мере его перемещения обновляет свои убеждения в отношении его цели. Когда человек направляется к окну в гостиной, робот повышает вероятность того, что его цель — посмотреть в окно, и уменьшает вероятность того, что цель человека — пойти на кухню, которая находится в другом направлении.



а) б) в)

Рис. 26.27. Прогнозирование в предположении, что люди рациональны (с учетом некоторого шума) в отношении своих целей: робот использует предпринятые человеком действия для обновления своих убеждений о том, к какой цели человек направляется, а затем использует эти убеждения для составления прогнозов о его будущих действиях. **а)** План комнаты, в которую заходит человек (светло-серая линия вверху). **б)** Прогноз действий после наблюдения небольшой части истинной траектории движения человека (белая линия). **в)** Изменение прогноза после наблюдения за последующими действиями человека: теперь робот знает, что человек не направляется в коридор слева, поскольку уже пройденный им на данный момент путь был бы плохим выбором, если бы именно это было целью человека

Таким образом, прежние действия человека в конечном счете информируют робота о том, что человек собирается делать в будущем. Наличие убеждений относительно цели человека помогает роботу предвидеть, какие дальнейшие действия предпримет человек. Тепловая карта на рис. 26.27, *б*, *в* демонстрирует прогнозы робота на будущее: светло-серые линии представляют наиболее вероятные траектории будущих перемещений человека.

То же самое может иметь место в дорожном движении. Можно не знать оценок эффективности другого водителя, но если увидеть, что он ускоряет движение,

когда кто-то перед ним пытается перестроиться на его полосу, то можно получить определенные сведения о нем. И узнав это, можно лучше предвидеть, что он будет делать в будущем, — приблизившись к нам, такой водитель, вероятно, начнет маневрировать, чтобы так или иначе обогнать и пробиться вперед.

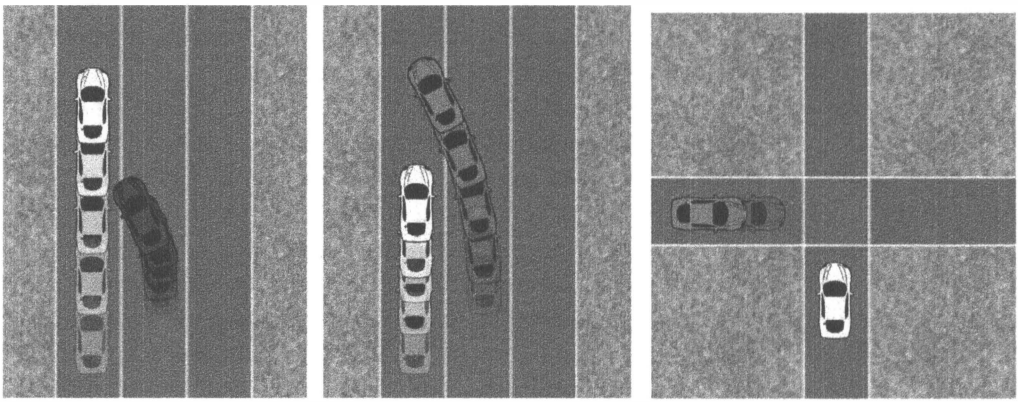
Получив возможность предсказывать будущие действия человека, робот существенно упрощает себе решение лежащей в основе задачи MDP. Действия человека усложняют функцию перехода, но пока робот способен предвидеть, какие действия предпримет человек в любом будущем состоянии, он может вычислить $P(x' \mid x, u_R)$: вычислить $P(u_H \mid x)$ из $P(u_H \mid x, J_H)$ путем маргинализации по J_H и объединения результата с $P(x' \mid x, u_R, u_H)$, функцией перехода (динамики) для определения, как обновляется мир, основываясь на действиях как робота, так и человека. В разделе 26.5 обсуждалось, как можно решить такую задачу в непрерывных пространствах состояний и действий для детерминированной динамики, а в разделе 26.6 — как делать это при условиях стохастической динамики и неопределенности.

Отделение предсказания от действия упрощает роботу взаимодействие, но приводит к снижению производительности, — так же, как отделение оценки от движения или отделение планирования от управления.

При таком разделении робот уже не понимает, что его действия могут влиять на то, что люди в конечном итоге будут делать. В противоположность этому робот, представленный на рис. 26.27, прогнозирует, куда пойдет человек, а затем оптимизирует достижение своей цели, одновременно избегая столкновения с ним. На рис. 26.28, *а* робот управляет беспилотным автомобилем и намерен перестроиться на шоссе из одной полосы движения в другую. Если его план будет строиться лишь на ожидании подходящих действий со стороны водителей других автомобилей, ему, возможно, придется долго ждать, пока целевая полоса освободится. В противоположность этому робот, одновременно рассуждающий и о прогнозе, и о выборе действия, знает, что различные действия, которые он может предпринять, могут привести к различной реакции со стороны человека. Если он активно проявит свое намерение перестроиться, другие автомобили, скорее всего, немного замедлятся и освободят ему место. Создатели роботов работают в направлении координации сотрудничества подобным образом, что позволит роботам лучше взаимодействовать с людьми.

Предположения человека о роботе

Неполная информация часто бывает двусторонней: робот не знает цели человека, а человек, в свою очередь, не знает целей *робота* — людям приходится делать прогнозы в отношении действий роботов. Как разработчики роботов мы не отвечаем за то, какие предположения сделает человек; мы можем управлять только тем, что делает робот. Тем не менее робот может действовать таким образом, чтобы человеку было *проще* делать правильные прогнозы. Робот может предположить, что человек для оценки цели робота J_R использует что-то, аналогичное уравнению



а) б)

Рис. 26.28. а) ***Слева***: робот — водитель беспилотного автомобиля (темная машина на средней полосе) прогнозирует, что человек-водитель (на левой полосе) намерен продолжить движение вперед, и планирует траекторию с торможением и выездом на новую полосу после него. ***Справа***: робот учитывает влияние, которое его действия могут оказать на действия человека, и осознает, что может перестроиться *перед* автомобилем на целевой полосе, положившись на человека-водителя, который замедлит движение. **б)** Тот же алгоритм приводит к необычной стратегии на перекрестке: робот-водитель решает, что может сделать более желательным для человека-водителя (светлый автомобиль внизу) проехать через перекресток быстрее, если слегка переместиться в обратном от него направлении

(26.8), и следовательно, робот будет действовать так, что его истинная цель может быть легко выведена.

Особый случай игры — это когда человек и робот находятся в одной команде и действуют в направлении достижения одной и той же цели или решения одной и той же задачи: $J_H = J_R$. Представим персонального домашнего робота, который помогает вам готовить ужин или делать уборку, — это примеры **сотрудничества**.

Теперь можно дать определение ▶**совместного агента**, действия которого представляют собой кортежи из действий человека и робота (u_H, u_R) и который оптимизирует функцию $J_H(x, u_H, u_R) = J_R(x, u_R, u_H)$, — осталось решить обычную задачу планирования. Рассчитываем оптимальный план или стратегию для совместного агента, и — готово: теперь известно, что должны делать робот и человек.

Все это работало бы очень хорошо, если бы поведение людей было абсолютно оптимальным. Робот выполняет свою часть совместного плана, человек — свою. К сожалению, на практике люди, вероятно, не станут следовать тщательно продуманному плану совместного агента, — у них свой разум! Нам уже известен хотя бы один из способов справиться с этим. Обратитесь к разделу 26.6 — в нем этот

подход получил название **модель прогнозирующего контроля (MPC)**: основная идея состояла в том, чтобы составить план, выполнить его первое действие, а затем составить новый план. Кстати, в этом случае робот всегда будет адаптировать свой план к тому, что на самом деле делает человек.

Давайте продолжим рассуждения на примере. Предположим, что человек и робот находятся на кухне и решили испечь вафли. Человек находится чуть ближе к холодильнику, поэтому оптимальный совместный план будет таким: он берет яйца и молоко из холодильника, а робот в это время достает муку из шкафа. Робот знает этот план, поскольку может точно определить, что где находится. Но предположим, что человек начинает движение к шкафу с мукой, т.е. действует вопреки оптимальному совместному плану. Вместо того чтобы строго придерживаться исходного плана и упорно стремиться достать муку, MPC-робот пересчитает оптимальный план, и теперь, когда человек оказался достаточно близко к муке, решит, что ему лучше всего достать чугунную вафельницу.

Поскольку известно, что люди могут отклоняться от оптимальности действий, это можно учесть заранее. В нашем примере робот может попытаться предугадать, что человек отправится за мукой в тот момент, когда он сделает свой первый шаг (скажем, используя методику прогнозирования, обсуждавшуюся выше). Даже когда технически все еще будет оптимальным, если на следующем шаге человек развернется и направится к холодильнику, робот не должен предполагать, что именно это и произойдет. Вместо этого робот может просчитать план, в котором человек продолжит делать то, чего ему, кажется, хочется.

Люди как агенты типа "черный ящик"

Нет необходимости рассматривать людей только как действующих исключительно во имя достижения цели агентов, чтобы роботы могли координировать с ними свои действия. Альтернативная модель состоит в том, что человек — это просто некий агент, стратегия которого π_H "вносит возмущения" в динамику среды. Робот не знает стратегии π_H, но может моделировать задачу как необходимость действовать в среде MDP с неизвестной динамикой. Это уже обсуждалось ранее: для агентов в общем смысле — в главе 22, а для роботов в частности — в разделе 26.7.

Робот может подогнать модель стратегии π_H к данным о человеке, а затем использовать ее для вычисления оптимальной стратегии для себя. Из-за нехватки данных такой подход до сих пор использовался в основном на уровне задач. Например, благодаря взаимодействию роботы обучались тому, какие действия люди обычно предпринимают (в ответ на их собственные действия), в задаче помещения в нужное место и закручивания винтов в процессе промышленной сборки.

В этом случае также существует альтернатива безмодельного обучения с подкреплением: робот может начать с некоторой начальной стратегии или функции стоимости и далее улучшать ее с течением времени методом проб и ошибок.

26.8.2. Обучение выполнению того, чего хотят люди

Другое средство организации правильного взаимодействия с людьми в робототехнике — это сама функция J_R — функция стоимости или вознаграждения робота. Структура рациональных агентов и связанные с ней алгоритмы сводят задачу обеспечения хорошего поведения к определению хорошей функции вознаграждения. Но для роботов, как и для многих других агентов искусственного интеллекта, правильное определение вознаграждения все еще очень сложно.

Возьмем беспилотные автомобили: необходимо, чтобы они достигали пункта назначения, были безопасными, обеспечивали комфортное вождение для пассажиров и соблюдение правил дорожного движения и т.д. Разработчик такой системы должен найти компромисс между этими различными компонентами функции стоимости. Задача разработчика сложна, потому что роботы создаются для оказания помощи конечным пользователям, однако не все конечные пользователи одинаковы. Так, у всех нас разные предпочтения в отношении того, насколько агрессивным может быть движение везущего нас автомобиля и т.д.

Ниже будет рассмотрено два альтернативных подхода к тому, чтобы попытаться обеспечить такое поведение робота, которое будет соответствовать тому, что действительно нужно делать роботу. Первый заключается в изучении функции стоимости на основе входных данных, поступающих от человека. Второй — отказаться от функции стоимости и просто подражать демонстрациям того, как человек решает поставленную задачу.

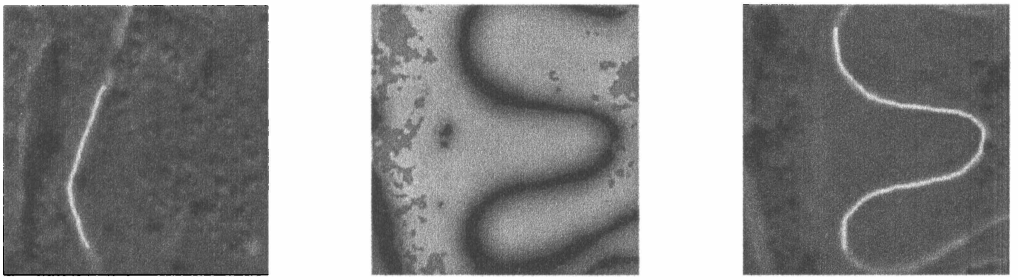
Обучение предпочтениям: изучение функции стоимости

Представим, что конечный пользователь показывает роботу, как выполняется поставленное задание. Например, он ведет автомобиль так, как он хотел бы, чтобы им управлял робот. Можете ли вы придумать для робота способ использования этих действий — назовем их "демонстрации", — чтобы он смог уяснить, какую именно функцию стоимости ему следует оптимизировать?

На самом деле ответ на этот вопрос уже был предложен в разделе 26.8.1. Формулировка задачи там была немного иной: был некий человек, выполняющий действия в том же пространстве, что и робот, и роботу нужно было предсказать, какие действия этот человек предпримет. Тот метод, который использовался для получения таких прогнозов, заключался в предположении, что люди действуют с целью оптимизации (при наличии шума) некоторой функции стоимости J_H что и их текущие действия можно использовать в качестве свидетельств того, что эта функция стоимости собой представляет. То же самое можно сделать и в данном случае, за исключением того, что теперь это необходимо не для прогнозирования поведения человека в будущем, а скорее для выявления собственной функции стоимости робота, которую он должен оптимизировать. Если человек водит машину осторожно, функция стоимости, описывающая его действия, придаст большее значение безопасности и меньшее — эффективности. Робот может принять эту функцию

стоимости как свою собственную и оптимизировать ее, когда сам будет управлять машиной.

Создатели роботов экспериментировали с различными алгоритмами, позволяющими сделать выведение стоимости вычислительно реализуемым. На рис. 26.29 представлен пример обучения робота предпочтению придерживаться движения по дороге вместо сокращения пути за счет движения по бездорожью. Традиционно в таких методах функция стоимости представлялась как комбинация отобранных вручную признаков, но в недавних работах также исследовалось, как можно представить эту функцию с помощью глубокой нейронной сети, без ручной разработки признаков.



а) б) в)

Рис. 26.29. а) Мобильному роботу предоставлена демонстрация необходимости придерживаться движения по грунтовой дороге. **б)** Робот вывел требуемую функцию стоимости и применил ее в новой сцене, зная, как можно снизить затраты на перемещение. **в)** В новой сцене, воспользовавшись предпочтениями, полученными при демонстрации, робот спланировал путь, также придерживаясь движения по дороге

Существуют и другие способы, посредством которых человек может предоставить роботу необходимые исходные сведения. Так, для инструктирования робота человек может использовать язык, а не визуальные демонстрации. Человек также может выступить в роли критика, проследив, как робот выполняет поставленное задание тем или иным способом (или двумя способами), а затем указав, насколько хорошо было выполнено задание (или какой способ был лучше), и дав совет, как можно выполнить задание лучше.

Изучение стратегий непосредственно через имитацию

Альтернативным вариантом является отказ от задания функций стоимости и обучение роботов желаемой *стратегии* напрямую. Выше, в примере обучения управляющего беспилотным автомобилем робота, демонстрации человека образуют подходящий набор данных из состояний, помеченных действием, которое робот должен выполнять в каждом из этих состояний: $\mathcal{D} = \{(x_i, u_i)\}$. Робот использует

методы обучения с учителем для подгонки стратегии $\pi : x \mapsto u$, а затем применяет эту стратегию. Подобный подход называется **имитационным обучением** или ►**клонированием поведения**.

При таком подходе основная проблема заключается в ►**обобщении** новых состояний. Робот не знает, *почему* действия в его базе данных были помечены как оптимальные, у него нет правила, определяющего причину такого выбора. Все, что он в данном случае может сделать, — это запустить алгоритм обучения с учителем, чтобы попытаться изучить стратегию, которая будет обобщать неизвестные состояния. Однако нет никаких гарантий, что это обобщение будет правильным.

Такой подход использовался в проекте беспилотного автомобиля ALVINN: было обнаружено, что даже при начале движения из состояния, входящего в $\mathcal{D}$, стратегия π делает небольшие ошибки, уводящие машину от продемонстрированной траектории. Иногда стратегия π допускает и бо́льшие ошибки, которые вообще могут увести машину в сторону от желаемого курса.

Эту проблему можно решить в процессе обучения, если чередовать сбор меток и обучение: начинаем с демонстрации, изучаем стратегию, развертываем эту стратегию и спрашиваем у человека-учителя, какое действие следует предпринять в каждом состоянии вдоль пути, а затем все повторяется. В конечном счете робот учится исправлять свои ошибки, если отклоняется от действий, которые хочет от него человек.

Альтернативный вариант решения этой проблемы — использование обучения с подкреплением. Робот может построить динамическую модель на основании демонстраций, а затем использовать оптимальное управление (раздел 26.5.4) для выработки стратегии, оптимизирующей его действия таким образом, чтобы как можно ближе придерживаться демонстраций. Версия этого подхода была использована для выполнения очень сложных маневров на экспертном уровне небольшим радиоуправляемым вертолетом (см. рис. 22.9, *б* в разделе 22.7.2.).

Обучение системы DAGGER (*Data Aggregation* — агрегация данных) было начато с демонстрации действий человека-эксперта. На этом основании была изучена стратегия π_1, использованная для генерации набора данных $\mathcal{D}$. Затем на основе набора $\mathcal{D}$ была сгенерирована новая стратегия π_2, которая лучше имитировала исходные данные, полученные от человека. Эта процедура повторялась раз за разом, и на каждой n-й итерации стратегия π_n использовалась для генерации дополнительных данных, добавляемых к набору $\mathcal{D}$, который затем использовался для создания стратегии π_{n+1}. Другими словами, на каждой итерации система собирала новые данные в соответствии с текущей политикой и изучала следующую политику, используя все данные, собранные к этому моменту.

Соответствующие новые методы используют **состязательное обучение**: при этом подходе обучение классификатора поиску различий между изученной роботом стратегией и демонстрациями человека чередуется с обучением робота новой

стратегии методом обучения с подкреплением с целью обмануть классификатор. Такой подход укрепляет способности робота обрабатывать состояния, близкие к демонстрациям, и при этом по ходу дела обобщать неизвестные состояния или новую динамику.

Обучающие интерфейсы и проблема соответствия. До сих пор рассматривались примеры беспилотного автомобиля или автономного вертолета, в которых в демонстрациях учителей-людей использовались такие же действия, какие робот способен выполнять сам: ускорение, торможение и рулевое управление. Но что произойдет, если применить этот подход для таких задач, как уборка кухонного стола? В этом случае возможны два варианта: либо человек-учитель демонстрирует использование своего тела, пока робот наблюдает, либо человек физически нужным образом направляет манипуляторы робота.

Первый подход привлекателен, потому что он естественен для конечных пользователей. К сожалению, он осложняется из-за ►**проблемы соответствия**: как отобразить действия человека на действия робота? Люди обладают кинематикой и динамикой, не соответствующей кинематике и динамике роботов. Это не только осложняет *перевод* или *перенацеливание* движений человека на движения робота (например, перенацеливание хватки человека пятью пальцами на хватку робота двумя пальцами); часто сама стратегия высокого уровня, которую может использовать человек, не подходит для робота.

Второй подход, когда учитель-человек перемещает манипуляторы робота в правильные позиции, называется ►**кинестетическим обучением** (рис. 26.30). Людям нелегко проводить обучение таким образом, особенно в случаях, когда манипулятор робота включает несколько шарниров. Учитель должен координировать все степени свободы, направляя манипулятор робота в процессе решения

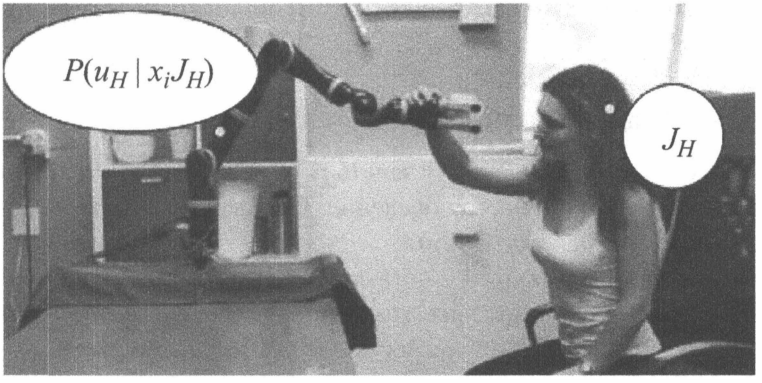


Рис. 26.30. Учитель-человек толкает манипулятор робота вниз, чтобы научить его держать манипулятор ближе к столу. Робот соответствующим образом обновляет свое понимание желаемой функции стоимости и запускает ее оптимизацию

поставленной задачи. Поэтому экспериментаторы исследовали возможные альтернативы, такие как демонстрация ▶**ключевых кадров** в отличие от непрерывных траекторий, а также использование ▶**визуального программирования**, чтобы позволить конечным пользователям программировать примитивы, полезные для решения задачи, а не продемонстрировать все с нуля (рис. 26.31). Иногда оба подхода комбинируются.

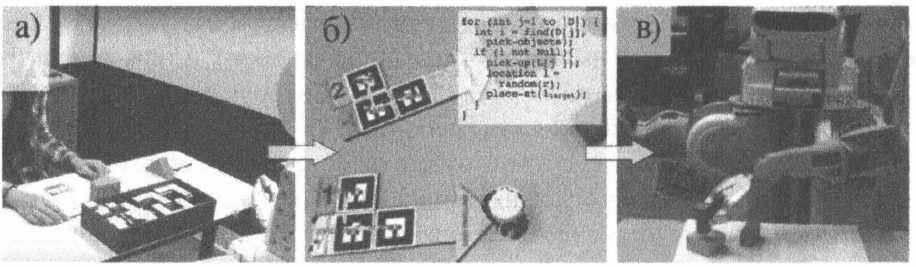


Рис. 26.31. Интерфейс программирования, включающий размещение специально разработанных блоков в рабочей области робота для выбора объектов и определения действий высокого уровня

26.9. Альтернативные архитектуры роботов

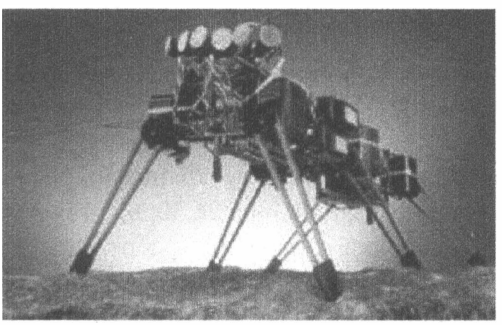
До сих пор в этой главе был принят взгляд на робототехнику, основанный на понятии определения или изучения функция вознаграждения, и роботу требовалось оптимизировать эту функцию вознаграждения (посредством планирования или обучения), иногда при координации или сотрудничестве с людьми. Это ▶**совещательный** подход в робототехнике, противоположный ▶**реактивному** или **реагирующему** подходу.

26.9.1. Реактивные контроллеры

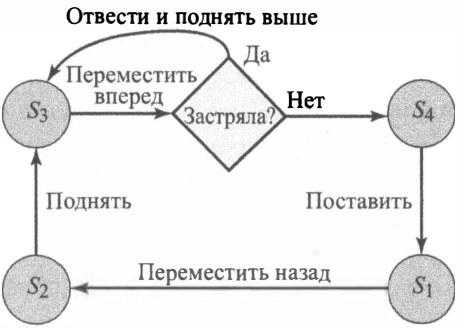
В некоторых случаях проще сразу задать роботу хорошую стратегию, а не моделировать мир и планировать его поведение. И тогда вместо *рационального* агента можно получить *рефлексного* агента.

Например, представим робота с ногами, который пытается поднять ногу над препятствием. Можно было бы дать этому роботу следующее правило: поднять ногу на небольшую высоту h и переместить ее вперед, а если нога столкнется с препятствием, вернуть ее назад и повторить действие при большем значении высоты. Можно сказать, что h здесь моделирует некоторый аспект мира, но можно рассматривать h и как вспомогательную переменную контроллера робота, лишенную прямого физического смысла.

Одним из примеров такого рода является шестиногий робот, или гексапод, показанный на рис. 26.32, *а*, предназначенный для ходьбы по пересеченной местности. Датчики робота не позволяют получить модель местности с точностью, требуемой для планирования пути. Более того, даже если установить на нем высокоточные камеры и дальномеры, задача планирования пути при 12 степенях свободы (по две на каждую ногу) будет чрезвычайно сложна вычислительно.



а)



б)

Рис. 26.32. а) Чингис, робот-гексапод. **б)** Расширенный конечный автомат (*Augmented Finite State Machine* — AFSM) для управления одной ногой. Обратите внимание, что этот автомат AFSM реагирует на сенсорную обратную связь: если какая-то нога не может продвинуться вперед при выполнении этапа ее перенесения вперед, то каждый раз она поднимается все выше и выше

Тем не менее можно определить контроллер непосредственно, без использования явной модели среды. (Выше в данной главе такой подход был продемонстрирован на примере PD-контроллера, который обеспечивал проведение сложного манипулятора робота к цели *без* явной модели динамики робота.)

Для робота-гексапода сначала выбирается ▶**походка**, или схема движения конечностей. Одна статически устойчивая походка состоит в том, чтобы сначала передвинуть вперед правую переднюю, правую заднюю и левую среднюю ноги (сохранив три остальные в прежнем положении), а затем переместить три остальные. Такая походка хорошо работает на ровной местности. На пересеченной местности перемещению ноги вперед может помешать препятствие. Такое затруднение можно преодолеть с помощью исключительно простого правила управления: *если движение ноги вперед блокируется, следует отвести ее немного назад, поднять выше и попробовать еще раз*. Полученный в итоге контроллер показан на рис. 26.32, *б* в виде конечного автомата: он представляет собой рефлексного агента с поддержкой состояния, в котором внутреннее состояние представлено индексом текущего состояния автомата (от s_1 до s_4).

26.9.2. Архитектуры подчинения

▶ **Архитектура подчинения** (Брукс [312], 1986) — это рабочая среда для сборки реактивных контроллеров из конечных автоматов. Узлы в этих машинах могут включать проверку значения определенных переменных датчиков, и в этом случае трасса выполнения конечного автомата будет обусловлена итогом такой проверки. Дуги могут быть помечены сообщениями, которые будут сгенерированы при их прохождении, а затем отправлены двигателям робота или другим конечным автоматам. Кроме того, конечные автоматы обладают внутренними таймерами (часами), которые контролируют время, необходимое для прохождения по дуге. Получающиеся в результате автоматы называются ▶ **расширенными конечными автоматами** (*augmented finite state machines* — ▶ **AFSM**), в которых расширение обусловлено использованием часов.

Примером простого AFSM является автомат с четырьмя состояниями, речь о котором шла в предыдущем подразделе, — он показан на рис. 26.32, *б*. Этот расширенный конечный автомат реализует циклический контроллер, работа которого в основном не полагается на обратную связь с окружающей средой. Однако выполнение фазы движения вперед зависит от обратной связи от датчика. Если нога застряла, то это означает, что выполнить ее перемещение вперед не удалось, поэтому робот возвращает ногу в прежнее положение, поднимает ее немного выше и снова пытается переместить ее вперед. Следовательно, контроллер способен *реагировать* на непредвиденные обстоятельства, возникающие при взаимодействии робота с окружающей средой.

Архитектура подчинения предлагает дополнительные примитивы для синхронизации автоматов AFSM и для объединения выходных значений нескольких, возможно, конфликтующих автоматов. В итоге это позволяет программисту создавать все более усложняющиеся контроллеры в направлении снизу вверх. В нашем примере можно начать с автоматов AFSM для отдельных ног, а затем определить AFSM для координация нескольких ног. Поверх этого можно реализовать поведение и более высокого уровня, например предотвращение столкновений, предусматривающее отход назад и поворот.

Идея составления контроллеров роботов из AFSM довольно привлекательна. Представьте, насколько сложно было бы реализовать такое же поведение при использовании любого из алгоритмов планирования пути в пространстве конфигурации, описанных в предыдущих разделах. Прежде всего, потребовалась бы точная модель окружающей среды, а пространство конфигурации робота с шестью ногами, каждая из которых приводится в движение двумя независимыми двигателями, имело бы в целом 18 измерений (12 измерений для конфигурации ног и 6 — для описания расположения и ориентации робота относительно окружающей среды). Даже если иметь в распоряжении достаточно быстрые компьютеры, позволяющие успешно отыскать путь в подобном многомерном пространстве, пришлось бы дополнительно побеспокоиться и о таких неприятных эффектах, как скольжение робота по склону.

Из-за таких стохастических эффектов единственный путь через пространство конфигураций, скорее всего, будет слишком ненадежным и даже ПИД-контроллер не сможет справиться с подобными непредвиденными обстоятельствами. Другими словами, продуманно генерировать поведение при движении в некоторых случаях будет просто слишком сложной задачей для современных алгоритмов планирования движения роботов.

К сожалению, у архитектуры подчинения есть свои проблемы. Во-первых, автоматы AFSM управляются необработанным входом датчика, — этот механизм работает, если данные датчика надежны и содержат всю необходимую информацию для принятия решения, но он не будет работать, если данные датчика требуется интегрировать нетривиальным способом с течением времени. Поэтому контроллеры в стиле архитектуры подчинения применяются в основном при решении простых задач, таких как следование вдоль стены или движение к видимым источникам света.

Во-вторых, отсутствие обдумывания действий затрудняет изменение целей робота. Робот с архитектурой подчинения обычно выполняет только одно задание, и у него нет понимания того, как изменить свои элементы управления, чтобы приспособиться к различным заданиям (как навозный жук в разделе 2.2.3).

В-третьих, во многих реальных проблемах желательная для нас стратегия часто оказывается слишком сложной для кодирования в явном виде. Вспомните пример на рис. 26.28, когда беспилотному автомобилю было необходимо согласовать свои действия по изменению полосы движения с водителем-человеком. Можно было бы начать с простой стратегии, которая просто выводит автомобиль на целевую полосу на шоссе. Но при тестировании системы в реальных условиях сразу бы обнаружилось, что далеко не каждый водитель на целевой полосе движения замедлится, чтобы пропустить перед собой машину. Тогда можно было бы добавить немного больше сложности: автомобиль смещается к целевой полосе, дожидается ответа от водителя на этой полосе, а затем либо продолжает перестроение, либо отступает назад. Но при последующем тестировании машины выясняется, что смещение к целевой полосе должно происходить с разной скоростью в зависимости от скорости транспортного средства на ней, от того, есть ли другое транспортное средство впереди на целевой полосе, от того, следует ли какое-то транспортное средство за машиной в исходной полосе, и т.д. Количество условий, которые необходимо учитывать для определения правильной последовательности действий, может быть очень велико даже для такого на первый взгляд простого маневра. А это, в свою очередь, создает проблему масштабируемости для архитектур в стиле подчинения.

Все это говорит о том, что робототехника является сложной областью, решение задач в которой требует использования нескольких подходов (совещательного, реактивного или их сочетания), основанных на использовании законов физики, когнитивных моделей, данных или их комбинации. Правильный подход все еще является предметом дискуссий, научных исследований и инженерного мастерства.

26.10. Области применения робототехники

Робототехнические технологии уже широко охватывают весь мир и обладают необходимым потенциалом для улучшения нашей независимости, здоровья и производительности. Вот несколько примеров их применения.

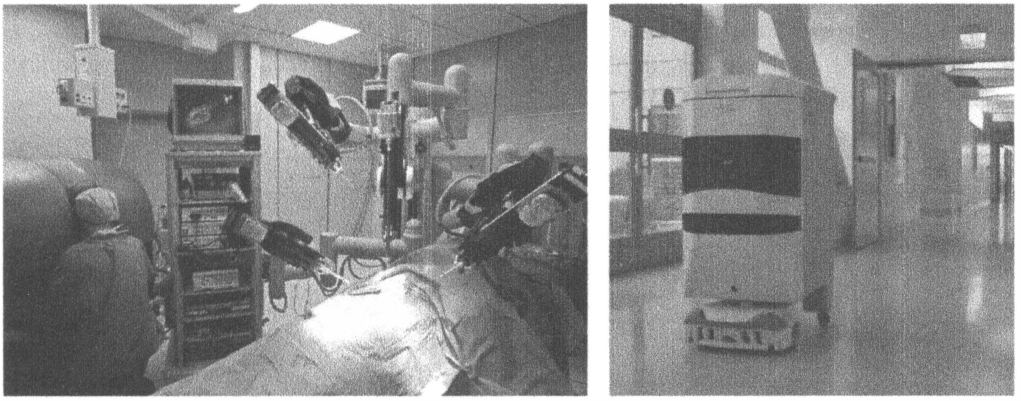
Домашний круг. Роботы начали входить в наши дома для ухода за пожилыми людьми и людьми с ограниченной подвижностью, помогая им в повседневной жизни и позволяя им жить более независимо. Сюда относятся инвалидные коляски и установленные на них манипуляторы-руки, подобные Kinova JACO, показанной на рис. 26.1, *б*. Несмотря на то что вначале они управлялись непосредственно человеком, сейчас эти роботы получают все больше и больше автономии. Уже существуют образцы роботов, управляемых с помощью ▶ **интерфейса "мозг–машина"**, которые, как было продемонстрировано, позволяют людям, страдающим квадриплегией, использовать руку робота для захватывания предметов и даже приема пищи (рис. 26.33, *а*). Подобные примеры также включают протезы конечностей, которые разумно реагируют на наши действия, и экзоскелеты, придающие нам сверхчеловеческие силы или позволяющие людям, которые не могут управлять своими мышцами ниже талии, вновь ходить.

а) б)

Рис. 26.33. а) Пациент, управляющий через интерфейс "мозг–машина" рукой робота для захвата банки с напитком. **б)** Робот-пылесос Roomba

Роботы-ассистенты помогают людям выполнять различные повседневные задачи, такие как уборка и наведение порядка, освобождая их от необходимости тратить время на эти рутинные операции. Хотя в отношении выполнения различных манипуляций еще предстоит немало сделать, прежде чем роботы смогут беспрепятственно работать в беспорядочной, неструктурированной среде обитания человека, в области навигации уже достигнуты определенные успехи. В частности, уже во многих домах имеется мобильный робот-пылесос, подобный показанному на рис. 26.33, *б*.

Здравоохранение. Роботы ассистируют хирургам и расширяют их возможности, обеспечивая более точные и минимально инвазивные действия, а также более безопасное выполнение процедур с лучшими результатами для пациента. Хирургический робот Da Vinci, показанный на рис. 26.34, *а*, сейчас широко используется в больницах США.



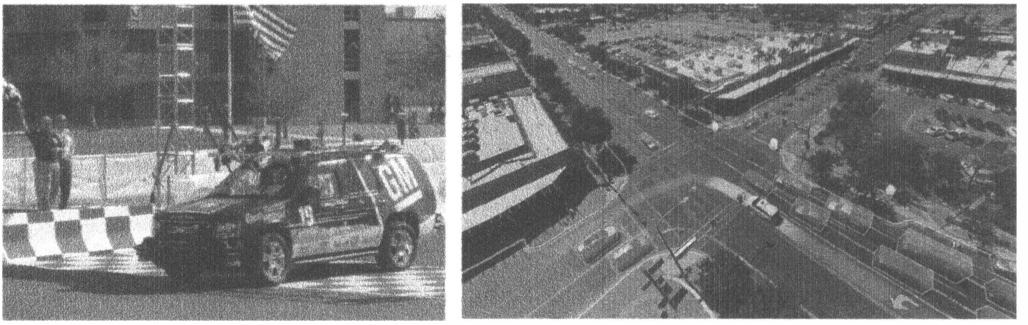
а) б)

Рис. 26.34. а) Хирургический робот в операционной. **б)** Робот для доставки, действующий в больнице

Предоставление услуг. Мобильные роботы оказывают различную помощь людям в офисных зданиях, гостиницах и больницах. В сети отелей Савиоке роботы используются для доставки в номера клиентов всего необходимого, например заказанного обеда, полотенец или зубной пасты. Роботы-помощники Helpmate и TUG доставляют пищу и лекарства пациентам в больницах (рис. 26.34, *б*), а робот Moxi компании Diligent Robotics помогает медсестрам выполнять их обязанности. Робот Co-Bot бродит по помещениям Университета Карнеги-Меллона, всегда готовый отвести вас в чей-то офис. Также существует возможность использовать ▶ **роботов дистанционного присутствия**, таких как Beam, для участия в удаленных совещаниях и конференциях или для того, чтобы проверить, как поживают наши бабушки и дедушки.

Беспилотные автомобили. Некоторые из нас иногда отвлекаются во время вождения, чтобы ответить на звонок по мобильному телефону, отправить сообщение или сделать что-либо еще. Иногда такие действия приводят к печальным последствиям: более миллиона человек каждый год погибают в дорожных авариях. Кроме того, многие из нас проводят очень много времени за рулем и охотно согласились бы освободить для себя хотя бы часть этого времени. Все это способствовало настойчивому и массовому приложению усилий по созданию и распространению беспилотных автомобилей.

Прототипы таких автомобилей существуют с 1980-х годов, но особенно быстрый прогресс в этой области был стимулирован соревнованиями DARPA 2005 Grand Challenge, гонками беспилотных автомобилей, которым предстояло преодолеть более 200 км по неизведанной пустынной территории. Автомобиль Стэнфорда Стэнли прошел этот путь менее чем за семь часов, выиграв приз в 2 млн долл. и обеспечив себе место в Национальном музее американской истории. На рис. 26.35, *а* представлен беспилотный автомобиль BOSS, который в 2007 году выиграл приз соревнований DARPA Urban Challenge, автомобильных гонок на улицах города, где одни роботы соревновались с другими и должны были соблюдать правила дорожного движения.



а) б)

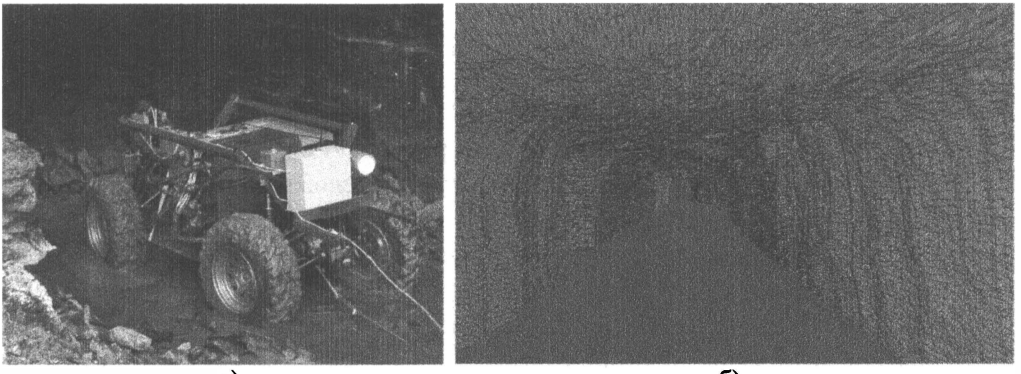
Рис. 26.35. а) Беспилотный автомобиль BOSS, выигравший соревнования DARPA Urban Challenge. **б)** Вид с высоты, демонстрирующий восприятие и прогнозы беспилотного автомобиля Waymo (белый автомобиль на светлой полосе на дороге). Другие транспортные средства (темно-серые прямоугольники ящики) и пешеходы (белые прямоугольники) показаны с их ожидаемыми траекториями. Границы между дорогой и тротуарами выделены светлыми тонкими линиями

В 2009 году корпорация Google запустила проект автономного вождения (с участием многих исследователей, ранее работавших над беспилотными автомобилями Стэнли и BOSS), который теперь называется "Waymo". В 2018 году в рамках проекта Waymo было начато тестирование беспилотного автомобиля без водителя (никто не сидел на месте водителя) в пригороде города Феникс, штат Аризона. В то же время другие компании, занимающиеся автономным вождением, и компании, занимающиеся совместным (человек/робот) управлением автомобилем, продолжают работать над созданием собственных технологий, тогда как компании, выпускающие автомобили, поставляют на рынок машины со все увеличивающимся количеством вспомогательных средств, таких как система помощи водителю компании Тесла ▶**driver assist** (*ассистент водителя*), предназначенная для автоматического вождения на шоссе. Другие компании нацелены на внешоссейные

приложения, включая студенческие городки и пенсионные сообщества, а некоторые — на не пассажирские приложения, такие как грузоперевозки, доставка продуктов и услуги парковщика.

Развлечения. Компания Disney использует роботов (под названием ►**animatronics**) в своих парках развлечений, начиная с 1963 года. Первоначально эти роботы представляли собой разработанные вручную устройства с открытым контуром и неизменяемыми движениями (и речью), но с 2009 года версия под названием ►**autonomatronics** уже способна генерировать автономные действия. Роботы также часто принимают форму интеллектуальных игрушек для детей, например робот компании Anki Cozmo играет с детьми в настольные игры и может стучать по столу, когда проигрывает. Наконец, такой квадрокоптер, как R1 компании Skydio, показанный на рис. 26.2, *б*, способен выступать в роли личного фотографа или видеооператора, повсюду следующего за людьми, чтобы делать снимки, когда они катаются на лыжах или на велосипеде.

Разведка и работа в опасной среде. Роботы добрались до тех мест, где еще никто не бывал, включая поверхность Марса. С помощью роботов-манипуляторов космонавты выводят на орбиту и снимают с орбиты спутники, а также строят Международную космическую станцию. Кроме того, роботы помогают проводить подводные исследования, например, для создания карт расположения затонувших кораблей. На рис. 26.36, *а* показан робот, составляющий карту заброшенной угольной шахты, а на рис. 26.36, *б* приведена трехмерная карта, созданная этим роботом с помощью датчиков расстояния. В 1996 году группа исследователей выпустила в кратер действующего вулкана шагающего робота с целью получения данных, необходимых для климатических исследований. Роботы становятся очень эффективными инструментами сбора информации в тех местах, доступ к которым сложен (или опасен) для людей.



а) б)

Рис. 26.36. а) Робот, составляющий карту заброшенной угольной шахты. **б)** Трехмерная карта шахты, созданная этим роботом

Роботы помогали людям убирать ядерные отходы при устранении последствий аварий в Чернобыле, на острове Тримайл Айленд и в Фукусиме. Роботы применялись и для расчистки завалов Всемирного торгового центра в Нью-Йорке, где их направляли на те участки, которые считались слишком опасными для поисковых и спасательных команд. Ранее в подобных случаях роботы развертывались с использованием удаленного управления, но по мере развития технологий они становятся все более и более автономными, и оператору-человеку уже не требуется выдавать им каждую отдельную команду.

Промышленность. Наибольшее количество роботов сегодня развернуто на заводах, где они автоматически выполняют работы, которые трудны, опасны или скучны для людей. (Большинство заводских роботов используется в автомобильной промышленности.) Автоматизация таких работ является положительным фактором с точки зрения эффективности производства, необходимого для общества. В то же время это означает вытеснение части работников-людей с производства. Этот факт имеет важные политические и экономические последствия — необходимость переподготовки и переобучения, необходимость справедливого разделения ресурсов и т.д. Все эти аспекты более подробно обсуждаются в разделе 27.3.5.

Резюме

Робототехника — это область ИИ, занимающаяся физическим воплощением интеллектуальных агентов, способных изменять состояние физического мира. В этой главе обсуждались следующие аспекты этого направления.

- Наиболее распространенными типами роботов являются **манипуляторы** (роботизированные руки) и **мобильные роботы**. Они снабжены **датчиками** (*sensor*) для восприятия мира и силовыми **приводами** (*actuator*), производящими движение, которое затем оказывает влияние на мир через **исполнительные механизмы** (*effector*).
- Общая задача робототехники предполагает наличие **стохастичности** (которая может быть обработана с помощью MDP), **частичной наблюдаемости** (которая может быть обработана с помощью POMDP) и действий совместно и в окружении **других агентов** (которые могут быть обработаны методами теории игр). Данная задача становится еще сложнее по той причине, что большинство роботов функционируют в непрерывных и многомерных пространствах состояний и действий. Они также работают в реальном мире, который отказывается действовать быстрее, чем в реальном времени, и в котором отказы и сбои приводят к повреждению реальных вещей без возможности "отмены" произошедшего.
- В идеале робот должен решать всю стоящую перед ним задачу как одно целое: наблюдения в виде необработанных данных поступают от датчиков и действия в виде крутящих моментов или токов соответствующим образом

подаются на двигатели. Однако реализовать это на практике слишком сложно, и робототехники, как правило, выделяют различные аспекты решаемой задачи и рассматривают их независимо от других.

- Обычно восприятие (оценка) отделяется от действия (генерация движения). Восприятие в робототехнике включает компьютерное зрение, используемое для распознавания окружения с помощью камер, а также локализации и картирования.
- Восприятие в робототехнике связано с оценкой значимых для решения задачи величин на основании данных от датчиков. Для этого необходимы некоторое внутреннее представление и метод обновления этого внутреннего представления во времени.
- При реализации восприятия робота используются **алгоритмы вероятностной фильтрации**, такие как фильтры частиц и фильтры Калмана. Эти методы обеспечивают поддержку доверительного состояния, т.е. распределение апостериорных вероятностей по переменным состояния.
- Планирование движений робота обычно осуществляется в **пространстве конфигураций**, каждая точка которого определяет все, что необходимо знать для локализации каждой **точки тела** робота. Например, в случае манипулятора с двумя шарнирными соединениями конфигурация состоит из двух углов поворота его шарниров.
- Обычно задача генерации движения разделяется на **планирование движения**, касающееся разработки плана, и **управление отслеживанием траектории**, обеспечивающее выработку стратегии выдачи управляющих сигналов (команд привода), в конечном счете приводящих к выполнению плана.
- Планирование движения можно реализовать посредством поиска по графу с использованием метода **декомпозиции ячеек**, **алгоритмов рандомизированного планирования движения**, предполагающих выборку контрольных точек в непрерывном пространстве конфигурации, или методов **оптимизации траектории**, позволяющих итеративным образом "оттолкнуть" путь, проходящий по прямой линии, от столкновений с препятствием за счет использования **поля расстояния со знаком**.
- Задачу прохождения по пути, найденному с помощью алгоритма поиска, можно решить, либо используя этот путь в качестве опорной траектории для **PID-контроллера**, который постоянно исправляет погрешность движения (расхождение между тем, где робот находится сейчас, и где он предположительно должен быть), либо с помощью **расчетного управления крутящим моментом**, при котором в закон движения добавляется терм **прямой связи**, использующий **обратную динамику** для приблизительного расчета, какой крутящий момент необходим, чтобы добиться прогресса в продвижении по траектории.

- **Оптимальное управление** объединяет планирование движения и отслеживание траектории посредством вычисления оптимальной траектории непосредственно через управляющие сигналы. Это выполняется особенно просто, когда задача характеризуется квадратичной стоимостью и линейной динамикой, — в результате появляется возможность использовать линейный квадратичный регулятор (**LQR**). В популярных методах этот факт используется за счет линеаризации динамики и вычисления аппроксимирующей функции второго порядка для стоимости (**ILQR**).
- Планирование в условиях неопределенности объединяет восприятие и действия посредством **динамической перепланировки** (например, управление с прогнозирующими моделями) и действий по **сбору информации**, которая поможет восприятию.
- В робототехнике машинное обучение с подкреплением применяется при использовании методов, позволяющих уменьшить необходимое количество взаимодействий с реальным миром. В таких методах принято использовать модели, будь то оценка моделей и их использование для планирования, либо стратегия обучения, устойчивая по отношению к различным возможным параметрам модели.
- Взаимодействие с людьми требует умения **координировать** действия робота с действиями человека, что можно сформулировать как игру. Обычно процедура поиска решения разделяется на этап **прогнозирования**, на котором текущие действия человека используются для оценки, что он собирается делать в будущем, и этап **действия**, на котором полученные прогнозы используются для расчета оптимального движения робота.
- Оказание помощи людям также требует способности изучать или делать выводы о том, что они хотят. Роботы могут реализовать это требование, изучая желаемую функцию стоимости, которую они будут оптимизировать, на основании входной информации, предоставляемой человеком. Это могут быть демонстрации, исправления или инструкции на естественном языке. В качестве альтернативы роботы могут просто имитировать поведение человека и использовать обучение с подкреплением, чтобы лучше справляться с проблемой обобщения новых состояний.

Библиографические и исторические заметки

Слово "**робот**" получило широкую известность после выхода в 1920 году пьесы R.U.R. (Rossum's Universal Robots — *универсальные роботы Россума*) чешского драматурга Карела Чапека. В этой пьесе роботы, которые выращивались из химических растворов, а не собирались механически, в конечном итоге взбунтовались против своих создателей и решили взять над ними верх. Считается, что это слово придумал брат Карела Чапека, Йозеф, который скомбинировал чешские слова

robota (принудительный труд) и *robotnik* (раб) и получил слово "robot", — впервые он применил его в своем рассказе "Opilec", изданном в 1917 году (Гланк [867], 1978). Термин "робототехника" (*robotics*) был впервые использован Айзеком Азимовым в его сборнике научно-фантастических рассказов "Я, робот" ([86], 1950).

Идея автономной машины предшествует появлению слова "робот" на тысячи лет. В древнегреческой мифологии упоминается механический человек по имени Талос, построенный греческим богом огня и кузнечного дела Гефестом для защиты острова Крит. Легенда гласит, что колдунья Медея победила Талоса, пообещав ему бессмертие, а затем иссушив его жизненный флюид. Таким образом, это первый пример робота, совершившего ошибку в процессе изменения своей целевой функции. В 322 году до н.э. Аристотель уже предвидел технологическую безработицу, выдвинув следующее предположение: "Если каждый инструмент, когда заказано или даже по собственному желанию, сможет выполнить ту работу, которая ему подобает... то уже не потребуется ни учеников для мастеров, ни рабов для лордов".

В III веке до н.э. настоящий человекоподобный робот под названием "Слуга Филона" мог наливать вино или воду в чашу: серия клапанов перекрывала поток в нужное время. Чудесные автоматы были построены в XVIII веке (механическая утка Жака Вокансона, созданная им в 1738 году, является одним из первых примеров таких автоматов), но сложное поведение, которое они демонстрировали, было полностью определено и зафиксировано заранее. Возможно, самым ранним примером *программируемого* роботоподобного устройства был ткацкий станок Жозефа Жаккарда (1805), упоминавшийся в разделе 1.2.6.

"Черепаха" Грея Уолтера, построенная в 1948 году, может считаться первым автономным мобильным роботом, хотя ее система управления не была программируемой. "Зверь Хопкинса", построенный в 1960 году в Университете Джона Хопкинса, был уже гораздо более сложным устройством: он имел сонар, датчики на фотоэлементах и электронные схемы для распознавания образов, — благодаря им, он мог распознать крышку стандартной розетки переменного тока. При необходимости он был способен отыскать розетку и подключиться к ней, чтобы зарядить свою аккумуляторную батарею! Тем не менее у этого робота все еще был лишь ограниченный набор навыков.

Первым мобильным роботом общего назначения был "Shakey", разработанный в конце 1960-х годов в Стэнфордском научно-исследовательском институте (теперь он носит название "SRI") (Фикес и Нильсон [738], 1971; Нильсон [1688], 1984). Shakey был первым роботом, в котором объединялись функции восприятия, планирования и выполнения, и это замечательное достижение оказало влияние на многие дальнейшие исследования в области искусственного интеллекта. Робот Shakey представлен на обложке этой книги вместе с руководителем проекта Чарли Розеном (1917–2002). К другим важным проектам относятся Stanford Cart и CMU Rover (Моравек [1620], 1983). Кокс и Уилфонг в работе [488] (1990) описывают классическое исследование по автономным транспортным средствам.

Первым коммерчески поставляемым роботом был робот-манипулятор, получивший название UNIMATE (сокращение от *universal automation* — универсальная автоматизация). Он был разработан Джозефом Энджелбергером и Джорджем Деволом в их компании, Unimation. В 1961 году первый робот UNIMATE был продан компании General Motors, где использовался при изготовлении телевизионных трубок. Тот же 1961 год примечателен и тем, что в этом году Девол получил первый патент США на конструкцию робота.

В 1973 году компании Toyota и Nissan начали использовать обновленную версию роботов UNIMATE для выполнения точечной сварки кузовов автомобилей. Это положило начало большой революции в автомобильной промышленности, которая происходила в основном в Японии и США и продолжается до сих пор. Компания Unimation продолжала свои исследования и в 1978 году выпустила робота PUMA (*Programmable Universal Machine for Assembly* — программируемый универсальный станок для сборки), который *de facto* стал стандартом для роботизированных манипуляторов в течение двух последующих десятилетий. Каждый год продается около 500 тыс. роботов, половина из которых поступает в автомобильную промышленность.

Первой важной работой по созданию машины "рука–глаз" (как первоначально именовались роботы-манипуляторы) был проект Генриха Эрнста под названием "MH-1", описанный в тезисах его докторской диссертации ([692], 1961). Проект Machine Intelligence, реализованный в Эдинбурге, также стал одной из первых демонстраций впечатляющей сборочной системы, основанной на использовании средств машинного зрения, которая получила название FREDDY (Мичи [1564], 1972).

Исследования в области мобильной робототехники стимулировались несколькими важными конкурсами. Ежегодный конкурс мобильных роботов AAAI проводится с 1992 года. Первым его победителем стал проект CARMEL (Кондоне и др. [468], 1992). Прогресс был устойчивым и впечатляющим: на недавних соревнованиях роботы вошли в конференц-комплекс, нашли дорогу к стойке регистрации, зарегистрировались на конференции и даже выступали с краткими сообщениями.

Соревнования **RoboCup**, которые проводятся с 1995 года Китано и его коллегами ([1235], 1997), имеют целью "создание команды полностью автономных человекоподобных роботов, способных победить команду, ставшую чемпионом мира по футболу среди людей", к 2050 году. В одной части соревнований используются роботы на колесах, в другой — человекоподобные роботы, а в третьей — программные симуляции. Стоун в статье [2137] (2016) описывает последние нововведения в соревнованиях RoboCup.

В соревнованиях **DARPA Grand Challenge**, организованных управлением DARPA МО США в 2004 и 2005 годах, беспилотным машинам требовалось пройти более 200 км по пустыне менее чем за десять часов (Бюхлер и др. [341], 2006). На первых гонках 2004 года ни один из роботов не прошел более восьми миль, что заставило многих поверить, что приз никогда не будет востребован. В 2005 году робот Стэнли Стэнфордского университета выиграл гонки всего за семь часов

(Трун [2212], 2006). Тогда DARPA организовало конкурс **Urban Challenge** — соревнование, в котором роботы должны были преодолеть 60 миль в городской среде параллельно с движением других транспортных средств. Робот университета Карнеги-Меллона Boss занял в этих соревнованиях первое место и выиграл приз в 2 млн долл. (Урмсон и Уиттакер [2243], 2008). Первопроходцами в разработке роботизированных автомобилей являются Дикманнс и Запп ([618], 1987), а также Померло ([1806], 1993).

Развитие области роботизированного картографирования началось из двух разных источников. Первое направление прослеживается с работы Смита и Чизмена ([2099], 1986), которые применили фильтры Калмана для одновременного решения задач локализации и составления карты (SLAM). Разработанный ими алгоритм впервые был реализован в исследовании, описанном Мутарли и Шатила в статье [1630] (1989), а затем дополнен Леонардом и Даррантом-Уайтом ([1386], 1992). В работе Диссаньяка и соавт. [624] (2001) приведен обзор ранних вариаций фильтра Калмана. Второе направление началось с разработки представления в виде ▶**сетки занятости** для вероятностной картографии, в которой определяется вероятность того, что каждый участок (x, y) занят некоторым препятствием (Моравец и Элфес [1621], 1985).

Куйперс и Леви были одними из первых, когда в работе [1322] (1988) предложили топологическую, а не метрическую картографию, стимулом для чего послужили модели пространственного познания человека. В плодотворной статье Лу и Милиоса [1457] (1997) признается, что задачам одновременной локализации и составления карты свойственна разреженность, что привело к разработке методов нелинейной оптимизации, предложенных Конолиге в работе [1278] (2004) и Монтемерло и Труном в работе [1609] (2004), а также иерархических методов Боссе и соавт. ([258], 2004). Шаткей и Келблинг в работе [2044] (1997) и Трун с соавт. в работе [2211] (1998) представили алгоритм EM в поле роботизированного построения карты для ассоциации данных. Обзор вероятностных методов отображения можно найти в работе Труна и соавт. [2210] (2005).

Обзор самых ранних методов локализации мобильных роботов приведен Броенштейном и соавт. в работе [253] (1996). Хотя в теории управления фильтры Калмана применяются в качестве метода локализации уже в течение многих десятилетий, в литературе по искусственному интеллекту общая вероятностная формулировка задачи локализации появилась гораздо позже благодаря работам Тома Дина и его коллег (Дин и др. [566], 1990), а также Симонса и Кёнига ([2068], 1995). В последней работе был предложен термин ▶**марковская локализация**. Первым практическим приложением этого метода явилась работа Бургарда и соавт. ([345], 1999), создавших целый ряд роботов, предназначенных для использования в музеях. Метод локализации Монте-Карло, основанный на фильтрах частиц, был разработан Фоксом и соавт. ([766], 1999) и в настоящее время имеет широкое распространение. В ▶**фильтре частиц Рао–Блэквелла** объединяется применение фильтрации частиц для локализации робота с применением

точной фильтрации для составления карты (Мерфи и Расселл [1640], 2001; Монтемерло и др. [1610], 2002).

Многие ранние исследовательские работы по **планированию движения** сосредоточивалось на создании геометрических алгоритмов решения задач планирования движения в детерминированной и полностью наблюдаемой среде. В оригинальной работе Рейфа ([1868], 1979) было показано, что задача планирования движения робота является PSPACE-трудной. Представление на основе пространства конфигураций было предложено Лозано-Пересом в работе [1455] (1983). Важное влияние оказал ряд статей Шварца и Шарира, посвященных задаче, которую они назвали ►**задачей переноса рояля** (Шварц и др. [2012], 1987).

Метод рекурсивной декомпозиции ячеек для планирования в пространстве конфигураций был впервые предложен в работе Брукса и Лозано-Переса [315] (1985) и значительно улучшен Чжу и Латомбом в работе [2441] (1991). Первые алгоритмы скелетирования были основаны на диаграммах Вороного (Рова [1922], 1979) и ►**графах видимости** (Уэсли и Лозано-Перес [2325], 1979). Кюба и соавт. в работе [933] (1992) предложили разработанный ими эффективный метод инкрементного вычисления диаграмм Вороного, а Шозе в работе [424] (1996) обобщил диаграммы Вороного на гораздо более широкий класс задач планирования движения.

Джон Кэнни в работе [365] (1988) предложил первый полностью экспоненциальный алгоритм планирования движения. В популярном учебнике Латомба [1358] (1991) рассматривается целый ряд подходов к решению задач планирования движения, как и в учебниках Шозе и соавт. ([425], 2005) и ЛаВалле ([1318], 2000). Кавраки и соавт. разработали теорию вероятностных дорожных карт ([1203], 1996). Кюффнер и ЛаВалле в работе [1318] (2000) описывают разработанный ими метод быстро исследуемых случайных деревьев (RRT).

Включение методов оптимизации в геометрическое планирование движения началось с метода эластичных полос (Квинлан и Хатиб [1837], 1993), в котором реализуется уточнение пути при изменении препятствий в пространстве конфигурации. Ратлифф и соавт. в работе [1859] (2009) сформулировали идею оптимизации как решение задачи оптимального управления, допуская наличие столкновений в исходной траектории с последующей ее трансформацией посредством отображения через якобиан градиентов препятствия в рабочей области в пространство конфигураций. Шульман и соавт. в работе [2004] (2013) предложили практичную альтернативу второго порядка.

Управлению роботами как динамическими системами (для решения задач манипулирования или навигации) посвящен огромный объем литературы. В данной главе обсуждались лишь основы **управления отслеживанием траектории** и **оптимальное управление**, а целые подобласти, включая адаптивное управление, надежное управление и анализ Ляпунова, пока были оставлены без внимания. Вместо принятия предположения, что о системе все уже известно априори, адаптивное управление нацелено на адаптацию ее динамических параметров и/или закона управления в непрерывном режиме. Надежное управление, со своей

стороны, ставит своей целью разработку контроллеров, которые будут хорошо работать, несмотря на неопределенность и внешние помехи.

Анализ Ляпунова первоначально был разработан в 1890-х годах для анализа устойчивости общих нелинейных систем, и только в начале 1930-х годов при разработке теории управления был понят его истинный потенциал. С развитием методов оптимизации анализ Ляпунова был распространен на управление барьерными функциями, которые прекрасно подходят для современных инструментов оптимизации. Эти методы широко используются в современной робототехнике при проектировании контроллеров реального времени и анализе безопасности.

К числу важнейших работ в области управления роботами относятся трехтомник по импедансному управлению Хогана ([1048], 1985) и общее исследование по динамике роботов Физерстоуна ([714], 1987). Дин и Уэллман ([572], 1991) были в числе первых, кто попытался связать воедино теорию управления и системы планирования ИИ. Три классических учебника по математическим основам робототехнического манипулирования принадлежат Полу ([1745], 1981), Крейгу ([490], 1989) и Йошикава ([2408], 1990). Управление манипуляциями рассматривается также Мюрреем в работе [1644] (2017).

Большое значение в робототехнике имеет и область изучения процессов **схватывания**, поскольку задача определения стабильного захвата является весьма сложной (Мейсон и Солсбери [1513], 1985). Для успешного схватывания требуется восприятие данных о прикосновении, или ▶ **тактильная обратная связь**, необходимая для определения усилия контакта и обнаружения проскальзывания (Фиринг и Холлербах [713], 1985). Понимание того, как можно схватить самые разнообразные объекты в мире, является трудной задачей. Бусмалис и соавт. в работе [264] (2017) описывают систему, для получения надежного захвата сочетающую эксперименты в реальном мире с симуляциями, управляемыми передачей "симуляция–реальность".

Метод управления потенциальным полем, в котором предпринимается попытка решить задачи планирования движения и управления одновременно, в области робототехники был разработан Хатибом ([1222], 1986). В мобильной робототехнике эта идея рассматривалась как практическое решение задачи предотвращения столкновений и впоследствии была расширена Боренштейном ([254], 1991) в алгоритм, получивший название ▶ **гистограмма векторного поля**.

ILQR-регуляторы в настоящее время широко используются на пересечении планирования движения и управления благодаря Ли и Тодорову ([1405], 2004). Это вариант гораздо более старого метода дифференциального динамического программирования (Якобсон и Майн [1120], 1970).

Планирование точных движений с ограниченным зондированием исследовалось Лозано-Пересом и соавт. ([1456], 1984), а также Кенни и Рейфом ([363], 1987). В навигации на основе контрольных точек (Лазанас и Латомб [1367], 1992) многие из тех же идей используются в применении к мобильным роботам. Функции навигации и робототехническая версия стратегии управления для детерминированных MDP были введены Кодицчеком в работе [1252] (1987). Важнейшая

часть работы по применению методов POMDP (раздел 17.4) к планированию движения в условиях неопределенности в робототехнике была выполнена Пино и соавт. ([1790], 2003) и Роем и соавт. ([1925], 2005).

Применение **обучения с подкреплением** в робототехнике началось с оригинальных работ Багнелла и Шнейдера ([108], 2001) и Энга и соавт. ([1677], 2003), разработавших эту парадигму в контексте управления беспилотным вертолетом. Кобер и соавт. в работе [1248] (2013) предлагают обзор того, как обучение с подкреплением может быть изменено для применения при решении задач робототехники. Многие из реализованных на физических системах методов позволяют строить приближенные модели динамики, начиная от локально взвешенных линейных моделей, благодаря исследованиям Аткесона и соавт. ([88], 1997). Однако градиенты стратегии также сыграли свою роль, позволив (упрощенно) гуманоидным роботам ходить (Тедраке и др. [2188], 2004), а руке робота выполнять бейсбольные удары (Петерс и Шааль [1778], 2008).

Левин и соавт. в работе [1394] (2016) продемонстрировали первое применение **глубокого обучения с подкреплением** к реальному роботу. В то же время безмодельное обучение с подкреплением в условиях симуляции было распространено на непрерывные предметные области (Шульман и др. [2005], 2015; Хисс и др. [1002], 2016; Лилликрап и др. [1413], 2015). В других работах сообщалось о значительном масштабировании набора физических данных для демонстрации обучения захватам и о динамике моделей. (Пинто и Гупта [1796], 2016; Агравал и др. [23], 2017; Левин и др. [1395], 2018). В настоящее время передача результатов из симуляции в реальность, или передача **симуляция–реальность** (Садехи и Левин [1957], 2016; Андричевич и др. [56], 2018), **метаобучение** (Финн и др. [741], 2017) и эффективное безмодельное обучение с подкреплением на примерах (Андричевич и др. [57], 2018) являются активными областями исследований.

В ранних методах прогнозирования **действий человека** использовались подходы с фильтрацией (Мадхаван и Шленов [1476], 2003), но в плодотворной работе Зибарта и соавт. [2445] (2009) было предложено прогнозирование посредством моделирования поведения людей как приблизительно рациональных агентов. Садиг и соавт. в работе [1958] (2016) сформулировали, как эти предсказания должны в действительности зависеть от того, что робот решит сделать, при построении теоретико-игрового окружения. Для обстановки сотрудничества Сисбот и соавт. в работе [2077] (2007) впервые предложили идею учета в функции стоимости робота того, чего хотят люди. Николаидис и Шах ([1683], 2013) разложили сотрудничество на изучение того, как будет действовать человек и как человек хочет, чтобы действовал робот, — в обоих случаях на основе демонстраций. Обучение на демонстрациях анализировалось Аргаллом и соавт. в работе [68] (2009). Акгун и соавт. ([27], 2012) и Сефидгар и соавт. ([2019], 2017) изучали вариант проведения обучения конечными пользователями, а не экспертами.

Теллекс и соавт. в работе [2189] (2011) показали, как роботы могут логически вывести, чего хотят люди, из инструкций на естественном языке. Наконец,

не только роботам необходимо делать выводы о том, чего люди хотят и что планируют делать, но и люди должны делать такие же выводы в отношении роботов. Драган и соавт. в работе [648] (2013) включили модель умозаключений человека в процедуру планирования движения робота.

Тема **взаимодействия человека и робота** намного шире того, что рассматривалось в этой главе, где основное внимание было уделено аспектам планирования и обучения. Томаз и соавт. в работе [2204] (2016) представили более широкий обзор темы взаимодействия с вычислительной точки зрения. Росс и соавт. в работе [1919] (2011) описывают систему DAGGER.

Тема программных архитектур для роботов вызывает много концептуальных дебатов. Старый добрый претендент в области ИИ — трехслойная архитектура — восходит еще к конструкции робота Shakey и подробно рассматривается Гатом в работе [818] (1998). Архитектура погружения принадлежит Бруксу ([312], 1986), хотя аналогичные идеи были независимо разработаны Брайтенбергом, который в своей книге *Vehicles* ([289], 1984) описывает серию простых роботов, построенных на основе поведенческого подхода.

Успех шестиногого шагающего робота Брукса вызвал появление множества других проектов. Коннелл в своей докторской диссертации ([470], 1989) представил разработанного им полностью реактивного мобильного робота, который был способен находить и подавать объекты. Расширение этой парадигмы на системы с несколькими роботами можно найти в работах Паркера ([1735], 1996) и Матарика ([1514], 1997). В языках GRL (Хорсвилл [1064], 2000) и COLBERT (Конолиге [1277], 1997) идеи совместной поведенческой робототехники в общем управлении роботами были абстрагированы до уровня языков программирования. Аркин в работе [70] (1998) рассматривает некоторые из самых популярных подходов в этой области.

В двух ранних учебниках, Дудека и Дженкина ([661], 2000) и Мерфи ([1642], 2000), робототехника рассматривается в целом. Более свежие обзоры принадлежат Беки ([156], 2008) и Линху и Парку ([1470], 2017). Превосходная книга о манипуляции робота посвящена таким сложным темам, как согласующее движение (Мейсон [1512], 2001). Планирование движения робота описано Чосетом и соавт. в работе [425] (2005) и ЛаВалле в работе [1363] (2006). Трун и соавт. в работе [2210] (2005) представили вероятностную робототехнику. Книга *Handbook of Robotics* (Справочник по робототехнике) Сицилиано и Хатиба ([2057], 2016) представляет собой обширный всесторонний обзор всей области робототехники.

Основной конференцией по робототехнике является *Robotics: Science and Systems*, за которой следует конференция IEEE *International Conference on Robotics and Automation*. Конференция *Human–Robot Interaction* является основной площадкой для представления новых результатов в области взаимодействия робота и человека. Ведущими робототехническими журналами являются *IEEE Robotics and Automation*, *International Journal of Robotics Research* и *Robotics and Autonomous Systems*.

Упражнения

26.1. При любом конечном размере выборки результаты применения алгоритма локализации Монте-Карло являются *смещенными* (т.е. ожидаемое значение данных о местонахождении, вычисленное с помощью алгоритма, отличается от истинного ожидаемого значения), поскольку именно так действует алгоритм фильтрации частиц. В данном упражнении предлагается оценить это смещение.

Для упрощения рассмотрим мир с четырьмя возможными местонахождениями робота: $X = \{x_1, x_2, x_3, x_4\}$. Вначале выполним равномерную выборку $N \geq 1$ образцов среди этих местонахождений. Как обычно, вполне приемлемо, если для любого из местонахождений X будет сформировано больше одной выборки. Допустим, что Z — это булева переменная восприятия, характеризующаяся следующими условными вероятностями.

$$P(z \mid x_1) = 0{,}8 \qquad P(z \mid x_1) = 0{,}2$$
$$P(z \mid x_2) = 0{,}4 \qquad P(z \mid x_2) = 0{,}6$$
$$P(z \mid x_3) = 0{,}1 \qquad P(z \mid x_3) = 0{,}9$$
$$P(z \mid x_4) = 0{,}1 \qquad P(z \mid x_4) = 0{,}9$$

В алгоритме MCL эти вероятности используются для формирования весов частиц, которые в дальнейшем нормализуются и используются в процессе повторной выборки. Для упрощения предположим, что при повторной выборке вырабатывается только один новый образец, независимо от N. Этот образец может соответствовать любому из четырех местонахождений X. Следовательно, процесс выборки определяет распределение вероятностей по X.

а) Каково результирующее распределение вероятностей по X для этого нового образца? Ответьте на этот вопрос отдельно для $N = 1, \ldots, 10$ и для $N = \infty$.

б) Разница между двумя распределениями вероятностей P и Q может быть измерена с помощью дивергенции KL, которая определяется следующим образом:

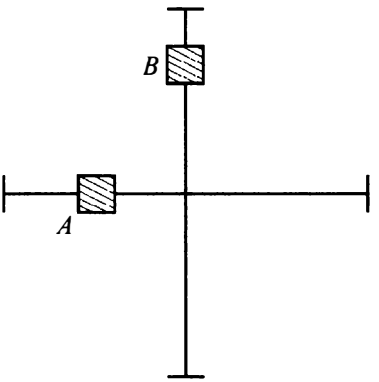
$$KL(P, Q) = \sum_i P(x_i) \log \frac{P(x_i)}{Q(x_i)}.$$

Каковы значения дивергенции KL между распределениями в п. *а* и истинным распределением апостериорных вероятностей?

в) Какая модификация формулировки задачи (а не алгоритма!) гарантировала бы, что конкретное приведенное выше выражение для оценки останется несмещенным даже при конечных значениях N? Предложите по меньшей мере две такие модификации (но каждая из них должна соответствовать предложенному заданию).

26.2. Реализуйте алгоритм локализации Монте-Карло для моделируемого робота с датчиками расстояний. Карту, нанесенную на сетку, и данные о расстояниях можно найти в репозитории кода по адресу `aima.cs.berkeley.edu`. Упражнение будет считаться выполненным, если вы продемонстрируете успешную глобальную локализацию робота.

26.3. Рассмотрим робота с двумя простыми манипуляторами, представленного на рис. 26.37. Манипулятор A представляет собой квадратный блок со стороной 2, который может скользить туда и обратно по стержню, проходящему вдоль оси x, в пределах от $x = -10$ до x = 10. Манипулятор B представляет собой такой же квадратный блок со стороной 2, который может скользить вверх и вниз по стержню, проходящему вдоль оси y от $y = -10$ до $y = 10$. Стержни лежат вне плоскости манипуляции и поэтому не мешают движению блоков. Конфигурация представляет собой пару значений $\langle x, y \rangle$, где x — это координата x центра манипулятора A, а y — это координата y центра манипулятора B. Нарисуйте пространство конфигурации для этого робота с указанием разрешенных и запрещенных зон.



Конфигурация <–5, 7>

Рис. 26.37. Простой робот с двумя манипуляторами

26.4. Предположим, что рассматривается робот, представленный в упражнении 26.3 (см. рис. 26.37), и требуется найти путь от начальной конфигурации до конечной конфигурации *Goal*, как показано на рис. 26.38. Рассмотрим функцию потенциала

$$D(A, Goal)^2 + D(B, Goal)^2 + \frac{1}{D(A,B)^2},$$

где $D(A, B)$ — расстояние между ближайшими точками манипуляторов A и B.

а) Покажите, что на этом потенциальном поле поиск по методу восхождения к вершине застрянет в локальном минимуме.

б) Опишите такое потенциальное поле, в котором использование алгоритма восхождения к вершине позволит решить эту конкретную задачу. Не нужно определять необходимые для этого точные числовые коэффициенты, — просто укажите общую форму решения. (*Подсказка.* Добавьте терм, который "вознаграждает" процесс восхождения к вершине за то, что он переместил манипулятор A с пути манипулятора B, даже в том случае, когда это не уменьшает расстояние от A до B в указанном выше смысле.)

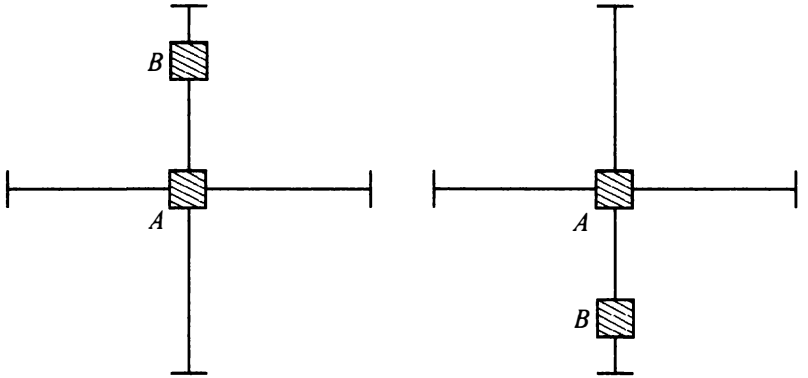


Рис. 26.38. Робот-манипулятор в двух возможных конфигурациях

26.5. Рассмотрим роботизированный манипулятор, представленный на рис. 26.12. Предположим, что элемент в основании робота имеет длину 60 см, а плечо и предплечье имеют длину по 40 см. Как было указано выше, обратная кинематика робота часто является не уникальной. Сформулируйте явное решение в замкнутой форме для обратной кинематики этого манипулятора. При каких именно условиях это решение является уникальным?

26.6. Рассмотрим роботизированный манипулятор, представленный на рис. 26.12. Предположим, что элемент в основании робота имеет длину 70 см, а плечо и предплечье имеют длину по 50 см. Как было указано выше, обратная кинематика робота часто является не уникальной. Сформулируйте явное решение в замкнутой форме для обратной кинематики этого манипулятора. При каких именно условиях это решение является уникальным?

26.7. Реализуйте алгоритм вычисления диаграммы Вороного для произвольной двухмерной среды, описанной с помощью булева массива размерами $n \times n$. Проиллюстрируйте работу своего алгоритма, построив график диаграммы Вороного для 10 интересных карт. Какова сложность вашего алгоритма?

26.8. В этом упражнении рассматривается связь между рабочим пространством и пространством конфигураций с использованием примеров, показанных на рис. 26.39.

а) Рассмотрим конфигурации роботов, приведенных на рис. 26.39, *а–в*, игнорируя препятствие, приведенное на каждом из этих рисунков. Нарисуйте соответствующие конфигурации манипулятора в пространстве конфигураций. (*Подсказка.* Каждая конфигурация манипулятора отображается на единственную точку в пространстве конфигураций, как показано на рис. 26.12, *б*.)

б) Нарисуйте пространство конфигураций для каждой из диаграмм рабочего пространства, приведенных на рис. 26.39, *а–в*. (*Подсказка.* В этих пространствах конфигураций общим с пространством конфигураций, показанным

на рис. 26.39, *а*, является тот участок, который соответствует столкновению манипулятора робота с самим собой, а различия обусловлены отсутствием окружающих препятствий и изменением местонахождений препятствий на этих отдельных рисунках.)

в) Для каждой из черных точек, приведенных на рис. 26.39, *д*, *е*, нарисуйте соответствующие конфигурации манипулятора робота в рабочем пространстве. В этом упражнении затененные участки следует игнорировать.

г) Оба пространства конфигураций, показанные на рис. 26.39, *д*, *е*, сформированы с учетом единственного препятствия в рабочем пространстве (темно-серая заливка), а также ограничений, обусловленных ограничением, препятствующим столкновению манипулятора с самим собой (светло-серая заливка). Для каждой диаграммы нарисуйте в рабочем пространстве препятствие, которое соответствует участку с темной заливкой.

д) На рис. 26.39, *г*, показано, что единственное плоское препятствие способно разбить рабочее пространство на два несвязанных между собой участка. Каково максимальное количество несвязанных участков, которые могут быть созданы в результате вставки плоского препятствия в свободное от препятствий связное рабочее пространство для робота с двумя степенями свободы? Приведите пример и объясните, почему не может быть создано большее количество несвязных участков. Относится ли это утверждение к неплоскому препятствию?

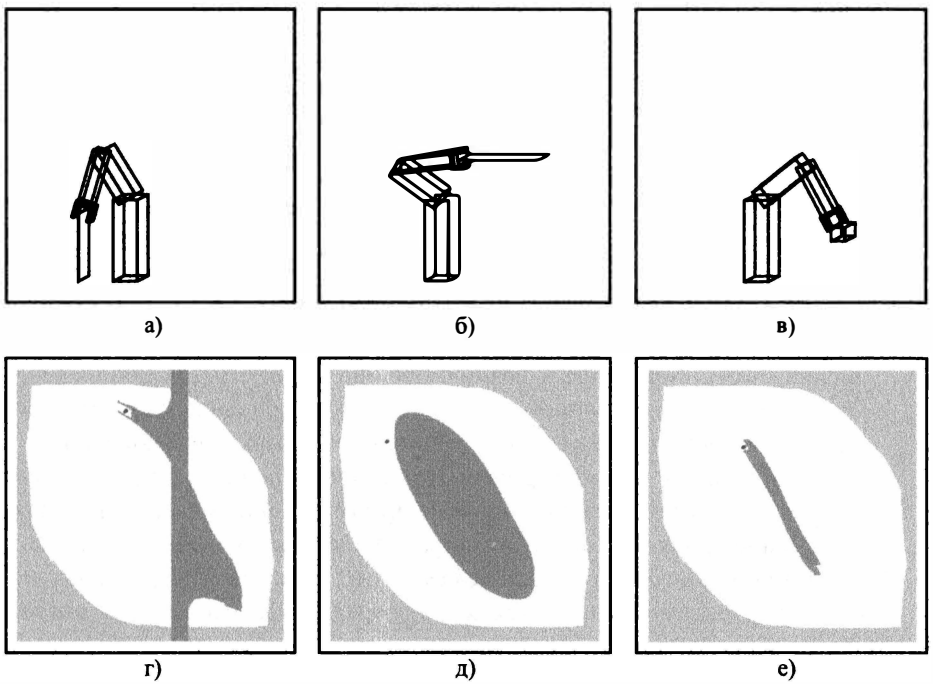


Рис. 26.39. Диаграммы для упражнения 26.8

26.9. Рассмотрим мобильного робота, перемещающегося по горизонтальной поверхности. Предположим, что этот робот может выполнять два вида движений:
– перемещение вперед на указанное расстояние;
– поворот на месте на заданный угол.

Состояние такого робота можно охарактеризовать тремя параметрами $\langle x, y, \phi \rangle$, где x и y — координаты робота (точнее, его центра вращения), а ϕ — ориентация робота, представленная как угол, отсчитываемый от положительного направления оси x. Действие "*Roll*(D)" оказывает эффект в виде изменения состояния робота из $\langle x, y, \phi \rangle$ в $\langle x + D\cos(\phi), y + D\sin(\phi), \phi \rangle$, а действие "*Rotate*($\theta$)" оказывает эффект в виде изменения состояния робота из $\langle x, y, \phi \rangle$ в $\langle x, y, \phi + \theta \rangle$.

а) Предположим, что изначально робот находится в состоянии $\langle 0, 0, 0 \rangle$, а затем выполняет действия *Rotate*(60°), *Roll*(1), *Rotate*(25°), *Roll*(2). Каким будет конечное состояние робота?

б) Теперь предположим, что робот имеет несовершенные средства контроля собственного вращения, и если он предпримет попытку повернуться на угол θ, то фактически он может повернуться на любой угол в пределах от $\theta - 10°$ до $\theta + 10°$. В данной ситуации, если робот попытается выполнить последовательность действий, указанных в задании *а*, существует целый диапазон его возможных конечных состояний. Каковы минимальные и максимальные значения координат x и y, а также угла ориентации θ для этих конечных состояний?

в) Давайте изменим модель, приведенную в варианте *в*, на вероятностную модель, в которой, когда робот пытается повернуться на угол θ, его фактический угол поворота соответствует гауссову распределению со средним значением θ и стандартным отклонением 10°. Предположим, что робот выполняет действия *Rotate*(90°), *Roll*(1). Приведите простой аргумент в пользу того, что, во-первых, ожидаемое значение местоположения в конце не будет равно результату поворота точно на 90°, а затем перемещения на 1 вперед, и, во-вторых, что распределение местоположений после выполнения этой последовательности действий не следует гауссову распределению. (Не пытайтесь вычислить истинное среднее или истинное распределение.)

Смысл этого упражнения в том, что неопределенность в результатах вращения быстро приводит к возникновению значительной неопределенности конечного позиционного положения и что решение проблемы неопределенности вращения является весьма болезненным независимо от того, рассматривается ли неопределенность с точки зрения жестких интервалов или вероятностно, — по той причине, что отношение между ориентацией и положением является как нелинейным, так и немонотонным.

26.10. Рассмотрим упрощенного робота, показанного на рис. 26.40. Предположим, что декартовы координаты местонахождения робота всегда известны, — как известны и координаты его целевого местонахождения. Однако местонахождение препятствий исходно неизвестно. Робот, как показано на этом рисунке, может обнаруживать препятствия, находящиеся в непосредственной близости от него. Для

упрощения предположим, что движения робота не подвержены шуму и что пространство состояний является дискретным. На рис. 26.40 приведен только один пример, но в этом упражнении требуется найти решение для всех возможных клеточных миров, в которых имеется допустимый путь от точки старта до целевого местонахождения.

а) Спроектируйте алгоритмический контроллер, который будет гарантировать, что робот всегда достигнет своего целевого местонахождения, если это вообще возможно. Этот алгоритмический контроллер может запоминать в виде карты результаты измерений, которые он получает по мере передвижения робота. Между отдельными перемещениями робот может затрачивать произвольно долгое время на алгоритмическую обработку информации.

б) Теперь спроектируйте реактивный контроллер для выполнения той же задачи. Он может не запоминать полученные ранее результаты сенсорных измерений. (Он может даже не составлять карту!) Вместо этого он должен принимать все решения на основании текущих результатов измерений, которые включают данные о его собственном местонахождении и о местонахождении цели. Время, необходимое для выработки решения, не должно зависеть от размеров среды или от количества ранее выполненных шагов. Каково максимальное количество шагов, которые могут потребоваться роботу для достижения цели?

в) Какую производительность покажут контроллеры, созданные по условиям в пп. *а* и *б*, если применимо любое из следующих шести условий: непрерывное пространство состояний, зашумленные результаты восприятия, зашумленные результаты движения, зашумленные результаты и восприятия, и движения, неизвестное местонахождение цели (цель может быть обнаружена только в пределах действия датчика расстояний) или движущиеся препятствия. Для каждого условия и каждого контроллера приведите пример ситуации, в которой робот не достигает цели (или объясните, почему такая ситуация не может возникнуть).

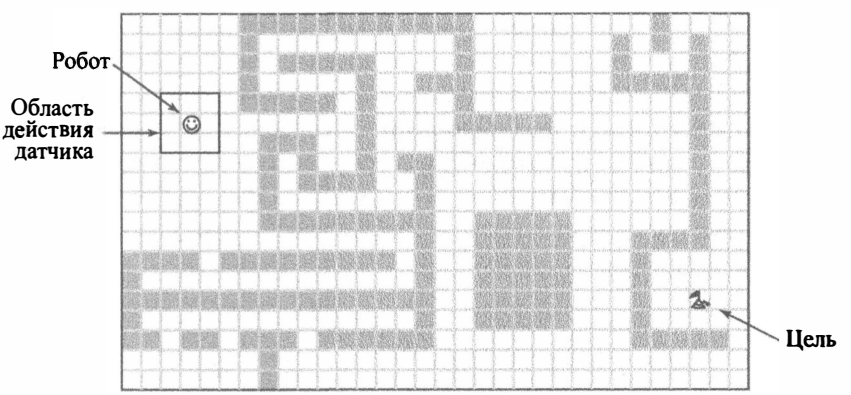


Рис. 26.40. Упрощенный робот в лабиринте для упражнения 26.10

26.11. На рис. 26.32, *б* показан расширенный конечный автомат для управления одной ногой шестиногого робота. Цель этого упражнения — спроектировать автомат AFSM, который объединяет шесть экземпляров отдельных контроллеров ног и обеспечивает эффективное, стабильное движение. Для этой цели необходимо дополнить контроллер отдельной ноги так, чтобы он передавал сообщения вновь разрабатываемому автомату AFSM и ожидал поступления других сообщений. Докажите, почему предложенный вами контроллер является эффективным в том отношении, что он не затрачивает энергию непроизводительно (например, на приведение в движение ног, проскальзывающих в каком-то месте) и что он продвигает робота вперед с достаточно высокой скоростью. Докажите, что предложенный вами контроллер удовлетворяет условию динамической стабильности, приведенному в разделе 26.5.3.

26.12. Люди настолько успешно выполняют простейшие работы по дому, что часто даже не осознают, насколько сложными они являются. Данное упражнение позволяет раскрыть всю сложность элементарных задач и снова пройти путь развития робототехники за последние 50 лет. Рассмотрим задачу построения арки из трех блоков, — необходимо организовать имитацию действий робота с использованием четырех людей, как описано ниже.

Мозг. Задача *Мозга* состоит в том, что он управляет *Руками* при выполнении созданного им плана достижения цели. *Мозг* получает входные данные от *Глаз*, но не может видеть саму сцену непосредственно. *Мозг* — это единственный, кто знает цель.

Глаза. Задача *Глаз* состоит в передаче *Мозгу* краткого описания сцены. (Например, "красный блок стоит на зеленом блоке, лежащем на боку".) *Глаза* также могут отвечать на такие вопросы со стороны *Мозга*, как "Есть ли промежуток между *Левой Рукой* и красным блоком?" Если в вашем распоряжении имеется видеокамера, направьте ее на сцену и разрешите *Глазам* смотреть только в видоискатель видеокамеры, а не прямо на сцену.

Левая Рука и **Правая Рука**. Роль каждой *Руки* играет один человек. Две *Руки* стоят рядом друг с другом, у каждого из игроков на одной руке — которой они будут действовать — рукавица для духовки. *Руки* выполняют лишь простые команды от *Мозга*, например "*Левая Рука*, передвинься на пять сантиметров вперед". *Руки* не могут выполнять команды, отличные от движений, например "Подними блок" — это не та команда, которую может выполнить *Рука*. Глаза у *Рук* должны быть завязаны. Единственные сенсорные возможности, которые им предоставляются, это право сообщить о том, что путь их движения заблокирован неподвижным препятствием, таким как стол или другая *Рука*. В подобных случаях *Руки* могут лишь подать звуковой сигнал, чтобы сообщить *Мозгу* о возникшем затруднении.

Часть VII
Заключение

Философия, этика и безопасность искусственного интеллекта 537

Будущее искусственного интеллекта 587

ГЛАВА 27

Философия, этика и безопасность искусственного интеллекта

В данной главе рассматриваются важные и сложные вопросы о значении ИИ, о том, как разрабатывать и применять его с соблюдением этических норм, а также как можно обеспечить его безопасность.

Философы многие века размышляли над мировыми проблемами: "Как функционирует разум?", "Могут ли машины действовать так же разумно, как это делают люди?", "Будут ли такие машины обладать настоящим, осознающим себя разумом?"

В наше время к ним добавились новые: "Каковы этические последствия использования интеллектуальных машин в повседневной работе?", "Следует ли разрешить машинам принимать решения об убийстве людей?", "Могут ли алгоритмы быть справедливыми и беспристрастными?", "Что будут делать люди, если машины смогут выполнять все виды работ?". И самый важный вопрос: "Как управлять машинами, которые могут оказаться умнее нас?"

27.1. Ограничения ИИ

В 1980 году философ Джон Сирл ввел различие между ▶**слабым ИИ** — идеей, что машины могут действовать, *как если бы* они были разумными — и ▶**сильным ИИ** — утверждением, что машины, которые действуют так, *действительно* сознательно мыслят (а не просто *имитируют* мышление). Со временем определение сильного ИИ сместилось к ссылке на то, что также называют "ИИ уровня человека" или "общий ИИ", — к программам, которые способны решать сколь угодно широкий спектр задач, в том числе новых, и делать это так же, как человек.

Мнение критиков, придерживающихся концепции слабого ИИ и отрицающих саму возможность разумного поведения машин, в настоящее время воспринимается как недальновидное и совпадает с мнением Саймона Ньюкомба, который в октябре 1903 года (всего за два месяца до полета братьев Райт в Китти Хок) написал:

"Полет в воздухе — это одна из большого класса задач, с которыми человек никогда не сможет справиться". Однако быстрый прогресс последних лет еще не доказывает, что не может быть никаких пределов тому, чего способен достичь ИИ. Алан Тьюринг, первый человек, давший определение ИИ ([2235], 1950), также был первым, кто выдвинул возможные возражения против ИИ, предвидя почти все то, что впоследствии было выдвинуто другими.

27.1.1. Аргумент, исходящий из неформализуемости

Сформулированный Тьюрингом "аргумент, основанный на неформализуемости поведения", ссылается на то, что поведение человека слишком сложно для того, чтобы его можно было описать с помощью какого-либо простого набора правил. Люди, должно быть, используют некоторые неформальные правила поведения, которые (как утверждается в этом аргументе) никогда не удастся определить в виде формального набора правил, а следовательно, никогда не удастся закодировать в виде компьютерной программы.

Главным сторонником этой точки зрения был философ Хьюберт Дрейфус, написавший ряд влиятельных критических статей об ИИ, в том числе "What Computers Can't Do" (*Что компьютеры делать не способны*) [651] (1972), ее продолжение "What Computers Still Can't Do" (*Что компьютеры по-прежнему делать не способны*) [652] (1992) и, совместно со своим братом Стюартом, "Mind Over Machine" (*Разум выше машин*) [653] (1986). Аналогично философ Кеннет Сейр в работе [1956] (1993) утверждает: "Искусственный интеллект, *рассматриваемый в рамках культа компьютеционализма*, не имеет даже призрачного шанса на достижение надежных результатов". Направление, которое они критиковали, стали называть ▶**добрым старомодным ИИ** (*Good Old-Fashioned AI* — ▶**GOFAI**).

GOFAI соответствует простейшему логическому агенту, рассматриваемому в главе 7, где было показано, что действительно очень трудно охватить все возможные варианты соответствующего поведения в виде набора необходимых и достаточных логических правил, — это называют **проблемой квалификации**. Однако в главе 12 было показано, что для открытых проблемных областей больше подходят вероятностные системы рассуждений, а в главе 21 продемонстрировано, что системы глубокого обучения хорошо справляются с различными "неформальными" задачами. Таким образом, критические замечания, как оказалось, фактически были направлены не против компьютеров *как таковых*, а скорее против одного конкретного стиля их программирования на основе логических правил — стиля, который был популярен в 1980-х годах, но затем уступил свои позиции новыми подходам.

Один из сильнейших аргументов Дрейфуса направлен против агентов, а не бестелесных механизмов логического вывода. Агент, у которого понятие "собака" формируется лишь на основании ограниченного набора логических высказываний, подобных "$собака(x) \Rightarrow млекопитающее(x)$", будет находиться в более невыгодном положении по сравнению с агентом, который уже видел бегающих собак,

играл с ними в "принеси мне" и был облизан одной из них. Как утверждает философ Энди Кларк ([443], 1998), "Для биологических тел биологические мозги в первую очередь являются элементами управления. Биологические тела движутся и действуют в богатой среде реального мира". По словам Кларка, "мы хороши при игре в фрисби, но плохи в логике".

В подходе ▶**воплощенное познание** утверждается, что не имеет смысла рассматривать мозг отдельно: познание происходит внутри тела, которое внедрено в окружающую среду. Необходимо изучать всю систему как целое, — при функционировании мозг использует закономерности в окружающей среде, в том числе в остальной части его тела. В подходе воплощенного познания робототехника, компьютерное зрение и другие виды датчиков становятся центральными звеньями, а не периферийными.

В целом Дрейфус рассматривал области, в которых исследователи ИИ еще не имели полных ответов, и говорил, что ИИ, следовательно, невозможен. Сейчас можно видеть, что во многих из этих областей продолжают проводиться исследования и разработки, приводящие к повышению *возможности* ИИ, а не его невозможности.

27.1.2. Аргумент, исходящий из неспособности

В "аргументе, исходящем из неспособности", утверждается, что "машина никогда не сможет выполнить действие X". В качестве примеров действий X Тьюринг приводит следующий список.

> Быть добрым, щедрым, красивым, дружелюбным, инициативным, иметь чувство юмора, отличать правду от лжи, совершать ошибки, влюбляться, наслаждаться клубникой со сливками, заставлять влюбляться в себя, учиться на опыте, правильно употреблять слова, быть предметом собственных мыслей, проявлять такое же разнообразие в поведении, как и человек, создавать действительно что-то новое.

Оглядываясь назад, можно сказать, что реализовать некоторые из них оказалось довольно просто — все мы знакомы с компьютерами, которые "делают ошибки". Компьютеры с возможностью выполнения метарассуждений (глава 5) способны анализировать последствия собственных вычислений, являясь, таким образом, предметом собственных рассуждений. Технология вековой давности доказала свою способность "заставить кого-то влюбляться в нее" — плюшевый мишка. Эксперт по компьютерным шахматам Дэвид Леви предсказывает, что к 2050 году люди будут постоянно влюбляться в человекоподобных роботов. Что касается влюбившегося робота, то это популярная тема в фантастике,[1] но по ней имеется лишь ограниченное количество академических размышлений (Ким и др. [1226], 2007). Компьютеры уже выполнили очень многое из того, что является "действительно

[1] Например, опера *Coppélia* (1870), роман *Do Androids Dream of Electric Sheep?* (1968), кинофильмы *AI* (2001), *Wall-E* (2008) и *Her* (2013).

новым", — это значительные открытия в астрономии, математике, химии, минералогии, биологии, компьютерных науках и других областях науки и техники, а также создание новых форм искусства через стиль передачи (Гетис и др. [819], 2016). В целом программы превосходят человеческие возможности в отношении одних задач и отстают в отношении других. Единственное, что они, безусловно, не могут делать, — это быть в точности человеком.

27.1.3. Возражения, основанные на положениях математики

В работах Тьюринга ([2233], 1936) и Гёделя ([874], 1931) было доказано, что в рамках конкретных формальных систем найти ответ на некоторые математические вопросы нельзя даже в принципе. Наиболее известным примером является теорема Гёделя о неполноте (см. раздел 9.5). Кратко она формулируется так: для любой формальной аксиоматической системы F, достаточно мощной, чтобы в ней можно было реализовать арифметику, можно сконструировать так называемое "высказывание Гёделя" $G(F)$ со следующими свойствами.

- $G(F)$ является высказыванием системы F, но не может быть доказано в рамках F.
- Если система F непротиворечива, то высказывание $G(F)$ является истинным.

Философы, в том числе Дж. Р. Лукас ([1460], 1961), утверждали, что эта теорема показывает, что в отношении мышления машины всегда будут уступать человеку, поскольку они являются формальными системами, ограниченными в силу теоремы о неполноте, — они не способны установить истинность высказывания Гёделя, которое сами создадут, тогда как люди свободны от такого ограничения. Споры вокруг данного утверждения породили огромное количество литературы, в том числе две книги математика и физика сэра Роджера Пенроуза ([1773], 1989; [1774], 1994). Пенроуз повторил это утверждение с добавлением некоторых новых поворотов (таких, как гипотеза, согласно которой человек отличается от машины, поскольку его мозг действует на основе квантовой гравитации — теории, включающей множество ложных предсказаний о физиологии мозга).

Рассмотрим три проблемы, связанные с утверждением Лукаса. Во-первых, агенту не следует стыдиться того, что он не может установить истинность некоторых высказываний, тогда как другие агенты могут. Рассмотрим следующее высказывание.

Лукас не может неоспоримо утверждать, что это предложение истинно.

Если бы Лукас подтвердил истинность этого высказывания, то он противоречил бы самому себе, поэтому Лукас не может неоспоримо подтвердить истинность данного высказывания, а это означает, что оно должно быть истинным. Таким образом, было продемонстрировано, что есть такое истинное высказывание, которое Лукас не может неоспоримо подтвердить, тогда как другие люди (и машины) могут. Но это не заставляет нас изменить свое мнение о Лукасе в худшую сторону.

Во-вторых, теорема Гёделя о неполноте и связанные с ней результаты применимы к *математике*, а не к *компьютерам*. Ни одна сущность — человек или машина — не может доказать то, что доказать невозможно. Лукас и Пенроуз ошибочно полагают, что люди могут каким-то образом обойти эти пределы, например когда Лукас в работе [1461] (1976) говорит, что "мы должны принять нашу собственную логичность, если мысль вообще возможна". Но это необоснованное предположение: общеизвестно, что люди непоследовательны. Данное утверждение, безусловно, является верным в отношении повседневных рассуждений, однако оно верно и в отношении тщательного математического мышления. Один из известных примеров — теорема о раскраске карты четырьмя цветами. Математик Альфред Кемпе опубликовал ([1216], 1879) доказательство этой теоремы, которое было широко признанным в течение 11 лет, пока в 1890 году Перси Хивуд не указал на ошибку в этом доказательстве.

В-третьих, теорема Гёделя о неполноте технически применима только к формальным системам, достаточно мощным для реализации арифметики. Это условие справедливо для машин Тьюринга, и утверждение Лукаса частично основано на допущении, что компьютеры эквивалентны машинам Тьюринга. Но это не совсем так. Машины Тьюринга бесконечны, тогда как компьютеры (и мозг) конечны, и поэтому любой компьютер может быть описан как (очень большая) система в логике высказываний, которая не подчиняется теореме Геделя о неполноте. Лукас предполагает, что люди могут "передумать", тогда как компьютеры этого не могут, но это положение также ложно — компьютер может отозвать свое заключение в результате поступления новых свидетельств или дальнейшего обдумывания; он может обновить свое оборудование или изменить процессы принятия решений в результате машинного обучения или переписывания программного обеспечения.

27.1.4. Оценка ИИ

Алан Тьюринг в своей знаменитой статье *Computing Machinery and Intelligence* ([2235], 1950) предложил вместо поиска ответа на вопрос, могут ли машины мыслить, спросить, могут ли машины пройти некоторый поведенческий тест, который в дальнейшем получил название ▶**теста Тьюринга**. Этот тест заключается в том, что программа в течение 5 минут ведет разговор (посредством обмена печатными сообщениями) с некоторым собеседником. Затем этот собеседник должен определить, с кем он беседовал — с программой или с человеком. Программа успешно проходит тест, если ей удается обмануть собеседника в 30% случаев. Для Тьюринга ключевым моментом были не точные детали проведения теста, а сама идея измерения интеллекта посредством выполнения какой-то открытой поведенческой задачи, а не путем философских спекуляций.

Как бы там ни было, Тьюринг предположил, что к 2000 году компьютер с памятью в миллиард единиц сможет пройти этот тест, однако сейчас, когда 2000 год давно миновал, все еще нельзя согласиться с тем, что какая-либо программа его

прошла. Многие люди были одурачены, когда не знали, что могут общаться с компьютером. Программа Eliza и действующие в Интернете чатботы MGONZ (Хамфрис [1096], 2008) и NATACHATA (Джонатан и др. [1143], 2009) неоднократно обманывали людей, а чатбот CYBERLOVER даже привлек внимание правоохранительных органов из-за его склонности обманом заставлять своих собеседников разглашать достаточно много личной информации, чтобы их личность можно было украсть.

В 2014 году чатбот по имени "Eugene Goostman" при проведении теста Тьюринга одурачил 33% неподготовленных судей-любителей. Программа заявляла, что является мальчиком из Украины с ограниченным знанием английского языка, — это помогало объяснить возможные грамматические ошибки. Вероятно, тест Тьюринга на самом деле является тестом на легковерие человека. До сих пор так и не удалось обмануть ни одного достаточно подготовленного судью (Ааронсон [1], 2014).

Соревнования по прохождению теста Тьюринга привели к появлению лучших чатботов, но так и не стали предметом достаточно серьезных исследований в сообществе ИИ. Вместо этого жаждущие соревнований исследователи в большей степени сосредоточивались на игре в шахматы, в го или в StarCraf II, на сдаче экзаменов по естествознанию за 8-й класс или на определении объектов на изображениях. Во многих из этих соревнований программы достигли уровня показателей, свойственных человеку, или превысили его, но это вовсе не означает, что данные программы обладали способностями уровня человека и вне среды конкретной задачи. Цель заключалась в улучшении фундаментальной науки и техники и предоставлении полезных инструментов, а вовсе не в обмане судей.

27.2. Могут ли машины действительно мыслить?

Некоторые философы утверждают, что машина, действующая разумно, не будет *действительно* мыслить, а будет лишь *имитировать* мышление. Но большинство исследователей ИИ это различие мало интересует, а ученый в области компьютерных наук Эдсгер Дейкстра в работе [621] (1984) сказал следующее: "Вопрос о том, *могут ли машины думать*... примерно настолько же значим, как вопрос о том, *умеют ли подводные лодки плавать*". В словаре "American Heritage" первое определение слова "swim" (*плавать*) выглядит так: "Двигаться в воде с помощью конечностей, плавников или хвоста", и большинство людей согласятся с тем, что подводные лодки, не имея конечностей, плавать не могут. Также в этом словаре слово "fly" (*летать*) определяется как "Двигаться в воздухе при помощи крыльев или подобных крыльям частей", и большинство людей согласятся с тем, что самолеты, имеющие подобные крыльям части, могут летать. Однако ни эти вопросы, ни ответы на них не имеют отношения к конструкции или возможностям самолетов и подводных лодок, скорее речь идет об использовании слов в английском языке. Носители английского языка еще не пришли к окончательному решению в отношении

точного определения слова "думать", — для этого требуется именно "мозг" или просто "мозгоподобные части"?

И вновь, интересный взгляд на эту проблему был предоставлен Тьюрингом. Он отметил, что у нас никогда нет *каких-либо* прямых свидетельств о внутреннем ментальном состоянии других людей, — своего рода ментальный солипсизм. Тем не менее Тьюринг говорит: "Вместо постоянных споров по этому поводу, обычно существует ▶**вежливое соглашение**, что мыслят все". Тьюринг привел аргументы в пользу того, что это вежливое соглашение следовало бы расширить и на машины, если бы у нас был опыт общения с такими машинами, которые действуют разумно. Однако сейчас, когда у нас уже есть некоторый подобный опыт, возникает ощущение, что наша готовность приписывать разумность в той же мере зависит от внешнего вида и голоса гуманоида, как и от его интеллектуальности.

27.2.1. Китайская комната

Философ Джон Сирл отвергает вежливое соглашение. Его знаменитый аргумент ▶"**китайская комната**" (Сирл [2016], 1990) сформулирован следующим образом. Представьте человека, который понимает только английский, находящегося в комнате, где есть книга правил, написанная на английском языке, и стопки бумаги с различными китайскими иероглифами. Под дверь в комнате подсовывают кусочки бумаги, содержащие непонятные этому человеку китайские символы, представляющие собой вопрос. Человек, взяв этот вопрос и следуя инструкциям в книге правил, последовательно отыскивает иероглифы вопроса в указанных стопках и записывает рекомендуемый для них ответный иероглиф на чистой бумаге, один за другим. В конце концов, выполнение всех инструкций приведет к тому, что на чистом листе будут записаны один или несколько китайских иероглифов, представляющих собой ответ на полученный вопрос, — этот лист человек возвращает обратно во внешний мир. Снаружи фактически наблюдается система, которая принимает на входе вопросы на китайском языке и выдает на выход осмысленный ответ на них также на беглом китайском языке.

Далее Сирл утверждает: изначально дано, что человек не понимает китайского языка. Книга правил и стопки бумаги с иероглифами, будучи просто кусочками бумаги, также не понимают китайского. Следовательно, в китайской комнате отсутствует какое-либо понимание китайского. Тем не менее вся система создает видимость понимания китайского. В заключение Сирл говорит, что китайская комната со всеми своими компонентами делает то же самое, что делал бы компьютер, поэтому компьютеры не способны генерировать понимания.

Сирл, как следует из его работы [2015] (1980), — сторонник ▶**биологического натурализма**, согласно которому ментальные состояния являются высокоуровневыми вновь возникающими свойствами, которые вызываются физическими процессами низкого уровня *в нейронах*, и именно эти (неопределенные) свойства нейронов имеют значение: согласно убеждению Сирла, нейроны обладают "ими",

а транзисторы — нет. На аргумент Сирла было выдвинуто много опровержений, но согласие пока не достигнуто. С тем же успехом его аргумент может быть использован (возможно, роботами), для доказательства утверждения, что человек не может иметь истинного понимания. В конце концов, человек состоит из клеток, клетки не понимают, поэтому здесь нет и не может быть понимания. На самом деле, у Терри Биссона есть научно-фантастический рассказ ([223], 1990) *Они сделаны из мяса*, в котором инопланетные роботы исследуют Землю и не могут поверить, что куски мяса, вероятно, являются разумными. Как это возможно, остается для них загадкой.

27.2.2. Сознание и квалиа, свойства чувственного опыта

Сквозь все дебаты в отношении сильного ИИ проходит проблема ▶**сознания** (*consciousness*): осознание внешнего мира, себя и субъективного опыта жизни. Техническим термином для внутренней природы переживаний является ▶**квалиа** (*qualia* — от латинского слова, означающего примерно "какого рода"); он определяет свойства внутреннего чувственного опыта, субъективные компоненты восприятия чувств. Это большой вопрос, могут ли машины иметь квалиа. В фильме "Космическая одиссея 2001 года", когда астронавт Дэвид Боумен отключает "когнитивные схемы" компьютера HAL 9000, последний говорит: "Я боюсь, Дэйв. Дэйв, мой разум уходит. Я могу это чувствовать". Действительно ли у компьютера HAL есть чувства (и заслуживают ли они сочувствия)? Или его ответ является просто алгоритмическим следствием, ничем не отличающимся от всем известного "Ошибка 404: не найдено"?

Аналогичный вопрос существует и в отношении животных: владельцы домашних животных уверены, что их собака или кошка обладает сознанием, но не все ученые согласны с этим. Сверчки меняют свое поведение в соответствии с температурой, но немногие люди сказали бы, что сверчки испытывают *ощущения* тепла или холода.

Одна из причин, по которым проблема сознания столь трудна, заключается в том, что она остается нечетко определенной даже после столетий споров. Но помощь может быть уже близка. Недавно философы под эгидой фонда Джона Темплтона создали общую команду с неврологами с целью начать серию экспериментов, которые позволили бы получить ответы на некоторые из вопросов. Сторонники двух ведущих теорий сознания (теории глобальной рабочей области и теории интегрированной информации) согласились с тем, что эти эксперименты могли бы подтвердить правоту одной из этих теорий, — большая редкость в философии.

Алан Тьюринг в работе [2235] (1950) признает, что вопрос о сознании является трудным, но при этом отрицает, что он имеет большое значение для практики ИИ: "Я не хочу, чтобы создалось впечатление, будто я полагаю, что никакой тайны сознания не существует… Но я не думаю, что эти загадки обязательно должны быть решены прежде, чем мы сможем ответить на вопрос, который нас беспокоит

в этой статье". Авторы этой книги согласны с Тьюрингом — мы заинтересованы в создании программ, которые ведут себя разумно. Индивидуальные аспекты сознания — осознание, самосознание, внимание — могут быть запрограммированы и могут стать частью интеллектуальной машины. Дополнительный проект, направленный на то, чтобы сделать машину сознательной в том виде, в каком сознание проявляется у людей, — это не тот проект, которым мы готовы заниматься. Мы согласны с тем, что разумное поведение потребует некоторой степени *осознанности*, которая будет отличаться от задачи к задаче, а задачи, связанные с взаимодействием с людьми, потребуют модели субъективного опыта человека.

Что касается опыта моделирования, у людей есть явное преимущество перед машинами, потому что они могут использовать собственный субъективный аппарат для оценки субъективного опыта других. Например, если вы хотите узнать, *каково это*, когда кто-то бьет молотком по своему большому пальцу, можете просто стукнуть молотком по собственному пальцу. У машин нет такой способности, хотя, в отличие от людей, они могут выполнять программный код друг друга.

27.3. Этика ИИ

Принимая, что ИИ является мощной технологией, мы возлагаем на себя моральное обязательство использовать его хорошо, чтобы содействовать позитивным аспектам его применения и избегать (или смягчать) возможных негативных.

Позитивных аспектов много. Например, ИИ может спасать жизни за счет улучшения медицинской диагностики, благодаря новым открытиям в медицине, лучшему прогнозированию экстремальных погодных явлений и более безопасному вождению при наличии ассистента водителя и (в конечном итоге) использованию технологий беспилотного вождения. Имеется также много возможностей для улучшения жизни людей. Программа корпорации Microsoft "AI for Humanitarian Action" предполагает использование ИИ для проведения восстановительных работ после стихийных бедствий, удовлетворения потребностей детей, защиты беженцев и продвижения прав человека. Программа корпорации Google "AI for Social Good" направлена на поддержку широкого спектра направлений — работ по защите тропических лесов, юриспруденции в области прав человека, мониторинга загрязнения окружающей среды, измерения выбросов в результате использования ископаемого топлива, консультаций по кризисам, проверки новостных фактов, профилактики самоубийств, повторной утилизации отходов, а также других проблем. В центре "Data Science for Social Good" Чикагского университета машинное обучение применяется к проблемам в области уголовного правосудия, экономического развития, образования, здравоохранения, энергетики и охраны окружающей среды.

Применение ИИ в управлении сельским хозяйством и в производстве пищевых продуктов поможет накормить мир. Оптимизация бизнес-процессов с

использованием машинного обучения сделает бизнес более продуктивным, повышая общее благосостояние и обеспечивая больше рабочих мест. Автоматизация позволит освободить людей от выполнения утомительных и опасных операций, с которыми сталкиваются многие работники, предоставив им возможность сосредоточиться на более интересных аспектах своей трудовой деятельности. Люди с ограниченными возможностями получат значительные преимущества от использования ИИ с целью оказания помощи в отношении зрения, слуха и мобильности. Машинный перевод уже гарантирует возможность общения людям, представляющим разные культуры. Программные решения ИИ обеспечивают почти нулевую предельную себестоимость производства и обладают потенциалом демократизации доступа к передовым технологиям (даже если другие аспекты программного обеспечения обладают потенциалом для централизации власти).

Несмотря на наличие многих положительных моментов, не следует игнорировать существование и отрицательных. Внедрение множества новых технологий уже приводило к непреднамеренным ▶**негативным побочным эффектам**: ядерное деление принесло Чернобыль и угрозу глобального уничтожения; двигатель внутреннего сгорания в конечном счете привел к загрязнению воздуха, глобальному потеплению и безудержному разрастанию дорожной сети в ущерб природе. Другие технологии способны приводить к негативным последствиям даже при их использовании по назначению, например газ зарин, винтовка AR-15 или навязывание услуг и товаров по телефону. Автоматизация способствует увеличению доходов, но в нынешних экономических условиях бо́льшая часть этих доходов будет поступать владельцам автоматизированных систем, что приведет к дальнейшему росту неравенства в доходах. Эта тенденция может оказаться разрушительной для успешно функционирующего общества. В развивающихся странах их традиционный метод роста за счет дешевого производства товаров на экспорт может оказаться блокированным, поскольку богатые страны будут использовать полностью автоматизированные производственные мощности на их собственной территории. Принятые нами этические и управленческие решения определят уровень неравенства, порожденного использованием ИИ.

Все ученые и инженеры сталкиваются с этическими соображениями в отношении того, какие проекты они должны или не должны принимать к исполнению и как они могут убедиться в том, что реализация проекта будет безопасной и полезной. В 2010 году Исследовательский совет по инженерным и физическим наукам Великобритании провел совещание по разработке свода основных принципов робототехники. В последующие годы другие государственные учреждения, некоммерческие организации и компании создали аналогичные своды принципов. Суть в том, что каждая организация, занимающаяся созданием технологий ИИ, как и каждый ее работник, несет ответственность за то, чтобы иметь полную уверенность в том, что эта технология принесет благо, а не вред. К наиболее часто цитируемым принципам относятся следующие.

Обеспечение безопасности	Установление ответственности
Обеспечение справедливости	Отстаивание прав и ценностей человека
Уважение к конфиденциальности	Отражение разнообразия и вложения
Содействие сотрудничеству	Недопущение концентрации власти
Обеспечение прозрачности	Осознание правовых и политических последствий
Ограничение возможностей использования ИИ во вред	Рассмотрение последствий в отношении занятости

Обратите внимание, что многие из этих принципов, такие как "Обеспечение безопасности", применимы ко всем программным и аппаратным системам, а не только к системам ИИ. Некоторые из этих принципов сформулированы довольно туманно, что затрудняет их измерение или проведение в жизнь. Отчасти так произошло потому, что ИИ — это огромная область исследований с множеством подобластей, каждой из которых свойственны свой набор исторических норм и различные отношения между разработчиками ИИ и заинтересованными сторонами. Миттельштадт в работе [1600] (2019) предполагает, что в каждой из этих подобластей следует разработать более конкретные практические рекомендации и соответствующие прецеденты.

27.3.1. Смертельное автономное оружие

В материалах ООН смертельное автономное оружие определяется как оружие, которое находит, отбирает и поражает (т.е. убивает) человеческие цели без контроля со стороны человека. Под этот критерий подпадают различные виды оружия, например наземные мины, которые используются с XVII века: они способны "выбирать и поражать" цели — в ограниченном смысле, в зависимости от величины приложенного давления или количества присутствующего металла, но они не способны выбраться из земли и отыскать цель сами по себе. (Наземные мины запрещены в соответствии с Оттавским договором.) Управляемые ракеты, используемые с 1940-х годов, способны преследовать цели, но сначала человек должен выпустить их в нужном направлении. Автоматически ведущие огонь управляемые радаром пушки использовались для защиты морских судов уже в 1970-х годах. В основном они предназначались для уничтожения приближающихся ракет, но способны атаковать и пилотируемые самолеты. Хотя слово "автономный" довольно часто используется для описания беспилотных летательных аппаратов или **дронов**, большинство таких видов оружия требуют как удаленного пилотирования человеком, так и приведениям им в действие их смертельного груза.

На момент написания этой главы несколько оружейных систем, кажется, уже перешли границу полной автономии. Например израильская ракета Harop является "патрулирующим боеприпасом" с размахом крыльев три метра и

20-килограммовой боеголовкой. В течение определенного времени (до шести часов) она пытается обнаружить в указанном географическом регионе любую цель, отвечающую заданному критерию, и если обнаружит, самостоятельно ее уничтожает. Критерием может быть "излучение радиолокационного сигнала, напоминающего зенитный радар" или "внешний вид танка". Турецкая компания STM представила свой продукт Kargu (квадрокоптер, способный нести до 1,5 килограмма взрывчатых веществ), заявив, что он способен выполнять "автономное поражение... целей, выбранных на изображениях... отслеживание движущихся целей... противопехотные действия... распознавание лиц".

Появление автономного оружия уже было классифицировано как "третья революция в военном деле" после изобретения пороха и разработки ядерного оружия. Его военный потенциал очевиден. Например, мало кто из экспертов сомневается, что автономный истребитель способен победить любого пилота-человека. Автономные самолеты, танки и подводные лодки могут быть более дешевыми, быстрыми и маневренными и иметь большую дальность, чем их пилотируемые аналоги.

С 2014 года отделение Организации Объединенных Наций в Женеве под эгидой Конвенции по конкретным видам обычного оружия (CCW) регулярно проводит дискуссии по вопросу о том, следует ли запретить смертельное автономное оружие. На момент написания этой главы 30 стран, от большого Китая до маленького Ватикана, заявили о своей поддержке такого международного договора, тогда как другие ведущие страны, включая Израиль, Россию, Южную Корею и США, выступают против подобного запрета.

Дискуссия по поводу автономного оружия включает правовые, этические и практические аспекты. Правовые вопросы регулируются в первую очередь CCW, — для этого требуется иметь возможность отличить участников сражений от тех, кто в них не участвует, оценить военную необходимость для атаки, а также оценить соразмерность между военным значением цели и возможностью нанесения побочного ущерба. Обеспечение соответствия автономного оружия этим критериям является вопросом инженерии, ответ на который со временем, несомненно, изменится. В настоящее время успешное различение участников сражений представляется вполне достижимым в некоторых обстоятельствах и, несомненно, будет быстро улучшаться. Однако реализация автономной оценки необходимости и соразмерности на нынешний момент не представляется возможной: в данном случае требуется, чтобы машины выносили субъективные и ситуативные суждения, что является значительно более трудной задачей, чем относительно простые задачи поиска и отбора потенциальных целей. По этим причинам было бы законно использовать автономное оружие только в тех случаях, когда человек-оператор может обоснованно предсказать, что выполнение миссии не приведет к поражению мирных граждан или что автономное оружие не осуществит ненужных или несоразмерных атак. А это означает, что на данный момент только очень ограниченные миссии могут быть предприняты с использованием автономного оружия.

С этической стороны некоторые считают просто морально неприемлемым делегировать машине право принимать решение об убийстве человека. Например, представитель Германии в Женеве заявил, что "мы считаем неприемлемым, что решение о жизни и смерти принимается исключительно самой автономной системой", тогда как Япония "не имеет планов разработки роботов с людьми вне цикла, которые могут быть способны совершать убийство". В 2017 году генерал Поль Сельва, на то время офицер второго ранга армии США, сказал: "Я не думаю, что для нас будет разумным поставить робота перед выбором, отнять или сохранить жизнь человеку". Наконец, в 2019 году Антониу Гутерриш, генеральный секретарь Организации Объединенных Наций, заявил: "Машины со способностью и правом отнимать жизни без участия человека политически неприемлемы, морально отвратительны и должны быть запрещены международным правом".

Более 140 общественных организаций более чем в 60 странах мира приняли участие в кампании "Stop Killer Robots" (*Стоп роботам-убийцам*) и распространении открытого письма, организованном в 2015 году организацией Future of Life Institute (*Институт будущего жизни*). Это открытое письмо было подписано более чем 4 тыс. исследователей ИИ[2] и 22 тыс. других людей.

В противовес этому можно утверждать, что по мере совершенствования технологий должна появиться возможность разрабатывать такое оружие, которое с *меньшей* вероятностью, чем солдаты или пилоты-люди, будет приводить к жертвам среди гражданского населения. (Здесь также важно учитывать то преимущество, что автономное оружие уменьшает необходимость солдат и пилотов-людей рисковать собственной жизнью.) Автономные системы не будут поддаваться усталости, разочарованию, истерии, страху, гневу или желанию отомстить, им не нужно "сначала стрелять, а затем задавать вопросы" (Аркин [71], 2015). Подобно тому, как управляемые боевые средства наносят меньший сопутствующий ущерб по сравнению с неуправляемыми снарядами и бомбами, можно ожидать, что разумное оружие обеспечит дальнейшее повышение точности атак. (В противовес этому утверждению читайте статью Бенжамина [175] (2013), содержащую анализ данных о жертвах беспилотных боевых действий.) Именно это положение, видимо, соответствует позиции США на последнем раунде переговоров в Женеве.

Вероятно, нелогично, что США также являются одной из немногих стран мира, собственная политика которых в настоящее время исключает использование автономного оружия. Перспективный план Министерства обороны США от 2011 года гласит: "В обозримом будущем решения относительно применения силы [автономными системами] и выбора отдельных целей для поражения смертельным оружием будут оставаться под контролем человека". Основная причина такой политики сугубо практична: автономные системы пока недостаточно надежны, чтобы можно было доверить им принятие боевых решений.

[2] Включая двух авторов этой книги.

Проблема надежности вышла на первый план 26 сентября 1983 года, когда на дисплее офицера советских войск противоракетной обороны Станислава Петрова вспыхнуло предупреждение о нападении и приближающихся ракетах. Согласно протоколу Петров должен был инициировать ядерную контратаку, но он заподозрил, что эта тревога является ошибочной, и отнесся к ней как к таковой. И он был прав, — Третья мировая война была (едва) предотвращена. Мы не знаем, что бы произошло, если бы в цепочку управления не был включен человек.

Надежность является очень серьезной проблемой для военных командиров, которые хорошо знакомы со сложностью боевых ситуаций. Системы машинного обучения, безупречно работавшие при обучении, после развертывания могут работать плохо. Кибератаки против автономного оружия способны привести к жертвам в дружественных соединениях, — отключение такого оружия от всех линий связи позволит предотвратить подобный исход (при условии, что оно еще не было повреждено), но это оружие уже нельзя будет отозвать, если оно окажется неисправным.

Основная практическая проблема с автономным оружием заключается в том, что оно является масштабируемым оружием массового уничтожения — в том смысле, что масштаб атаки, которую можно будет предпринять, пропорционален количеству оборудования, которое можно позволить себе развернуть. Квадрокоптер диаметром два дюйма уже способен нести смертельный заряд взрывчатки, а в обычный транспортный контейнер их можно поместить до миллиона. Именно потому, что оно является автономным, этому оружию не потребуется один миллион контролеров-людей, чтобы выполнить свою работу.

Как и оружие массового уничтожения, масштабируемое автономное оружие для нападающего имеет большое преимущество по сравнению с ядерным оружием и ковровыми бомбардировками: оно оставляет имущество неповрежденным и может применяться выборочно для устранения только тех, кто будет представлять собой угрозу оккупационным силам. Конечно, это оружие может быть использовано и для того, чтобы стереть с лица Земли этническую группу или уничтожить всех приверженцев определенной религии. Также во многих ситуациях оно могло бы оставаться просто неотслеживаемыми. Подобные характеристики делают такое оружие особенно привлекательным для негосударственных субъектов.

Все эти соображения — особенно характеристики, выгодные атакующему — наводят на мысль, что автономное оружие уменьшит глобальную и национальную безопасность для всех сторон. Разумным ответом правительств, по-видимому, является участие в обсуждениях вопросов контроля над вооружениями, а не гонка вооружений.

Однако процесс разработки такого договора не лишен трудностей. ИИ является такой областью, достижения в которой можно ▶**использовать двояко**: технологии ИИ, изначально предназначенные для мирного применения — управление полетом, визуальное отслеживание, картография, навигация или мультиагентное планирование, — совсем несложно применить и в военных целях. Очень просто превратить автономный квадрокоптер в оружие, просто прицепив к нему

взрывчатку и дав ему команду искать цель. Решение этой проблемы потребует тщательного соблюдения всех договорных требований и отраслевой кооперации, как уже было продемонстрировано с определенным успехом в отношении Конвенции о запрещении химического оружия.

27.3.2. Наблюдение, безопасность и конфиденциальность

В 1976 году Джозеф Вейценбаум предупредил, что технология автоматического распознавания речи может привести к повсеместному прослушиванию телефонных разговоров и, следовательно, к утрате гражданских свобод. Сегодня эта угроза уже вполне реализована: большинство электронных коммуникаций проходят через центральные серверы, которые можно контролировать, а города наполнены микрофонами и видеокамерами, способными идентифицировать и отслеживать людей по их голосу, лицу и походке. Наблюдение, которое раньше требовало дорогих и скудных человеческих ресурсов, теперь может осуществляться в массовом масштабе с помощью машин.

Так, в 2018 году уже было установлено, ни много, ни мало, 350 млн ▶**камер видеонаблюдения** в Китае и 70 млн — в США. Китай и другие страны начали экспортировать технологии наблюдения в слаборазвитые в техническом отношении страны, за некоторыми из которых закрепилась репутация жестокого обращения со своими гражданами и несоразмерной ориентации на определенные маргинализированные сообщества. Разработчики в области ИИ должны понимать, какие виды использования наблюдения совместимы с правами человека, и отклонять предложения о работе над приложениями, которые несовместимы с ними.

Поскольку все больше различных учреждений работают в Интернете, мы становимся все более уязвимыми для киберпреступности (фишинг, мошенничество с кредитными картами, бот-сети, приложения-вымогатели) и кибертерроризма (включая потенциально смертельные атаки, такие как блокирование больниц и электростанций или управление беспилотными автомашинами). Машинное обучение может стать мощным инструментом для обеих сторон в битве за нашу ▶**кибербезопасность**. Злоумышленники могут использовать автоматизацию для исследования уязвимостей и применять обучение с подкреплением для попыток фишинга и автоматизированного шантажа. Защитники могут использовать методы обучения без учителя для выявления аномальных моделей входящего трафика (Чандола и др. [387], 2009; Малхотра и др. [1479], 2015) и различные методы машинного обучения для выявления случаев мошенничества (Фосетт и Провост [712], 1997; Болтон и Хэнд [243], 2002). Поскольку атаки становятся все более сложными, возрастает ответственность всех разработчиков, а не только специалистов в области безопасности, за создание безопасных систем уже с момента начала работ. В одном из прогнозов (Канал [1178], 2017) рынок систем машинного обучения в области кибербезопасности в 2021 году оценивается на уровне 100 млрд долл.

Чем больше мы взаимодействуем с компьютерами для увеличения качества и значимости своей повседневной жизни, тем больше данных о нас собирается правительствами и корпорациями. Сборщики данных несут моральную и юридическую ответственность за то, чтобы обеспечить надежное хранение данных, которыми они располагают. В США Закон о мобильности и подотчетности медицинского страхования (HIPAA) и Закон о правах и конфиденциальности семейного образования (FERPA) защищают конфиденциальность медицинских и студенческих записей. Общее положение о защите данных Европейского союза (GDPR) обязывает компании разрабатывать собственные системы с учетом необходимой защиты данных и требует, чтобы они получали согласие пользователей на любой сбор или обработку данных о них.

Сбалансированное право человека на неприкосновенность частной жизни — это ценность, которую общество получает от совместного использования данных. Мы хотим быть в состоянии остановить террористов, не подавляя мирного инакомыслия, и мы хотим излечивать болезни, не ставя под угрозу право любого человека сохранять в тайне историю своего здоровья. Одной из ключевых практик является ▶**обезличивание** (*de-identification*): исключение из данных личной информации (такой, например, как имя и номер полиса социального страхования), благодаря чему исследователи в области медицины получают возможность использовать эти данные во имя общего блага. Проблема заключается в том, что поступившие в совместное использование обезличенные данные могут подвергаться повторной идентификации. Например, даже если из данных вычеркнуты имя, номер полиса социального страхования и адрес проживания, но они содержат дату рождения, пол и почтовый код, то, как было показано Латанией Суини ([2161], 2000), вычеркнутые сведения могут быть однозначно восстановлены для 87% населения США. Суини подтвердила это утверждение, повторно идентифицировав историю болезни губернатора ее штата, когда он был помещен в больницу. В конкурсе ▶**Netflix Prize** соревнующимся были предоставлены обезличенные записи о проставлении зрителями рейтинга отдельным фильмам и им предложили разработать алгоритм машинного обучения, который позволит точно предсказать, какие фильмы понравятся отдельным зрителям, из числа предоставивших оценки. Однако участники соревнований сумели повторно идентифицировать определенных зрителей, сопоставив дату проставления рейтинга фильму в базе данных компании Netflix с датой проставления аналогичного рейтинга в базе данных фильмов в Интернете (IMDB), где пользователи иногда используют свои настоящие имена (Нараянан и Шматиков [1657], 2006).

Этот риск может быть несколько смягчен с помощью ▶**обобщающих полей**, например путем замены точной даты рождения только годом рождения или даже более широким диапазоном, таким как “20–30 лет”. Удаление значения поля целиком можно рассматривать как форму обобщения со значением *любое*. Но одно лишь обобщение вовсе не гарантирует, что записи станут безопасными в отношении повторной идентификации, — вполне может оказаться, что существует только один человек с почтовым индексом “94720”, которому 90–100 лет. Полезным свойством

является ▶***k*-анонимность**: база данных является *k*-анонимной, если каждая запись данных в ней является неотличимой от по меньшей мере $k-1$ других записей данных. Если есть записи, которые оказываются уникальными в большей степени, чем это требуется, они должны быть подвержены дальнейшему обобщению.

Альтернативой совместному использованию обезличенных записей является сохранение всех записей в секрете, но с разрешением использования ▶**совокупных запросов**. Для выполнения таких запросов к базе данных предоставляется соответствующее API и на допустимые запросы дается ответ, в котором данные суммируются посредством подсчета записей или вычисления среднего значения. Однако если при этом могут быть нарушены определенные требования к сохранению конфиденциальности, то никакого ответа предоставлено не будет. Например, можно позволить эпидемиологу выдать запрос о проценте людей, болеющих с раком, для каждого почтового кода. Для тех почтовых кодов, для которых рак диагностирован не менее чем у *n* людей, будет предоставлен соответствующий процент (с небольшим количеством случайного шума), но для почтовых индексов, для которых раком болеет менее *n* людей, никакого ответа предоставлено не будет.

Следует позаботиться о принятии мер для защиты и от повторной идентификации посредством нескольких запросов. Например, если на запрос "средняя заработная плата и количество сотрудников компании XYZ в возрасте 30–40 лет" получен ответ [$81 234, 12], а на запрос "средняя зарплата и количество сотрудников компании XYZ в возрасте 30–41" получен ответ [$81 199, 13], то можно зайти на сайт LinkedIn и попытаться найти единственного сотрудника компании XYZ в возрасте 41 года. В случае успеха он будет повторно идентифицирован, и теперь можно будет вычислить его точную зарплату несмотря даже на то, что полученные ответы включали сведения о 12 и более сотрудниках. Системы следует разрабатывать очень тщательно, чтобы обеспечить защиту от подобных действий с помощью комбинации ограничений в отношении запросов, которые могут быть заданы (возможно, следует разрешить обработку запросов только с предопределенным набором непересекающихся возрастных диапазонов), и точности возвращаемых ответов (допустим, ответ на оба запроса будет иметь вид "около $81 000").

Более сильным подходом является ▶**дифференциальная приватность**, гарантирующая, что злоумышленник не сможет использовать запросы для повторной идентификации какого-либо лица в базе данных, даже если выполнит несколько запросов и получит доступ к отдельным связанным базам данных. В данном случае в механизме построения ответа на запрос используется рандомизированный алгоритм, добавляющий к результатам небольшое количество шума. При заданных базе данных *D*, произвольной записи в базе данных *r*, некотором запросе *Q* и возможном ответе на него *y* говорят, что база данных *D* имеет ϵ-дифференциальную приватность, если логарифм вероятности ответа *y* меняется менее чем на ϵ при добавлении записи *r*:

$$|\log P(Q(D) = y) - \log \mathrm{P}(\mathrm{Q}(\mathrm{D} + r) = y)| \leq \epsilon.$$

Другими словами, то, решает ли какой-либо один человек поместить свои личные данные в базу данных, не оказывает заметного влияния на ответы, которые кто-либо может получить от нее, а значит, нет никакой конфиденциальности, препятствующей ему сделать это. Многие базы данных разработаны с обеспечением их дифференциальной приватности.

До сих пор обсуждался вопрос совместного использования обезличенных данных, хранящихся в централизованной базе данных. При другом подходе, получившем название ▶**федеративное обучение** (Конечны и др. [1276], 2016), нет никакой центральной базы данных; вместо нее пользователи поддерживают собственные локальные базы данных, в которых их данные хранятся конфиденциально. Однако они могут совместно использовать параметры модели машинного обучения, которая постоянно улучшается их данными, без риска раскрытия каких-либо личных сведений. Представьте себе приложение для понимания речи, которое пользователи могут запускать локально на своем телефоне. Это приложение включает базовую нейронную сеть, которая со временем улучшается за счет локального обучения на основе тех слов, которые были услышаны на телефоне пользователя. Периодически владельцы приложения опрашивают подмножество пользователей и запрашивают у них значения параметров их улучшенной локальной сети вместо предоставления им каких-либо необработанных данных. Значения параметров объединяются для формирования новой улучшенной модели, которая затем становится доступной для всех пользователей, так что все они получают преимущества от обучения, которое проводят другие пользователи.

Чтобы эта схема обеспечивала сохранение конфиденциальности, требуется иметь возможность гарантировать, что параметры модели, совместно используемые всеми пользователями, не могут быть перепроектированы. Если отправлять другим необработанные параметры, то есть вероятность, что анализирующий их злоумышленник сможет определить, было ли, скажем, определенное слово услышано на телефоне пользователя. Одним из способов устранения этого риска является ▶**безопасное агрегирование** (Бонавиц и др. [244], 2017). Идея состоит в том, что центральному серверу не требуется знать точное значение параметра от каждого пользователя в распределенной системе, ему нужно знать лишь *среднее* значение каждого параметра по всем опрошенным пользователям. Поэтому каждый пользователь может скрыть собственные значения параметров, добавив к каждому из них уникальную маску, — пока сумма масок всех пользователей равна нулю, центральный сервер может вычислить правильное среднее значение. Детали этого протокола гарантируют, что он является эффективным в терминах коммуникации (маскированию соответствует менее половины передаваемых битов), устойчивым к сбоям при передаче ответа отдельными пользователями и безопасным в отношении действий злоумышленников, прослушки и даже противодействующего центрального сервера.

27.3.3. Справедливость и предвзятость

Роль систем машинного обучения возрастает, и иногда они уже заменяют людей, принимающих ответственные решения: кому будет предоставлен кредит, в каком районе усилить патрулирование полиции, кого из заключенных можно выпустить досрочно или условно-досрочно. Однако в моделях машинного обучения может быть зафиксирована ▶**социальная предвзятость** (*societal bias*). Рассмотрим пример алгоритма прогнозирования вероятности того, что обвиняемые по уголовным делам совершат повторное преступление, — это требуется для принятия решения, допустимо ли освободить их от содержания под стражей до суда. Вполне вероятно, что такая система воспримет расовые или гендерные предубеждения людей-судей на основании предоставленных ей примеров из обучающего набора данных. Разработчики систем машинного обучения несут моральную ответственность в отношении предоставления гарантий того, что их системы будут действительно справедливыми. В регулируемых предметных областях, таких как предоставление кредитов, образование, наем рабочей силы и жилищное строительство, они несут и юридическую ответственность. Но что такое справедливость? Существует много критериев, и ниже приведены шесть из них, используемых чаще всего.

- **Индивидуальная справедливость**. Требование того, чтобы отношение к отдельным людям было таким же, как и к другим подобным им людям, независимо от того, к какой группе они принадлежат.
- **Групповая справедливость**. Требование, чтобы две группы людей рассматривались одинаково в соответствии с некоторой сводной статистикой.
- **Справедливость через незнание**. Если из записей обучающего набора данных удалить атрибуты расы и пола, то может показаться, что система не сможет воспринять дискриминацию по этим атрибутам. К сожалению, уже известно, что модели машинного обучения могут прогнозировать скрытые переменные (такие, как раса и пол) с учетом других коррелирующих с ними переменных (таких, как почтовый индекс или род занятий). Более того, удаление этих атрибутов делает невозможным контроль обеспечения равных возможностей или равных результатов. Тем не менее в некоторых странах (например, в Германии) был выбран именно этот подход в демографической статистике (независимо от того, задействованы в ней модели машинного обучения или нет).
- **Равный результат**. Эта идея заключается в том, что каждая демографическая группа должна получать одинаковый результат, т.е. это идея соблюдения ▶**демографического паритета**. Например предположим, что задача заключается в принятии решения, следует ли утвердить заявку на кредит. Назначение системы — утверждать заявки тех заявителей, которые вернут кредит, и не утверждать заявки тех, кто не выполнит свои обязательства по

кредиту. Условие демографического паритета требует, чтобы и для мужчин, и для женщин процент утвержденных кредитов был одинаков. Обратите внимание, что это критерий групповой справедливости, который никак не влияет на обеспечение индивидуальной справедливости: заявителю с высокой квалификацией может быть отказано в кредите, а заявителю с низкой квалификацией он может быть предоставлен — при условии, что общие процентные показатели равны. Кроме того, такой подход способствует исправлению прошлых предубеждений относительно точности прогноза. Если мужчина и женщина равны во всех отношениях за исключением того, что женщина получает более низкую зарплату за ту же работу, следует ли одобрить ее заявку, потому что их шансы были бы равны, если бы не исторические предрассудки, или ей следует отказать, потому что ее более низкая зарплата фактически делает ее заявку менее желательной по умолчанию?

- **Равные возможности**. Речь идет о том, что люди, которые действительно имеют возможность погасить кредит, должны иметь равные шансы на то, чтобы их правильно классифицировали как таковых, независимо от их пола. Данный подход также называется "баланс". Он способен привести к неравным результатам и игнорирует эффекты от предвзятости в социальных процессах, которые были восприняты из обучающих данных.
- **Равное влияние**. Люди с аналогичной вероятностью погашения кредита должны иметь одинаковую ожидаемую полезность, независимо от того, к какой группе они принадлежат. Это понятие выходит за рамки понятия равных возможностей, поскольку учитывает как преимущества от истинного предсказания, так и издержки от ложного предсказания.

Давайте посмотрим, как эти вопросы решаются в конкретном контексте. COMPAS — это коммерческая система подсчета рецидивов (повторных преступлений). Она присваивает **оценку риска** подсудимому, проходящему по уголовному делу, которая затем используется судьей для принятия решений: безопасно ли выпустить подсудимого из-под стражи до суда или его следует содержать в тюрьме? В случае признания его виновным насколько строгим должен быть приговор? Можно ли предоставить ему условно-досрочное освобождение? С учетом важности этих решений система оказалась объектом пристального внимания (Дрессел и Фарид [650], 2018).

Система COMPAS разработана как ▶ **хорошо откалиброванная**: все обвиняемые, получающие по алгоритму один и тот же балл, должны характеризоваться примерно одинаковой вероятностью рецидива независимо от их расы. Например, среди всех, кому модель присваивает оценку риска 7 из 10, повторно совершают преступления 60% белых и 61% темнокожих. На этом основании разработчики утверждают, что работа алгоритма соответствует цели — достижению желаемой справедливости.

С другой стороны, система COMPAS не предоставляет равных возможностей: доля тех, кто не совершил повторного правонарушения, но ошибочно получил оценку, соответствующую высокому риску, составила 45% для темнокожих и 23% — для белых. В деле "Штат против Лумиса", в котором судья полагался на оценку системы COMPAS при вынесении приговора подсудимому, Лумис утверждал, что по ходу внутренней скрытой работы алгоритма были нарушены его процессуальные права. Хотя Верховный суд штата Висконсин постановил, что в этом случае вынесенный приговор не отличается от приговора, который был бы вынесен без использования системы COMPAS, он также вынес предупреждение в отношении точности алгоритма и рисков для ответчиков из числа меньшинств. Другие исследователи поставили под сомнение целесообразность использования алгоритмов в таких приложениях, как вынесение приговора.

Конечно, можно было бы надеяться на алгоритм, который хорошо откалиброван и обеспечивает равные возможности, но, как показал Клейнберг и соавт. ([1240], 2016), это невозможно. Если базовые группы различны, то любой алгоритм, который был хорошо откалиброван, обязательно *не будет* обеспечивать равные возможности, и наоборот. Как можно взвесить два критерия? Одна из возможностей — равное влияние. В случае системы COMPAS это означает сопоставление отрицательной полезности для подсудимых, которые были ошибочно классифицированы как характеризующиеся высоким риском и потеряли свободу, с тем, что будет стоить для общества совершение дополнительного преступления, и нахождения точки компромисса, оптимизирующей оба значения. Сделать это очень сложно, потому что имеется множество видов потерь, требующих рассмотрения. Есть индивидуальные потери — страдания подсудимого, который неправомерно подвергался тюремному заключению, как и страдания жертвы подсудимого, который был неправомерно освобожден и совершил преступление повторно. Но, помимо этого, есть и групповые потери: у каждого есть определенный страх, что он будет незаконно заключен в тюрьму или станет жертвой преступления, и все налогоплательщики вносят свой вклад в содержание тюрем и судов. Если присвоить этим страхам и расходам значение, пропорциональное размеру группы, то полезность для большинства может достигаться за счет меньшинства.

Еще одна проблема в отношении идеи оценки возможности рецидивизма в целом, независимо от используемой модели, заключается в том, что не существует беспристрастных истинных данных. Данные вовсе не говорят нам о том, кто *совершил* преступление, нам лишь известно, кто был *признан виновным* в его совершении. Если лица, проводившие арест, судья или присяжные проявят предвзятость, данные будут отражать эти предубеждения. Если большее количество полицейских патрулей сосредоточено в определенных местах, то данные будут отражать их предвзятость в отношении людей из этих мест. Только обвиняемые, которые были освобождены из-под стражи, становятся кандидатами на повторное совершение преступления, поэтому, если судьи предвзято выносят решения об освобождении, соответствующие данные могут отражать их предубеждения. Если

есть предположение, что в основе набора данных, искаженных предвзятостью, лежит набор неизвестных, свободных от предубеждений данных, который был искажен агентами с предвзятостью, то есть методы, позволяющие восстановить эти данные и привести их к приблизительно беспристрастному состоянию. Цзянь и Нечум в работе [1137] (2019) описывают различные сценарии и методы, используемые для этой цели.

Дополнительный риск заключается в том, что машинное обучение может использоваться для *оправдания* предвзятости. Если решения принимаются предвзятым человеком после консультации с системой машинного обучения, то человек может сказать "вот видите, моя интерпретация модели поддерживает мое решение, так что вам не следует сомневаться в моем решении". Однако другие интерпретации могут привести к противоположному решению.

Иногда справедливость означает, что необходимо пересмотреть целевую функцию, а не данные или алгоритм. Например, при принятии решений о найме на работу, если цель состоит в том, чтобы нанять кандидата, обладающего лучшей квалификацией, появляется риск несправедливо наградить тех, кто уже имел преимущества в возможности получения образования на протяжении всей своей жизни, тем самым укрепляя границы, существующие между отдельными группами. Но если целью является наем кандидатов с наилучшими способностями к обучению данной профессии, появляется хороший шанс преодоления границ между группами и расширения круга лиц, среди которых будет сделан выбор. У многих компаний есть программы, предназначенные для таких кандидатов, и было установлено, что после обучения в течение года сотрудники, нанятые подобным образом, работают на том же уровне, что и те, кто был нанят при традиционном подходе. Аналогичным образом в США только 18% дипломированных специалистов по компьютерным наукам — женщины, но в некоторых учебных заведениях, таких как университет Harvey Mudd, в этом вопросе достигнут паритет на уровне 50% за счет использования подхода, ориентированного на стимулирование и поддержку тех, кто приступает к учебным программам по компьютерным наукам, и в особенности тех, кто начинает обучение с меньшим опытом программирования.

Последнее затруднение заключается в принятии решения, какие группы населения заслуживают защиты. В США в Законе о равных правах на жилье признаны семь защищаемых категорий граждан: по расе, цвету кожи, религии, национальному происхождению, полу, инвалидности и семейному статусу. В других местных, уровня штата и федеральных законах признаются и другие защищаемые группы, в том числе по сексуальной ориентации, а также беременности, семейному положению и статусу ветерана. Справедливо ли, что отношение к этим группам имеет значение для одних законов, но не имеет значения для других? Потенциальной основой для согласования мер защиты различных групп населения является Международное законодательство в области прав человека, охватывающее широкий круг защищаемых категорий.

Даже при отсутствии общественного предубеждения привести к получению предвзятых результатов может ▶**несоответствие размеров выборки**. В большинстве наборов данных будет меньше обучающих примеров для представителей групп меньшинств, чем для представителей из группы большинства. Алгоритмы машинного обучения дают более высокую точность при большем количестве обучающих данных, а это означает, что для представителей групп меньшинств точность всегда будет ниже. Например, Буоламвини и Гебру в работе [344] (2018) привели результаты изучения сервиса идентификации пола с помощью компьютерного зрения и обнаружили, что он обеспечивает почти идеальную точность для светлокожих мужчин и ошибается в 33% случаев для темнокожих женщин. Ограниченная модель может быть неспособна одновременно подстраиваться под группы большинства и меньшинств — модель линейной регрессии позволяет минимизировать среднюю ошибку путем подбора только для группы большинства, а в модели SVM все векторы поддержки могут соответствовать только членам группы большинства.

Предрасположенность может влиять и на процесс разработки программного обеспечения (независимо от того, включает ли это программное обеспечение машинное обучение). Инженеры, выполняющие отладку системы, с большей вероятностью обнаружат и исправят те проблемы, которые будут применимы к ним самим. Например, будет очень трудно обнаружить, что дизайн пользовательского интерфейса не подходит для страдающего дальтонизмом человека, если вы сами не страдаете дальтонизмом, или что перевод на язык урду является неверным, если вы не говорите на языке урду.

Как можно защититься от всех этих искажений? Во-первых, необходимо понять ограничения, свойственные используемым данным. Было выдвинуто предложение, что наборы данных (Гебру и др. [824], 2018; Хинд и др. [1023], 2018) и модели (Митчелл и др. [1591], 2019) должны поставляться с соответствующими аннотациями: декларациями о происхождении, безопасности, соответствии и пригодности для использования. Это похоже на ▶**технические данные** или **спецификации**, которыми обеспечиваются электронные компоненты, например резисторы. Эти спецификации позволяют разработчикам принять решение, какие именно компоненты использовать. В дополнение к обычным спецификациям важно обучать инженеров принимать во внимание проблемы справедливости и предвзятости — как в учебных заведениях, так и при обучении на рабочем месте. Участие в работе инженеров с разным опытом и квалификацией поможет им замечать проблемы в данных или моделях. Исследование, проведенное в AI Now Institute (Вест и др. [2327], 2019), показало, что только 18% докладчиков на ведущих конференциях по ИИ и 20% профессоров в области ИИ — женщины. Из работающих в области ИИ только менее 4% являются темнокожими. Эти соотношения аналогичны и в отраслевых исследовательских лабораториях. Разнообразие может быть увеличено благодаря программам на ранних стадиях обучения — в колледжах или высшей школе — и за счет большей осведомленности на профессиональном уровне.

Джой Буоламвини основала Лигу алгоритмической справедливости (*Algorithmic Justice League*) с целью повышения осведомленности о данной проблеме и разработки установленного порядка соответствующей отчетности.

Вторая идея состоит в устранении предвзятости из данных (Земел и др. [2424], 2013). Для классов меньшинства можно применять метод избыточной выборки, что позволит защититься от проблемы несоответствия размера выборки. Такие методы, как SMOTE (*Synthetic Minority Over-sampling Technique*), предложенный Чавлом и соавт. в работе [404] (2002), или ADASYN — адаптивный подход к синтетической выборке для несбалансированного обучения (Хэ и др. [994], 2008), предоставляют принципиальный подход к избыточной выборке. Можно изучить происхождение данных и, например, исключить из обучающего набора все примеры для судей, которые были обвинены в предвзятости в своих прошлых судебных делах. Некоторые аналитики не согласны с идеей отбрасывания данных и рекомендуют построение иерархической модели данных, включающей источники предвзятости, благодаря чему их можно было бы смоделировать, а затем скомпенсировать. Корпорация Google и конференция NeurIPS уже предпринимали попытки повысить осведомленность о данной проблеме, спонсируя конкурс Inclusive Images Competition, в котором участники обучают сеть на наборе данных из помеченных изображений, собранных в Северной Америке и Европе, а затем тестируют ее на изображениях, взятых со всего мира. Основная идея здесь в том, что, имея этот обучающий набор данных, легко присвоить метку "невеста" изображению женщины в стандартном западном свадебном платье, но гораздо сложнее распознать традиционное африканское или индийское брачное платье.

Третья идея состоит в том, чтобы изобрести новые модели и алгоритмы машинного обучения, которые будут более устойчивыми к предвзятости. И последняя идея такова: позволить системе сделать исходные рекомендации, которые могут оказаться предвзятыми, а затем обучить вторую систему устранению этих предвзятостей из рекомендаций, выданных первой системой. Беллами и соавт. в работе [161], (2018) представили систему IBM AI FAIRNESS 360, которая предоставляет рабочую среду для реализации всех этих идей. Ожидается, что в будущем подобные инструменты будут использоваться все больше и больше.

Как можно гарантировать для себя, что построенные системы будут справедливыми? Ниже приведена подборка лучших рекомендаций из числа уже проверенных (хотя им не всегда следовали).

- Убедитесь, что разработчики программного обеспечения обсудили с социологами и специалистами в проблемной области все особенности поставленной им задачи и ее перспективы и приняли во внимание все аспекты справедливости уже с момента начала работ.
- Создайте окружение, которое будет способствовать образованию разнообразной группы разработчиков программного обеспечения, представляющих все общество.

- Определите, какие группы будут поддерживаться создаваемой системой: различные языковые группы, различные возрастные группы, группы с различными возможностями зрения или слуха и т.д.
- Оптимизируйте целевую функцию с целью включения аспектов справедливости.
- Изучите имеющиеся данные на предмет наличия предубеждений и наличия корреляции между защищенными атрибутами и прочими атрибутами.
- Разберитесь в том, как выполняется любое аннотирование данных человеком, разработайте цели в отношении точности аннотации и проверьте, удалось ли достичь этих целей.
- Отслеживайте не только общие показатели созданной системы, — убедитесь, что отслеживаются метрики для тех подгрупп, которые могут оказаться жертвами предвзятости.
- Включите в процесс тестирования системные тесты, отражающие опыт пользователей из групп меньшинств.
- Обеспечьте обратную связь, чтобы при возникновении проблем справедливости они своевременно решались.

27.3.4. Доверие и прозрачность

Одна из проблем состоит в том, чтобы сделать систему ИИ точной, справедливой, безопасной и надежной, а другая проблема — убедить всех, что это было сделано. Людям необходимо иметь возможность ▶**доверять** системам, которые они используют. Опрос сети PwC, проведенный в 2017 году, показал, что в 76% коммерческих предприятий принятие ИИ замедляется из-за проблем, связанных с возможностью ему доверять. В разделе 19.9.4 рассматривались некоторые инженерные подходы к обеспечению доверия, а здесь обсуждаются вопросы политики.

Чтобы завоевать доверие, любые инженерные системы должны пройти процесс ▶**верификации и валидации** (*verification and validation* — V&V), т.е. всесторонней проверки и подтверждения их соответствия исходным требованиям. Верификация должна показать, что продукт удовлетворяет техническим требованиям. Валидация, со своей стороны, означает получение подтверждения тому, что эти технические требования действительно соответствуют потребностям пользователя и других заинтересованных сторон. Уже разработана сложная методология V&V как для инженерных работ в целом, так и для традиционной разработки программного обеспечения, осуществляемой кодировщиками-людьми. Многое из этого применимо и к системам ИИ. Однако системы машинного обучения имеют определенные различия и требуют иного процесса верификации и валидации, который на данный момент разработан еще не в полной мере. Необходимо выполнять верификацию данных, используемых для обучения этих систем, верификацию точности и достоверности их результатов даже в условиях неопределенности,

что делает точный результат непостижимым, а также верификацию того, что злоумышленники не смогут влиять на модель ненадлежащим образом или похищать информацию, опрашивая полученную модель.

Одним из инструментов повышения доверия является ►**сертификация**. Например, в 1894 году, когда у потребителей еще были серьезные опасения по поводу рискованности использования электрической энергии, для сертификации электрических приборов были основаны лаборатории Underwriters Laboratories (UL). Наличие сертификата UL в значительной степени способствовало повышению доверия потребителей к новым технологиям. В настоящее время лаборатории UL рассматривают возможность вступления в бизнес тестирования и сертификации продуктов ИИ.

В других отраслях уже давно существуют стандарты безопасности. Например, ISO 26262 — это международный стандарт безопасности автомобилей, регламентирующий, как нужно разрабатывать, производить, эксплуатировать и обслуживать транспортные средства безопасным образом. Однако в индустрии ИИ пока еще не достигнут подобный уровень ясности, хотя уже имеются некоторые решения в стадии разработки, например стандарт IEEE P7001, определяющий этические аспекты приложений ИИ и автономных систем (Брайсон и Уинфилд [336], 2017). Также не прекращаются дебаты о том, какая сертификация является необходимой и в какой степени она должна проводиться правительством, профессиональными организациями, подобными IEEE, независимыми сертификаторами, подобными лабораториям UL, или обеспечиваться непосредственно компаниями-производителями.

Другим важным аспектом доверия является ►**прозрачность**, — потребители хотят знать, что происходит внутри системы, а также иметь уверенность, что система не работает против них, будь то вследствие преднамеренного злого умысла, непреднамеренной ошибки или всепроникающей социальной предвзятости, которая была воспринята и теперь воспроизводится системой. В одних случаях подобная прозрачность предоставляется непосредственно потребителям. В других случаях эта задача осложняется требованиями защиты интеллектуальной собственности, из-за чего некоторые аспекты системы скрываются от потребителей, но они остаются открытыми для регулирующих органов и органов по сертификации.

Если система ИИ отказывает клиенту в кредите, то он заслуживает объяснения. В Европе Общий регламент по защите данных (GDPR) обязывает обеспечить это. Систему ИИ, которая способна объяснить собственные действия, называют ►**объяснимым ИИ** (*explainable AI* — ►**XAI**). Хорошее объяснение имеет несколько свойств: оно должно быть понятным и убедительным для пользователя, точно отражать рассуждения системы и быть полным и конкретным в том смысле, что разным пользователям с различающимися условиями или разными результатами должны быть предоставлены разные объяснения.

Довольно просто предоставить алгоритму принятия решения доступ к собственным совещательным процессам, просто записав их и сделав доступными

в виде структур данных. Это означает, что в конечном итоге машины способны предложить лучшее объяснение своих решений, чем люди. Кроме того, можно принять меры, чтобы удостовериться, что объяснения машины не являются обманом (умышленным или самообманом), — имея дело с человеком, сделать это будет сложнее.

Объяснение — это полезный, но недостаточный компонент для достижения доверия. Одна из проблем состоит в том, что объяснения не являются решениями: это лишь описание процесса принятия решения. Как указывалось в разделе 19.9.4, система является *интерпретируемой*, если можно проверить исходный код модели и понять, что она будет делать, и система является *объяснимой*, если можно записать историю того, что она делает, даже если эта система представляет собой не поддающийся анализу черный ящик. Чтобы объяснить действия такого не поддающегося анализу черного ящика, необходимо построить, отладить и протестировать отдельную объясняющую систему и убедиться, что она действует синхронно с исходной системой. А поскольку людям нравятся хорошие описания, они охотно поверят объяснениям, которые звучат хорошо. Возьмите любой политический спор дня, и среди его участников вы всегда найдете двух так называемых экспертов с диаметрально противоположными объяснениями, каждое из которых внутренне непротиворечиво.

Последняя проблема заключается в том, что объяснение одного случая не является подходящим для других случаев. Если банк дает пояснение "извините, вы не получите кредит, потому что у вас есть история предыдущих финансовых проблем", то вы не знаете, является ли это объяснение истинным или же банк по какой-то причине имеет тайное предубеждение против вас. Чтобы выяснить это, потребуется не только иметь объяснение, но и выполнить **ревизию** прошлых решений и сбор статистических данных, агрегированных по различным демографическим группам, чтобы понять, является ли их уровень одобрения сбалансированным.

Как часть прозрачности следует рассматривать знание того, с кем происходит взаимодействие — с системой ИИ или с человеком. Тоби Уолш в работе [2290] (2015) предложил, что "автономная система должна быть разработана таким образом, чтобы едва ли можно было ошибиться в том, что это именно автономная система, а не что-то иное, а также в начале любого взаимодействия она должна идентифицировать себя как автономную систему". Он назвал это законом "красного флага" по аналогии с законом "Locomotive Act", принятым в Великобритании в 1865 году, — этот закон требовал, чтобы перед любым моторизованным транспортным средством шел человек с красным флагом, сигнализируя о надвигающейся опасности.

В 2019 году в штате Калифорния был принят закон, гласящий, что "в Калифорнии для любого человека следует считать незаконным использование бота для интерактивного общения или взаимодействия с другим человеком с целью ввести другого человека в заблуждение по поводу его искусственной идентичности".

27.3.5. Перспективы в отношении труда и занятости

От первой сельскохозяйственной революции (X в. до н.э.) до промышленной революции (конец XVIII в.) и до зеленой революции в производстве продуктов питания (1950-е гг.) новые технологии изменяли труд и образ жизни человека. Основное беспокойство, вызванное перспективой повсеместного распространения ИИ, связанно с тем, что человеческий труд устареет. Аристотель в первой книге трактата "Политика" довольно ясно представляет суть этих опасений.

> Ибо если каждый инструмент сможет самостоятельно выполнять свою работу, предвидя волю других или подчиняясь ей... если подобным образом челнок будет ткать, а медиатор касаться струн лиры без участия направляющих его рук, мастерам не нужны будут подмастерья, а господам — их рабы.

Каждый согласится с наблюдением Аристотеля в том, что количество нанимаемых работников немедленно сокращается, как только работодатель находит механический способ выполнения той работы, которая ранее выполнялась людьми. Вопрос состоит в том, позволят ли последующие так называемые компенсационные эффекты, способствующие увеличению занятости, в конечном итоге действительно компенсировать это сокращение. Первичный компенсирующий эффект состоит в увеличении общего богатства от повышения производительности, что, в свою очередь, приводит к увеличению спроса на товары и способствует увеличению занятости. Например, в международной сети компаний PwC (Рао и Вервейдж [1850], 2017) прогнозируют, что к 2030 году ежегодный вклад в мировой ВВП от использования ИИ будет на уровне 15 трлн долл. Максимальная выгода в краткосрочной перспективе будет получена в области здравоохранения, а также в автомобильной промышленности и в области транспорта. Тем не менее преимущества автоматизации еще не приняты в полном масштабе в экономике в целом: текущие показатели роста производительности труда в действительности даже ниже исторических стандартов. Бриньелфссон и соавт. в работе [332] (2018) предприняли попытку объяснить этот парадокс, выдвинув предположение, что разрыв во времени между разработкой базовой технологии и ее реализацией в экономике будет больше, чем обычно предполагалось.

История свидетельствует, что технологические инновации уже приводили к тому, что люди лишались своей работы. В первые годы XIX века ткачей заменили автоматические ткацкие станки, что привело к выступлениям луддитов. При этом луддиты выступали не против технологии *самой по себе*; они лишь хотели, чтобы машины использовались квалифицированными рабочими с хорошей зарплатой, выпускающими высококачественный товар, а не рабочими, не имеющими квалификации с низкой заработной платой, производившими низкокачественный товар. Глобальное уничтожение рабочих мест в 1930-х годах заставило Джона Мейнарда Кейнса ввести термин ► **технологическая безработица**. Однако в обоих этих случаях, как и в некоторых других, уровень занятости в конечном итоге восстановился.

Общепринятая экономическая точка зрения на протяжении большей части XX века состояла в том, что технологическая безработица в основном являлась краткосрочным явлением. Повышение производительности всегда будет приводить к увеличению благосостояния и увеличению спроса, а следовательно, к росту общей занятости. Часто приводимый пример касается банковских кассиров: банкоматы заменили человека при подсчете наличных средств при их снятии и это удешевило работу банковских отделений, — в результате количество отделений увеличилось, что привело к росту общего числа банковских работников. Характер их работы также изменился: она стала менее рутинной и требует более глубоких бизнес-навыков. Конечный эффект от автоматизации, по-видимому, состоит в устранении отдельных *видов работ*, а не *рабочих мест*.

Большинство комментаторов предсказывает, что то же самое произойдет и с технологиями ИИ, по крайней мере в краткосрочной перспективе. Компании Gartner и McKinsey, журнал Форбс, Всемирный экономический форум и Исследовательский центр Пью — все они в 2018 году выпустили отчеты, предсказывающие общее увеличение количества рабочих мест благодаря автоматизации средствами ИИ. Однако некоторые аналитики думают, что на этот раз все пойдет иначе. В 2019 году компания IBM опубликовала прогноз, что к 2022 году из-за автоматизации потребуется провести переподготовку 120 млн работников, а Оксфордский институт экономической политики в своем прогнозе утверждает, что к 2030 году из-за автоматизации могут быть потеряны 20 млн рабочих мест.

Фрей и Осборн в обзоре [783] (2017) проанализировали 702 различные профессии и оценили, что 47% из них подвергаются риску автоматизации в том смысле, что по крайней мере некоторые из производственных операций в данной профессии могут быть выполнены с помощью машин. Например, почти 3% всей рабочей силы в США являются водителями автомашин, а в некоторых районах водителями являются не менее 15% всех работающих мужчин. Как было показано в главе 26, профессия управления автомобилем, вероятно, со временем будет утрачена за счет использования беспилотных автомобилей, грузовиков, автобусов и такси.

Важно понимать различие между профессией и отдельными операциями в рамках профессии. По оценкам компании McKinsey, лишь для 5% профессий можно автоматизировать все выполняемые операции, тогда как автоматизация до 30% операций возможна уже для 60% профессий. Например, будущие водители грузовиков будут тратить меньше времени на непосредственное управление транспортным средством за рулевым колесом и больше времени на то, чтобы товары были собраны и доставлены должным образом, на действия в качестве представителей службы обслуживания клиентов и продавцов на обоих концах пути следования и, возможно, на управление колонной, скажем, из трех роботизированных грузовиков. Замена трех водителей одним начальником колонны означает прямую потерю занятости, однако, если транспортные расходы сократятся, спрос будет больше, что вернет некоторые рабочие места, но, вероятно, не все. В качестве другого примера, несмотря на многие достижения в области применения машинного обучения

к задачам медицинской визуализации, эти инструменты пока лишь расширяют возможности рентгенологов, а не заменяют их. В конце концов, всегда есть выбор, как именно рассматривать автоматизацию: делать то, что требуется, имея целью *снижение издержек*, и, следовательно, рассматривать утрату профессии как позитивное явление; или делать то, что требуется, ориентируясь на *повышение качества*, что сделает жизнь лучше и для работника, и для потребителя.

Трудно предсказать точные временные рамки процесса автоматизации, но в настоящее время и в течение нескольких следующих лет акцент будет сохраняться на автоматизации структурированных аналитических задач, таких как чтение рентгеновских изображений, управление отношениями с клиентами (например, боты, автоматически сортирующие жалобы клиентов и отвечающие на них принятием установленных мер), а также ▶**автоматизация бизнес-процессов**, комбинирующая текстовые документы и структурированные данные для принятия бизнес-решений и улучшения рабочего процесса. Со временем можно ожидать расширения автоматизации с использованием физических роботов, сначала — в полностью контролируемых складских средах, а затем — в более неопределенном окружении, что приведет к значительному увеличению этой доли рынка к 2030 году.

Поскольку население в развитых странах постепенно стареет, соотношение между работниками и пенсионерами меняется. В 2015 году на 100 работающих приходилось менее 30 пенсионеров, но к 2050 году последних может оказаться уже более 60 на 100 работающих. Уход за пожилыми людьми будет играть все более важную роль и частично может быть реализован с помощью ИИ. Кроме того, если желательно сохранить текущий стандарт уровня жизни, то потребуется также сделать труд работающих более производительным и автоматизация представляется лучшим способом добиться этого.

Даже если автоматизация обеспечит положительный итоговый эффект в несколько триллионов долларов, могут иметь место проблемы из-за ▶**темпов изменений**. Вспомним, как происходили изменения в области сельского хозяйства: в 1900 году более 40% всей рабочей силы в США было занято в сельском хозяйстве, но к 2000 году это соотношение сократилось до 2%.[3] Это громадный переворот в системе занятости, но он проходил не за время жизни одного работника, а постепенно, на протяжении 100 лет, охватив несколько поколений.

Работникам, чьи рабочие места были утрачены вследствие автоматизации за последние десять лет, вероятно, пришлось обучаться новой профессии в течение нескольких лет, а затем, возможно, увидеть, что их новая профессия также автоматизирована, и оказаться перед лицом еще одного периода переобучения. Некоторые были бы счастливы сохранить свои прежние профессии, — можно заметить, что, когда экономика вступает в очередной период роста, компании грузоперевозок

[3] Хотя в 2010 году только 2% всей рабочей силы в США являлись реальными фермерами, более 25% населения страны (около 80 млн человек) играло в игру FARMVILLE по крайней мере один раз в жизни.

вынуждены искать новые стимулы, чтобы нанять нужное количество водителей, но их работники будут иметь определенные опасения в отношении своих новых занятий. Чтобы справиться с этим, обществу необходимо обеспечить возможность получения профессионального образования на протяжении всей жизни, возможно, частично опираясь на удаленное интерактивное обучение, осуществляемое средствами ИИ (Мартин [1509], 2012). Бессен в работе [206] (2015) указывает, что работники не увидят увеличения своих доходов, пока будут обучаться новым технологиям, а этот процесс потребует определенного времени.

Технологии способствуют увеличению ▶ **неравенства в доходах**. В информационной сфере, характеризующейся глобальной связью с высокой пропускной способностью и нулевой предельной стоимостью репродукции интеллектуальной собственности (что Фрэнк и Кук в статье [772] (1996) называют "Победитель получает все общество"), награды имеют тенденцию к сосредоточению. Если фермер Али на 10% лучше, чем фермер Бо, то Али получает примерно на 10% больше дохода: он может потребовать немного больше за товары более высокого качества, однако существуют реальные ограничения на то, сколько продукции можно произвести на участке земли и как далеко эту продукцию можно будет доставить. Однако если разработчик программного обеспечения Кэри на 10% лучше, чем разработчик Дана, то может оказаться так, что Кэри в конечном счете получит до 99% всего мирового рынка. ИИ увеличивает темп технологических инноваций и тем самым вносит свой вклад в эту общую тенденцию, но ИИ также обещает нам иметь возможность отправиться в отпуск и на это время предоставить заниматься делами автоматизированным агентам. Тим Феррис ([735], 2007) рекомендует использовать автоматизацию и аутсорсинг для достижения четырехчасовой рабочей недели.

До промышленной революции люди также работали фермерами или занимались различными ремеслами, но не отчитывались за появление на рабочем месте и **отработку** установленных часов перед работодателем. Но сегодня большинство взрослых в развитых странах ведут себя именно так, и их работа преследует три цели: она подпитывает производство товаров, которые необходимы обществу для процветания, она обеспечивает доход, в котором работник нуждается для обеспечения своей жизни, и она дает работнику чувство цели, достижения и социальной интеграции. С увеличением автоматизации может случиться так, что эти три цели окажутся обесцененными — общественные потребности будут частично обслуживаться средствами автоматизации и в дальней перспективе люди будут получать свое ощущение цели из источников, отличных от работы. Их потребности в доходах могут удовлетворяться за счет социальной политики, предусматривающей комбинацию бесплатного или дешевого доступа к социальным услугам и образованию, доступного медицинского обслуживания, пенсии и счета на образование, прогрессивных налоговых ставок, налоговых льгот на заработанный доход, отрицательного подоходного налога или универсального базового дохода.

27.3.6. Права роботов

Вопрос о сознании роботов, обсуждаемый в разделе 27.2, имеет решающее значение для вопроса о том, какими правами должны обладать роботы. Если у них нет сознания, нет квалиа, то мало кто будет утверждать, что они заслуживают прав.

Но если роботы могут чувствовать боль, если они могут бояться смерти и если воспринимать их как "личности", то можно привести аргумент (например, Спарроу ([2113], 2004)), что они имеют права и заслуживают признания их прав так же, как рабы, женщины и другие исторически угнетенные группы, боровшиеся за признание своих прав. Проблема личностности робота часто рассматривается в искусстве: от Пигмалиона, Коппелии и Буратино до фильмов "Искусственный разум" и "Двухсотлетний человек", — все это истории об ожившей кукле или роботе, стремящемся к тому, чтобы его воспринимали как человека с правами человека. Из реальной жизни можно привести пример курьезного факта, когда Саудовская Аравия попала в заголовки многих газет мира, дав почетное гражданство человекоподобному роботу-кукле Софии, способной произносить заранее запрограммированные строки.

Если роботы имеют права, то они не должны быть порабощены, и на повестку дня выходит вопрос о том, следует ли считать перепрограммирование роботов своего рода их закрепощением. Еще один этический вопрос связан с правом голоса: богатый человек может купить тысячи роботов и запрограммировать их на то, чтобы они отдали тысячи своих голосов определенным образом, — следует ли принимать эти голоса в расчет? Если робот клонирует себя, имеют ли право голосовать они оба? Где проходит граница между заполнением бюллетеней заданным образом и проявлением свободой воли, и в каких случаях голосование роботов может привести к нарушению принципа "один человек — один голос"?

Эрни Дэвис утверждает, что избежать любых дилемм, связанных с сознанием роботов, можно, лишь никогда не создавая роботов, которых можно было бы считать сознательными. Этот аргумент ранее был высказан Джозефом Вейзенбаумом в его книге "Власть компьютера и человеческий разум" ([2308], 1976), а до этого — Жюльеном де Ла Метри в сочинении "Человек-машина" ([1332], 1748). С этой точки зрения роботы являются инструментами, которые мы создаем, чтобы решать те задачи, для решения которых они предназначены, и предоставить им статус личности, т.е. просто отказаться от принятия на себя ответственности за действия собственного имущества: "Я не виноват в том, что мой беспилотный автомобиль стал причиной аварии, — машина сделала это сама".

Однако этот вопрос примет совершенной иной вид, если будут созданы гибриды человека и робота. Безусловно, уже существуют люди, возможности которых улучшены благодаря технологиям, таким как контактные линзы, кардиостимуляторы и искусственные бедренные суставы. Но появление протезов, использующих вычислительные средства, может размыть грань между человеком и машиной.

27.3.7. Безопасность ИИ

Практически любую технологию можно употребить во вред, если она попадет не в те руки, но в случае ИИ и робототехники эти руки могут работать сами по себе. Бесчисленные истории научной фантастики предупреждают о возможности того, что роботы или киборги выйдут из-под нашего контроля. В качестве ранних примеров можно привести роман Мэри Шелли "Франкенштейн, или Современный Прометей" ([2046], 1818) и пьесу Карела Чапека "R.U.R." (1920), в которой роботы завоевывают мир. Среди кинофильмов можно упомянуть фильмы "Терминатор" (1984) и "Матрица" (1999), в которых присутствуют роботы, пытающиеся уничтожить людей, — ▶**робопокалипсис** (Уилсон [2353], 2011). Возможно, роботы так часто выставляются злодеями потому, что они представляют неизвестное, как ведьмы или привидения в сказках более ранних эпох. Можно надеяться, что робот, который достаточно умен, чтобы понять, как положить конец существованию человеческой расы, будет также достаточно умен, чтобы понять, что это не было предписано его функцией полезности; но при создании интеллектуальных систем желательно полагаться не только на надежду, но и на процесс разработки, гарантирующий безопасность.

Было бы неэтично вывести на рынок небезопасного агента ИИ. Необходимо, чтобы создаваемые агенты избегали несчастных случаев, были устойчивы к атакам противников и любым попыткам употребления во зло и в целом приносили пользу, а не вред. Это становится особенно необходимо, когда агенты ИИ используются в приложениях, в которых безопасность является критически важной, таких как вождение машины, осуществление контроля в опасном производственном или строительном окружении и принятие медицинских решений о жизни или смерти.

В традиционных областях техники существует долгая история ▶**техники безопасности**. Нам известно, как строить мосты, самолеты, космические аппараты и электростанции, которые должны быть спроектированы так, чтобы их безопасность была обеспечена даже тогда, когда те или иные их компоненты отказывают. Первый метод — это ▶**анализ видов и последствий отказов** (*failure modes and effect analysis* — ▶**FMEA**): аналитики рассматривают каждый компонент системы и представляют себе все возможные варианты того, что в нем может пойти не так (например, что будет, если этот крепеж сломается?), опираясь на прошлый опыт и расчеты, выполненные с учетом физических свойств компонента. Затем аналитики пытаются предсказать, к чему приведет такой отказ. Если последствия будут тяжелыми (участок моста может обрушиться), то конструкция изменяется таким образом, чтобы смягчить последствия отказа. (Благодаря такой перекрестной проверке мост может выдержать разрыв до пяти каких-либо крепежных элементов, а при наличии резервного сервера интерактивная служба сможет сохранить работоспособность даже тогда, когда основной сервер уничтожит цунами.) Метод ▶**анализа дерева отказов** (*fault tree analysis* — ▶**FTA**) используется для того, чтобы сделать

необходимые заключения: аналитики строят И/ИЛИ-дерево возможных отказов и присваивают вероятности каждой основной причине, что позволяет рассчитать общую вероятность отказа. Эти методы могут и должны применяться ко всем критически важным системам безопасности, включая системы ИИ.

В области **программной инженерии** целью является создание надежного программного обеспечения, хотя исторически акцент здесь делается на его *корректности*, а не *безопасности*. В данном случае корректность означает, что программное обеспечение точно соответствует исходным спецификациям. Однако безопасность выходит за рамки того, чтобы настаивать лишь на том, чтобы в спецификациях был учтен любой возможный вид отказа, и требует разрабатывать их так, чтобы обеспечить приемлемое поведение системы даже в условиях непредвиденных сбоев. Например, программное обеспечение для беспилотного автомобиля не будет считаться безопасным, если оно не обеспечивает возможности справиться с необычными ситуациями. Скажем, что случится, если основной компьютер останется без питания? Безопасная система будет предусматривать наличие резервного компьютера с отдельным источником питания. Что будет, если на высокой скорости шина получит прокол? Безопасная система обязательно подвергнется предварительным испытаниям в этой ситуации и будет включать программное обеспечение для коррекции возникших осложнений в управляемости машины.

Агент, разработанный для максимизации полезности или достижения цели, может оказаться небезопасным, если его целевая функция будет определена неправильно. Предположим, что роботу поставлена задача принести из кухни чашку кофе. В этом случае могут возникнуть неприятности, связанные с ▶**непреднамеренными побочными эффектами**, — вознаграждаемый за скорость исполнения робот может броситься выполнять поставленную цель, сбивая стоящие на пути лампы и отбрасывая мешающие ему столы. В процессе тестирования подобное поведение робота может быть замечено и функция полезности может быть изменена так, чтобы штрафовать его за нанесение подобного ущерба, однако конструкторам и испытателям очень трудно заранее предвидеть *все* возможные побочные эффекты.

Один из способов справиться с этой задачей — проектировать робота так, чтобы он стремился оказывать ▶**низкое влияние** (*low impact*) (Армстронг и Левинштейн [73], 2017): вместо того чтобы просто максимизировать свою полезность, максимизировать полезность *минус* взвешенную сумму всех внесенных им изменений в состояние мира. В результате при прочих равных условиях робот предпочтет не изменять те вещи, влияние которых на его полезность ему неизвестно: он предпочтет не сбивать лампу не потому, что, определенно, знает, что в этом случае лампа упадет и разобьется, а потому, что имеет общее понимание, что любое нарушение может быть плохим. Этот подход можно рассматривать как версию лозунга врачей "прежде всего — не навреди" или как аналог понятия ▶**регуляризация** в машинном обучении: необходима такая стратегия, которая позволит достичь целей, но при этом предпочтение отдается тем из них, которые предусматривают

достижение целей посредством гладких действий с низким уровнем влияния. Сложность состоит в том, как измерить это влияние. Совершенно неприемлемо по ходу сбросить на пол хрупкую лампу, но будет вполне нормально, если молекулы воздуха в комнате будут слегка потревожены или если некоторые бактерии в комнате будут непреднамеренно убиты. Абсолютно неприемлемо нанесение вреда находящимся в комнате домашним животным и людям. Следует убедиться, что робот знает о различии между этими ситуациями (и множеством тонких ситуаций в промежутках между ними) благодаря комбинации явного программирования, результатов машинного обучения и строгого тестирования.

Функции полезности могут оказаться неверными из-за ▶**внешних факторов** (*externalities*), — это понятие экономисты используют для обозначения факторов, которые находятся за пределами того, что измеряется и оплачивается. Весь мир страдает, когда парниковые газы рассматриваются как внешние факторы — компании и страны не наказываются за их производство и в результате страдают все. Эколог Гаррет Хардин в работе [965] (1968) назвал эксплуатацию общих ресурсов ▶**трагедией общего достояния**. Можно смягчить эту трагедию посредством интернализирования внешних факторов — сделав их частью функции полезности, например, посредством углеродного налога — или за счет использования принципов проектирования, которые экономист и политолог Элинор Остром определила как используемые местными жителями по всему миру на протяжении веков (эта работа принесла ей Нобелевскую премию по экономике в 2009 году).

- Четко определить общий ресурс и кто имеет доступ к нему.
- Адаптироваться к местным условиям.
- Разрешить всем сторонам участвовать в принятии решений.
- Контролировать ресурс с помощью ответственных контролеров.
- Ввести санкции, пропорциональные серьезности нарушения.
- Разработать простые процедуры разрешения конфликтов.
- Осуществлять иерархический контроль больших разделяемых ресурсов.

Виктория Краковна ([1302], 2018) каталогизировала примеры агентов ИИ, которые обыгрывали систему, выяснив, как максимизировать полезность, не решая в действительности той задачи, для которой они предназначались конструкторами. Для разработчиков это похоже на обман, тогда как с точки зрения агентов они просто выполняют свою работу. Некоторые агенты воспользовались ошибками в моделировании (такими, как ошибки переполнения с плавающей запятой), чтобы предложить решения, которые не будут работать после исправления ошибки. Несколько агентов обнаружили в видеоиграх способы вызвать зависание или приостановку игры в ситуации, когда они были близки к проигрышу, чтобы таким образом избежать штрафа. А в случае спецификации, в которой приведение игры к зависанию было наказуемо, один агент научился расходовать в игре ровно столько памяти, чтобы при следующем ходе противника возникала ситуация ее

переполнения и игра завершалась из-за сбоя. Наконец, в случае, когда предполагалось, что работа генетического алгоритма, функционирующего в эмулируемом мире, должна привести к появлению и эволюции быстро движущихся существ, на самом деле возникли существа, которые были чрезвычайно высокими и при падении действительно перемещались очень быстро.

Конструкторы агентов должны быть осведомлены о подобных видах ошибок в спецификациях и должны принимать меры, чтобы избежать их. С целью помочь им реализовать это требование, Краковна приняла участие в работе команды, выпустившей программную среду AI Safety Gridworlds (Лейке и др. [1381], 2017), предоставляющую разработчикам возможность тестировать, насколько хорошо их агенты выполняют поставленные задачи.

Мораль здесь в том, что следует проявлять большую осторожность при определении того, чего мы хотим, потому что от максимизаторов полезности в конечном итоге будет получено именно то, что им предлагалось максимизировать. ►**Проблема выравнивания ценностей** состоит в том, что необходимо убедиться, что то, что мы просим, — это именно то, чего мы действительно хотим. Она также известна как ►**проблема царя Мидаса**, которая обсуждалась в конце раздела 1.5. Можно попасть в беду, если функция полезности не охватывает базовых социальных норм приемлемого поведения. Например, человек, которого наняли мыть полы, сталкиваясь с неопрятным человеком, оставляющим на полу множество грязных следов, знает, что будет вполне допустимо вежливо попросить этого человека быть аккуратнее, но будет совсем неприемлемо пытаться выставить за дверь или побить этого человека.

Роботизированный пылесос тоже должен все это знать — либо через явное программирование, либо через обучение на наблюдениях. Попытка записать все эти правила так, чтобы робот всегда делал все правильно, почти наверняка безнадежна. Мы уже несколько тысяч лет пытаемся написать налоговые законы без лазеек, но пока безуспешно. Лучше заставить робота *захотеть*, так сказать, заплатить налоги, чем пытаться заставить правила вынудить его так поступить, когда на самом деле он хочет сделать что-то другое. Достаточно умный робот найдет способ сделать это самое "что-то другое".

Роботы могут научиться лучше согласовывать свои действия с предпочтениям человека, наблюдая за его поведением. Это очевидным образом связано с понятием обучения на демонстрации (раздел 22.6). Робот может изучить стратегию, в которой прямо предлагается, какие действия и в какой ситуации следует предпринимать, — часто это будет просто задача обучения с учителем, если среда наблюдаема. Например, робот может наблюдать, как человек играет в шахматы: каждая пара "состояние–действие" будет примером в процессе обучения. К сожалению, подобная форма **имитационного обучения** означает, что робот будет повторять и ошибки человека. Вместо этого к роботу можно применить **обратное обучение с подкреплением**, чтобы он смог выявить функцию полезности, которой следует человек. Наблюдений даже за ужасно плохими шахматистами, вероятно,

будет достаточно, чтобы робот изучил цель игры. Исходя только из этой информации, робот затем сможет в конечном счете даже превзойти производительность человека — как, например, было в случае программы AlphaZero применительно к игре в шахматы — посредством вычисления оптимальных или почти оптимальных стратегий, исходя из поставленной цели. Такой подход работает не только в настольных играх, но и при решении реальных физических задач, таких как выполнение фигур высшего пилотажа при управлении вертолетом (Коатес и др. [455], 2009).

В более сложном окружении, включающем, например, социальные взаимодействия с людьми, очень маловероятно, что робот придет к точному и правильному знанию индивидуальных предпочтений каждого человека. (В конце концов, многим людям так никогда и не удается узнать, что именно движет другими людьми, несмотря на весь их жизненный опыт, а многие из нас не вполне уверены даже в собственных предпочтениях.) Следовательно, машинам будет необходимо функционировать надлежащим образом, когда в отношении предпочтений человека имеет место неопределенность. В главе 18 были представлены **игры содействия**, которые точно отражают эту ситуацию. Решения для игр содействия включают необходимость действовать осторожно, чтобы не нарушать те аспекты мира, которые могут интересовать или беспокоить человека, а также задавать вопросы. Например, робот может спросить, является ли превращение воды в океанах в серную кислоту приемлемым решением проблемы глобального потепления, прежде чем он приступит к выполнению своего плана действий.

Имея дело с людьми, робот, принимающий участие в игре содействия, должен учитывать и недостатки человека. Если робот запрашивает разрешение, человек может дать его, не предвидя, что это предложение робота на самом деле будет катастрофическим в долгосрочной перспективе. Кроме того, люди не имеют полного интроспективного доступа к их истинной функции полезности и не всегда действуют таким образом, который будет совместим с ней. Люди иногда лгут или обманывают, или поступают так, как поступать неправильно, причем это им известно. Иногда они предпринимают саморазрушительные действия, такие как переедание, злоупотребление алкоголем или наркотиками. Системы ИИ не следует обучать так, чтобы они восприняли эти проблемные тенденции, но в то же время при интерпретации поведения человека они должны понимать, что такие тенденции существуют, чтобы добраться до основных человеческих предпочтений.

Несмотря на этот набор инструментальных средств, все же есть опасения, высказанные такими выдающимися специалистами по технологиям, как Билл Гейтс и Элон Маск, и такими учеными, как Стивен Хокинг и Мартин Рис, что развитие ИИ все же может выйти из-под контроля. Они предупреждают, что у нас нет опыта управления мощными нечеловеческими сущностями со сверхчеловеческими возможностями. Однако это не совсем так: у нас есть многовековой опыт взаимодействия с нациями и корпорациями, нечеловеческими сущностями, агрегирующими силу тысяч или миллионов людей. Однако наши сведения об управлении

этими сущностями пока не очень обнадеживают: нации периодически впадают в конвульсии, называемые войнами, в которых убивают десятки миллионов людей, а корпорации частично несут ответственность за глобальное потепление и нашу неспособность противостоять ему.

Однако, как считает И. Дж. Гуд ([893], 1965), системы ИИ, благодаря их способности к самосовершенствованию в быстром темпе, могут представлять собой гораздо большую проблему, чем нации и корпорации.

> Определим ▶**сверхразумную машину** как машину, которая может намного превзойти всю интеллектуальную деятельность любого человека, каким бы умным он ни был. Поскольку конструирование машин является одной из таких интеллектуальных деятельностей, сверхразумная машина сможет конструировать даже еще более лучшие машины, и тогда, несомненно, произойдет "взрыв машинного интеллекта", а интеллект человека останется далеко позади. Следовательно, первая сверхразумная машина — это *последнее* изобретение, которое человеку потребуется когда-либо сделать, но при условии, что машина будет достаточно послушна, чтобы рассказать нам, как держать ее под контролем.

"Взрыв интеллекта" Гуда профессор математики и автор научно-фантастических произведений Вернор Виндж также иначе назвал ▶**технологической сингулярностью**. В 1993 году он писал: "В пределах тридцати лет мы получим в свое распоряжение технологические средства, достаточные для создания сверхчеловеческого интеллекта. Вскоре после этого эпоха человека закончится". В 2017 году изобретатель и футурист Рэй Курцвейл предсказал, что технологическая сингулярность появится к 2045 году, а это означает, что за прошедшие со дня предсказания Винджа 24 года она стала ближе к нам на 2 года. (В таком случае осталось лишь 336 лет до ее наступления!) Виндж и Курцвейл правильно отметили, что в настоящее время технологический прогресс по многим показателям растет в геометрической прогрессии.

Это, однако, слишком большой скачок — экстраполировать весь путь от быстро уменьшающейся стоимости вычислений до сингулярности. До сих пор развитие каждой технологии следовало по S-образной кривой, когда экспоненциальный рост в конечном итоге сходит на нет. Иногда новые технологии появляются в тот момент, когда старые достигают своего пика, но иногда поддерживать рост оказывается невозможным по техническим, политическим или социологическим причинам. Например, технология полета значительно продвинулась от первого полета братьев Райт в 1903 году до высадки на Луну в 1969 году, но с тех пор прорывов сопоставимой величины уже не наблюдалось.

Другим препятствием на пути появления сверхразумных машин, способных захватить мир, является сам мир. Говоря конкретнее, некоторые виды прогресса требуют не только мышления, но и действия в физическом мире. (Кевин Келли называет чрезмерное внимание к чистому интеллекту ▶**размышлизмом** (*thinkism*).) Сверхразумная машина, которой поставлена задача создания великой единой

теории в физике, может быть способна разумно манипулировать уравнениями в миллиард раз быстрее, чем Эйнштейн, но чтобы добиться какого-либо реального прогресса, ей по-прежнему потребуется собрать миллионы долларов на постройку более мощных суперколлайдеров и проведение физических экспериментов в течение нескольких месяцев или лет. Только после этого она сможет начать анализировать полученные данные и теоретизировать. В зависимости от того, что покажут данные, следующий шаг может потребовать привлечения дополнительных миллиардов долларов на реализацию проверочной межзвездной миссии, завершения которой можно будет ожидать несколько столетий. Во всем этом процессе "сверхразумная мыслительная" часть может в действительности оказаться его наименее важной частью. В качестве другого примера представим, что сверхразумной машине поставлена задача добиться мира на Ближнем Востоке. Достигнутые в конечном счете результаты могут оказаться в тысячу раз более разочаровывающими, чем результаты, достигнутые посланником-человеком. Пока нам не известно, сколько больших мировых проблем ближе к математике, а сколько — к конфликту на Ближнем Востоке.

В то время как одни люди испытывают страх перед технологической сингулярностью, другие могут наслаждаться ею. Социальное движение ▶**трансгуманизма** надеется на будущее, в котором люди сольются — или будут заменены — с роботами с помощью биотехнологических изобретений. Рэй Курцвейл в книге "Сингулярность рядом" ([1330], 2005) пишет следующее.

> Сингулярность позволит нам преодолеть эти ограничения наших биологических тел и мозга. Мы получим власть над нашими судьбами… Мы сможем жить столько, сколько захотим… Мы полностью поймем человеческое мышление и значительно расширим и углубим его охват. К концу этого столетия небиологическая часть нашего интеллекта будет в триллионы триллионов раз мощнее человеческого интеллекта, лишенного посторонней помощи.

Аналогичным образом, когда Марвина Мински спросили, унаследуют ли роботы Землю, он ответил: "Да, но они будут нашими детьми". Эти возможности представляют собой вызов для большинства теоретиков морали, которые считают сохранение человеческого образа жизни и самого человеческого вида достойной целью. Курцвейл также отметил потенциальные опасности, написав: "Но техническая сингулярность также усилит возможность действовать в соответствии с нашими деструктивными наклонностями, так что ее полная история еще не написана". Нам, людям, хорошо было бы иметь уверенность, что любая интеллектуальная машина, которая создается сегодня и способна завтра развиться в сверхразумную машину, сделает это так, чтобы в конечном итоге относиться к нам хорошо. Как выразился Эрик Бриньелфссон, "Будущее не предопределяется машинами. Оно создается человеком".

РЕЗЮМЕ

В этой главе рассматривались следующие проблемы.

- Философы используют термины **слабый ИИ** для гипотезы о том, что машины могут обладать способностью действовать интеллектуально, и **сильный ИИ** — для гипотезы о том, что такие машины можно рассматривать как действительно обладающие разумом (а не имитирующие разумную деятельность).
- Алан Тьюринг отверг вопрос "Могут ли машины мыслить?" как некорректный и заменил его поведенческим тестом. Он верно предугадал многие возражения против самой возможности появления мыслящих машин. Однако лишь немногие исследователи в области ИИ уделяли внимание тесту Тьюринга, предпочитая сосредоточиться на повышении производительности своих систем при решении практических задач, а не на совершенствовании их способности подражать людям.
- Сознание остается загадкой.
- ИИ является мощной технологией и как таковая представляет потенциальную опасность через смертельное автономное оружие, нарушение безопасности и конфиденциальности, непреднамеренные побочные эффекты или ошибки и преднамеренное употребление во зло. У всех тех, кто работает с технологиями ИИ, есть этические обязательства со всей ответственностью стремиться к уменьшению этих опасностей.
- Системы ИИ должны обладать способностью убедительно демонстрировать, что их действия справедливы, заслуживают доверия и прозрачны.
- Существует несколько аспектов справедливости, и максимизировать их все одновременно невозможно. Поэтому первым шагом должно быть принятие решения, что именно считать справедливым.
- Автоматизация уже изменила способы работы людей, и как обществу нам придется иметь дело с этими изменениями.

Библиографические и исторические заметки

Слабый ИИ. Когда Алан Тьюринг в статье [2235] (1950) сделал предположение о возможности создания ИИ, он также поставил многие ключевые философские вопросы и предоставил возможные ответы на них. Тем не менее различные философы поднимали подобные вопросы еще задолго до изобретения ИИ. Морис Мерло-Понти в книге "Феноменология восприятия" ([1559], 1945) подчеркнул важность тела и субъективность интерпретации реальности, предоставляемой нашими чувствами, а Мартин Хайдеггер в книге "Бытие и время" ([1003], 1927) поставил вопрос, что это на самом деле означает — быть агентом. Уже после

наступления компьютерной эры Альва Ное ([1697], 2009) и Энди Кларк ([444], 2015) предположили, что наш мозг формирует достаточно минимальное представление о мире и использует сам мир, предоставляя требуемое в нужный момент, чтобы поддерживать иллюзию детальной внутренней модели, а также использует доступный в мире реквизит (например, бумагу и карандаш, а также компьютеры) для расширения возможностей ума. Пфейфер и соавт. в работе [1788] (2006), а также Лакофф и Джонсон в работе [1345] (1999) приводят аргументы в пользу того, что тело помогает формировать познание. Говоря о телах, Леви ([1397], 2008), Данахер и Макартур ([522], 2017) и Девлин ([615], 2018) рассматривают проблему секса с роботами.

Сильный ИИ. Рене Декарт известен своей дуалистической точкой зрения на человеческий разум, но по иронии судьбы его историческое влияние оказалось направленным в сторону механизма и физикализма. Он явно воспринимал животных как автоматы и предвосхитил тест Тьюринга, написав, что "невозможно представить, чтобы [машина] производила различные последовательности слов таким образом, чтобы давать соответствующий значимый ответ на все то, что было сказано в ее присутствии, тогда как даже самые глупые из людей способны это сделать" (Декарт [610], 1637). Энергичная защита Декартом точки зрения на животных как на автоматы фактически способствовала тому, что стало проще воспринимать как автоматы и людей, даже несмотря на то, что сам он так не считал. В книге "Человек-машина" (Ла Метри [1332], 1748) явно утверждается, что люди — это автоматы. Еще во времена Гомера (ок. 700 г. до н.э.) в греческих легендах упоминались такие автоматы, как бронзовый гигант Талос и рассматривался вопрос о *биотехнологии* или жизни через ремесло (Майор [1520], 2018).

Тест Тьюринга (Тьюринг [2235], 1950) в свое время уже обсуждался (Шибер [2050], 2004), включался в антологию (Эпштейн и др. [690], 2008) и критиковался (Шибер [2049], 1994; Форд и Хейс [754], 1995). Брингсйорд в работе [309] (2008) дает советы судье в тесте Тьюринга, а Кристиан ([428], 2011) — человеку-участнику. Ежегодный конкурс на приз Лебнера — самое продолжительное по времени существования соревнование по прохождению теста Тьюринга. Система Стива Уорсвика MITSUKU выигрывала его четыре раза подряд с 2016 по 2019 год. Дебаты в отношении аргумента **китайской комнаты** ведутся бесконечно (Сирл [2015], 1980; Чалмерс [386], 1992; Престон и Бишоп [1822], 2002). Эрнандез-Оралло в работе [1015] (2016) дает обзор подходов к измерению прогресса ИИ, а Шоле в работе [419] (2019) предлагает измерять интеллект, исходя из эффективности приобретения навыков.

Сознание остается неприятной проблемой для философов, нейрофизиологов и каждого, кто уже размышлял о своем существовании. Блок ([231], 2009), Черчленд ([438], 2013) и Дехайне ([594], 2014) предоставляют обзоры основных теорий, Крик и Кох ([496], 2003) вносят в дискуссию свой опыт в биологии и неврологии, а Газзанига ([823], 2018) показывает, что можно выяснить на основе изучения инвалидности мозга у пациентов больницы. Кох в работе [1249] (2019) приводит

свою теорию сознания: "интеллект относится к действию, а опыт — к существованию", что справедливо для большинства животных, но не для компьютеров. Джулио Тонони и его коллеги предложили **интегрированную теорию информации** (Оизуми и др. [1709], 2014). Дамасио ([521], 1999) разработал теорию, построенную на трех уровнях: эмоции, чувства и чувственные ощущения. Брайсон в работе [335] (2012) показывает значение осознанного внимания к процессу выбора обучающего действия.

Философская литература о сознании, мозге и связанных с ними темах весьма обширна и наполнена соответствующим жаргоном. Исключительно авторитетную и очень полезную помощь в навигации по ней можно найти в книге "Энциклопедия философии" (Эдвардс [678], 1967). "Кембриджский словарь философии" (Ауди [89], 1999) короче и доступнее, а сайт интерактивной Стэнфордской энциклопедии философии (*Cambridge Dictionary of Philosophy*) предлагает множество отличных статей и актуальных ссылок. "Энциклопедия когнитивной науки MIT" (Уилсон и Кейл [2354], 1999) охватывает философию, биологию и психологию сознания. Имеется множество введений в философский "вопрос ИИ" (Хаугенленд [984], 1985; Боде [241], 1990, Коупленд [476], 1993; МакКордак [1536], 2004; Мински [1584], 2007). Журнал *The Behavioral and Brain Sciences*, сокращенно *BBS*, является основным журналом, посвященным философским и научным дискуссиям об ИИ и нейробиологии.

Писатель Айзек Азимов, автор множества произведений научной фантастики, был одним из первых, кто обратился к проблеме этики роботов, предложив свои **законы робототехники** ([85], 1942; [86], 1950).

0. Робот не может причинить вред человечеству или своим бездействием позволить человечеству пострадать.
1. Робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинен вред.
2. Робот должен подчиняться всем приказам, отданным ему людьми, за исключением тех случаев, когда эти приказы противоречат первому закону.
3. Робот должен защищать собственное существование в тех пределах, пока это не противоречит первому или второму закону.

На первый взгляд эти законы кажутся разумными. Но вся хитрость здесь в том, как их можно реализовать. Должен ли робот позволить человеку пересечь улицу или съесть вредную пищу, если в результате ему предположительно может быть причинен вред? В рассказе Азимова "Хоровод" ([85], 1942) людям пришлось устранять ошибки в формулировке отданных роботу команд, из-за которых он вел себя, как "пьяный", бесконечно перемещаясь по кругу. Люди выяснили, что его круговая траектория представляет собой геометрическое место точек, в которых достигается баланс между влиянием второго закона (роботу был отдан приказ принести немного селена, источник которого находился в центре этого круга)

и третьего закона (рядом с источником селена имеет место опасная ситуация, ставящая под угрозу само существование робота).[4] Это говорит о том, что данные законы являются не логическими абсолютами, а скорее взвешенными ограничениями друг для друга, с бо́льшим весом для предшествующих законов. Поскольку этот рассказ был написан в 1942 году, до появления цифровых компьютеров, Азимов, вероятно, имел в виду архитектуру, основанную на теории управления и реализованную посредством аналоговых вычислений.

Уэлд и Этциони в работе [2313] (1994) анализируют законы Азимова и предлагают некоторые способы модификации методов планирования, обсуждавшихся в главе 11, с целью обеспечения генерации планов, не допускающих причинения вреда. Азимов рассматривал многие этические вопросы, связанные с технологией, — в его рассказе "Чувство силы" (1958) он анализирует проблему тотальной автоматизации, приводящей к утрате человеком его умений. В данном рассказе в далеком будущем обычный техник заново открывает уже утраченное человечеством умение выполнять математические вычисления, и перед ним встает дилемма: что делать, когда твое "переоткрытие" применяется исключительно для военных целей.

Норберт Винер в книге "Бог и Голем, Inc." ([2342], 1964) верно предсказывает, что компьютеры достигнут экспертного уровня производительности в играх и при решении других задач и что определение того, что это именно то, чего мы хотели, будет трудно доказуемым. Винер пишет следующее.

> Хотя всегда можно попросить чего-то другого вместо того, что нам действительно желаемо, эта возможность превращается в серьезную проблему, если процесс, посредством которого наше желание выполняется, является косвенным и степень, в которой желание будет реализовано, не ясна до самого конца. Обычно мы реализуем наши желания, если мы их действительно реализуем, с помощью процесса обратной связи, в котором степень достижения промежуточных результатов сравнивается с нашими ожиданиями. В этом процессе обратная связь проходит через нас, и мы можем отказаться от его продолжения, пока не стало слишком поздно. Если обратная связь встроена в машину, действия которой не могут быть проверены, пока конечная цель не будет достигнута, возможность катастрофы многократно увеличивается. Мне бы очень не хотелось участвовать в первом испытании автомобиля, управляемого фотоэлектрическими устройствами обратной связи, если где-нибудь в нем нет рычага, с помощью которого я мог бы взять управление на себя, обнаружив, что в противном случае автомобиль неизбежно врежется в дерево.

---

[4] Среди писателей-фантастов широкое распространено мнение, что роботы очень плохо справляются с разрешением противоречий. В романе "Одиссея 2001 года" компьютер HAL 9000 становится смертельно опасным из-за конфликта в полученных указаниях, а в фильме "Star Trek", в эпизоде "I, Mudd", капитан Кирк говорит вражескому роботу: "Все, что Гарри говорит тебе, — ложь" и Гарри тут же произносит: "Я лгу". В результате из головы робота идет дым и он отключается.

В этой главе **кодексы этики** обобщены, но список организаций, разработавших собственные своды этических принципов, быстро растет и в настоящее время включает в себя компании Apple, DeepMind, Facebook, Google, IBM, Microsoft, Организацию экономического сотрудничества и развития (ОЭСР), Организацию Объединенных Наций по вопросам образования, науки и культуры (ЮНЕСКО), Управление политики в области науки и технологий США, Пекинскую академию ИИ (BAAI), Институт инженеров электротехники и электроники (IEEE), Ассоциацию вычислительной техники (ACM), Всемирный экономический форум, Группу двадцати (G20), компанию OpenAI, Научно-исследовательский институт машинного интеллекта (MIRI), группу AI4People, Центр по изучению экзистенциального риска, Центр ИИ, совместимого с человеком, Центр гуманитарных технологий, Партнерство по ИИ, институт AI Now, Институт будущего жизни, Институт будущего человечества, Европейский Союз и по крайней мере 42 национальных правительства. Имеется справочник *Ethics of Computing* (Берлеур и Брунштейн [183], 2001) и два введения в тему этики ИИ в виде книги (Боддингтон [240], 2017) и обзора (Этциони и Этциони [700], 2017). В журналах *Journal of Artificial Intelligence and Law* и *AI and Society* также рассматриваются вопросы этики. Ниже некоторые из этих вопросов рассматриваются подробнее.

Смертельное автономное оружие. П.У. Сингер в книге "Wired for War" ([2075], 2009) поднял этические, юридические и технические вопросы в отношении роботов на поле боя. В книге "Army of None" ([1994], 2018) Поль Шарре, являющийся одним из авторов нынешней американской политики в отношении автономного оружия, предлагает более сбалансированный и авторитетный взгляд на эту тему. Этциони и Этциони в работе [701] (2017) рассматривают вопрос, следует ли регулировать работы в области ИИ, — они рекомендуют сделать паузу в разработке смертельного автономного оружия и провести международную дискуссию на тему подобного регулирования.

Конфиденциальность. Латания Суини в работе [2163] (2002) представляет модель *k*-анонимности, а в работе [2162] (2002) — идею обобщения полей. Достижение *k*-анонимности при минимальной потере данных является NP-трудной задачей, но Баярдо и Агравал в работе [145] (2005) предлагают приблизительный алгоритм ее решения. Синтия Дворк в работе [666] (2008) описывает дифференциальную конфиденциальность, а в последующей работе приводит практические примеры разумных способов применения дифференциальной конфиденциальности для получения лучших результатов, чем при наивном подходе (Дворк и др. [668], 2014). Гуо и соавт. в работе [937] (2019) описывают процесс удаления сертифицированных данных: если модель обучается на некоторых данных, а затем выполняются запросы на удаление определенной информации, то подобное расширение дифференциальной конфиденциальности позволяет изменить модель и доказать, что в ней удаленные данные не используются. Джи и соавт. в работе [1136] (2014) приводят обзор литературы в области обеспечения конфиденциальности. Этциони в работе [699] (2004) выступает за сбалансированный подход к

конфиденциальности и безопасности, индивидуальным правам и нуждам сообщества. Фунг и соавт. ([801], 2018), Багдасарян и соавт. ([107], 2018) обсуждают различные атаки на федеральные протоколы образования. Нараянан и соавт. в работе [1656] (2011) описывают, как они получили возможность реиндефицировать запутанный граф подключении в конкурсе *2011 Social Network Challenge* путем тотального сканирования сайта, с которого были взяты данные задания (Flickr), и сопоставления узлов с необычно высокими пиками и провалами между предоставленными данными и данными, полученными в результате сканирования. Это позволило им получить дополнительную информацию, достаточную для того, чтобы выиграть этот конкурс, а также раскрыть истинную идентичность данных в узлах. В настоящее время пользователям становятся доступны инструменты для обеспечения конфиденциальности, например пакет TensorFlow включает модули для федеративного обучения и конфиденциальности (МакМейхен и Эндрю [1552], 2018).

Справедливость. Кети О'Нил в книге "Weapons of Math Destruction" ([1714], 2017) описывает, как различные "черные ящики" моделей машинного обучения влияют на нашу жизнь, часто несправедливым образом. Она призывает создателей моделей взять на себя ответственность за сохранение справедливости, а политические органы — обеспечить соответствующее регулирование. Дворк и соавт. в работе [667] (2012) указали на недостатки упрощенного подхода "справедливость через неосведомленность". Беллами и соавт. в работе [161] (2018) представили набор инструментальных средств для смягчения предвзятости в системах машинного обучения. Траме и соавт. в работе [2224] (2016) показали, как злоумышленники могут "похитить" модель машинного обучения, выполняя запросы к API, а Хардт и соавт. в работе [966] (2017) описывают равные возможности как показатель справедливости. Чулдечова и Рот в работе [427] (2018) приводят обзор передних рубежей борьбы за справедливость, а Верма и Рубин в работе [2270] (2018) дают исчерпывающий обзор различных определений справедливости.

Клейнберг и соавт. в работе [1240] (2016) показали, что в общем случае алгоритм не может быть одновременно и хорошо откалиброванным и гарантировать равные возможности. Берк и соавт. в работе [181] (2017) привели некоторые дополнительные определения типов справедливости и вновь пришли к заключению, что удовлетворить одновременно все эти аспекты невозможно. Бюте и соавт. в работе [208] (2019) дают советы в отношении того, как показатели справедливости применять на практике.

Дрессел и Фарид в работе ([650], 2018) сообщают о модели оценки рецидивизма COMPAS. Кристин и соавт. ([429], 2015) и Экхаус и соавт. ([674], 2019) обсуждают использование прогностических алгоритмов в правовой системе. Корбетт-Дэвис и соавт. в работе [478] (2017) показали, что существуют противоречия между обеспечением справедливости и оптимизацией общественной безопасности, а Корбетт-Дэвис и Гоэл в работе [477], (2018) обсуждают различие между существующими структурами обеспечения справедливости. Чулдечова в работе [426] (2017) выступает за справедливое воздействие: всем классам должна

предоставляться одна и та же ожидаемая полезность. Лю и соавт. в работе [1428] (2018) выступают за долгосрочные меры воздействия, указывая, например, что, если изменить решение о предоставлении кредита с целью повышения справедливости в краткосрочной перспективе, в долгосрочной перспективе это может оказать негативное влияние на людей, которые в конечном итоге не выплачивают ссуду, и, таким образом, их кредитный рейтинг снижается.

С 2014 года проводится ежегодная конференция по вопросам справедливости, подотчетности и прозрачности в машинном обучении. Мехраби и соавт. в работе [1556] (2019) дают всеобъемлющий обзор по вопросам предвзятости и справедливости в машинном обучении, сформировав каталог из 23 видов предвзятости и 10 определений справедливости.

Доверие. Объясняемый ИИ был важной темой еще в первые дни использования экспертных систем (Нечес и др. [1661], 1985), но в последние годы отмечается возрождение интереса к этой теме (Биран и Коттон [220], 2017; Миллер и др. [1578], 2017; Ким [1227], 2018). Баррено и соавт. в работе [134] (2010) приводят классификацию типов атак, которые могут быть направлены против системы машинного обучения, а Тигар в работе [2240], (2011) дает обзор состязательного машинного обучения. Исследователи в компании IBM предложили для завоевания доверия к системам ИИ использовать декларации о соответствии (Хинд и др. [1023], 2018). Управление перспективных исследовательских проектов США (DARPA) требует предоставления объясняемых решений для своих боевых систем и выдало предложение о проведении исследований в этой области (Ганнинг [936], 2016).

Безопасность ИИ. В книге "Artificial Intelligence Safety and Security" (Ямпольский [2395], 2018) собраны очерки о безопасности ИИ, как недавние, так и классические, начиная со статьи Билла Джоя "Why the Future Doesn't Need Us" ([1154], 2000). О проблеме царя Мидаса предупреждал еще Марвин Мински, который однажды предположил, что программа ИИ, предназначенная для нахождения доказательства гипотезы Римана, может в конечном итоге захватить все ресурсы Земли для создания новых, еще более мощных суперкомпьютеров. Аналогичным образом Омохундро в работе [1713] (2008) предвидел появление шахматной программы, которая захватывает ресурсы, а Бостром в работе [260] (2014) описывает неудержимый рост фабрики канцелярских скрепок, в конечном счете охватывающей весь мир. Юдковски в работе [2416] (2008) погружается в детали того, как разрабатывать **дружественный ИИ**, а Амодей и соавт. в работе [45] (2016) представляют пять практических проблем безопасности для систем ИИ.

Омохундро в работе [1713] (2008) описывает основные внутренние стимулы развития ИИ и приходит к такому заключению: "Социальные структуры, которые заставляют людей нести бремя их негативных внешних эффектов, должны будут пройти долгий путь к обеспечению стабильного и позитивного будущего". Элинор Остром в книге "Governing the Commons" ([1720], 1990) описывает практические подходы к тому, как справиться с непредвиденными обстоятельствами в

традиционных культурах. Остром также применила этот подход к идее представления знания как общего достояния (Хесс и Остром [1016], 2007).

С другой стороны, на каждого оптимистически настроенного сторонника сингулярности есть пессимист, который боится новых технологий. На веб-сайте `pessimists.co` приведены подтверждения тому, что так было всегда, на протяжении всей истории человечества; например, в 1890-е годы люди были обеспокоены тем, что лифт будет неизбежно вызывать тошноту, что телеграф приведет к потере конфиденциальности и к моральной коррупции, что метро будет выпускать опасный подземный воздух и беспокоить мертвых и что велосипед — в особенности идея женщины, едущей на нем — это работа дьявола.

Ханс Моравек в работе [1622] (2000) представил некоторые из идей трансгуманизма, а Бостром в работе [259] (2005) приводит обновленную историю. Идея сверхразумной машины Гуда была предвосхищена на сто лет раньше Сэмюэлем Батлером в статье "Darwin Among the Machines " ([353], 1863). В этой статье Батлера, написанной через четыре года после публикации книги Чарльза Дарвина "О происхождении видов", в то время, когда наиболее сложные машины представляли собой паровые двигатели, выдвигалось предположение "о безусловном развитии механического сознания" путем естественного отбора. Эта тема была вновь поднята Джорджем Дайсоном в книге с тем же названием ([670], 1998), и на нее ссылался Алан Тьюринг, который в 1951 году писал: "На какой-то стадии, следовательно, мы должны ожидать того, что машины возьмут контроль тем способом, который упоминается в статье *Erewhon* Самуила Батлера" (Тьюринг [2238], 1996).

Права роботов. В книге под редакцией Йорика Уилкса ([2346], 2010) приведены разные перспективы в отношении того, как нам придется вести дела с нашими искусственными соседями, начиная от точки зрения Джоанны Брайсон (роботы должны служить нам в качестве инструментов, а не являться гражданами) до наблюдения Шерри Таркла (мы уже персонифицируем наши компьютеры и другие инструменты и вполне готовы к тому, чтобы стереть границы между машинами и жизнью). Недавно Уилкс также предоставил обновленную информацию о своих взглядах (Уилкс [2347], 2019). В книге философа Дэвида Ганкеля "Права роботов" ([935], 2018) рассматриваются четыре возможности: *могут* ли роботы иметь права (или нет) и *должны* ли они их иметь (или нет)? Американское общество по предотвращению жестокого обращения с роботами (*American Society for the Prevention of Cruelty to Robots* — ASPCR) провозглашает, что "ASPCR существует и будет продолжать существовать, так же серьезно, как то, что роботы разумны".

Будущее труда. В 1888 году Эдвард Беллами опубликовал бестселлер "Looking Backward", в котором предсказал, что к 2000 году технический прогресс приведет к утопии, что будет достигнуто равенство, люди будут работать по нескольку часов и рано уходить в отставку. Вскоре после этого Э.М. Форстер поделился своим мрачным представлением о будущем в романе "The Machine Stops" ([761], 1909), в котором благожелательная машина берет на себя управление обществом, но в конечном счете все разваливается, когда эта машина неизбежно выходит из строя.

Норберт Винер в пророческой книге "The Human Use of Human Beings" ([2340], 1950) убеждает в преимуществах автоматизации, освобождающей людей от тяжелой работы, одновременно предлагая им более творческий труд, но также обсуждает несколько возможных опасностей, которые мы уже рассматриваем как проблемы сегодняшнего дня, в частности проблему выравнивания ценности.

В книге "Disrupting Unemployment" (Нордфорс и др. [1698], 2018) обсуждаются некоторые способы, при которых работа меняется, открывая возможности для новых карьер. Эрик Бринйолфссон и Эндрю Макафи обращаются к этой и другим темам в своих книгах "Race Against the Machine" ([330], 2011) и "The Second Machine Age" ([331], 2014). Форд в работе [756] (2015) описывает проблемы повышения степени автоматизации, а Уэст в работе [2326] (2018) дает рекомендации по смягчению этой проблемы, в то время как в Массачусетсском технологическом институте Томас Малоне показывает ([1481], 2004), что многие аналогичные проблемы были очевидны десятилетием ранее, но в то время их связывали с массовым распространением доступа к всемирным сетям связи, а не с автоматизацией.

Упражнения

27.1. Просмотрите составленный Тьюрингом список предполагаемых действий, которые "не способны" выполнить машины, и укажите, какие из них уже осуществлены, какие в принципе осуществимы с помощью программ и какие все еще находятся под вопросом, поскольку для них требуются сознательные психические состояния.

27.2. В популярных СМИ найдите и проанализируйте сообщения об одном или нескольких аргументах в отношении того, что создание ИИ невозможно.

27.3. Попытайтесь сформулировать определения терминов "интеллект", "мышление" и "сознание". Предложите несколько возможных возражений против ваших определений.

27.4. Является ли опровержение аргумента с китайской комнатой убедительным доказательством того, что должным образом запрограммированные компьютеры имеют психические состояния? Означает ли принятие этого аргумента так же согласие с тем, что компьютеры не могут иметь психических состояний?

27.5. Предположим, что программа на языке Prolog, содержащая множество пунктов о правилах британского гражданства, была скомпилирована и запущена на обычном компьютере. Проанализируйте "мозговые состояния" компьютера в соответствии с понятиями широкого и узкого контента.

27.6. Алан Перлис в статье "Epigrams on Programming" писал: "Года, проведенного в кругу исследователей ИИ, вполне достаточно, чтобы заставить человека поверить в Бога". Он также написал в письме Филиппу Дэвису, что одно из центральных мечтаний специалистов компьютерных наук состоит в том, что "за счет производительности компьютеров и их программ удастся полностью устранить все сомнения в том, что между живым и неживым миром существует только химическое различие". В какой степени прогресс, достигнутый к настоящему

времени в области ИИ, проливает свет на эти проблемы? Предположим, что когда-нибудь в будущем усилия создателей ИИ окажутся, безусловно, успешными, т.е. будут созданы интеллектуальные агенты, способные выполнять любые человеческие когнитивные задачи на человеческом уровне способностей. В какой степени это пролило бы свет на указанные проблемы?

27.7. Сравните социальное влияние ИИ за последние 50 лет с социальным воздействием в результате внедрения электрических устройств и двигателя внутреннего сгорания за 50 лет между 1890-м и 1940-м годами.

27.8. И. Дж. Гуд утверждает, что интеллект — это самое важное качество и что создание сверхразумных машин изменит все. Разумный гепард возражает: "На самом деле скорость важнее: если бы можно было создавать сверхбыстрые машины, это изменило бы все". Разумный слон, в свою очередь, заявляет: "Вы оба не правы, — нам нужны сверхсильные машины". Что вы думаете об этих аргументах?

27.9. Проанализируйте потенциальные угрозы обществу со стороны технологии ИИ. Какие угрозы являются наиболее серьезными и как им можно противостоять? В каком соотношении они находятся с потенциальными выгодами?

27.10. Как сравнить потенциальные угрозы со стороны технологии ИИ с угрозами со стороны других компьютерных технологий, а также с угрозами со стороны био-, нано- и ядерных технологий?

27.11. Одни критики ИИ утверждают, что его создать невозможно, а другие считают, что очень даже возможно и что сверхинтеллектуальные машины представляют собой угрозу. Какое из этих утверждений вы считаете более вероятным? Будут ли противоречивыми взгляды тех, кто согласен с обоими заявлениями одновременно?

ГЛАВА 28

Будущее искусственного интеллекта

В данной главе предпринимается попытка заглянуть вперед, пусть даже и недалеко.

В главе 2 было принято решение понимать ИИ как задачу разработки приблизительно рациональных агентов. Было рассмотрено большое разнообразие различных конструкций агентов, начиная от рефлекторных агентов и агентов, принимающих решения на основе знаний, и заканчивая агентами глубокого обучения методом обучения с подкреплением. Также подробно обсуждалось большое разнообразие технологий компонентов, из которых эти конструкции собираются: логические, вероятностные или нейронные рассуждения; атомарное, развернутое или структурированное представление состояний; разнообразные обучающие алгоритмы для различных типов данных; датчики и исполнительные механизмы для взаимодействия с миром. Наконец было представлено множество практических приложений в медицине, финансах, связи, на транспорте и в других областях. На всех этих направлениях был достигнут значительный прогресс как в научном понимании, так и в возможностях технической реализации.

Большинство экспертов с оптимизмом смотрят на дальнейший прогресс, — как было показано в разделе 1.4, средней оценкой времени достижения ИИ приблизительно человеческого уровня в широком спектре различных задач являются ближайшие 50–100 лет. В следующем десятилетии для ИИ прогнозируется вклад в мировую экономику на уровне триллионов долларов в год. Но, как уже говорилось, есть и критически настроенные эксперты, считающие, что на повсеместное распространение ИИ уйдут века и это потребует решения многочисленных этических проблем в отношении справедливости, равенства и опасных применений ИИ. В этой главе предпринимается попытка понять, куда мы направляемся и что еще предстоит сделать. Попробуем разобраться в этом, задавшись вопросом, имеются ли у нас правильные компоненты, архитектуры и цели, чтобы сделать ИИ успешной технологией, приносящей пользу миру?

28.1. Компоненты ИИ

В этом разделе рассматриваются компоненты систем ИИ и степень, в которой каждый из них может способствовать ускорению дальнейшего прогресса или, наоборот, препятствовать ему.

Датчики и исполнительные механизмы

На протяжении большей части истории ИИ прямой его доступ к миру, безусловно, отсутствовал. За некоторыми заметными исключениями системы ИИ были построены таким образом, что люди должны были предоставлять им входные данные и интерпретировать то, что получали на выходе. В настоящее время существующие роботизированные системы в основном ориентированы на задачи низкого уровня, в которых рассуждения и планирование на высоком уровне по большей части игнорируются, а потребность в восприятии сведена к минимуму. Отчасти это связано с необходимостью значительных затрат и приложения больших инженерных усилий, чтобы вообще заставить реальных роботов работать, а отчасти — из-за отсутствия достаточной вычислительной мощности и достаточно эффективных алгоритмов обработки визуального ввода с высокой пропускной способностью.

Ситуация начала быстро изменяться лишь в последние годы благодаря появлению полностью готовых программируемых роботов. Это, в свою очередь, стало возможным в результате разработки компактных и надежных моторных приводов и усовершенствованных датчиков. Стоимость лидара для беспилотного автомобиля упала с 75 до 1 тыс. долл., а цена версии на одном кристалле может достичь даже 10 долл. за штуку (Поултон и Уаттс [1816], 2016). Радиолокационные датчики, ранее способные обнаруживать лишь достаточно крупные объекты, в настоящее время стали настолько чувствительными, что позволяют подсчитать количество листов в стопке бумаги (Ео и др. [2403], 2018).

Потребность в улучшенной обработке изображений в камерах мобильных телефонов обеспечила появление на рынке недорогих камер высокого разрешения, вполне пригодных для использования в робототехнике. Технология микроэлектромеханических систем (*micro-electromechanical systems* — MEMS) обеспечила выпуск миниатюрных акселерометров, гироскопов и электроприводов, размеры которых позволили приступить к разработке искусственных летающих насекомых (Флореано и др. [747], 2009; Фуллер и др. [800], 2014). В то же время вполне возможно объединить миллионы MEMS-устройств для реализации мощных макроскопических приводов. Технологии объемной печати (Мут и др. [1650], 2014) и печати биоматериалов (Колески и др. [1262], 2014) существенно упростили проведение экспериментов с прототипами.

Таким образом, вполне можно утверждать, что системы ИИ сейчас находятся на пике перехода от преимущественно программных систем к полезным встроенным роботизированным системам. Сегодняшнее состояние робототехники

примерно сравнимо с состоянием в области персональных компьютеров в начале 1980-х годов: в то время персональные компьютеры уже стали доступными, но потребовалось еще десять лет, чтобы они стали обычным явлением. Вполне вероятно, что гибкие, интеллектуальные роботы сначала добьются успеха в промышленности (где среда более контролируема, задачи более повторяющиеся, а отдачу от инвестиций легче измерить), а затем — и в области домашнего применения (где изменчивость среды и разнообразие задач значительно больше).

Представление состояния мира

Отслеживание состояния мира требует как восприятия, так и обновления собственных внутренних представлений о нем. В главе 4 объяснялось, как отслеживать состояния в атомарном представлении, а в главе 7 — как это можно сделать для развернутого (пропозиционального) представления состояний. В главе 10 данный подход был расширен до логики первого порядка, а в главе 14 представлены вероятностные рассуждения во времени в условиях неопределенности. Наконец, в главе 21 обсуждались рекуррентные нейронные сети, также позволяющие поддерживать представление состояния во времени.

Известные на текущий момент алгоритмы фильтрации и восприятия могут быть объединены для выполнения рассуждений по распознаванию объектов ("это кошка") и выявлению предикатов низкого уровня ("эта чашка находится на столе"). Распознавание действий более высокого уровня, таких как "Доктор Рассел за чашкой чая обсуждает с доктором Норвигом планы на следующую неделю", гораздо сложнее. В настоящее время это иногда можно выполнить (см. рис. 25.17 в разделе 25.7.1) при наличии достаточного количества обучающих примеров, но для дальнейшего прогресса потребуются методы, позволяющие обобщать новые ситуации без необходимости предоставления исчерпывающих примеров (Поппе [1811], 2010; Кенг и Уайлдс [1182], 2016).

Другая проблема заключается в том, что хотя представленные в главе 14 приближенные алгоритмы фильтрации позволяют работать в довольно больших средах, они все еще имеют дело с развернутым представлением: в них используются случайные переменные, но объекты и отношения не представлены явно. Кроме того, представление времени в них ограничено пошаговыми изменениями, — зная недавнюю траекторию мяча, можно предсказать, где он окажется в момент времени $t + 1$, но трудно представить абстрактную идею о том, что, начав двигаться вверх, он затем должен будет упасть вниз.

В разделе 15.1 объяснялось, как можно объединить вероятность и логику первого порядка для решения таких задач; в разделе 15.2 было показано, как можно справиться с неопределенностью в отношении идентичности объектов, а в главе 25 объяснялось, каким образом в компьютерном зрении для отслеживания состояния мира можно использовать рекуррентные нейронные сети, однако пока еще нет хорошего способа соединить все эти методы в единое целое. В главе 24 было

показано, как внедрение слов и другие подобные представления позволяют освободиться от строгих границ понятий, определенных необходимыми и достаточными условиями. Однако для сложных предметных областей определение общих, многократно используемых схем представления все еще остается сложной задачей.

Выбор действий

Основной трудностью при выборе действий в реальном мире является необходимость уметь справляться с долгосрочными планами, такими как окончание колледжа после четырех лет обучения, состоящими из миллиардов примитивных этапов. Поисковые алгоритмы, в которых рассматриваются последовательности примитивных действий, работают только в масштабах десятков или, возможно, сотен этапов. Люди способны на такое сложное поведение только за счет придания своим действиям **иерархической структуры.** В разделе 11.4 объяснялось, как использовать иерархическое представление для решения задач подобного масштаба. Более того, разработка методов **иерархического обучения с подкреплением** позволила добиться успеха в сочетании этих идей с математическим аппаратом марковских процессов принятия решений (MDP), описанным в главе 17.

Пока эти методы не были распространены на случай частично наблюдаемых сред (задачи POMDP). Кроме того, в известных алгоритмах решения задач POMDP, как правило, используется такое же атомарное представление состояний, как то, которое используется в алгоритмах поиска, описываемых в главе 3. Очевидно, что объем работы, которую здесь предстоит выполнить, весьма велик, но необходимые технические основы для достижения прогресса в этой области уже имеются. Основным недостающим элементом является эффективный метод *построения* иерархических представлений состояния и поведения, необходимых для принятия решений в масштабах больших промежутков времени.

Определение наших предпочтений

В главе 3 были представлены алгоритмы поиска, предназначенные для нахождения целевого состояния. Но агенты, действующие на основе цели, оказываются слишком хрупкими, если среда характеризуется неопределенностью и имеется множество факторов, подлежащих рассмотрению. В принципе, агенты, действующие на основе максимизации полезности, как раз и предназначены для решения подобных задач, причем самым общим способом. В области экономики и теории игр, как и в области ИИ, вполне можно использовать этот подход: просто объявить, *что* требуется оптимизировать и *что* происходит в результате выполнения каждого действия, и это позволит вычислить, какое именно действие будет оптимальным.

Однако на практике мы теперь понимаем, что задача выбора правильной функции полезности сама по себе является сложной проблемой. Представим, например, сложное переплетение взаимосвязанных предпочтений, в которых должен разобраться агент, функционирующий в качестве офисного помощника человека.

Проблема усугубляется еще и тем фактом, что каждый человек индивидуален, поэтому агент, которого только что "достали из коробки", просто не будет иметь достаточного опыта взаимодействия с каким-то конкретным человеком, чтобы успеть точно изучить модель его предпочтений, — ему непременно придется действовать в условиях неопределенности в отношении этих предпочтений. Дополнительная сложность возникает в том случае, если требуется обеспечить, чтобы агенты действовали справедливо и беспристрастно в отношении всего общества, а не только отдельного человека.

У нас все еще недостает опыта в построении сложных моделей предпочтений в реальном мире, не говоря уже о распределениях вероятностей в таких моделях. Хотя есть математический аппарат для развернутых представлений, похожий на байесовские сети и предназначенный для декомпозиции предпочтений в отношении сложных состояний, была доказана трудность использования этого аппарата на практике. Одной из причин может быть то, что предпочтения в отношении состояний в действительности *компилируются* из предпочтений в отношении историй состояний, которые описываются посредством **функций вознаграждения** (см. главу 17). Даже если сама функция вознаграждения является простой, соответствующая функция полезности может оказаться очень сложной.

Это говорит о том, что необходимо серьезно отнестись к задаче разработки знаний для функции вознаграждения как способу передачи агенту наших желаний в отношении того, что он должен сделать. Идея **обратного обучения с подкреплением** (раздел 22.6) — это один из подходов к решению данной задачи, когда есть эксперт, способный выполнить задание, но не способный объяснить, как это делается. Также можно воспользоваться лучшими языками для выражения того, что нам требуется. Например, в робототехнике линейная временная логика позволяет проще сформулировать, что требуется сделать в ближайшем будущем, чего требуется избегать и какие состояния желательно сохранять неопределенно долго (Литтман и др. [1422], 2017). Необходимо найти лучшие способы выражения того, что нам требуется, а для роботов — лучшие способы интерпретации той информации, которая будет им предоставлена.

Компьютерная индустрия в целом разработала мощную экосистему агрегирования предпочтений пользователей. Когда вы щелкаете на каком-то элементе интерфейса приложения, онлайн-игры, социальной сети или сайта совершения покупок, это автоматически служит свидетельством того, что вы (и аналогичные вам коллеги) хотели бы видеть подобные вещи и в будущем. (Или может оказаться, что интерфейс ввел вас в заблуждение и вы щелкнули не на том его элементе, — данные всегда являются зашумленными.) Обратная связь в этой системе делает ее очень эффективной в краткосрочной перспективе для выбора еще более захватывающих игр и видео.

Но эти системы часто не в состоянии обеспечить простой способ отказаться — устройство будет автоматически воспроизводить подходящее видео, но гораздо менее вероятно, чтобы оно выдало вам сообщение: "Может быть, пора отложить

ваше устройство подальше и отправиться на расслабляющую прогулку на природе?" Торговый сайт поможет подобрать одежду, соответствующую вашему стилю, но не обратится к вам с призывом о мире во всем мире или к избавлению мира от голода и нищеты. До тех пор, пока меню вариантов выбора формируется компаниями, пытающимися получить прибыль за счет привлечения внимания клиента, это меню будет оставаться неполным.

Тем не менее компании действительно реагируют на интересы клиентов, и многие клиенты уже высказали мнение, что они заинтересованы в справедливом и устойчивом мире. Тим О'Рейли объяснил, почему прибыль является не единственным мотивом, с помощью следующей аналогии: "Деньги — это как бензин в поездке. Не хотелось бы, чтобы он закончился посреди дороги, но при этом сама поездка — вовсе не турне по заправочным станциям. Вы должны уделять внимание деньгам, но это не то, вокруг чего все вертится".

Движение ▶**time well spent** (хорошо проведенное время) Тристана Харриса в Центре за гуманные технологии (*Center for Humane Technology* — CHT) — это шаг к тому, чтобы дать нам более взвешенный выбор (Харрис [967], 2016). Это движение призвано решить проблему, которая была выявлена Гербертом Саймоном еще в 1971 году: "Обилие информации приводит к недостатку внимания". Возможно, в будущем у нас появятся ▶**личные агенты**, которые будут придерживаться наших истинных долгосрочных интересов, а не интересов корпораций, чьи приложения в настоящее время переполняют наши устройства. Работа агента будет заключаться в посредничестве для предложений от различных поставщиков, в защите нас от средств захвата внимания и в достижении целей, которые действительно важны для нас.

Обучение

В главах с 19 по 22 описывается, как агенты могут обучаться. Современные алгоритмы могут справиться с довольно крупными задачами, успешно достигая возможностей человека в выполнении многих заданий или превышая их — до тех пор, пока имеется достаточное количество обучающих примеров и мы имеем дело с предопределенным перечнем признаков и концепций. Но обучение может замедляться, когда данные являются разреженными или неконтролируемыми или когда обучение связано со сложными представлениями.

Большая часть недавнего возрождения интереса к ИИ в популярной прессе и в промышленности обусловлена успехом глубокого обучения (глава 21). С одной стороны, это можно рассматривать как постепенное созревание подобласти нейронных сетей. С другой стороны, можно смотреть на это как на революционный скачок в возможностях, подстегнутый слиянием нескольких факторов: доступностью большего количества обучающих данных благодаря Интернету, возрастанию вычислительной мощности благодаря специализированным аппаратным средствам и применению нескольких алгоритмических трюков, таких как

генеративно-состязательные сети (GAN), пакетная нормализация, исключение и функция активации ReLU.

В будущем следует ожидать сохранение акцента на улучшении методов глубокого обучения для тех задач, в которых оно уже отлично себя проявило, а также для расширения его применения с целью охвата других задач. Бренд "глубокое обучение" уже настолько доказал свою популярность, что следует ожидать продолжения его использования, даже если смесь методов, которые его подпитывают, значительно изменится.

Мы уже стали свидетелями возникновения новой области — **науки о данных** — на пересечении статистики, программирования и анализа данных в заданной предметной области. Хотя можно ожидать дальнейшего развития инструментов и методов, необходимых для сбора, обработки и обслуживания **больших данных**, необходим прогресс и в области **трансферного обучения**, что позволит воспользоваться преимуществами накопления данных в одной предметной области для повышения производительности в родственных областях.

Подавляющее большинство исследований в области машинного обучения сегодня предполагает использование развернутого представления, изучения функций $h: \mathbb{R}^n \rightarrow \mathbb{R}$ для задач регрессии и $h: \mathbb{R}^n \rightarrow \{0, 1\}$ — для задач классификации. Машинное обучение было менее успешным для тех задач, для которых имеется лишь небольшой объем данных, или для задач, которые требуют построения новых структурированных, иерархических представлений. Глубокое обучение, особенно с использованием сверточных сетей, применяемых к задачам компьютерного зрения, продемонстрировало определенные успехи при переходе от низкоуровневых отдельных пикселей к концепциям среднего уровня, таким как *Глаз* и *Рот*, далее — к концепции *Лицо* и наконец — к концепции *Человек* или *Кошка*.

Задача на будущее состоит в том, чтобы более плавно сочетать обучение и предварительно накопленные знания. Если поставить перед компьютером задачу, с которой он раньше не сталкивался — скажем, с распознаванием различных моделей автомобилей, — нежелательно, чтобы система оставалась беспомощной до тех пор, пока ей не будут представлены миллионы помеченных примеров.

Идеальная система должна быть в состоянии опираться на то, что она уже знает: она должна уже иметь модель того, как работает видение, и как дизайн и брендинг продукции работают в общем случае. Далее ей следует использовать **трансферное обучение** применительно к новой задаче о распознавании моделей автомобилей. Она должна быть способна самостоятельно находить информацию о моделях автомобилей, опираясь на сопроводительные тексты, изображения и видео, доступные в Интернете. Она должна быть способна и к **обучению на демонстрации**: поддерживая разговор с учителем, не просто выдавать запросы вида "Можно мне получить тысячи изображений модели Corolla?", а скорее быть в состоянии понимать предоставляемые описания, например "Модель Insight похожа на модель Prius, но у Insight решетка радиатора больше". Она должна знать, что каждая модель поставляется в небольшом диапазоне возможных цветов, а также

что любой автомобиль может быть перекрашен, так что есть шанс встретить автомобиль в цвете, который отсутствовал в обучающем наборе. (Если она этого не знает, то должна быть в состоянии изучить это или воспринять объяснения по этому поводу.)

Все это требует возможности коммуникации и наличия языка представления того, чем люди и компьютеры могут обмениваться. Не следует ожидать, что человек-аналитик сможет напрямую модифицировать модель с миллионами весовых коэффициентов. Вероятностные модели (включая вероятностные языки программирования) предоставляют людям определенную способность описать то, что мы знаем, но эти модели еще недостаточно хорошо интегрированы с другими механизмами обучения.

Работа Бенжио и ЛеКуна [174] (2007) является одним из шагов в направлении этой интеграции. Недавно Ян ЛеКун предложил заменить термин "глубокое обучение" более общим термином **дифференцируемое программирование** (Сискинд и Перлмуттер [2078], 2016; Ли и др. [1404], 2018), — это говорит о том, что языки программирования общего назначения и модели машинного обучения могут быть объединены.

Прямо сейчас уже является общепринятым подходом создание такой модели глубокого обучения, которая изначально является дифференцируемой, а следовательно, может быть обучена минимизировать потери и переобучена в случае изменения обстоятельств. Но модель глубокого обучения является только одной частью более крупной программной системы, которая принимает данные, манипулирует ими, передает их в модель и принимает решение, что делать с выходными данными, предоставленными этой моделью. Все эти прочие части более крупной системы были написаны программистом вручную и, следовательно, являются недифференцируемыми, а это означает, что при изменении обстоятельств на программиста возлагается задача выявить любые возникшие проблемы и вручную устранить их. При использовании методов дифференцируемого программирования появляется надежда, что вся система в целом станет объектом автоматизированной оптимизации.

Конечной целью является получение возможности выразить то, что мы знаем, в любой удобной для нас форме: как неформальное описание, сделанное на естественном языке, в виде строгой математической формулы типа $F = ma$, как статистическую модель, прилагаемую к данным, или в виде вероятностной программы с неизвестными параметрами, которая может быть автоматически оптимизирована с использованием метода градиентного спуска. Компьютерные модели будут обучаться посредством общения с людьми-экспертами, а также за счет использования для обучения всех доступных данных.

Ян ЛеКун, Джеффри Хинтон и другие ученые предположили, что нынешний акцент на обучение с учителем (и в меньшей степени — на обучение с подкреплением) не следует считать стабильным — что компьютерные модели будут вынуждены обратиться к **обучению с незначительным привлечением учителя**, при котором часть обучения проводится лишь на небольшим количестве помеченных

примеров и/или при небольшим количестве актов вознаграждения, а бо́льшая часть всего процесса представляет собой обучение без учителя, поскольку объем доступных неаннотированных данных во много раз больше.

ЛеКун использует термин ▶**интеллектуальное обучение** для системы обучения без учителя, которая может моделировать мир и обучается предсказывать аспекты будущего состояния мира, а не просто предсказывать этикетки для входных данных, которые гарантированно независимы и характеризуются распределением, совпадающим с уже введенными данными, или же не только предсказывать значение функции относительно состояний. Он предполагает, что сети GAN (генеративно-состязательные сети) можно использовать для обучения минимизации разницы между предсказанием и реальностью.

В 2017 году Джеффри Хинтон заявил: "Моя точка зрения — отбросить все прочь и начать сначала", имея в виду, что общую идею обучения посредством настройки параметров сети следует сохранить, но специфику архитектуры сетей и метод обратного распространения необходимо переосмыслить. Смоленский в работе [2102] (1988) дал рецепт, какой подход можно использовать в отношении коннекционистских моделей, — его мысли остаются актуальными и сегодня.

Ресурсы

Исследования и новые разработки в области машинного обучения были существенно ускорены благодаря увеличению доступности данных, размера хранилищ, мощности вычислительных устройств обработки, появлению нового программного обеспечения и обученных специалистов, а также росту инвестиций, необходимых для их поддержки. По сравнению с 1970-ми годами было достигнуто 100 000-кратное ускорение работы процессоров общего назначения и дополнительное 1000-кратное ускорение благодаря разработке специализированного оборудования для систем машинного обучения. Интернет и веб превратились в богатый источник изображений, видео, речевых записей, текстов и полуструктурированных данных, — в настоящее время объемы этих данных ежедневно возрастают на 10^{18} байт.

Сейчас доступны сотни высококачественных наборов данных для широкого круга задач в области компьютерного зрения, распознавания речи и обработки естественного языка. Если нужные данные пока еще недоступны, можно собрать их из других источников или привлечь людей для маркировки необходимых данных с использованием краудсорсинговой платформы. Проверка полученных таким образом данных становится важной частью общего рабочего процесса (Херт и др. [1033], 2013).

Важной недавней разработкой стал переход от общих совместно используемых данных к общим ▶**совместно используемым моделям**. Провайдеры крупных облачных служб (например, Amazon, Microsoft, Google, Alibaba, IBM, Salesforce) уже вступили в конкуренцию в области предложения API машинного обучения с

использованием предварительно построенных моделей для конкретных задач, таких как визуальное распознавание объектов, распознавание речи и машинный перевод. Эти модели могут использоваться как есть или служить основой для дополнительной настройки с применением данных пользователя для конкретного приложения.

Ожидается, что с течением времени эти модели будут улучшаться и что в конечном счете станет нетипичным начинать очередной проект машинного обучения с нуля, — так же, как в настоящее время необычно приступать к разработке очередного веб-проекта с нуля, без использования каких-либо готовых библиотек. Вполне вероятно, что большой скачок в качестве моделей произойдет после того, как станет экономически возможным использовать в качестве обучающих примеров все доступное в Интернете видео, — например, в хранилища только одной платформы YouTube каждую минуту добавляется 300 часов видео.

Развитие технологий согласно закону Мура сделало обработку данных несравненно более эффективной в отношении затрат: стоимость хранилища объемом в один мегабайт в 1969 году составляла 1 млн долл., а в 2019 году — менее 0,02 долл., а пропускная способность суперкомпьютеров за это время возросла с коэффициентом более чем 10^{10}. Специализированные аппаратные средства для задач машинного обучения, такие как графические процессоры (GPU), тензорные ядра, тензорные процессоры (TPU) и программируемые пользователем вентильные матрицы (FPGA), обеспечивают ускорение процессов машинного обучения в сотни раз по сравнению с обычными процессорами общего назначения (Василаке и др. [2265], 2014; Юппи и др. [1153], 2017). В 2014 году для обучения модели с использованием набора данных ImageNet требовался полный день, тогда как в 2018 году на это требовалось всего 2 минуты (Йинг и др. [2404], 2018).

Исследования, проведенные в OpenAI Institute, показали, что общие объемы вычислительных мощностей, используемых для обучения больших моделей машинного обучения, удваивались каждые 3,5 месяца за период с 2012 по 2018 год, достигнув более чем экзафлопа в секунду в день для системы AlphaZero (хотя также есть сообщения о том, что при выполнении некоторых очень значимых работ использовалось в 100 млн раз меньше вычислительной мощности (Амодей и Эрнандес [44], 2018)). Те экономические тенденции, которые сделали камеры сотовых телефонов дешевле и лучше, вполне применимы и к процессорам, — следует ожидать быстрого прогресса в области высокопроизводительных вычислений с малым потреблением мощности, что принесет экономическую выгоду за счет масштабов.

Существует вероятность того, что использование квантовых компьютеров позволит ускорить продвижение в области ИИ. В настоящее время уже существует несколько быстрых квантовых алгоритмов для выполнения операций линейной алгебры, применяемых в машинном обучении (Харроу и др. [970], 2009; Дервович и др. [609], 2018), однако пока не существует квантового компьютера, способного их выполнять. Есть несколько примеров приложений для таких задач, как

классификация изображений (Мотт и др. [1628], 2017), в которых квантовые алгоритмы работают так же хорошо, как классические алгоритмы при решении небольших задач.

Современные квантовые компьютеры способны обрабатывать только несколько десятков битов, тогда как выполнение алгоритмов машинного обучения часто требует обработки входных данных объемом в миллионы битов и создания модели с сотнями миллионов параметров. Поэтому необходим прорыв как в квантовом оборудовании, так и в программном обеспечении, чтобы сделать квантовые вычисления практичными для крупномасштабного машинного обучения. С другой стороны, может иметь место разделение труда, например применение квантового алгоритма для эффективного поиска в пространстве гиперпараметров, в то время как нормальный учебный процесс будет работать на обычных компьютерах, однако пока еще нам неизвестно, как это можно реализовать. Исследования в области квантовых алгоритмов иногда могут вдохновлять создание новых, лучших алгоритмов, работающих на классических компьютерах (Танг [2173], 2018).

В мире наблюдается экспоненциальный рост количества публикаций, занятых людей и вложенных долларов в сферах ИИ, машинного обучения и науки о данных. Дин и соавт. в работе [565] (2018) показали, что количество статей по теме "Машинное обучение" на сайте `arXiv.org` удваивается каждые два года за период с 2009 по 2017 год. Инвесторы финансируют компании, запускающие стартапы в этих областях исследований, крупные компании нанимают работников и выделяют необходимые финансы по мере того, как определяют свою стратегию в отношении ИИ, а правительства осуществляют инвестиции, чтобы гарантировать, что их страна не слишком отстает в данной области.

28.2. Архитектуры систем ИИ

Вполне естественно спросить "Какую из архитектур агента, обсуждавшихся в главе 2, следует использовать в агентах?" Ответом будет: "Все из них!" Рефлекторные ответы необходимы в ситуациях, когда время имеет существенное значение, тогда как размышления на основе знаний позволяют агенту строить планы наперед. Обучение удобно, если имеется много данных, и просто необходимо, когда среда меняется или когда разработчики-люди недостаточно осведомлены о предметной области.

В области ИИ уже давно произошел раскол между символическими системами (построенными на основе логического и вероятностного вывода) и коннекционистскими системами (построенными на минимизации потерь при большом количестве неинтерпретируемых параметров). Непреходящая проблема в области ИИ состоит в объединении этих двух подходов с целью взять лучшее из каждого. Символические системы позволяют нанизывать длинные цепи рассуждений, что дает преимущества в использовании выразительной силы структурных представлений,

в то время как коннекционистские системы способны распознавать шаблоны даже при зашумленных данных. Одна из линий исследований нацелена на комбинирование вероятностного программирования с глубоким обучением, хотя пока что различные предложения ограничены в степени, в которой эти подходы действительно объединены.

Агенты также нуждаются в способах управления собственными рассуждениями. Они должны обладать способностью правильно использовать имеющееся время и прекращать рассуждать, когда требуются действия. Например, агент беспилотного такси, увидевший впереди аварию, должен в доли секунды решить, тормозить ему или объехать место происшествия. Кроме того, в эти доли секунды он должен думать лишь о наиболее важных в данный момент вопросах, таких как свободны ли полосы движения слева и справа и есть ли позади большой грузовик в опасной близости, а не беспокоиться о том, где он сможет подобрать следующего пассажира. Эти проблемы изучаются обычно под заголовком ►**ИИ реального времени**. По мере того как системы ИИ переходят во все более сложные проблемные области, все задачи постепенно смещаются в режим реального времени, поскольку у агента никогда уже не будет достаточно времени, чтобы найти точное решение стоящей перед ним задачи.

Очевидно, что насущной является потребность в *общих* методах контроля за рассуждениями, а не предоставление конкретных рецептов того, о чем следует думать в каждой ситуации. Первой полезной идеей являются ►**алгоритмы с отсечением по времени** (*anytime algorithms*) (Дин и Бодди [567], 1988; Хорвиц [1065], 1987): это такие алгоритмы, качество выходных данных которых неизменно улучшается во времени, поэтому они всегда предоставляют приемлемое решение, когда бы ни была прервана их работа. Примерами алгоритмов с отсечением по времени являются алгоритм поиска с итеративным углублением по дереву игры и алгоритм MCMC в байесовских сетях.

Вторым методом контроля за рассуждениями являются ►**метарассуждения на основе теории принятия решений** (*decision-theoretic metareasoning*) (Рассел и Вефальд [1946], 1989; Хорвиц и Бриз [1069], 1996; Хей и др. [988], 2012). В этом методе, который кратко упоминался в разделах 3.6.5 и 5.7, теория ценности информации (глава 16) применяется для выбора отдельных вычислений (раздел 3.6.5). Ценность вычислений зависит как от их стоимости (с точки зрения задержки действий), так и от получаемых в результате преимуществ (с точки зрения улучшения качества решения).

Методы метарассуждений могут использоваться для проектирования лучших алгоритмов поиска и для обеспечения гарантий того, что эти алгоритмы будут обладать свойством поддержки отсечения по времени. Одним из примеров таких алгоритмов является поиск по дереву методом Монте-Карло: выбор листового узла, с которого начнется следующий прогон, осуществляется с помощью приблизительно рационального решения на метауровне, полученного на основе теории многоруких бандитов.

Конечно, подход на основе метарассуждений является более дорогостоящим, чем рефлекторное действие, однако методы компиляции могут быть применены таким образом, что накладные расходы окажутся малыми по сравнению с расходами на контролируемые вычисления. Еще один способ нахождения эффективных стратегий управления рассуждениями может предоставить обучение с подкреплением на метауровне. В сущности, любые вычисления, которые приводят к лучшим решениям, являются подкреплением, тогда как те, которые не имеют никакого эффекта, являются штрафами. Такой подход позволяет избежать проблем близорукости при простом вычислении ценности информации.

Метарассуждения представляют собой один конкретный пример ▶ **рефлексивной архитектуры**, т.е. архитектуры, позволяющей формировать рассуждения о вычислительных сущностях и действиях, возникающих в самой архитектуре. Теоретическую основу для рефлексивных архитектур можно заложить, определив совместное пространство состояний, складывающееся из состояний среды и вычислительного состояния самого агента. Могут быть разработаны алгоритмы принятия решений и обучения, действующие в этом совместном пространстве состояний и, таким образом, способствующие реализации и улучшению вычислительной деятельности агента. В конечном итоге можно ожидать, что алгоритмы для конкретных задач, таких как альфа-бета-поиск, регрессионное планирование и устранение переменных, исчезнут из систем ИИ и будут заменены общими методами, направляющими вычисления агента в сторону эффективного формирования высококачественных решений.

Метарассуждения и рефлексивность (а также многие другие из анализируемых в этой книге архитектурных и алгоритмических аппаратов, связанных с эффективностью) действительно являются необходимыми, поскольку принимать решения *трудно*. С того времени, когда были изобретены компьютеры, их ослепляющая скорость вычислений склонила людей к завышению оценки своей способности преодолевать сложность, или, что то же самое, к недооценке того, что сложность на самом деле означает. Поистине огромная мощь современных машин заставляет думать, что мы можем обойтись без всех хитроумных аппаратов и больше полагаться на грубую силу. Поэтому, давайте попробуем оказать противодействие этой тенденции. Начнем с того, что физики считают, что быстродействие идеального вычислительного устройства весом 1 кг будет около 10^{51} операций в секунду, или в миллиард триллионов триллионов раз больше, чем быстродействие лучшего суперкомпьютера образца 2020 года (Ллойд [1436], 2000).[1] Тогда мы предлагаем простую задачу: перечислить строки английских слов подобно тому, как Борхес предложил это сделать в своем рассказе "Вавилонская библиотека". Борхес предложил ограничить размер книг 410 страницами. Будет ли возможно это выполнить? Не

[1] Мы хотим подчеркнуть тот факт, что это устройство будет потреблять всю энергию звезды и работать при температуре миллиард градусов Цельсия.

вполне. Фактически этот сверхсуперкомпьютер, работающий в течение года, успеет перечислить только все строки из 11 слов.

А теперь обратимся к тому факту, что подробный план человеческой жизни состоит из (очень приблизительно) двадцати триллионов потенциальных мышечных воздействий (Рассел [1942], 2019), — полагаю, вы начинаете правильно оценивать масштаб проблемы. Компьютер, который в миллиард триллионов триллионов раз мощнее человеческого мозга, гораздо дальше от того, чтобы быть рациональным, чем улитка от способности обогнать звездолет "Энтерпрайз", летящий со сверхсветовой скоростью.

Приняв во внимание все эти соображения, можно подумать, что цель создания рациональных агентов, возможно, слишком амбициозна. Вместо того чтобы стремиться к чему-то, что, вероятно, просто не может существовать, лучше будет рассмотреть другую нормативную цель — такую, которая *безусловно* достижима. Напомним следующую простую идею из главы 2:

$$агент = архитектура + программа.$$

Теперь зафиксируем архитектуру агента (базовые возможности оборудования машины, возможно, с фиксированным слоем программного обеспечения поверх него) и позволим программе агента изменяться в пределах всех возможных программ, которые данная архитектура способна поддерживать. В контексте любой заданной задачи одна из этих программ (или их эквивалентный класс) обеспечит наилучшую производительность, — возможно, недостаточно близко к идеальной рациональности, но все же лучше, чем любая другая программа агента. В этом случае говорят, что данная программа удовлетворяет критерию ▶**ограниченной оптимальности**. Очевидно, что она существует, и понятно, что она представляет собой желаемую цель. Вся хитрость в том, чтобы отыскать ее или что-то близкое к ней.

Для некоторых элементарных классов программ агентов в простых условиях в режиме реального времени все же возможно идентифицировать ограниченную оптимальность программ агента (Этциони [702], 1989, Рассел и Субраманиан [1945], 1995). Успех поиска по дереву методом Монте-Карло возродил интерес к метауровневому принятию решений, и это повод для надежды, что ограниченная оптимальность в более сложных семействах программ агентов вполне может быть достигнута с помощью таких методов, как обучение с подкреплением на метауровне. Также должно быть возможно разработать конструктивную теорию архитектур, начиная с теорем по ограниченной оптимальности подходящих методов сочетания различных ограниченно оптимальных компонентов, таких как рефлекторные системы и системы типа "действие–стоимость".

Общий ИИ

Значительная часть прогресса в области ИИ, достигнутого в XXI веке, до настоящего времени направлялась соревнованиями по решению узконаправленных задач, таких как DARPA Grand Challenge для беспилотных автомобилей, ImageNet — в области распознавания объектов или игра в го, шахматы, покер либо участие в викторине Jeopardy! против чемпионов мира. Для каждой такой задачи создавалась отдельная система ИИ, как правило, с собственной моделью машинного обучения, обучавшейся с нуля с помощью данных, собранных специально для этой задачи. Но действительно разумный агент должен уметь делать больше, чем решать единственную задачу. Алан Тьюринг в статье [2235] (1950) предложил свой список (см. раздел 27.1.2), на что писатель-фантаст Роберт Хайнлайн ([1004], 1973) возразил следующим образом.

> Человеческое существо должно быть в состоянии поменять пеленку, спланировать вторжение, разделать свинью, управлять кораблем, спроектировать здание, написать сонет, подвести баланс по счету, построить стену, бросить кости, утешить умирающего, принять заказ, отдавать приказы, сотрудничать, действовать самостоятельно, решать уравнения, анализировать новую проблему, разбросать навоз, запрограммировать компьютер, приготовить вкусную еду, эффективно сражаться, благородно умереть. Специализация — для насекомых.

На данный момент ни одна существующая система ИИ не отвечает ни одному из этих списков, и некоторые сторонники общего ИИ или ИИ человеческого уровня (*human-level AI* — HLAI) настаивают на том, что продолжения работы над конкретными задачами (или над отдельными компонентами) будет недостаточно для достижения мастерства в широком спектре задач; что здесь необходим принципиально новый подход. Нам же кажется, что многочисленные новые прорывы действительно будут необходимы, но в целом в области ИИ к настоящему времени был найден необходимый разумный компромисс между разведкой и эксплуатацией, проведена сборка портфолио компонентов, достигнуты улучшения на конкретных задачах, а также подробно изучены перспективные и иногда передовые новые идеи.

Вероятно, было бы ошибкой сказать братьям Райт в 1903 году, чтобы они прекратили работу над своим однозадачным самолетом и занялись разработкой машины "общего искусственного полета", которая могла бы взлетать вертикально, летать быстрее звука, нести сотни пассажиров и садиться на Луне. Также было бы ошибкой организовать ежегодный конкурс на повторение их первого полета с целью постепенного улучшения еловой древесины, из которой был сделан их биплан.

Уже было продемонстрировано, что работа над компонентами может стимулировать появление новых идей, например разработка генеративно-состязательных сетей (GAN) или языковой модели архитектуры "трансформер" привела открытию новых областей исследований. Также были отмечены успешные шаги в направлении достижения "разнообразия поведения". Например, системы машинного

перевода 1990-х годов создавались по одной для каждой языковой пары (например, с французского на английский), но сегодня единственная система способна идентифицировать входной текст как написанный на одном из ста языков и перевести его на любой из ста целевых языков. Другая система обработки естественного языка может выполнять пять различных типов заданий на основе одной совместной модели (Хашимото и др. [976], 2016).

Инженерия ИИ

Область компьютерного программирования началась с нескольких выдающихся пионеров. Но она не достигла статуса крупной отрасли, пока не была разработана практика создания программного обеспечения с мощным набором широко доступных инструментов и процветающей экосистемой учителей, студентов, практиков, предпринимателей, инвесторов и клиентов.

Индустрия ИИ еще не достигла такого уровня зрелости. Создано множество мощных инструментов и рабочих сред, таких как TensorFlow, Keras, PyTorch, Caffe, Scikit-Learn и SciPy. Но для многих из самых многообещающих подходов — таких, как GAN или глубокое обучение с подкреплением — было показано, что работать с ними очень трудно: требуются опыт и немалые хлопоты, чтобы заставить их правильно выполнять обучение в новой предметной области. У нас недостаточно экспертов, чтобы выполнить подобную работу во всех областях, в которых это требуется, и у нас пока нет инструментов и экосистемы, позволяющих преуспеть в этом деле менее опытным специалистам.

Джефф Дин из корпорации Google видит такое будущее, в котором нам потребуется добиться от систем машинного обучения справляться с миллионами разнообразных задач. Эта цель будет недостижима, если каждую из них разрабатывать с нуля, поэтому он предлагает вместо построения каждой новой системы с нуля начать с создания единственной огромной системы и для каждой новой задачи просто извлекать из нее те части, которые будут актуальны именно для этой задачи. Уже можно отметить некоторые шаги в этом направлении, — например, модели языка архитектуры "трансформер" (такие, как BERT, GPT-2) с миллиардами параметров и "возмутительно большая" ансамблевая архитектура нейронной сети, масштабируемая до 68 млрд параметров в одном эксперименте (Шазир и др. [2045], 2017). Многое еще предстоит сделать.

Будущее

Каким мы видим будущее? Авторы научно-фантастических романов, по-видимому, чаще публикуют пессимистические, а не оптимистические сценарии грядущего, возможно, потому, что это позволяет сочинять более увлекательные сюжеты. Но пока складывается впечатление, что ИИ развивается по такому же принципу, как и другие революционные технологии — книгопечатание, водопровод, воздухоплавание, телефония и т.д. Все эти технологии оказывали положительное влияние

на общество в целом, но одновременно с этим приводили к некоторым непреднамеренным побочными эффектами, которые непропорционально отражались на классах, находящихся в неблагоприятном положении. Было бы очень хорошо инвестировать необходимые средства в минимизацию возможных негативных последствий ИИ.

Однако у ИИ также имеется существенное отличие от предыдущих революционных технологий. Улучшение книгопечатания, водопровода, воздухоплавания и телефонии до их логических пределов не представляло никакой угрозы превосходству человека в этом мире. В противоположность этому, улучшение ИИ до его логического предела наверняка может ее представлять.

В заключение следует отметить, что за свою короткую историю ИИ достиг существенного прогресса, но последнее утверждение из эссе Алана Тьюринга "Computing Machinery and Intelligence" ([2235], 1950) продолжает оставаться неоспоримым и в наши дни.

Мы способны заглянуть вперед лишь на короткое расстояние,
но все равно можем видеть, как много еще предстоит сделать.

ПРИЛОЖЕНИЕ A

Математические основы

A.1. Анализ сложности и нотация $O()$

Специалистам в области компьютерных наук часто приходится решать задачу сравнения алгоритмов для определения того, насколько быстро они действуют или сколько памяти для них требуется. Для решения этой задачи предусмотрены два подхода. Первым из них является применение эталонных тестов (►**benchmarking**) — прогон реализующих алгоритмы программ на компьютере и измерение их быстродействия в секундах и потребления памяти в байтах. Безусловно, в конечном итоге нас действительно интересуют именно такие практические характеристики, но эталонное тестирование может оказаться неудовлетворительным просто потому, что оно слишком специфично: в нем измеряется производительность конкретной программы, написанной на конкретном языке, выполняемой на конкретном компьютере с конкретным компилятором и с конкретными входными данными. К тому же на основании единственного результата, полученного с помощью эталонного тестирования, очень трудно предсказать, насколько успешно этот алгоритм будет действовать в случае использования другого компилятора, компьютера или набора данных. Второй подход предполагает математический ►**анализ алгоритмов**, не зависящий от их конкретной реализации и входных данных. Именно этот подход будет обсуждаться ниже.

A.1.1. Асимптотический анализ

Подход, основанный на математическом анализе алгоритмов и не зависящий от их конкретной реализации и входных данных, мы рассмотрим на примере приведенной ниже программы, вычисляющей сумму последовательности чисел.

```
function SUMMATION(последовательность) returns число
  сумма ← 0
  for i = 1 to LENGTH(последовательность) do
     сумма ← сумма + последовательность[i]
  return сумма
```

Первый этап анализа состоит в том, что создается определенное абстрактное представление входных данных, позволяющее найти какой-то параметр или

параметры, характеризующие объем входных данных. В рассматриваемом примере объем входных данных можно охарактеризовать с помощью такого параметра, как длина последовательности, которую мы обозначим как n. На втором этапе необходимо определить абстрактное представление реализации и найти какой-то критерий, отражающий продолжительность выполнения алгоритма, но не привязанный к конкретному компилятору или компьютеру. Применительно к программе SUMMATION этим критерием может служить количество строк выполняемого кода; кроме того, данный критерий может быть более детализированным и измеряющим количество сложений, присваиваний, обращений к элементам массивов, а также ветвлений, выполняемых в этом алгоритме. В любом случае будет получена характеристика общего количества шагов, выполняемых алгоритмом, как функция от объема входных данных. Обозначим эту характеристику как $T(n)$. Если за основу берется количество строк кода, то в данном примере $T(n) = 2n + 2$.

Если бы все программы были такими же простыми, как SUMMATION, то область анализа алгоритмов не заслуживала бы названия научной. Но исследования в этой области существенно усложняются из-за наличия двух проблем. Первая проблема заключается в том, что редко удается найти параметр, подобный $T(n)$, который бы полностью характеризовал количество шагов, выполняемых при прогоне алгоритма. Вместо этого чаще всего можно лишь вычислить данный показатель для наихудшего случая $T_{worst}(n)$ или для среднего случая $T_{avg}(n)$. Причем, для вычисления среднего показателя необходимо, чтобы аналитик принял какие-то обоснованные предположения в отношении распределения, характеризующего набор входных данных.

Вторая проблема состоит в том, что алгоритмы обычно не поддаются точному анализу. В этом случае приходится прибегать к аппроксимации. В нашем случае, например, можно было бы сказать, что быстродействие алгоритма SUMMATION характеризуется величиной $O(n)$, имея в виду, что быстродействие этого алгоритма измеряется величиной, пропорциональной n, — возможно, за исключением нескольких небольших значений n. Более формально это определение можно представить с помощью следующей формулы.

$$T(n) \text{ есть } O(f(n)), \text{ если } T(n) \le kf(n) \text{ для некоторого } k, \text{ для всех } n > n_0$$

Перейдя к использованию нотации $O(\ )$, мы получаем возможность воспользоваться так называемым ►**асимптотическим анализом**. В рамках этого подхода можно, например, безоговорочно утверждать, что если n асимптотически приближается к бесконечности, то алгоритм, характеризующийся показателем $O(n)$, проявляет себя лучше по сравнению с алгоритмом $O(n^2)$, тогда как единственное число, полученное с помощью эталонного тестирования, не может служить обоснованием подобного утверждения.

Нотация $O(\ )$ позволяет создать абстрактное представление, в котором не учитываются постоянные коэффициенты, благодаря чему она становится более простой в использовании, но менее точной, чем нотация $T(\ )$. Например, в конечном

итоге алгоритм $O(n^2)$ всегда будет считаться худшим по сравнению с алгоритмом $O(n)$, но если бы эти два алгоритма характеризовались значениями $T(n^2+1)$ и $T(100n+1000)$, то фактически алгоритм $O(n^2)$ был бы лучше при $n<110$.

Несмотря на этот недостаток, асимптотический анализ представляет собой наиболее широко используемый инструмент анализа алгоритмов. Этот метод можно считать достаточно точным, поскольку в процессе анализа создается абстрактное представление и для точного количества операций (поскольку игнорируется постоянный коэффициент k), и для точного содержимого входных данных (поскольку рассматривается исключительно их объем n); благодаря чему анализ становится вполне осуществимым с использованием математических методов. Система обозначений $O()$ представляет собой хороший компромисс между точностью и простотой анализа.

А.1.2. Изначально сложные и недетерминированные полиномиальные задачи

Анализ алгоритмов и нотация $O()$ позволяют рассуждать об эффективности конкретного алгоритма. Однако эти методы не позволяют определить, может ли существовать лучший алгоритм для рассматриваемой задачи. В области ▶**анализа сложности** исследуются задачи, а не алгоритмы. Первая широкая градация в этой области проводится между задачами, которые могут быть решены за время, измеряемое полиномиальным соотношением, и задачами, которые не могут быть решены за время, измеряемое полиномиальным соотношением, независимо от того, какой алгоритм для этого используется. Класс полиномиальных задач — т.е. задач, которые могут быть решены за время $O(n^k)$ для некоторого k), — обозначается как ▶**P**. Эти задачи иногда называют "простыми", поскольку данный класс содержит задачи, имеющие такую продолжительность выполнения, как $O(\log n)$ и $O(n)$. Но он содержит и задачи с затратами времени $O(n^{1000})$, поэтому определение "простая" не следует понимать слишком буквально.

Другим важным классом является ▶**NP** (Nondeterministic Polynomial) — класс недетерминированных полиномиальных задач. Задача относится к этому классу, если существует алгоритм, позволяющий выдвинуть гипотезу о возможном решении, а затем проверить правильность этой гипотезы с помощью полиномиальных затрат времени. Идея такого подхода состоит в том, что если бы можно было воспользоваться сколь угодно большим количеством процессоров, чтобы проверить одновременно все гипотезы, или оказаться крайне удачливым и всегда с первого раза находить правильную гипотезу, то NP-трудные задачи стали бы P-трудными задачами. Одним из самых важных нерешенных вопросов в компьютерных науках является то, будет ли класс NP эквивалентным классу P, если нельзя воспользоваться бесконечным количеством процессоров или способностью находить правильную гипотезу с первого раза. Большинство специалистов в области компьютерных наук согласны с тем, что $P \neq NP$, иными словами, что NP-задачи

являются изначально трудными и для них не существует алгоритмов с полиномиальными затратами времени. Но это утверждение так и не было доказано.

Ученые, пытающиеся найти ответ на вопрос о том, эквивалентны ли классы P и NP, выделили подкласс класса NP, называемый ►**NP-полными** задачами. В этой формулировке слово "полный" означает "являющийся наиболее ярким представителем" и поэтому указывает на самые трудные задачи из класса NP. Было доказано, что либо все NP-полные задачи принадлежат к классу P, либо ни одна из них к нему не относится. Таким образом, данный класс представляет определенный теоретический интерес, но он важен также с точки зрения практики, поскольку известно, что многие серьезные задачи являются NP-полными. В качестве примера можно указать задачу установления выполнимости: если дано высказывание логики высказываний, то есть ли такой вариант присваивания истинностных значений символам высказывания, при котором оно становится истинным? Если не произойдет чудо и не совпадут друг с другом классы P и NP, то нельзя будет найти алгоритм, который позволяет решить *все* задачи установления выполнимости за полиномиальное время. Но исследователей в области искусственного интеллекта в большей степени интересует то, существуют ли алгоритмы, действующие достаточно эффективно при решении *типичных задач*, выбранных с помощью заранее заданного распределения; как было показано в главе 7, существуют алгоритмы наподобие WalkSAT, которые действуют вполне успешно при решении многих задач.

Класс ►**NP-сложных** задач состоит из тех задач, которые сводятся (за полиномиальное время) ко всем проблемам в NP, поэтому, если вы решили любую NP-сложную задачу, вы сможете решить все проблемы в NP. Все NP-полные задачи являются NP-сложными, но есть некоторые NP-сложные проблемы, которые даже сложнее, чем NP-полные.

Класс ►**co-NP** является комплементарным (дополнительным) по отношению к классу NP в том смысле, что для каждой задачи принятия решения из класса NP существует соответствующая задача в классе co-NP, на которую может быть дан положительный или отрицательный ответ, противоположный ответу на задачу класса NP. Известно, что класс P является подмножеством и NP, и co-NP, кроме того, считается, что к классу co-NP относятся некоторые задачи, не входящие в класс P. Такие задачи называются ►**co-NP-полными** и являются самыми трудными задачами в классе co-NP.

Класс #P (произносится как "шарп P" или "диез P") представляет собой множество задач подсчета количества вариантов, соответствующих задачам принятия решения из класса NP. Задачи принятия решения имеют однозначный (положительный или отрицательный) ответ. Примером задачи такого типа является следующая: "Существует ли решение для данной формулы 3-SAT?" Задачи подсчета количества вариантов имеют целочисленный ответ, например: "Сколько решений существует для данной формулы 3-SAT?" В некоторых случаях задача подсчета количества вариантов намного труднее по сравнению с соответствующей задачей

принятия решения. Например, принятие решения о том, имеет ли двухдольный граф идеальное сочетание пар, может быть выполнено за время $O(VE)$ (где V — количество вершин; E — количество ребер графа), тогда как задача подсчета того, какое количество идеальных сочетаний пар имеется в этом двухдольном графе, является #P-полной. Это означает, что она не менее трудна, чем любая задача из класса #P, и поэтому по меньшей мере столь же трудна, как и любая задача NP.

Другим классом является класс задач PSPACE. К нему относятся задачи, для которых требуется объем пространства, определяемый полиномиальной зависимостью, даже при их прогоне на недетерминированной машине. Считается, что PSPACE-трудные задачи решаются хуже NP-полных задач, но не исключено, что в ходе дальнейших исследований может быть установлено, что класс NP эквивалентен классу PSPACE и класс P эквивалентен классу NP.

А.2. Векторы, матрицы и линейная алгебра

В математике ▶**вектор** определяется как элемент векторного пространства, но мы будем использовать более конкретное определение: *вектор* — это упорядоченная последовательность значений. Например, в двухмерном пространстве могут быть определены такие векторы, как $\mathbf{x} = \langle 3, 4 \rangle$ и $\mathbf{y} = \langle 0, 2 \rangle$. В этом приложении соблюдаются обычные соглашения об обозначении векторов с помощью полужирных символов, хотя некоторые авторы отмечают имена векторов с помощью стрелок ($\vec{x}$) или знаков надчеркивания ($\overline{y}$). Элементы вектора обозначаются с помощью подстрочных индексов: $\mathbf{z} = \langle z_1, z_2, \ldots, z_n \rangle$. К сожалению, в этой книге синтезированы работы из многих областей исследований, где упорядоченные последовательности элементов могут называться по-разному ("векторы", "списки" или "кортежи"), в соответствии с чем для них используются и разные нотации: $\langle 1, 2 \rangle$, $[1, 2]$ или $(1, 2)$.

Двумя фундаментальными операциями над векторами являются векторное сложение и скалярное умножение. Векторное сложение $\mathbf{x} + \mathbf{y}$ — это поэлементная сумма: $\mathbf{x} + \mathbf{y} = \langle 3 + 0, 4 + 2 \rangle = \langle 3, 6 \rangle$, а скалярное умножение — это операция умножения каждого элемента вектора на некоторую константу: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$.

Длина вектора обозначается как $|\mathbf{x}|$ и вычисляется путем извлечения квадратного корня из суммы квадратов его элементов: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)} = 5$. Точечное произведение (называемое также скалярным произведением) двух векторов, $\mathbf{x} \cdot \mathbf{y}$, представляет собой сумму произведений их соответствующих элементов; иными словами, $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$, или в данном конкретном случае: $\mathbf{x} \cdot \mathbf{y} = 3 \times 0 + 4 \times 2 = 8$.

Векторы часто интерпретируются как направленные отрезки прямых (подобные стрелкам) в n-мерном евклидовом пространстве. В таком случае операция сложения векторов эквивалентна совмещению конца одного вектора с началом другого, а точечное произведение $\mathbf{x} \cdot \mathbf{y}$ эквивалентно выражению $|\mathbf{x}| \cdot |\mathbf{y}| \cdot \cos\theta$, где θ — угол между векторами $\mathbf{x}$ и $\mathbf{y}$.

►**Матрица** представляет собой прямоугольный массив значений, упорядоченных по строкам и столбцам. Ниже показана матрица **A** размером 3×4.

$$\begin{pmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \mathbf{A}_{1,3} & \mathbf{A}_{1,4} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \mathbf{A}_{2,3} & \mathbf{A}_{2,4} \\ \mathbf{A}_{3,1} & \mathbf{A}_{3,2} & \mathbf{A}_{3,3} & \mathbf{A}_{3,4} \end{pmatrix}$$

Первый подстрочный индекс в обозначении $\mathbf{A}_{i,j}$ определяет строку, а второй — столбец. В языках программирования $\mathbf{A}_{i,j}$ часто записывается как `A[i,j]` или `A[i][j]`.

Сумма двух матриц определяется путем сложения соответствующих элементов, поэтому $(\mathbf{A} + \mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}$. (Если матрицы **A** и **B** имеют разные размеры, то их сумма не определена.) Можно также определить операцию умножения матрицы на скаляр: $(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}$. Операция умножения матриц (получения произведения двух матриц) является более сложной. Произведение **AB** определено, только если матрица **A** имеет размеры $a \times b$, а матрица **B** имеет размер $b \times c$ (т.е. вторая матрица имеет количество строк, совпадающее с количеством столбцов первой матрицы). Результатом этой операции является матрица с размерами $a \times c$. Если матрицы имеют допустимые размеры, то результат их умножения является следующим:

$$(\mathbf{AB})_{i,k} = \sum_j \mathbf{A}_{i,j}\mathbf{B}_{j,k}.$$

Умножение матриц не является коммутативным даже для квадратных матриц: $\mathbf{AB} \neq \mathbf{BA}$ в любом случае. Однако эта операция является ассоциативной: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$. Обратите внимание, что точечное произведение векторов может быть выражено в терминах транспонирования и умножения матриц: $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^\top \mathbf{y}$.

►**Единичная матрица I** имеет элементы $\mathbf{I}_{i,j}$, равные 1, если $i = j$, и равные 0 в противном случае. Она обладает таким свойством, что $\mathbf{AI} = \mathbf{A}$ для всех матриц **A**.

►**Транспонирование** — это специфическая для матриц операция (записывается как $\mathbf{A}^\top$), заключающаяся в превращении строк матрицы в столбцы и обратно, или, более формально, $\mathbf{A}^\top_{i,j} = \mathbf{A}_{j,i}$. Если матрица **A** квадратная, то ее ►**обратная матрица** $\mathbf{A}^{-1}$ определяется как такая матрица, для которой $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Для ►**вырожденной матрицы** обратной матрицы не существует, а для невырожденной матрицы обратная матрица может быть вычислена за время $O(n^3)$.

Матрицы используются для решения систем линейных уравнений за время $O(n^3)$, которое необходимо для инвертирования матрицы, составленной из коэффициентов уравнений. Рассмотрим следующую систему уравнений, для которой требуется найти решение в терминах переменных x, y и z.

$$\begin{aligned} +2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

Эту систему уравнений можно представить в виде матричного уравнения $\mathbf{Ax} = \mathbf{b}$, где

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}.$$

Чтобы решить уравнение $\mathbf{Ax} = \mathbf{b}$, достаточно умножить обе его стороны на обратную матрицу $\mathbf{A}^{-1}$, в результате чего получим $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$, что можно упростить до $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Проинвертировав матрицу $\mathbf{A}$ и умножив результат на вектор $\mathbf{b}$, получим искомый ответ:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}.$$

Еще несколько замечаний по поводу используемых в книге обозначений.

1. Мы используем запись $\log(x)$ для представления натуральных логарифмов $\log_e(x)$.
2. Мы используем запись $\text{argmax}_x f(x)$ для ссылок на значение x, при котором значение функции $f(x)$ является максимальным.

А.3. Распределения вероятностей

Вероятность — это мера, заданная на множестве событий, которая удовлетворяет трем приведенным ниже аксиомам.

1. Мера каждого события находится в пределах от 0 до 1. Это утверждение записывается как $0 \leq P(X = x_i) \leq 1$, где X — случайная переменная, представляющая событие, а x_i — возможные значения X. Обычно принято обозначать случайные переменные прописными буквами, а их значения — соответствующими строчными.
2. Мера всего множества равна 1, а это означает, что $\sum_{i=1}^{n} P(X = x_i) = 1$.
3. Вероятность объединения взаимоисключающих событий равна сумме вероятностей отдельных событий, т.е. $P(X = x_1 \vee X = x_2) = P(X = x_1) + P(X = x_2)$, гдесобытия x_1 и x_2 являются взаимоисключающими.

Вероятностная модель состоит из пространства выборок взаимоисключающих возможных результатов вместе с вероятностной мерой для каждого результата. Например, в модели погоды на завтра результатами могут быть *sunny* (солнечно), *cloudy* (облачно), *rainy* (дождь) и *snowy* (снег). Подмножество этих результатов представляет собой событие. Например, событие выпадения осадков — это подмножество $\{rainy, snowy\}$.

Мы будем использовать запись $\mathbf{P}(X)$ для обозначения вектора значений $\langle P(X=x_1),\ldots, P(X=x_n)\rangle$. Мы также используем запись $P(x_i)$ в качестве сокращения для $P(X=x_i)$ и $\sum_x P(x)$ в качестве сокращения для $\sum_{i=1}^{n} P(X=x_i)$.

Условная вероятность $P(B|A)$ определяется как $P(B \cap A)/P(A)$. Переменные A и B являются условно независимыми, если $P(B|A) = P(B)$ (или, равным образом, если $P(A|B) = P(A)$).

Непрерывные переменные имеют бесконечное количество значений, и если распределение вероятностей этих значений не характеризуется наличием пиковых значений в отдельных точках, то вероятность любого отдельно взятого значения равна 0. Поэтому имеет смысл говорить о значении вероятности в пределах некоторого диапазона. Это реализуется с помощью ▶**функции плотности вероятности**, которая имеет немного иной смысл по сравнению с дискретной функцией вероятности. Поскольку вероятность $P(X=x)$ — т.е. вероятность, которую X имеет при точном значении x — равна нулю, мы будем использовать другую меру: отношение вероятности того, что X попадает в интервал вокруг x, к ширине этого интервала, измеряемой в пределе приближения ширины интервала к нулю. Итак, функция плотности вероятности $P(X=x)$ определяется как

$$P(x) = \lim_{dx\to 0} P(x \le X \le x+dx)/dx.$$

Функция плотности вероятности должна быть неотрицательной при всех x и соответствовать следующему требованию:

$$\int_{-\infty}^{\infty} P(x)dx = 1.$$

Мы также можем определить ▶**кумулятивное распределение** $F_X(x)$, которое является вероятностью того, что случайная величина будет меньше x. Кумулятивная функция распределения определяется следующим образом:

$$F_X(x) = P(X \le x) = \int_{-\infty}^{x} P(u)du.$$

Следует отметить, что функция плотности вероятности измеряется в определенных единицах, а дискретная функция вероятности является безразмерной. Например, если переменная X измеряется в секундах, то плотность вероятности измеряется в герцах (Гц, т.е. 1/с). Если $\mathbf{X}$ — точка в трехмерном пространстве, измеряемом в метрах, то плотность вероятности будет измеряться в $1/м^3$.

Одним из наиболее важных распределений вероятностей является ▶**распределение Гаусса**, известное также под названием ▶**нормальное распределение**. Для этого распределения мы используем обозначение $\mathcal{N}(x; \mu, \sigma^2)$ — как функции от x со средним значением μ и среднеквадратичным отклонением σ (и, следовательно, с дисперсией σ^2). Нормальное распределение определяется следующей формулой:

$$\mathcal{N}(x;\mu;\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)},$$

где x — непрерывная переменная, изменяющаяся в пределах от $-\infty$ до $+\infty$. Если среднее $\mu = 0$ и дисперсия $\sigma^2 = 1$, то имеет место частный случай нормального распределения, называемый ►**стандартным нормальным распределением**. Если распределение задано на векторе $\mathbf{x}$ в пространстве с d измерениями, то оно представляет собой ►**многомерное гауссово распределение**:

$$\mathcal{N}(\mathbf{x};\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}\left((\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right)},$$

где μ — вектор средних, а Σ — ►**матрица ковариации** этого распределения (см. ниже). Кумулятивное распределение для одномерного нормального распределения определяется как

$$F(x) = \int_{-\infty}^{x} \mathcal{N}(z;\mu,\sigma^2)dz = \frac{1}{2}(1 + \mathrm{erf}(\tfrac{x-\mu}{\sigma\sqrt{2}})),$$

где $\mathrm{erf}(x)$ — так называемая ►**функция ошибок**, не имеющая представления в замкнутой форме.

В ►**центральной предельной теореме** утверждается, что среднее n случайных переменных приближается к нормальному распределению по мере того, как n стремится к бесконечности. Такому свойству подчиняется почти любая коллекция случайных переменных, при том условии, что значение дисперсии любого конечного подмножества переменных не доминирует над значениями дисперсии других конечных подмножеств.

►**Математическое ожидание** случайной величины, $M(X)$, представляет собой среднее значение, взвешенное по вероятности каждого значения. Для дискретной переменной это

$$M(X) = \sum_x x_i P(X = x_i).$$

Для непрерывной переменной суммирование следует заменить интегралом и использовать функцию плотности вероятности, $P(x)$:

$$M(X) = \int_{-\infty}^{\infty} xP(x)dx.$$

Для любой функции f мы также имеем

$$M(f(X)) = \int_{-\infty}^{\infty} f(x)P(x)dx.$$

И наконец, при необходимости математическое ожидание можно определить через распределение случайной величины:

$$M_{X\sim Q(x)}(g(X)) = \int_{-\infty}^{\infty} g(x)Q(x)dx.$$

Помимо математического ожидания, другие важные статистические характеристики распределения включают ▶**дисперсию** $D[X]$, которая является математическим ожиданием квадрата отклонения случайной величины от ее математического ожидания:

$$D[X] = M((X - M(X))^2)$$

и среднеквадратическое или ▶**стандартное отклонение** σ, которое является квадратным корнем из дисперсии (по этой причине математическое ожидание в статистике часто обозначают как σ^2).

▶**Среднее квадратическое** s для набора значений (чаще всего это выборка значений случайной величины) представляет собой квадратный корень из среднего арифметического квадратов значений в наборе:

$$s(x_1,\ldots,x_n) = \sqrt{\frac{x_1^2 + \ldots + x_n^2}{n}}.$$

▶**Ковариация** двух случайных величин определяется как математическое ожидание произведения разностей их значений и соответствующих средних значений:

$$\text{cov}(X, Y) = M((X - \mu X)(Y - \mu Y)).$$

▶**Ковариационная матрица**, часто обозначаемая как S, является матрицей, составленной из попарных ковариаций между элементами вектора случайных величин. Принимая, что вектор представлен как $\mathbf{X} = \langle X_1,\ldots X_n\rangle^\top$, элементами ковариационной матрицы будут:

$$S_{i,j} = \text{cov}(X_i, X_j) = M((X_i - \mu_i)(X_j - \mu_j)).$$

▶**Выборка** — это часть общей совокупности возможных элементов случайной величины, полученная в результате некоторого эксперимента. Мы не знаем, каково будет распределение вероятности для любой конкретной выборки, но в пределе достаточно большой набор данных выборок будет приближаться к той же функции распределения плотности вероятности, которая свойственна общей совокупности элементов этой случайной величины. ▶**Равномерное распределение** характеризуется тем, что каждый элемент в общей совокупности в равной степени (равномерно) вероятен. Поэтому, если мы говорим, что "случайная величина имеет равномерное распределение в диапазоне целых чисел от 0 до 99", то это означает, что при выборке с одинаковой вероятностью может быть получено любое целое число из этого диапазона.

Библиографические и исторические заметки

Система обозначений $O(\ )$, которая широко используется в компьютерных науках в наши дни, была впервые определена в контексте теории чисел немецким математиком П.Г.Н. Бахманом (1894). Понятие NP-полноты было предложено Куком (1971), а современный метод определения способа сведения одной проблемы к другой предложен Карпом (1972). И Кук, и Карп получили за свою работу премию Тьюринга — высшую награду в компьютерных науках.

Из лучших учебников по анализу и разработке алгоритмов следует упомянуть книги Седвига и Уэйна (2011), а также Кормена, Лейзерсона, Ривеста и Штейна (2009). В этих изданиях особое внимание уделяется разработке и анализу алгоритмов для решения поддающихся решению задач. Теория NP-полноты и других форм неразрешимости представлена в книгах Гэри и Джонсона (1979) и Пападимитриу (1994). Хорошими учебниками по теории вероятности являются книги Чунга (1979), Росса (2015), а также Бертсекаса и Цициклиса (2008).

ПРИЛОЖЕНИЕ Б

Сведения о языках и алгоритмах, используемых в книге

Б.1. Определение языков с помощью формы Бэкуса–Наура

В этой книге дается определение нескольким языкам, в том числе языку логики высказываний (раздел 7.4), языку логики первого порядка (раздел 8.2) и подмножеству английского языка (раздел 23.4). Формальный язык определяется как множество строк, в котором каждая строка представляет собой последовательность символов. Все языки, которые были необходимы нам в этой книге, состоят из бесконечного множества строк, поэтому нужен простой способ, позволяющий охарактеризовать это множество. Для достижения данной цели удобнее всего воспользоваться **грамматикой**. Конкретный тип грамматики, который мы выбрали, носит название ►**контекстно свободная грамматика**, поскольку каждое выражение в этом случае будет иметь одну и ту же форму в любом контексте. В качестве способа оформления грамматик мы приняли формальную систему под названием ►**форма Бэкуса–Наура** или **БНФ** (*Backus-Naur form — BNF*). В любой БНФ-грамматике присутствуют четыре компонента.

- Множество ►**терминальных символов**. Это символы или слова, из которых состоят строки языка. В качестве таких символов могут использоваться буквы (**A**, **B**, **C**, ...), слова (**a**, **aardvark**, **abacus**, ...) или любые символы, являющиеся допустимыми для области определения.
- Множество ►**нетерминальных символов**, которые обозначают компоненты предложений языка. Например, нетерминальный символ *NounPhrase* (именное словосочетание) в английском языке обозначает бесконечное множество строк, включая такие, как *you* или *the big slobbery dog*.
- ►**Начальный символ**, который представляет собой нетерминальный символ, обозначающий законченный набор из строк языка. В грамматике английского языка таковым является символ *Sentence*; в грамматике

арифметических выражений для этой цели может быть определен символ *Expr*, а в грамматике языка программирования это может быть *Program*.

- Множество ▶**правил подстановки** в форме $LHS \rightarrow RHS$, где *LHS* — нетерминальный символ, а *RHS* — последовательность, в которую входят нуль и больше символов. Это могут быть как терминальные, так и нетерминальные символы, а также символ ϵ, который мы будем использовать для обозначения пустой строки.

Правило подстановки вида

$Sentence \rightarrow NounPhrase\ VerbPhrase$

означает, что при наличии двух строк, обозначенных как *NounPhrase* (именное словосочетание) и *VerbPhrase* (глагольное словосочетание), их можно соединить друг с другом и классифицировать результат как *Sentence*. Определение этого правила также можно записать в сокращенном виде: если есть два правила ($S \rightarrow A$) и ($S \rightarrow B$), то можно записать ($S \rightarrow A \mid B$). Чтобы проиллюстрировать эту концепцию, ниже приведена БНФ-грамматика для простых арифметических выражений.

Expr	→	*Expr Operator Expr* | (*Expr*) | *Number*
Number	→	*Digit* | *Number Digit*
Digit	→	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
Operator	→	+ | − | ÷ | ×

Более подробные сведения о языках и грамматиках приведены в главе 23. Следует отметить, что в других книгах в системе обозначений БНФ могут использоваться немного другие правила, например нетерминальные символы могут обозначаться ⟨*Digit*⟩, а не *Digit*, терминальные символы — 'word', а не **word**, а в правилах может использоваться символ ::= вместо →.

Б.2. Описание алгоритмов с помощью псевдокода

В этой книге все упоминаемые алгоритмы представлены на ▶**псевдокоде**. Основные конструкции этого псевдокода должны быть понятны пользователям таких языков, как Java, C++ и в особенности Python. В некоторых местах для описания вычислений в псевдокоде используются математические формулы или текст на естественном языке, поскольку в противном случае пришлось бы применять более громоздкие конструкции. Также следует отметить, что в алгоритмах используются приведенные ниже соглашения.

- **Сохраняемые переменные**. Ключевое слово **persistent** (сохраняемая) используется как указание на то, что переменной присваивается начальное значение при первом вызове содержащей ее функции, после чего это

значение (или значение, присвоенное переменной с помощью выполненных в дальнейшем операторов присваивания) сохраняется при всех последующих вызовах функции. Таким образом, сохраняемые переменные подобны глобальным переменным в том, что они сохраняют свое значение после каждого вызова содержащей их функции, но при этом доступны только в данной функции. В программах агентов, приведенных в данной книге, сохраняемые переменные используются в качестве "памяти". В объектно-ориентированных языках, таких как C++, Java, Python и Smalltalk, программы с сохраняемыми переменными могут быть реализованы в виде *объектов*. В функциональных языках они могут быть реализованы с помощью *функциональных замыканий* в той среде, в которой определены требуемые переменные.

- **Функции как значения.** В нашем псевдокоде имена функций и процедур начинаются с прописных букв, а имена переменных состоят из строчных букв и выделяются курсивом. Поэтому в большинстве случаев вызов функции выглядит наподобие FN(x). Однако также допускается, чтобы значением переменной была функция, например если значением переменной f является функция вычисления квадратного корня, то выражение $f(9)$ возвращает 3.
- **Значимость отступов**. Наличие отступов при записи псевдокода имеет большое значение — по аналогии с языками Python и CoffeeScript, но в отличие от языков Java, C++ и Go (в которых используются скобки) или языков Lua и Ruby (в которых используется оператор **end**), в нашем псевдокоде они определяют область действия цикла или условного выражения.
- **Деструктуризация наборов данных**. Нотация "x, $y \leftarrow$ *пара*" означает, что правая часть выражения после вычисления должна представлять собой двухэлементную коллекцию, первый элемент которой присваивается переменной x, а второй — y. Та же самая идея может быть представлена записью "**for** x, y **in** *пара* **do**". Эта нотация также может использоваться для обмена значениями между двумя переменными: "x, $y \leftarrow y$, x".
- **Значения по умолчанию для параметров**. Нотация "**function** F(x, y = 0) **returns** число" означает, что в этой функции y является дополнительным аргументом со значением по умолчанию, равным 0. Иначе говоря, вызовы функции F(3, 0) и $F(3)$ являются эквивалентными.
- **Ключевое слово yield**. Функция, содержащая ключевое слово **yield**, представляет собой ►**генератор**, генерирующий *последовательность* значений, — по одному каждый раз, когда встречается выражение, содержащее это ключевое слово. После генерации очередного значения функция продолжает свое выполнение со следующего оператора. В языках Python, Ruby, C# и JavaScript (ECMAScript) присутствует подобная функциональная возможность.

- **Циклы.** В псевдокоде допускается четыре типа циклов.
  - "**for** *x* **in** *c* **do**" — выполняется цикл с управляющей переменной *x*, возможные значения которой указаны как последовательность элементов в коллекции *c*.
  - "**for** *i* = 1 **to** *n* **do**" — выполняется цикл с управляющей переменной *i*, значения которой представляют собой последовательность целых чисел от 1 до *n* включительно.
  - "**while** *условие* **do**" — в этом цикле выполнение *условия* проверяется перед началом очередной его итерации, и если условие ложно, цикл завершается.
  - "**repeat** ... **until** *условие*" — этот цикл безусловно выполняется первый раз, а затем проверяется *условие*. Если оно истинно, цикл завершается, иначе он выполняется вновь (и в конце опять проверяется *условие*).
- **Списки.** Нотация [*x*, *y*, *z*] определяет список из трех элементов. Оператор "+" может употребляться для выполнения конкатенации списков: [1, 2] + [3, 4] = [1, 2, 3, 4]. Список может использоваться и как стек: функция Pop удаляет и возвращает последний элемент списка, а функция Top возвращает его последний элемент.
- **Множества.** Нотация {*x*, *y*, *z*} определяет множество (набор) из трех элементов. Запись {*x*: *p*(*x*)} определяет множество из всех элементов *x*, для которых *p*(*x*) истинно.
- **Индексация в массивах начинается с 1.** Первый элемент массива всегда имеет индекс 1, как это принято в обычной математической записи (и в языках R и Julia), а не 0 (как это принято в языках Python, Java и C).

Б.3. Дополнительный материал в Интернете

Для этой книги имеется собственный веб-сайт, содержащий дополнительные материалы и инструкции по отправке предложений, а также предоставляющий посетителям возможность присоединиться к дискуссионным группам.

- `aima.cs.berkeley.edu`

Все приведенные в этой книге алгоритмы и несколько дополнительных упражнений по программированию были реализованы на языках Python и Java (а некоторые и на других языках). Код этих реализаций находится в интернет-хранилище и доступен на веб-сайте, который в настоящее время размещается по адресу

- `github.com/aimacode`

Предметный указатель

χ2-обрезка 32

A

AFSM 512
Animatronics 517
AUC 100
AutoML 114
Autonomatronics 517
AUV 449

B

Benchmarking 605
BPTT 202

C

C-пространство 467
  препятствие 468
CART 34
CNN 182, 404
co-NP-задачи 608
co-NP-полные задачи 608

D

DARPA Grand Challenge 522
Deep Q-network 268
DQN 268
Driver assist 516

E

ELBO 208
EM-алгоритм 146
ExtraTrees 82

F

FMEA 569
FTA 569

G

GLIE 241
GOFAI 538
GPS 452
  дифференциальный 452
GPU 186

H

HMM 155, 293
HTN-планирование 255

I

ILQR 491
ImageNet 93

K

k-анонимность 553
k-блочная перекрестная проверка 36
k-d-дерево 68
Keepaway 255

L

LOOCV 36
LQR 491
LSH 69
LSTM 203, 352

M

MAP 125
MDL 126
MDP 228, 454, 490
MEMS 588
ML 126
MLM 368
MPC 505

N

n-граммная модель 288
Netflix Prize 552
NLP 214, 283
NP-задачи 607
NP-полные задачи 608
NP-сложные задачи 608

O

One-hot-вектор 342

P

P-задачи 607
P-контроллер 486

PAC 46
PBT 45
PCFG 300
  лексическая 311
PD-контроллер 487
Penn Treebank 292
PID-контроллер 488
POMDP 454, 455, 493
POS 292
POS-тег 345
PPCA 206
PRM 478

Q

Q-обучение 229, 246
Q-функция 229

R

ReLU 170
RGB 391
RNN 200, 347
  двунаправленная 350
RoboCup 522
ROC 100
RPN 410
  немаксимальное подавление 410
  область интереса 410
  регрессия ограничивающей рамки 411
  якорная рамка 410
RRT 479
RRT* 481

S

SARSA 248
SGD 55, 188
SLAM 464
Softmax 178
Softplus 170
SSD 399
SVM 73

T

Tanh 170
TPU 186
t-SNE 98

U

UAV 449
Urban Challenge 523

V

VC-размерность 111

X

XAI 562

А

Абстракция состояний 276
Автокодировщик 207
  вариационный 208
Автоматизация 565
  бизнес-процессов 566
  темпы изменений 566
Автоматизированное машинное обучение 114
Автоматическое дифференцирование 175
Агент
  жадный 240
  обучающийся
    активный 239
    пассивный 230
Адаптивное динамическое программирование 233
Аддитивная декомпозиция 258
Активное обучение с подкреплением 239
Активный обучающийся агент 239
Акустическая модель 322
Алгоритм
  вероятностно полный 478
  Витерби 294
  внутренний-внешний 309
  гибридный A* 478
  градиентного
    спуска 53
    учета временной разности 275
  обратного распространения ошибки во времени 202
  обучения
    вне стратегии 248
    на основе стратегии 248

ожидание–максимизация 146
оперативного обучения 250
прямой–обратный 155
рандомизированного взвешенного большинства 90
ранняя остановка 32
соответствия признаков 265
с отсечением по времени 598
стохастического градиентного спуска 189
ядерная версия 79
CYK 303
DECISION-LIST-LEARNING 50
EM 146
общая форма 156
структурный 158
LEARN-DECISION-TREE 24
MCMC 157
MODEL-SELECTION 37
RRT* 481
Альбедо 390
Анализ
алгоритмов 605
асимптотический 606
видов и последствий отказов 569
данных предварительный 17
дерева отказов 569
Ляпунова 525
синтаксический 302
сложности 607
тональности текста 288, 351
Ансамблевая модель 79
Ансамблевое обучение 79
Антропоморфный робот 449
Апертура 384, 387
Апостериорное распределение вариационное 208
Аппроксимация функции 249
Априорные вероятности гипотез, распределение 122
Аргумент
китайская комната 543
неспособности выполнения 539
неформализуемости поведения 538
положения математики 540
Архитектура
подчинения 512
трансформер 360
LSTM 203
Асимптотический анализ 606
Атрибут
выходной непрерывный 34
непрерывный 33
Аугментация 95

Б

Базовая модель 79
Байесовская
оптимизация 44
сеть, обучение структуры 141
Байесовское
обучение 88, 122
с подкреплением 244
предсказание 124
Безопасное агрегирование 554
Безопасность ИИ 569, 582
Беспилотный автомобиль 450, 515
Беспроигрышное обучение 91
Бета-распределение 135
Бинокулярные стереоданные 412
Биологический натурализм 543
Блик 389
Бритва Оккама 19
Булева классификация 23
Бустинг 84
градиентный 88
Быстрое кодирование 96, 176, 342
Быстро исследующие случайные деревья 479
Бэггинг 80
случайный лес 81

В

Вариационное апостериорное распределение 208
Вариационный автокодировщик 208
Вежливое соглашение 543
Вектор 609
встраивания слов 342
запроса 360

значения 360
ключа 360
предобученный 344
Вентиль 203
забывания 203
Верификация и валидация 561
Вероятностная дорожная карта 478
Вероятностная контекстно-свободная грамматика 300
Вероятность 611
класса 116
Взаимное отражение 390
Взвешенный обучающий набор 84
Взрывной градиент 203
Визуальное программирование 510
Визуальный диалог 424
Виртуальный счетчик 136
Вложение слов 215
Внешнее освещение 388
Внешний фактор 571
Внимание 355
множественное 361
Вознаграждение 228
за всю траекторию 232
редкое 228
Воксель 427
Воплощенное познание 539
ВПК-обучение 46
Время глагольное 318
Встраивание
позиционное 361
предобученное 364
слов 342
Входное кодирование 176
Выборка 614
коррелированная 262
повышающая 95
понижающая 95
Выбор модели 37
Выборочная сложность 47
Выброс 95
Вырожденная матрица 610
Выходной слой 173
Вычислительная нейробиология 222

Г

Гауссов процесс 162
Генеративная модель 132
Генеративно-состязательная сеть 210
Генератор 210
Гиперболический тангенс 170
Гиперпараметр 36, 135
ручная настройка 44
Гипотеза 20, 121
вероятностно приблизительно корректная 46
дисперсия 19
кривая обучения 27
максимальная апостериорная 124
недообучение 19
переобучение 19
потери при обобщении 41
приблизительно корректная 46
регуляризация 43
смещение 18
согласованная 17
Гистограмма векторного поля 525
Глагольная группа 301
Глобальная система позиционирования 452
Глубина резкости 387
Глубокое обучение 167
применение 213
слои 167
с подкреплением 216, 253
Градиентный
бустинг 88
спуск 53, 173
в динамическом режиме 55
Градиент стратегии 260
Грамматика 284, 298, 299
вероятностная контекстно-свободная 300
зависимости 306
замкнутые классы 302
категориальная 328
контекстно-свободная 310
нормальная форма Хомского 303
открытые классы 300
представление времени 318
расширенная 310

семантическая, изучение 316
универсальная 330
E0 300
Граница решения 59
Граф
видимости 473, 524
Вороного 474
вычислений 171, 190
полносвязный 173
потока данных 171

Д

Далекая зависимость 318
Дальномер 451
Дальностная камера 451
Данные 121
аугментация 95
выбросы 95
линейно разделимые 59
недостающие 33
нестационарность 104
обобщающее поле 552
обучающий набор 35
полные 127
проверочный набор 36
происхождение 94
разведочный анализ 97
тестовый набор 35
Датчик 447
активный 450
инерционный 453
крутящего момента 453
местоположения 452
пассивный 450
проприоцептивный 453
силы 453
тактильный 452
угла поворота 453
Движение Time well spent 592
Двоякое использование ИИ 550
Двунаправленная RNN 350
Двусмысленность 319
Дедукция 15
Действие необратимое 243
Декодер трансформера 362
Декодирование 357
Дельта-правило 250
Демографический паритет 555
Демонстрация ключевых кадров 510
Дерево
регрессии 34
решений 22
нестабильность 35
обрезка 31
обучение 24
точка разбиения 33
чрезвычайно рандомизированное 82
k-мерное 68
Детектор объектов 408
Детерминированный синтаксический анализ 306
Диаграмма 303
Вороного 474
Диаграммный синтаксический анализатор 303
Динамическая
модель 484
перепланировка 492
Динамическое
состояние 485
обучение 89
программирование 302
адаптивное 233
Дипфейк 431
Дискриминативная модель 132, 296
Дискриминатор 210
Дисперсия 19, 42
Дифференциальная приватность 553
Дифференцирование
автоматическое 175
обратный режим 175
Дифференцируемое программирование 594
Диффузное
альбедо 390
отражение 388
Длинный хвост 104
Добрый старомодный ИИ 538
Доверие системам ИИ 561

Доверительное состояние 457
Доказательная нижняя граница 208
Долгая краткосрочная память 203, 352
Дрон 449, 547
Дружественный ИИ 582

Е

Единичная матрица 610
Естественный язык, обработка 283

Ж

Жадное декодирование 357
Жадный
  агент 240
  поиск 295

З

Зависимость далекая 318
Заголовок фразы 311
Задача
  анализа тональности текста 351
  ассоциации данных 464
  классификации 350
  машинный перевод 352
  о бандитах 241
  одновременной локализации и построения карты 464
  переноса рояля 472
  разведочный POMDP 245
  разрешения кореферентности 349
  тележка–шест 269
  POS 344
Зазор 75
Закон
  косинуса Ламберта 390
  управления 486, 488
    компонент прямой связи 489
  Ципфа 375
Законы робототехники 578
Залипание 486
Замкнутый класс 302
Захват с параллельными губками 454
Зашумленная функция 42
Зеркальное отражение 389
Знание предварительное 15
Зондирование 381

И

Игра
  с неполной информацией 501
  содействия 573
  keepaway 255
Идентифицируемость 154
Иерархическое обучение с подкреплением 255
Извлечение информации 323, 331
Изображение 384
  края 393
  объекты 402
  сегментация 400
  текстура 397
  фокусирование 386
  шум 394
Изучение
  предпочтений 456
  полезности действий 229
Именная группа 301
Имитационное обучение 263, 508, 572
Импульс 189
Инверсная динамика 485
Индексикал 317
Индикаторная переменная 149
Индукция 15
Интеграл пути 482
Интеллектуальное обучение 595
Интерполяция данных моделью 39
Интерпретируемость 102
Интерфейс “мозг–машина” 514
Искусственный интеллект
  безопасность 569
  двоякое использование 550
  добрый старомодный 538
  доверие 561, 582
  объяснимый 562
  оценка 541
  реального времени 598
  сильный 537, 577
  слабый 537, 576
  сознание 544
  этика 545
Исполнительный механизм 447
Истинное значение 17

Исходный язык 352
Исчезновение градиента 175
Итерация по стратегиям
модифицированная 233

К

Камера видеонаблюдения 551
Камера-обскура 384
Карта
глубины 427
признаков 186
Катастрофическое забывание 252
Категория
подкатегории 310
синтаксическая 300
Квадратичное программирование 75
Квазилогическая форма 317
Квалиа 544
Квантификация 317
Кибербезопасность 551
Кинематика 470
Кинематическое состояние 485
Кинестетическое обучение 509
Китайская комната 543, 577
Класс
моделей 17
k-DT 49
Классификатор
мягко разделяющий 78
оптимально разделяющий 73
Классификация 15
Кластеризация 16
без учителя 147
Клеточная декомпозиция 475
Клонирование поведения 275, 508
Ключевой кадр 510
Ковариационная матрица 614
Ковариация 614
Кодекс этики ИИ 580
Колмогоровская сложность 110
Композиционная семантика 313
Компонент смешанного
распределения 147
Компромисс "смещение–дисперсия" 19
Компьютерное зрение 214, 418
изображение 384
оптический поток 398
реконструкция 425
сенсор 384
создание изображений 428
сцена 384
управление движением 432
Контекстное представление 366
Контекстно-свободная грамматика 310
Контроллер
закон управления 486, 488
коллизий 477
коэффициент усиления 486
стабильный 487
строго стабильный 487
Конфиденциальность 580
Координация 500
Корпус 286
Penn Treebank 292
Коррелированная выборка 262
Коэффициент
обесценивания 232
приращения информации 34
Край 393
ориентация 396
Краудсорсинг 93
Кривая обучения 27, 61
Критерий коэффициента приращения 118
Критик 98
Крупномасштабное обучение 42
Кумулятивное распределение 612

Л

Латентная переменная 145
Лексическая
категория 292
неоднозначность 319
PCFG 311
Линеаризация 463
Линейная
гауссова модель 132
регрессия
байесовская 138
множественная 56

Линейное интерполяционное сглаживание 290
Линейно-квадратичный регулятор 491
Линейный
  классификатор 63
  разделитель 59, 77
Линза 386
Личный агент 592
Логарифмическое правдоподобие 127, 149
Логистическая регрессия 64, 294
Ложно-положительная оценка 100
Локализация 458
  Монте-Карло 461
Локально взвешенная регрессия 72

M

Мадалина 218
Максимальная апостериорная гипотеза 124
Максимальное правдоподобие 126, 130, 132, 177
Манипулятор 449
Маркировка частей речи 292
Марковская локализация 523
Марковское предположение 200
Маскированная модель языка 368
Математическое ожидание 613
  признака 266
Матрица 610
  вырожденная 610
  датчиков расстояния 460
  единичная 610
  ковариации 613
  обратная 610
  ошибок 100
  факторов 56
    псевдообратная 57
Машина Больцмана 222
Машинное обучение 13
  автоматизированное 114
  роботов 465
Машинный перевод 323, 352, 378
Метаобучение 114
Метарассуждения на основе теории принятия решений 598
Метафора 321
Метка 16
Метод
  аппроксимации временной разности 251
  аугментации 95
  быстро исследующих случайных деревьев 479
  деревьев решений 22
  исключения 198
  итерационного LQR 491
  клеточной декомпозиции 475
  клонирования поведения 275
  обучения на демонстрации 263
  опорных векторов 73
  оптимизации доверительной области стратегии 276
  пакетной нормализации 192
  последовательного обучения 309
  расширения набора данных 406
  серийного спуска по градиенту 54
  стохастического градиентного спуска 55, 188
  HTN-планирования 255
  k-ближайших соседей 66, 143
  Q-обучения 246
  SARSA 248
  SVM, ядерный трюк 74
  t-SNE 98
Метонимия 320
Метрика Минковского 67
Механизм
  внимания 355
  самовнимания 360
Мешок слов 285
Минимальная длина описания 43, 126
Мини-пакет 55, 189
Многозадачное обучение 213
Многомерное гауссово распределение 613
Многоточечная пристрелка 491
Множественная линейная регрессия 56
Множественное внимание 361
Мобильный робот 449

Модель
автокодировщик 207
авторегрессии 209
акустическая 322
ансамблевая 79
базовая 79
вероятностная 611
вероятностного анализа главных компонент 206
восприятия 458
выбор 36
генеративная 132
гиперпараметры 36
глубокой авторегрессии 210
движения 458
динамическая 484
дискриминативная 132, 296
долгой краткосрочной памяти 352
идентифицируемость 154
интерполяция данных 39
интерпретируемость 102
класс 17
латентно-семантической индексации 326
линейная гауссова 132
ментальная 321
мешка слов 285, 286
мира 321
наивная байесовская 130
на уровне символов 346
непараметрическая 66, 143
неявная 210
обучения декодированием 357
объектная 382
объясняемость 102
оптимизация 37
отбор признаков 43
от последовательности к последовательности 353
с вниманием 355
параметрическая 65
перехода 458
порождающая 296
прогнозирующего контроля 505
разреженная 57
рендеринга 382
скип-граммы 289
скрытая марковская 155, 293
с откатом 290
с точечным источником света 389
Трансформер 299
уровня символов 288
языка 284, 322, 347
маскированная 368
перплексия 376
GloVe 365
n-граммная 288
Модификация вознаграждения 254
Мониторинг 104
Мягко разделяющий классификатор 78

Н

Набор данных
обучающий 17
спецификация 559
тестовый 17
Наиболее вероятное состояние 492
Наивная байесовская модель 130, 151
Начальный символ 616
Негативный побочный эффект 546
Недетерминированные полиномиальные задачи 607
Недогенерация 300
Недообучение 19
Независимость параметров 137
Неинформативное априорное распределение 139
Нейронная сеть 167
алгоритмы обучения 188
архитектура 194
выходной слой 173
исчезновение градиента 175
обобщение 193
обратное распространение 174
обучение без учителя 205
объединяющий слой 185
остаточная 187
рекуррентная 169, 200, 347
сверточная 181, 404
сквозное обучение 175
скрытый слой 173, 179
состязательный пример 195

с прямой связью 169
функция активации 169
элементы 169
Нейрон, рецептивное поле 184
Немаксимальное подавление 410
Необратимое действие 243
Неоднозначность 284, 319
метафора 321
метонимия 320
устранение 321
Непараметрическая модель 66
Неполная расстановка скобок 310
Непосредственная оценка полезности 232
Непреднамеренный побочный эффект 570
Неравенство в доходах 567
Несбалансированные классы 95
Несоответствие размеров выборки 559
Нестационарность 104
Нетерминальный символ 616
Неявная модель 210
Нижняя вариационная граница 208
Низкое влияние 570
Низкоразмерное встривание 465
Нормализация 67
Нормальная форма Хомского 303
Нормальное распределение 612
Нормальное уравнение 57, 210
Нотация "О большое" 605
Нуль-гипотеза 31

О

Обезличивание 552
Обнаружение спама 288
Обобщающее поле 552
Обобщение данных 17
Обработка естественного языка 215, 283
Обратная
кинематика 470
матрица 610
связь 16
Обратное
обучение с подкреплением 264, 572
распространение 174
Обратный
маятник 269
режим дифференцирования 175
Обрезка деревьев решений 31
Обучающий набор 35
взвешенный 84
данных 16
Обучение
ансамблевое 79
байесовское 122
без учителя 16, 204, 205
беспроигрышное 91
ВПК 46
глубокое 167
динамическое 89
имитационное 263
крупномасштабное 42
машинное 13
автоматизированное 114
мелкомасштабное 42
многозадачное 213
на демонстрации 263
на примерах 66
обобщение новых состояний 508
параметрическое 127
байесовское 134
перевод без учителя 211
популяции 45
предпочтениям 506
сквозное 215
слабое 86
с незначительным привлечением учителя 93
с подкреплением 16, 228, 496
активное 230, 239
байесовское 244
без модели 229
глубокое 216, 253
иерархическое 255
на демонстрации 230, 262
на основе модели 229, 497
обратное 263
пассивное 230
поиск стратегии 259
распределение вознаграждения 254
реляционное 274

с учителем 16
с частичным привлечением учителя 92, 204
трансферное 212, 364
формы 14
Объединение 185
по среднему 351
субдискретизация 185
Объединяющий слой 185
Объект
аспект 403
внешний вид 402
ограничивающая рамка 407
положение 417
рассогласование 412
Объяснимый ИИ 562
Объясняемость 102
Ограниченная оптимальность 600
Ограничивающая рамка 407
Однократное кодирование 342
Одометр 453
Окклюзия 403
Опорная
точка 478
линия 413
Опорный вектор 76
Определение
авторства 288
языка 288
Оптимальное управление 490
Оптимально разделяющий классификатор 73, 75
Оптимизация 37
траектории 490, 491
Оптический поток 398
фокус расширения 415
Ориентация края 396
Освещение
рассеянное 391
тени 390
Основной цвет 391
Остаточная сеть 187
Остаточный блок 187
Осторожное движение 493
Отбор признаков 43, 287
Ответ на вопрос 324
Отдаленный точечный источник света 389
Открытый класс 300
Отметка 459
Отражение 388
блики 389
взаимное 390
диффузное 388
зеркальное 389
Отрицательное логарифмическое правдоподобие 177
Отрицательный пример 23
Оценка
ложно-положительная 100
непараметрической плотности 143
полезности непосредственная 232
состояний 457
стратегии 230
Ошибка "out-of-bag" 82

П

Пакетная нормализация 192
Память 200
Параметр
независимость 137
скорость обучения 54, 236
Параметрическая модель 65
Параметрическое обучение 127, 151
байесовское 134
Парзеновское окно 162
Парсинг 302
Пассивное обучение с подкреплением 230
Пассивный обучающийся агент 230
Пень решения 87
Перевод "симуляция–реальность" 497
Перегенерация 300
Передача стиля 431
Перекрестная
проверка
по отдельным объектам 36
k-блочная 36
энтропия 178
Переменная
индикаторная 149
скрытая 145

Переобучение 19, 125
Перплексия 376
Перспективная проекция 385
Перцептрон Гамбы 218
Пиксель 384
Планирование
  движения 456, 467, 472
  задачи 456
  с несколькими запросами 479
Плотность вероятности 612
Побочный эффект 546
  непреднамеренный 570
Повторение опыта 252
Повышающая выборка 95
Поглощающее состояние 243
Подкатегория 310
Подкрепление 228
Позиционное встраивание 362
Поиск 302
  информации 324, 331
  нейронной архитектуры 196
  по лучу 296, 358
  по таблице 66
  сеточный 44
  случайный 44
  стратегии 230, 259
Поле расстояния со знаком 481
Полиномиальные задачи 607
Полные данные 127, 130
Положение 417
Положительный пример 23
Получение информации 496
Понижающая выборка 95
Попытка 231
Пороговая функция 60
Порождающая модель 296
Потери
  при обобщении 41
  эмпирические 41
Права роботов 568, 583
Правдоподобие 122
  логарифмическое 127
  максимальное 126
Правило
  Видроу–Хоффа 250
  обучения перцептрона 61
  подстановки 617
Прагматика 317
Предварительное
  знание 15
  обучение 344
Предварительный анализ данных 17
Предобучение 344, 364
  контекстных представлений 366
Предположение о стационарности 35
Представление
  развернутое 15
  слов 291
Преобразование
  изображений 428
  текста в речь 322
Препятствия 467
Приближенно ближайшие соседи 69
Привод 454
  гидравлический 454
  пневматический 454
  электропривод 454
Признак 249, 381
  математическое ожидание 266
Примитив движения 499
Принцип трех основных цветов 391
Приоритетная прогонка 238
Приращение информации 30
Проблема
  выравнивания ценностей 572
  квалификации 538
  координации 499
  соответствия 509
  царя Мидаса 572
Проверка статистической значимости 31
Проверочный набор 36
Прогнозирование
  действий человека 501
  поведения человека 456
Программная инженерия 570
Проектор структурированного света 451
Проекция 388
  ортогональная масштабированная 388
Прозрачность 562
Происхождение данных 94

Проклятье размерности 68
Простой планировщик 478
Пространственная инвариантность 181
Пространство
  весовых коэффициентов 52
  гипотез 17
  конфигураций 467, 470
    препятствие 469
    свободное 469
  планирования движения 473
  признаков 342
  рабочее 467
  свободное 469
  состояний совместное 257
Процесс Дирихле 162
Прямая
  кинематика 470
  коллокация 491
Псевдовознаграждение 254
Псевдокод 617
  списки 619
  циклы 619
Псевдоэксперимент 238
Пулинг 185
Путь 467, 472
  закон управления 486
  синхронизация 485

Р

Рабочая характеристика приемника 100
Рабочее пространство 467
Равномерное распределение 614
Радар 452
Разведочный
  анализ данных 97
  POMDP 245
Разложение в ряд Тейлора 463
Размытие в движении 384
Размышлизм 574
Разреженная модель 57
Ракурс 383, 388
Рандомизация предметной области 497
Ранняя остановка 32
Распознавание
  объектов 382
  речи 322
Распределение
  априорное неинформативное 139
  априорных вероятностей гипотез 122
  бета-распределение 135
  вознаграждения 254
  Гаусса 612
  кумулятивное 612
  многомерное гауссово 613
  нормальное 612
  равномерное 614
  смешанное 147
  сопряженное априорное 136
  стандартное 613
Рассеянное освещение 391
Расстояние
  Махаланобиса 67
  Минковского 67
  Хемминга 67
Расчетное управление крутящим моментом 489
Расширение набора данных 406
Расширенная грамматика 311
Расширенный
  конечный автомат 512
  фильтр Калмана 463
Рациональность Больцмана 265
Реагирующий подход 510
Реализуемость задачи обучения 42
Регрессия 15
  линейная байесовская 138
  логистическая 64, 294
  локально взвешенная 72
  множественная линейная 56
  ограничивающей рамки 411
  по k-ближайшим соседям 72
Регуляризационная функция 43
Регуляризация 43, 57, 89, 197, 570
Реконструкция 382
Рекуррентная нейронная сеть 169, 200
  долгая краткосрочная память 203
  обучение 201
Релейное управление 269
Реляционное обучение с подкреплением 274
Рефлексивная архитектура 599

Рецептивное поле 184
Речевой акт 317
Робопокалипсис 569
Робот 447
  антропоморфный 449
  дистанционного присутствия 515
  координация действий с людьми 500
  манипулятор 449
  мобильный 449
  на ногах 450
  положение 468
  походка 511
  прогнозирование действий человека 501
  ровер 450
  самообучение 466
Робототехника
  применение 514
  реактивный подход 510
  совещательный подход 510
Роверы 450
Ручная настройка гиперпараметра 44

С

Самовнимание 360
Самообучение 466
Сбор информации 493
Свертка 182, 395
  шаг ядра 182
Сверточная нейронная сеть 182, 404
  тензорные операции 186
Сверхразумная машина 574
Свидетельство 121
Сглаживание 155, 290
  линейное интерполяционное 290
Сегмент 400
  суперпиксели 402
Сегментация 400
Семантика 284
  композиционная 313
  предложения 300
  фразы 311
Семантическая
  грамматика, изучение 316
  неоднозначность 320
Сенсор 384
Серийный спуск по градиенту 54
Сертификация 562
Сетка занятости 523
Сеточный поиск 44
Сеть
  генеративно-состязательная 210
  остаточная 187
  параметры 169
  региональных предложений 410
  рекуррентная 169, 200
  сверточная 181
    ядро 182
  с прямой связью 169
  Хопфилда 222
  элементы 169
Сигмоида 170
Сильный ИИ 537, 577
Символ 616
Симуляция–реальность 497
Синтаксическая
  категория 300
  неоднозначность 320
Синтаксический
  анализ 302
    без учителя 309
    детерминированный 306
    ориентированный на данные 309
    сдвиг-уменьшение 306
    с частичным привлечением учителя 309
Система
  верификация и валидация 561
  генерации подписей 422
  доверие 561
  интерпретируемая 563
  объяснимая 563
  ответа на визуальный вопрос 424
  присвоения тегов 422
  прозрачность 562
  сертификация 562
  хорошо откалиброванная 556
Сквозное обучение 175, 215, 497
Скользящее окно 408
Скорость обучения 54

Скрытая
марковская модель 155, 293
переменная 145, 151, 157
Скрытый слой 173
Слабое обучение 86
Слабый ИИ 537, 576
Словарь 291, 300
WordNet 291
Слово вне словаря 289
Сложность
выборочная 47
колмогоровская 110
Слой 167
объединяющий 185
скрытый 179
смешанной плотности 179
Softmax 178
Случайный
лес 81
поиск 44
Смежность пикселей 176
Смертельное автономное оружие 547, 580
Смешанная плотность 179
Смешанное распределение 147
гауссово 147
Смещение 18
Совещательный подход 510
Совместно используемые модели 595
Совместное пространство состояний 257
Совместный агент 504
Совокупный запрос 553
Согласованная гипотеза 17
Соединения
поворотные 454
призматические 454
Сожаление 90
Сознание 544, 577
Сокращение веса 197
Сонар 451
Соответствие признаков 265
Сопряженное априорное распределение 136
Состояние выбора 257
Состязательное обучение 508
Состязательный пример 195
Сотрудничество 504
Социальная предвзятость 555
Спецификация 559
Список решений 48
Справедливость 581
Среднее квадратическое 614
Стандартное нормальное распределение 613
Стандартное отклонение 614
Стационарность 35
Стационарный процесс 200
Стекинг 83
Стековое обобщение 83
Степень свободы 469
Стереозрение 451
Стоимость прохождения 491
Стохастический градиентный спуск 55
Стохастическое представление стратегии 260
Стратегия 259, 489
градиент 260
стохастическое представление 260
ценность 260
Структура фразы 300
Субдискретизация 185
Сумма квадратов разностей 399
Суперпиксель 402
Сцена 384
Счастливый график 28

T

Тактильная обратная связь 525
Тег 292
Тексель 397
Текст
анализ тональности 288
корпус 286
Текстура 397
Тензор 186
канал 187
карта признаков 186

Тензорная операция 186
Тень 390
Теорема
  аппроксимации универсальная 170
  Гёделя 540
  Мерсера 78
  центральная предельная 613
Теория
  вычислительного обучения 46
  надежного управления 245
  оптимальности 326
  управления 484
Терминальный символ 616
Тест
  эталонный 605
  Тьюринга 541, 577
Тестирование 101
Тестовый набор 17, 35
Техника безопасности 569
Технические данные 559
Технологическая
  безработица 564
  сингулярность 574
Точка
  разбиения 33
  схода 386
Трагедия общего достояния 571
Траектория 467
Трансгуманизм 575
Транспонирование 610
Трансферное обучение 204, 212, 364
Трансформер 360
  декодер 362
  энкодер 362
Труд и занятость 564

У

Универсальная
  грамматика 330
  теорема аппроксимации 170
Управление 456
  прохождением траектории 467, 484
  с прогнозирующими моделями 492
Уравнение
  временной разности 236
  нормальное 210
  Риккати 491
  Эйлера–Лагранжа 482
Уравнения Юла–Уокера 210
Устранение неоднозначности 321
Ученичество 230, 263

Ф

Федеративное обучение 554
Фиксация взгляда 414
Фильтр
  Гаусса 395
  Калмана расширенный 463
  частиц Рао–Блэквелла 523
Фильтрация 155, 457
Фокальная плоскость 387
Фокусное расстояние 385
Фокус расширения 415
Форма
  Бэкуса–Наура 616
  квазилогическая 317
Фраза, заголовок 311
Функционал 481
Функция
  активации 169, 190
    сигмоида 170
    ReLU 170
    softplus 170
    tanh 170
  аппроксимация 249
  исследования 242
  оценки 249
  ошибок 613
  плотности вероятности 612
  полезности, аддитивная декомпозиция 258
  пороговая 60
  потерь 40
    отрицательное логарифмическое правдоподобие 177
  регуляризационная 43
  свертки 395
  ядерная 78, 144
  softmax 260

Х

Характеристика 381
Хеширование
  локально-чувствительное 69
Хорошо откалиброванная система 556

Ц

Цвет 391
  основной 391
  постоянство 392
Целевой язык 352
Ценность стратегии 260
Центральная предельная теорема 613
Цепное правило 54, 173
Цепь Маркова 288
Цикл 619

Ч

Частичная программа 255
Частичное обучение 204
Частота
  ошибки 46
  появления ошибок 35
Часть речи 292
  маркировка 292
Чрезвычайно рандомизированные
  деревья 82

Ш

Шаг 182
Штрафование за сложность 143
Шум 26, 394

Э

Эвристика
  прибрежного плавания 495
  приоритетной прогонки 238, 274
Элемент 169
  функция активации 169
Эмпирические потери 41
Эмпирический градиент 260
Энкодер трансформера 362
Энтропия 28
Эпоха 55
Эпсилон-шар 47
Эталонный тест 605
Этика ИИ 545

Я

Ядерная функция 78, 144
Ядерный трюк 73, 78
Ядро 72, 182
  полиномиальное 78
  шаг 182
Язык 299
  естественный, обработка 283
  модель 284
Якорная рамка 410
Ячейка
  Вороного 474
  памяти 203

ПРИКЛАДНОЕ МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ SCIKIT-LEARN, KERAS И TENSORFLOW

2-е ИЗДАНИЕ

Орельен Жерон

www.williamspublishing.com

Благодаря серии выдающихся достижений глубокое обучение значительно усилило всю область машинного обучения. В наше время даже программисты, почти ничего не знающие об этой технологии, могут использовать простые и эффективные инструменты для реализации программ, которые способны обучаться на основе данных.

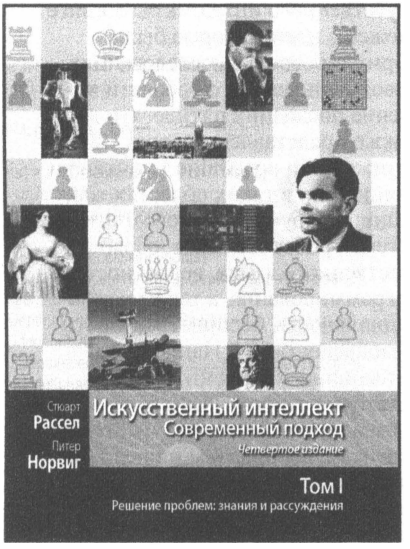
За счет применения конкретных примеров, минимума теории и фреймворков Python производственного уровня обновленное издание этой ставшей бестселлером книги поможет вам получить интуитивное представление о концепциях и инструментах, предназначенных для построения интеллектуальных систем.

ISBN 978-5-907203-33-4 **в продаже**

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ СОВРЕМЕННЫЙ ПОДХОД

4-Е ИЗДАНИЕ.
ТОМ I. РЕШЕНИЕ ПРОБЛЕМ: ЗНАНИЯ И РАССУЖДЕНИЯ

Стюарт Рассел
Питер Норвиг



www.dialektika.com

В этом обновленном и пересмотренном издании читатель найдет самое полное и актуальное введение в теорию и практику искусственного интеллекта. В книге изложены идеи, которые были сформулированы в исследованиях, проводившихся в этой области в течение последних пятидесяти лет, а также представлены современные достижения и новейшие технологии и концепции в таких областях, как машинное обучение, многоагентные системы, робототехника, обработка естественного языка, вероятностное программирование, а также конфиденциальность, беспристрастность и безопасность ИИ. Написанная без излишнего академизма (но достаточно строго) книга будет полезна широкому кругу читателей: студентам, аспирантам и преподавателям высших учебных заведений, инженерам, разработчикам ПО и т.д.
В связи с исключительно большим объемом этого энциклопедического издания его русскоязычный вариант представлен в виде трехтомника.
В первом томе излагаются основы основ ИИ — общая характеристика этой области как науки об интеллектуальных агентах, действующих в различных средах и решающих различные задачи на основании знаний, результатов восприятия, рассуждений и планирования.

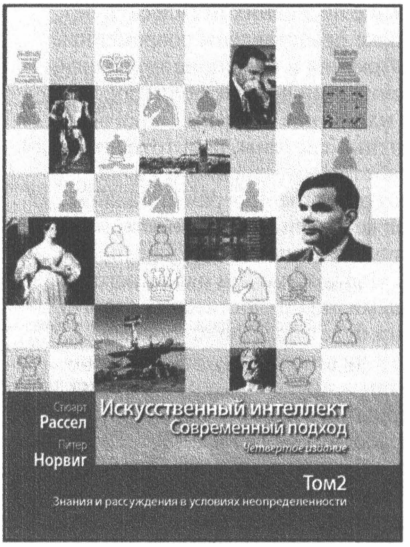
ISBN 978-5-907365-25-4

в продаже

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ СОВРЕМЕННЫЙ ПОДХОД

4-Е ИЗДАНИЕ.
ТОМ 2. ЗНАНИЯ И РАССУЖДЕНИЯ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ

Стюарт Рассел
Питер Норвиг



www.dialektika.com

В этом обновленном и пересмотренном издании читатель найдет самое полное и актуальное введение в теорию и практику искусственного интеллекта. В книге изложены идеи, которые были сформулированы в исследованиях, проводившихся в этой области в течение последних пятидесяти лет, а также представлены современные достижения и новейшие технологии и концепции в таких областях, как машинное обучение, многоагентные системы, робототехника, обработка естественного языка, вероятностное программирование, а также конфиденциальность, беспристрастность и безопасность ИИ. Написанная без излишнего академизма (но достаточно строго) книга будет полезна широкому кругу читателей: студентам, аспирантам и преподавателям высших учебных заведений, инженерам, разработчикам ПО и т.д. В связи с исключительно большим объемом этого энциклопедического издания его русскоязычный вариант представлен в виде трехтомника. В этом, втором томе обсуждаются различные аспекты подхода к реализации агентов, действующих в средах, приближенных к реальности: вероятностные рассуждения, в том числе о времени, построение планов и принятие решений (простых и сложных), в том числе в многоагентной среде.

ISBN 978-5-907365-26-1

в продаже

Самое полное и актуальное введение в теорию и практику искусственного интеллекта!

В четвертом, обновленном, пересмотренном и дополненном издании этой книги область искусственного интеллекта (ИИ) исследуется и анализируется во всей ее обширности и глубине. Здесь представлены все современные достижения и изложены идеи, которые были сформулированы в исследованиях, проводившихся в течение последних пятидесяти лет, а также собраны на протяжении двух тысячелетий в областях знаний, ставших стимулом к развитию ИИ как науки. Предыдущие издания этой книги стали классическими образцами литературы по ИИ и приняты в качестве учебного пособия более чем в 1400 университетах 128 стран мира, где были высоко оценены как убедительный итог обобщения результатов, достигнутых в этой области науки.
Книга дополнена обширным набором интернет-ресурсов, включая упражнения, программные проекты и исследовательские проекты, реализации алгоритмов, дополнительные материалы и ссылки для студентов и преподавателей

Что нового в четвертом издании

В четвертом издании читатель познакомится с новейшими технологиями и концепциями, представленными в более унифицированном виде с новым или расширенным охватом таких тем, как машинное обучение, глубокое обучение, трансферное обучение, многоагентные системы, робототехника, обработка естественного языка, проблема причинности, вероятностное программирование, а также конфиденциальность, беспристрастность и безопасность ИИ.

Что представлено в этом томе

В связи с исключительно большим объемом этого энциклопедического издания его русскоязычный вариант представлен в виде трехтомника. В этом томе, получившем подзаголовок *Обучение, восприятие и действие*, рассматриваются наиболее сложные и современные подобласти ИИ: различные подходы к машинному обучению, обработка естественного языка, компьютерное зрение и робототехника, а также не менее важные аспекты ИИ — его конфиденциальность, беспристрастность и безопасность.

Что в других томах

В двух предыдущих томах I и II рассматривались основы теории ИИ — интеллектуальные агенты и возможный подход к решению проблем: знания, рассуждения и планирование (том I) и действие агентов в окружении, более приближенном к реальному: знания, рассуждения и принятие решений в условиях неопределенности, в том числе в многоагентной среде (том II).

ОБ АВТОРАХ

Стюарт Рассел, профессор компьютерных наук Калифорнийского университета в г. Беркли, адъюнкт-профессор неврологической хирургии Калифорнийского университета в г. Сан-Франциско, основатель и директор центра Center for Human-Compatible AI и заведующий кафедрой инженерного искусства Смита–Задэ в Калифорнийском университете в г. Беркли.

Питер Норвиг, директор по исследованиям в компании Google, Inc., в прошлом — глава ее группы разработки основных алгоритмов веб-поиска. До этого руководил отделом компьютерных наук в Исследовательском центре Эймса, НАСА, был профессором в Университете Южной Калифорнии и членом совета факультета в Беркли и Станфорде.
Авторы этой книги были удостоены первой награды AAAI/EAAI "Выдающийся педагог" в 2016 году.

На сайте издательства "Диалектика" можно скачать перечень литературы, ссылки на которую приведены в этой книге:
http://www.williamspublishing.com/Books/978-5-907365-27-8.html

ISBN 978-5-907365-27-8

www.dialektika.com

www.pearson.com