ВЕДЬ ЭТО ТАК ПРОСТО!

Глубокое обучение

для чайников®

Издательство ДИАЛЕКТИКА

Почему глубокое обучение является столь важной технологией

Эксперименты с глубоким обучением в среде языка Python

Примеры основных типов приложений глубокого обучения

Джон Пол Мюллер
Лука Массарон

Авторы книги *Python и наука о данных для чайников, 2-е издание*

Глубокое обучение

для чайников®

Deep Learning

by John Paul Mueller and Luca Massaron

for dummies®
A Wiley Brand

Джон Пол Мюллер и Лука Массарон

Глубокое обучение

для чайников®

Київ
Комп'ютерне видавництво
"ДІАЛЕКТИКА"
2020

УДК 004.855
М98

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

Мюллер, Дж. П., **Массарон**, Л.

М98 Глубокое обучение для чайников/Джон Пол Мюллер, Лука Массарон; пер. с англ. В.А. Коваленко. — Киев. : "Диалектика", 2020. — 400 с. : ил. — Парал. тит. англ.

ISBN 978-617-7874-07-1 (укр.)
ISBN 978-1-119-54304-6 (англ.)

Глубокое обучение предоставляет средства для выявления шаблонов в данных, лежащих в основе онлайновых операций, медицине, исследованиях, социальных сетях и во многих элементах повседневной жизни. В этой книге предоставлена вся информация, необходимая для устранения загадочности этой темы, а также описаны все основные технологии, связанные с ней. Вскоре вы сможете разобраться в весьма запутанных алгоритмах и найдете простую и безопасную среду для экспериментов с глубоким обучением.

УДК 004.855

ISBN 978-617-7874-07-1 (укр.)
ISBN 978-1-119-54304-6 (англ.)

Оглавление

Введение 18

Часть 1. Появление глубокого обучения 25
Глава 1. Введение в глубокое обучение 27
Глава 2. Знакомство с принципами машинного обучения 45
Глава 3. Получение и использование языка Python 69
Глава 4. Использование инфраструктуры глубокого обучения 99

Часть 2. Основы лубокого обучения 119
Глава 5. Обзор матричной математики и оптимизации 121
Глава 6. Основы линейной регрессии 141
Глава 7. Введение в нейронные сети 163
Глава 8. Построение простой нейронной сети 183
Глава 9. Переход к глубокому обучению 197
Глава 10. Сверточные нейронные сети 215
Глава 11. Введение в рекуррентные нейронные сети 239

Часть 3. Взаимодействие с глубоким обучением 255
Глава 12. Классификация изображений 257
Глава 13. Передовые CNN 275
Глава 14. Обработка текстов на естественном языке 295
Глава 15. Создание произведений изобразительного искусства и музыки 315
Глава 16. Построение генеративно-состязательных сетей 327
Глава 17. Глубокое обучение с подкреплением 341

Часть 4. Великолепные десятки 357
Глава 18. Десять приложений, требующих глубокого обучения 359
Глава 19. Десять инструментов глубокого обучения 369
Глава 20. Десять профессий, использующих глубокое обучение 379

Содержание

Об авторах 15
Посвящение Джона 16
Посвящение Луки 16
Благодарности Джона 17
Благодарности Луки 17

Введение 18

О книге 18
Соглашения, принятые в книге 19
Глупые предположения 20
Источники дополнительной информации 21
Что дальше 22
Ждем ваших отзывов! 23

Часть 1. Появление глубокого обучения 25

Глава 1. Введение в глубокое обучение 27

Определение смысла глубокого обучения 28
Начнем с искусственного интеллекта 28
Роль искусственного интеллекта 30
Сосредоточимся на машинном обучении 34
Переход от машинного обучения к глубокому обучению 36
Использование глубокого обучения в реальном мире 38
Концепция обучения 39
Решение задач глубокого обучения 39
Использование глубокого обучения в приложениях 39
Среда программирования для глубокого обучения 40
Преодоление заблуждений, связанных с глубоким обучением 42
Экосистема запуска 43
Когда глубокое обучение неприменимо 43

Глава 2. Знакомство с принципами машинного обучения 45

Определение машинного обучения 46
Как работает машинное обучение 46
Это все чистая математика 48

Разные стратегии обучения 48
Обучение, проверка и тестирование данных 51
Поиск обобщения 52
Знакомство с предубеждением 53
Сложность модели 54
Разнообразие способов обучения 54
Бесплатных обедов не бывает 55
Пять основных подходов 55
Несколько разных подходов 58
В ожидании следующего прорыва 62
Размышления об истинном использовании машинного обучения 62
Преимущества машинного обучения 63
Границы машинного обучения 65

Глава 3. Получение и использование языка Python 69

Работа с языком Python в этой книге 70
Получение собственного экземпляра Anaconda 71
Получение пакета Anaconda от Continuum Analytics 71
Установка Anaconda на Linux 72
Установка Anaconda на MacOS 73
Установка Anaconda на Windows 74
Загрузка наборов данных и примеров кода 78
Использование Jupyter Notebook 79
Определение хранилища кода 81
Получение и использование наборов данных 86
Создание приложения 87
Понятие ячеек 88
Добавление ячеек документации 89
Использование ячеек других типов 90
Использование отступов 90
Добавление комментариев 91
Понятие комментариев 92
Применение комментариев для напоминания 93
Применение комментариев для предотвращения выполнения кода 95
Получение справки по языку Python 95
Работа в облаке 96
Использование наборов данных Kaggle и ядер 96
Использование Google Colaboratory 97

Глава 4. Использование инфраструктуры глубокого обучения 99
Знакомство с инфраструктурой 100
Определение различий 100
Популярность инфраструктур 101
Определение инфраструктуры глубокого обучения 104
Выбор конкретной инфраструктуры 105
Работа с бюджетными инфраструктурами 106
Caffe2 106
Chainer 107
PyTorch 108
MXNet 108
Microsoft Cognitive Toolkit/CNTK 109
Инфраструктура TensorFlow 109
Понятно, почему TensorFlow так хорош 109
Упрощение TensorFlow с помощью TFLearn 112
Использование Keras для большего упрощения 113
Получение экземпляра TensorFlow и Keras 114
Исправление ошибки инструментов сборки C++ в Windows 116
Доступ к новой среде в Notebook 117

Часть 2. Основы лубокого обучения 119

Глава 5. Обзор матричной математики и оптимизации 121
В какой математике вы действительно нуждаетесь 122
Работа с данными 123
Создание и работа с матрицей 124
Скалярные, векторные и матричные операции 125
Создание матрицы 126
Умножение матриц 127
Расширенные матричные операции 129
Расширение анализа до тензоров 131
Эффективное использование векторизации 133
Интерпретация обучения как оптимизации 134
Функции стоимости 135
Нисходящая кривая ошибок 136
Обучение в правильном направлении 137
Обновление 139

Глава 6. Основы линейной регрессии 141
Объединение переменных 142
Простая линейная регрессия 143
Переход к множественной линейной регрессии 144
Включение градиентного спуска 145
Линейная регрессия в действии 146
Смешивание типов переменных 148
Моделирование ответов 148
Моделирование признаков 149
Работа со сложными отношениями 150
Переход на вероятности 152
Обеспечение бинарного ответа 152
Преобразование числовых оценок в вероятности 153
Прогноз правильных признаков 155
Определение результата несовместимых признаков 155
Избежание переобучения с использованием выборки и регуляризации 156
Изучение по одному примеру за раз 158
Использование градиентного спуска 158
Чем отличается SGD 158

Глава 7. Введение в нейронные сети 163
Знакомство с невероятным перцептроном 164
Функциональность перцептрона 164
Достижение предела неразделяемости 166
Уменьшение сложности нейронными сетями 169
Нейрон 169
Прямая передача данных 171
Еще глубже в кроличью нору 173
Использование обратного распространения для настройки обучения 176
Борьба с переобучением 180
Понятие проблемы 180
Открываем черный ящик 181

Глава 8. Построение простой нейронной сети 183
Понятие нейронных сетей 184
Базовая архитектура 185
Документирование основных модулей 187
Решение простой задачи 190

Внутренняя работа нейронных сетей 192
Выбор правильной функции активации 193
Полагаясь на умный оптимизатор 195
Установка рабочей скорости обучения 196

Глава 9. Переход к глубокому обучению 197

Данные везде 198
Учет влияния структуры 199
Следствия закона Мура 200
Что меняет закон Мура 201
Преимущества дополнительных данных 202
Последствия больших данных 203
Своевременность и качество данных 203
Ускорение обработки 205
Использование мощного оборудования 205
Другие инвестиции 206
Отличие глубокого обучения от других форм искусственного интеллекта 207
Добавление большего количества слоев 207
Изменение активаций 210
Добавление регуляризации в ходе отсева 211
Поиск еще более разумных решений 212
Использование дистанционного обучения 213
Перенос обучения 213
Обучение от начала до конца 214

Глава 10. Сверточные нейронные сети 215

Начнем обзор CNN с распознавания символов 216
Основы изображения 216
Как работает свертка 219
Свертка 220
Упрощение использования подвыборки 224
Архитектура LeNet 226
Обнаружение краев и фигур в изображениях 231
Визуализация сверток 231
Успешные архитектуры 233
Перенос обучения 235

Глава 11. Введение в рекуррентные нейронные сети 239
Рекуррентные сети 240
Моделирование последовательностей с использованием памяти 241
Распознавание и перевод речи 242
Размещение правильной подписи на картинках 245
Долгая краткосрочная память 246
Определение различий в памяти 247
Архитектура LSTM 248
Открывая интересные варианты 250
Получение необходимого внимания 252

Часть 3. Взаимодействие с глубоким обучением 255

Глава 12. Классификация изображений 257
Конкурсы по классификации изображений 258
ImageNet и MS COCO 259
Магия приращения данных 261
Распознавание дорожных знаков 265
Подготовка данных изображения 266
Выполнение задачи классификации 269

Глава 13. Передовые CNN 275
Различные задачи классификации 276
Локализация 278
Классификация нескольких объектов 278
Аннотирование нескольких объектов на изображениях 279
Сегментирование изображений 280
Восприятие объектов в их окружении 282
Как работает RetinaNet 283
Использование кода Keras-RetinaNet 284
Предотвращение преднамеренных атак на приложения глубокого обучения 289
Обманные пиксели 290
Взлом с помощью наклеек и других артефактов 292

Глава 14. Обработка текстов на естественном языке 295
Обработка языка 296
Определение понимания как лексического анализа 297
Помещение всех документов в набор 299

Запоминание имеющих значение последовательностей 301
Понимание семантики по векторным представлениям слов 302
Использование искусственного интеллекта для анализа настроений 307

Глава 15. Создание произведений изобразительного искусства и музыки 315

Учимся подражать искусству и жизни 316
Передача художественного стиля 317
Сведение проблемы к статистике 319
Что глубокое обучение создать не может 320
Подражание художнику 321
Определение нового произведения на основе одного примера 321
Объединение стилей для создания нового произведения искусства 323
Визуализация мечты нейронных сетей 324
Использование сети для сочинения музыки 324

Глава 16. Построение генеративно-состязательных сетей 327

Создание состязающихся сетей 328
Поиск ключа в состязании 329
Достижение более реалистичных результатов 331
Рост поля деятельности 338
Создание реалистичных фотографий знаменитостей 338
Улучшение деталей и преобразование изображений 339

Глава 17. Глубокое обучение с подкреплением 341

Играя в игру с нейронными сетями 342
Знакомство с обучением с подкреплением 343
Имитация игровой среды 345
Q-обучение 348
Объяснение Alpha-Go 351
Если вы собираетесь выиграть 352
Применение самообучения в масштабе 354

Часть 4. Великолепные десятки 357

Глава 18. Десять приложений, требующих глубокого обучения 359

Восстановление цвета на черно-белых видео и изображениях 360
Аппроксимация позы человека в реальном времени 361
Анализ поведения в реальном времени 361

Перевод 362
Оценка потенциала солнечной энергии 363
Победа над людьми в компьютерных играх 363
Генерация голоса 364
Прогнозирование демографии 365
Создание произведений искусства из реальных фотографий 366
Прогнозирование природных катастроф 367

Глава 19. Десять инструментов глубокого обучения 369

Компиляция математических выражений с использованием Theano 369
Дополнение TensorFlow с помощью Keras 370
Динамическое вычисление графиков с помощью Chainer 371
Создание среды, похожей на MATLAB, с помощью Torch 372
Динамическое выполнение задач с помощью PyTorch 373
Ускорение исследований в области глубокого обучения с использованием CUDA 373
Поддержка бизнес-потребностей с Deeplearning4j 375
Получение данных с использованием Neural Designer 376
Алгоритмы обучения с использованием Microsoft Cognitive Toolkit (CNTK) 377
Полное использование возможностей графического процессора с помощью MXNet 377

Глава 20. Десять профессий, использующих глубокое обучение 379

Управление людьми 380
Улучшение медицинского обслуживания 380
Разработка новых устройств 381
Обеспечение поддержки клиентов 382
Просмотр данных новыми способами 383
Ускорение выполнения анализа 383
Создание лучшей рабочей среды 384
Исследование неясной или подробной информации 385
Проектирование зданий 386
Повышение безопасности 387

Об авторах

Джон Мюллер — внештатный автор и технический редактор. Умение писать у него в крови, на настоящий момент он является автором 112 книг и более чем 600 статей. Разнообразие тем распространяется от работы с сетями до искусственного интеллекта и от управления базами данных до автоматического программирования. Некоторые из его книг затрагивают темы науки о данных, машинного обучения и алгоритмов. Его навыки технического редактирования помогли больше чем 70 авторам усовершенствовать содержимое своих произведений. Джон оказывает услуги технического редактирования различным журналам, консультирует и пишет сертификационные экзамены. Обратитесь к блогу Джона по адресу `http://blog.johnmuellerbooks.com/`. Джон доступен в Интернете по адресу `John@JohnMuellerBooks.com`. У него также есть веб-сайт по адресу `http://www.johnmuellerbooks.com/`, а также представительство на Amazon по адресу `https://www.amazon.com/John-Mueller/`.

Лука Массарон — аналитик данных и директор по маркетинговым исследованиям, специализирующийся на многомерном статистическом анализе, машинном обучении и изучении ожиданий потребителей с больше чем десятилетним опытом в решении реальных проблем и консультации заинтересованных лиц на основании обоснованных доводов, статистики, данных и алгоритмов. Предпочитая простоту ненужной изощренности, он верит, что в науке о данных можно многого достичь, поняв и применив ее основы. Лука также является экспертом Google (GDE) в области машинного обучения.

Посвящение Джона

Эта книга посвящена моим исключительно добрым соседям, Донни (Donnie) и Шеннон Томпсон (Shannon Thompson). Они переопределили для меня термин “сосед” так много раз, что я потерял счет.

Посвящение Луки

Я хотел бы посвятить эту книгу моей семье, Юкико (Yukiko) и Амелии (Amelia), моим родителям, Ренцо (Renzo) и Лиции (Licia), и семье Юкико, Ёсики (Yoshiki), Такайо (Takayo) и Макико (Makiko).

Благодарности Джона

Благодарю мою жену Ребекку. Хотя ее уже нет, ее дух в каждой написанной мной книге и в каждом слове на каждой странице. Она верила в меня, когда не верил никто.

Рус Маллен (Russ Mullen) заслужил благодарность за техническое редактирование этой книги. Он привнес точность и глубину в материал, который вы видите здесь. Рус работал исключительно усердно, помогая в исследованиях для этой книги, находя труднодоступные URL-адреса, а также делая множество предложений.

Мэтт Вагнер (Matt Wagner), мой агент, заслуживший благодарности за то, что помог мне получить первый контракт и заботился обо всех подробностях, которые большинство авторов действительно не учитывают. Я всегда ценю его помощь. Приятно знать, что кто-то хочет тебе помочь.

Многие люди прочитали всю эту книгу или ее часть, чтобы помочь мне усовершенствовать подход, проверить примеры и вообще предоставить мне отзыв о том, что им не понравилось. Эти добровольные помощники помогли слишком во многом, чтобы упомянуть здесь все. Я особенно ценю усилия Евы Битти (Eva Beattie), Гленна А. Рассела (Glenn A. Russell), г-на Освальдо Теллеса Альмиралла (Mr. Osvaldo Téllez Almirall) и Симоны Скардапейн (Simone Scardapane), которые внесли общий вклад, прочитали всю книгу и самоотверженно посвятили себя этому проекту.

И наконец, я хотел бы поблагодарить Кэти Мор (Katie Mohr), Сьюзен Кристоферсен (Susan Christophersen) и остальную часть редакционного и технического коллектива.

Благодарности Луки

Я очень благодарен моей семье, Юкико, Амелии, за их поддержку и терпение. Я также хочу поблагодарить Симону Скардапейн, доцента Римского университета Ла Сапиенца и коллегу-эксперта Google, которая предоставила неоценимые отзывы во время написания этой книги.

Введение

Когда говоришь с некоторыми людьми о глубоком обучении, они думают о какой-то глубокой темной тайне, но глубокое обучение вовсе не является загадкой, вы используете его каждый раз, когда разговариваете со своим смартфоном, так что вы ежедневно носите его с собой. На самом деле, вы найдете глубокое обучение везде. Например, вы обнаружите его использование во многих сетевых приложениях и даже при совершении покупок. Вы окружены глубоким обучением и даже не понимаете этого, что делает изучение глубокого обучения необходимым, поскольку вы можете использовать его, чтобы делать гораздо больше, чем вы возможно думаете.

У других людей совершенно иной взгляд на глубокое обучение, который в реальности не имеет никаких оснований. Они полагают, что глубокое обучение послужит причиной какого-то страшного апокалипсиса, но на самом деле это невозможно с современными технологиями. Скорее всего, кто-то найдет способ использовать глубокое обучение для создания фальшивых людей, используемых при совершении преступлений или обмане правительства на тысячи долларов. Тем не менее, роботы-убийцы, безусловно, не предвидятся в ближайшем будущем.

Принадлежите ли вы к первой части общества или ко второй, мы надеемся, что вы прочтете эту книгу, чтобы понять, на что на самом деле способно глубокое обучение. В решении повседневных задач эта технология, вероятно, способна сделать намного больше, чем вы думаете, но она также имеет ограничения, и вам нужно знать о них.

О книге

Работая с этой книгой, вы получите доступ к большому количеству примеров кода, который будет работать на стандартной системе Mac, Linux или Windows. Вы также можете запускать код по сети, используя нечто вроде Google Colab. (Мы предоставляем указания о том, как получить необходимую для этого информацию.) Специальное оборудование, такое как GPU, позволит примерам работать быстрее. Тем не менее, смысл этой книги в том, что вы можете создавать код для глубокого обучения независимо от того, какой у вас

компьютер, если вы готовы подождать, пока его выполнение не завершится. (Мы укажем, запуск каких примеров может занять много времени.)

Первая часть этой книги содержит начальную информацию, чтобы вы не растерялись совершенно, прежде чем начать изучение. Вы узнаете, как устанавливать различные нужные вам продукты, и получите представление о некоторых важных математических задачах. Первые примеры более похожи на стандартную регрессию и машинное обучение, но вам нужна эта основа, чтобы получить полное представление о том, что глубокое обучение может сделать для вас.

После того, как вы пройдете эти начальные этапы, вы начнете делать довольно удивительные вещи. Например, вы узнаете, как создавать произведения искусства и выполнять другие задачи, для которых, возможно, понадобится много кода и некоторые специальные аппаратные средства. К концу книги, вы будете поражены тем, что вы можете сделать, даже если у вас нет высоких степеней по машинному или глубокому обучению.

Соглашения, принятые в книге

Здесь используются соглашения, общепринятые в компьютерной литературе.

- Новые термины в тексте выделяются *курсивом*. Чтобы обратить внимание читателя на отдельные фрагменты текста, также применяется *курсив*.
- Текст программ, функций, переменных, URL веб-страниц и другой код представлен моноширинным шрифтом.
- Все, что придется вводить с клавиатуры, выделено **полужирным моноширинным** шрифтом.
- Знакоместо в описаниях синтаксиса выделено *курсивом*. Это указывает на необходимость заменить знакоместо фактическим именем переменной, параметром или другим элементом, который должен находиться на этом месте:
  `BINDSIZE=` (*максимальная ширина колонки*) `*` (*номер колонки*)`.`
- Пункты меню и названия диалоговых окон представлены следующим образом: Menu Option (Пункт меню).

Текст некоторых абзацев выделен специальным стилем. Это примечания, советы и предостережения, которые помогут обратить внимание на наиболее важные моменты в изложении материала и избежать ошибок в работе.

СОВЕТ

Советы хороши тем, что позволяют сэкономить время или упростить решение некой задачи. Советы в этой книге описывают экономящие время методики или содержат указатели на ресурсы, с которыми имеет смысл ознакомиться, чтобы получить максимум пользы от изучения языка Python применительно к науке о данных.

ВНИМАНИЕ!

Мы не хотим походить на строгих родителей или каких-то маньяков, но вам не следует делать то, что отмечено данной пиктограммой. В противном случае вы можете обнаружить, что ваше приложение работает не должным образом, вы получите неправильные ответы из, казалось бы, пуленепробиваемых уравнений или (в худшем случае) потеряете данные.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Увидев эту пиктограмму, знайте, что это дополнительный совет (или методика). Вы могли бы найти его очень полезным или слишком скучным, но он может содержать решение, необходимое для запуска программы. Пропускайте эти разделы, если хотите.

ЗАПОМНИ!

Если вы не вынесли ничего из некой главы или раздела, то запомните хотя бы материал, отмеченный этой пиктограммой. Такой текст обычно содержит наиболее важную информацию, которую следует знать для работы с языком Python или успешного выполнения задач, связанных с наукой о данных.

Глупые предположения

Возможно, вам будет трудно поверить, что мы о вас что-то думали — в конце концов, мы еще даже не встречались! Хотя большинство предположений действительно глупы, мы сделали их, чтобы обеспечить некую отправную точку для книги.

Вы должны быть знакомы с инфраструктурой, которую хотите использовать, поскольку данная книга не предлагает никаких рекомендаций в этом отношении. (Однако, в главе 3 приведены инструкции по установке Anaconda, а в главе 4 вы можете установить инфраструктуры TensorFlow и Keras, используемые в этой книге.) Чтобы предоставить вам максимум информации о языке Python и его применении для глубокого обучения, в этой книге не обсуждаются никакие проблемы специфичные для инфраструктур. Вам нужно знать, как устанавливать и использовать приложения и вообще работать с выбранной вами инфраструктурой, прежде чем начинать работу с этой книгой.

Вы должны знать, как работать с языком Python. В Интернете вы можете найти множество таких учебных пособий (см. примеры на `https://www.w3schools.com/python/` и `https://www.tutorialspoint.com/python/`).

Эта книга не является учебником по математике. Да, вы увидите здесь много примеров сложной математики, но основной упор делается на то, чтобы помочь вам использовать Python и науку о данных для выполнения задач глубокого обучения, а не преподавания математической теории. Мы приводим несколько примеров, в которых также обсуждается использование машинного обучения в применении к глубокому обучению. Главы 1 и 2 дадут вам лучшее представление о том, что вам нужно знать для успешного использования этой книги.

Эта книга также предполагает, что вы можете получить доступ к Интернету. Повсюду приведены многочисленные ссылки на сетевые материалы, которые расширят ваш опыт обучения. Но эти дополнительные источники полезны только в том случае, если вы действительно их найдете и используете.

Источники дополнительной информации

Эта книга — не конец вашего изучения языка Python или глубокого обучения, а только начало. Чтобы она стала для вас максимально полезной, мы предоставляем дополнительные источники информации. Получая от вас письма по электронной почте, мы сможем ответить на возникшие у вас вопросы, а также подсказать, как обновления Python или связанных с ним надстроек влияют на содержание книги. Вы также можете использовать следующие замечательные источники.

- **Шпаргалка.** Вы помните, как использовали в школе шпаргалки, чтобы получить лучшие оценки на контрольной? Да, это разновидность шпаргалки. В ней содержится ряд заметок о малоизвестных задачах, решаемых с помощью Python, машинным обучением и наукой о данных, известные не каждому. Шпаргалка находится в конце книги. Она содержит действительно полезную информацию, такую как наиболее распространенные ошибки программирования, вызывающие у людей затруднения при использовании языка Python.
- **Обновления.** Рано или поздно все изменяется. Например, мы могли не заметить грядущих изменений, когда смотрели в свои хрустальные шары во время написания этой книги. Когда-то это просто означало, что книга устарела и стала менее полезной, но теперь вы можете найти ее обновления по адресу `www.dummies.com`, если будете искать по названию этой книги. Кроме этих обновлений,

имеет смысл посетить блог автора по адресу http://blog.johnmuellerbooks.com/, содержащий ответы на вопросы читателей и связанные с книгой полезные материалы.

» **Сопутствующие файлы**. Эй! Кто действительно хочет набрать весь код в книге и восстановить все эти нейронные сети вручную? Большинство читателей предпочитают тратить свое время на работу с Python, выполнение задач по машинному или глубокому обучению и просмотр интересных вещей, которые они могут сделать, а не набирать текст. К счастью для вас, примеры, используемые в книге, доступны для скачивания, поэтому все, что вам нужно сделать, это прочитать книгу, чтобы изучить использование языка Python для методов глубокого обучения. Вы можете найти эти файлы на сайте www.dummies.com. Ищите по названию этой книги и прокрутите вниз появившуюся страницу до изображения обложки книги и щелкните по нему. Затем нажмите кнопку More about This Book (Подробнее об этой книге) и на открывшейся странице перейдите на вкладку Downloads (Загрузки). Сопутствующие файлы можно также загрузить со страницы русского издания книги по адресу: http://www.dialektika.com/books/978-5-907203-59-4.html

Что дальше

Пришло время приступить к изучению языка Python и для приключений в области глубокого обучения! Если вы абсолютный новичок в Python и его использовании для задач глубокого обучения, вам следует начать с главы 1 и продвигаться по книге со скоростью, позволяющей вам освоить как можно больше материала.

Если вы новичок, который спешит начать работу с языком Python для глубокого обучения как можно быстрее, вы можете перейти к главе 3 с пониманием того, что некоторые темы впоследствии могут оказаться немного запутанными. Переход к главе 4 — это нормально, если у вас уже установлена Anaconda (программный продукт, использованный в книге), но обязательно просмотрите хотя бы главу 3, чтобы вы знали, какие предположения мы сделали при написании этой книги.

Для выполнения задач глубокого обучения эта книга использует комбинацию TensorFlow и Keras. Даже если у вас есть некоторый опыт, ознакомьтесь с главой 4, чтобы понять, как настроить среду, используемую для этой книги. Неправильная настройка среды в соответствии с инструкциями почти наверняка приведет к сбоям при попытке запустить код.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info.dialektika@gmail.com`

WWW: `http://www.dialektika.com`

1 Появление глубокого обучения

В ЭТОЙ ЧАСТИ...

- Как глубокое обучение влияет на окружающий мир
- Взаимосвязь между глубоким обучением и машинным обучением
- Создание собственной настройки Python
- Необходимость среды в глубоком обучении

Глава 1

Введение в глубокое обучение

В ЭТОЙ ГЛАВЕ...

- Понятие глубокого обучения
- Работа с глубоким обучением
- Разработка приложений глубокого обучения
- Учет ограничений глубокого обучения

Вы, наверное, много слышали о глубоком обучении. Термин появляется повсеместно и, кажется, относится ко всему. В действительности, *глубокое обучение* (deep learning) — это подмножество *машинного обучения* (machine learning), которое, в свою очередь, является подмножеством *искусственного интеллекта* (artificial intelligence — AI). Первая задача этой главы — помочь вам понять, что такое глубокое обучение на самом деле и как оно применяется в современном мире. Вас это может удивить, но глубокое обучение — это не единственная игра в городе; есть и другие методы анализа данных. На самом деле, глубокое обучение решает определенный набор задач, когда дело доходит до анализа данных, поэтому вы вполне можете использовать другие методы и даже не знать об этом.

Глубокое обучение — это только часть искусственного интеллекта, но весьма важная часть. Вы можете видеть применение методов глубокого обучения для ряда задач, но не для всех. Фактически, некоторые люди связывают

глубокое обучение с задачами, которые оно выполнить не может. Следующим шагом в изучении глубокого обучения является понимание того, что оно может сделать, и чего не может.

В ходе работы над этой книгой вы напишете приложения, полагающиеся на глубокое обучение для обработки данных и создающие желаемый результат. Конечно, вам нужно немного узнать о среде программирования, прежде чем сможете что-то сделать. Несмотря на то, что в главе 3 обсуждается, как установить и настроить язык Python, используемый для демонстрации глубокого обучения в этой книге, сначала вам нужно узнать немного больше о доступных возможностях.

Глава завершается обсуждением того, почему глубокое обучение не должно быть единственным методом обработки данных в вашем наборе инструментов. Да, глубокое обучение может выполнять удивительные задачи при правильном использовании, но оно также может создавать серьезные проблемы, если его применить к задачам, для которых оно не слишком подходит. Иногда, для выполнения некоторых задач, вам стоит обратиться к другим технологиям или выяснить, какие еще технологии использовать с глубоким обучением, чтобы обеспечить наиболее эффективное и элегантное решение конкретных проблем.

Определение смысла глубокого обучения

Понимание глубокого обучения начинается с точного определения терминов. В противном случае вам будет трудно отделить рекламу от реальности в том, на что способно глубокое обучение. Глубокое обучение является частью как искусственного интеллекта, так и машинного обучения, как показано на рис. 1.1. Чтобы понять глубокое обучение, следует начать с внешней стороны, т.е. с искусственного интеллекта, а затем пройти путь через машинное обучение и, наконец, определить глубокое обучение. Следующие разделы помогут вам в этом процессе.

Начнем с искусственного интеллекта

Утверждение, что искусственный интеллект — это искусственный интеллект, не говорит вам на самом деле ничего осмысленного, вот почему возникает так много дискуссий и разногласий по поводу этого термина. Да, вы можете утверждать, что происходящее является искусственным, а не естественным. Однако интеллектуальная часть в лучшем случае неоднозначна. Люди определяют интеллект по-разному. Но вы можете сказать, что интеллект подразумевает определенные умственные усилия, состоящие из следующих действий.

Рис. 1.1. Глубокое обучение — это подмножество машинного обучения, которое является подмножеством искусственного интеллекта

- **Обучение**. Возможность получать и обрабатывать новую информацию.
- **Рассуждение**. Возможность манипулирования информацией различными способами.
- **Понимание**. Учет результата манипулирования информацией.
- **Понимание истины**. Определение достоверности обрабатываемой информации.
- **Понимание взаимоотношений**. Выяснение, как достоверные данные взаимодействуют с другими данными.
- **Учет смысла**. Применение истины к конкретным ситуациям в соответствии с их отношениями.
- **Отделение факта от веры**. Определение, адекватно ли подтверждаются данные доказуемыми источниками, неизменно демонстрирующими достоверность.

Список может быть довольно длинным, но даже в таком виде он подлежит толкованию любым, кто считает его достоверным. Однако, как видно из списка, интеллект зачастую следует процессу, который компьютерная система вполне может имитировать в ходе симуляции.

1. Установить цель на основании потребностей или желаний.
2. Оценить ценность любой относящейся к цели информации, известной в настоящее время.
3. Собрать дополнительную информацию, способную относится к цели.
4. Манипулировать данными так, чтобы они приобрели форму, соответствующую существующей информации.
5. Определить отношения между существующей и новой информацией, а также ее истинную ценность.
6. Определить, достигнута ли цель.
7. Изменить цель в свете новых данных и их влияния на вероятность успеха.
8. Повторять этапы 2 – 7 по мере необходимости, пока цель не будет достигнута (обнаружена истина) или пока не будут исчерпаны возможности ее достижения (обнаружена ложь).

ЗАПОМНИ!

Даже если вы можете создавать алгоритмы и предоставлять доступ к данным для поддержки этого процесса на компьютере, возможности компьютера в области интеллекта сильно ограничены. Например, компьютер не способен что-либо понять, поскольку для манипулирования данными он полагается на машинные процессы с использованием чисто математического механизма. Аналогично, компьютеры не могут легко отличить правду от неправды. Фактически, ни один компьютер не может полностью реализовать какую-либо умственную деятельность, описанную в приведенном выше списке описания интеллекта.

Размышляя об искусственном интеллекте, следует учитывать цели людей, которые развивают искусственный интеллект. Цель заключается в подражании интеллекту человека, а не в его копировании. На самом деле компьютер не мыслит, но создает видимость мышления. Тем не менее, фактически компьютер поддерживает только логко-математическую форму интеллекта. Компьютер умеренно успешен в имитации визуально-пространственного и телесно-кинестетического интеллекта. Компьютер обладает низкими способностями в области межличностного и лингвистического интеллекта. Однако, в отличие от людей, компьютер неспособен имитировать внутриличностный или творческий интеллект.

Роль искусственного интеллекта

Как упоминалось в предыдущем разделе, первая концепция, которую важно осознать, — это то, что искусственный интеллект действительно не имеет никакого отношения к человеческому интеллекту. Да, некоторые модели

искусственного интеллекта созданы для симуляции человеческого интеллекта, но это только симуляция. Размышляя об искусственном интеллекте, обратите внимание на взаимосвязь между поиском цели, обработкой данных, используемой для достижения этой цели, и сбором данных для лучшего понимания цели. Для достижения результата, искусственный интеллект полагается на алгоритмы, и результат может иметь некое отношение к человеческим задачам или методам их решения, а может и не иметь. С учетом этого искусственный интеллект можно классифицировать четырьмя способами.

- **Действующий по-человечески.** Когда компьютер действует, как человек, он лучше проходит тест Тьюринга, в котором компьютер побеждает, когда его невозможно отличить от человека (см. детали на `http://www.turing.org.uk/scrapbook/test.html`). К этой категории относится все, что, по мнению средств массовой информации, является искусственным интеллектом. Такие технологии используются для обработки текстов на естественном языке, для представления знаний, автоматизации формулирования логических выводов и машинного обучения (все четыре способны проходить тест). Первоначальный тест Тьюринга не подразумевал физического контакта. Более новый, полный тест Тьюринга действительно подразумевает физический контакт в форме перцепционной способности взаимодействия, а значит, для успеха компьютеру требуются машинное зрение и робототехника. Современные технологии склоняются к идее достижения цели, а не полного подражания людям. Например, братья Райт (Wright Brothers) добились успеха в создании самолета не точным копированием полета птиц; птицы, скорее, подсказали идеи, которые привели к созданию аэродинамики, которая, в конечном счете, привела к полету человека. Цель в том, чтобы летать. И птицы, и люди решают одинаковую задачу, но используют разные подходы.
- **Думающий по-человечески.** Когда компьютер думает, как человек, он решает такие задачи как вождение автомобиля, требующие интеллекта схожего с человеческим (а не механических процедур). Чтобы удостовериться, думает ли программа, как человек, необходим некоторый метод выяснения того, как думают люди, определив подход когнитивного моделирования. Эта модель полагается на три методики.
  - **Самоанализ.** Выявление и документирование методик, используемых для достижения цели при контроле собственных мыслительных процессов.
  - **Психологическая проверка.** Наблюдение за поведением человека и внесение результатов в базу данных подобных поведений

других людей при подобном стечении обстоятельств, задаче, ресурсах и условиях окружающей среды (помимо всего прочего).

- **Нейровизуализация.** Контроль мозговой деятельности непосредственно, с использованием различных аппаратных средств, таких как компьютерная томография (Computerized Axial Tomography — CAT), позитронно-эмиссионная томография (Positron Emission Tomography — PET), магнитно-резонансная томография (Magnetic Resonance Imaging — MRI) и магнитоэнцефалография (Magnetoencephalography — MEG).

После создания модели можно написать использующую ее программу. С учетом огромного разнообразия человеческих мыслительных процессов и трудностей их точного представления в программе результаты в лучшем случае носят экспериментальный характер. Эта категория, мышление по-человечески, зачастую используется в психологии и других областях, в которых моделируется человеческий мыслительный процесс для создания реалистичных симуляций.

» **Думающий рационально.** Изучение процесса мышления людей с использованием некого стандарта позволяет выработать правила, описывающие типичное человеческое поведение. Человека считают рациональным, когда он следует этим правилам поведения в пределах определенных уровней отклонения. Компьютер, который думает рационально, полагается на заранее заданные правила поведения для выработки направлений взаимодействия с окружающей средой на основании имеющихся данных. Цель этого подхода заключается в как можно более логичном решении проблем. Как правило, этот подход подразумевает создание базовой методики решения задачи, которая впоследствии будет модифицирована для фактического решения конкретной проблемы. Другими словами, решение проблемы в принципе зачастую отличается от ее решения на практике, но отправная точка все же нужна.

» **Действующий рационально.** Изучение того, как люди действуют в неких ситуациях, и при определенных условиях, позволяет определить, какие методики эффективны. Компьютер, действующий рационально, полагается на заранее заданные действия при взаимодействии с окружающей средой на основании условий, внешних факторов и имеющихся данных. Как и при рациональном мышлении, рациональные действия зависят от решения в принципе, которое может и не оказаться полезным практически. Однако рациональное действие предоставляет базовую линию, относительно которой компьютер может начать переговоры об успешном завершении задачи.

ЧЕЛОВЕК ПРОТИВ РАЦИОНАЛЬНЫХ ПРОЦЕССОВ

Человеческие процессы отличаются от рациональных процессов. Процесс *рационален*, если на основании текущей информации он всегда делает правильные вещи с идеальной эффективностью. Короче говоря, если рациональные процессы руководствуются книгой, то они полагают книгу абсолютно правильной. Человеческие процессы задействуют инстинкт, интуицию и другие переменные, которые не обязательно учтут книгу и могут даже не рассматривать имеющиеся данные. Например, рациональный способ управления автомобилем подразумевает неукоснительное следование правилам. Однако дорожный трафик не рационален. Если вы будете точно следовать правилам дорожного движения, то закончите влипшим во что-нибудь, поскольку другие водители не следуют правилам абсолютно точно. Поэтому, чтобы быть успешным, беспилотный автомобиль должен действовать разумно, а не рационально.

Сегодня искусственный интеллект используется в очень многих приложениях. Единственная проблема в том, что технология работает настолько хорошо, что вы даже не подозреваете о ее существовании. Фактически вы можете быть крайне удивлены, обнаружив, как много устройств в вашем доме уже использует эту технологию. Искусственный интеллект используется миллионами экземпляров и вполне безопасен, хотя и весьма драматичен по природе. Вот лишь несколько способов использования искусственного интеллекта.

- **Обнаружение мошенничества.** Вы получаете из банка запрос, платили ли вы своей кредитной карточкой за определенную покупку. Банк не любопытен; он просто обеспокоен тем фактом, что некто мог сделать покупку, используя вашу карточку. Искусственный интеллект банка обнаружил незнакомый шаблон расходов и предупредил, кого следует.
- **Планирование ресурсов.** Многим организациям необходимо эффективное планирование использования ресурсов. Например, больнице, вероятно, придется решать, куда поместить пациента, исходя из его потребностей, доступности квалифицированного персонала и ожидаемого периода пребывания пациента в больнице.
- **Комплексный анализ.** Люди нередко нуждаются в комплексном анализе, когда приходится учитывать слишком много факторов. Например, один и тот же набор симптомов может свидетельствовать о нескольких проблемах. Чтобы спасти пациенту жизнь, врачу или другому специалисту может понадобиться помощь в своевременной постановке диагноза.

- **Автоматизация.** Любая форма автоматизации может извлечь пользу из применения искусственного интеллекта для реакции на непредусмотренные изменения или события. Проблема некоторых типов автоматизации сегодня заключается в том, что непредусмотренное событие, такое как нахождение объекта в неположенном месте, фактически может привести к ее отказу. Добавление искусственного интеллекта к автоматизации может позволить справиться с непредвиденным событием и продолжить работу, как будто ничего не случилось.
- **Клиентская служба.** На другом конце линии клиентской службы, в которую вы сегодня звонили, вполне могло и не быть человека. Автоматизация сегодня достаточно хороша, чтобы, используя различные сценарии и ресурсы, справиться с подавляющим большинством вопросов. При корректных интонациях голоса (также предоставляемых искусственным интеллектом) вы не сможете даже с уверенностью сказать, что говорили с компьютером.
- **Системы безопасности.** Большинство систем безопасности современных машин различных видов полагается на искусственный интеллект, чтобы перехватить управление транспортным средством в критической ситуации. Например, многие автоматические тормозные системы полагаются на искусственный интеллект, чтобы остановить автомобиль исходя из всех исходных данных, которые может предоставить транспортное средство, например направления заноса.
- **Эффективность машин.** Искусственный интеллект может помочь контролировать машину так, чтобы добиться максимальной эффективности. Искусственный интеллект контролирует использование ресурсов, чтобы система не превышала скорости и для других задач. Каждая унция (28,3495 г) мощности будет использована точно так, как необходимо, чтобы оказать желаемую услугу.

Сосредоточимся на машинном обучении

Машинное обучение является одним из подмножеств искусственного интеллекта, и единственным, обсуждаемым в этой книге. Целью машинного обучения является симуляция процесса обучения человека, чтобы приложение могло приспособиться к неопределенным или неожиданным ситуациям. Для решения этой задачи машинное обучение использует алгоритмы анализа огромных наборов данных.

ЗАПОМНИ!

В настоящее время машинное обучение не способно обеспечить такой вид искусственного интеллекта, как в кино (машина не может учиться интуитивно, как человек); он может имитировать только определенные виды обучения, и только в узком диапазоне. Даже наилучшие алгоритмы не могут думать, чувствовать, обладать любой формой самосознания или иметь свободу выбора. Из-за этих ограничений в восприятии, машинам очень сложно понять те характеристики, которые для человека являются основополагающими. Машины не осознают себя.

На что машинное обучение способно, так это выполнять прогнозирующую аналитику намного быстрее, чем любой человек. В результате машинное обучение может помочь людям работать эффективнее. Да, искусственный интеллект в нынешнем состоянии способен выполнить выдающийся анализ, но смысл его результатов все еще должны осознавать люди, они же принимают необходимые моральные и этические решения на его основании. По существу машинное обучение обеспечивает лишь обучающую часть искусственного интеллекта, и эта часть ничуть не готова создать искусственный интеллект того вида, который вы видите в фильмах.

ЗАПОМНИ!

Основная причина несоответствия между обучением и интеллектом — в человеческом предположении, что простой способности машины справляться со своей работой (обучение) уже достаточно для сознания (интеллект). Это предположение ничем не подтверждено для машинного обучения. То же самое происходит, когда люди полагают, что компьютер преднамеренно создает для них проблемы. Компьютер не имеет эмоций, а потому действует только на основании предоставленных данных и инструкций для их обработки. Истинный искусственный интеллект получится только тогда, когда компьютеры, наконец, смогут подражать следующей сложной комбинации, используемой в природе.

- **Генетика.** Медленное обучение из поколения в поколение.
- **Обучение.** Быстрое обучение на базе организованных источников.
- **Исследование.** Спонтанное обучение на базе средств массовой информации и общения между собой.

Чтобы привести концепции машинного обучения в соответствие с тем, что на самом деле может делать машина, необходимо рассмотреть конкретные случаи применения машинного обучения. Полезно также узнать об использовании машинного обучения и вне областей, традиционных для искусственного

интеллекта. Вот несколько случаев использования машинного обучения, которые могли бы и не быть связаны с искусственным интеллектом.

- **Управление доступом.** Во многих случаях доступ либо разрешают, либо запрещают. Смарт-карта сотрудника предоставляет доступ к ресурсу точно так же, как и ключ, на протяжении веков. Некоторые замки действительно позволяют задавать время и даты, когда их можно открывать, но такой контроль — не ответ на каждую потребность. Используя машинное обучение, можно определить, может ли сотрудник получить доступ к данному ресурсу, исходя из его роли и потребности. Например, сотрудник может получить доступ к учебной комнате, если обучение является его ролью.
- **Защита животных.** Океан может казаться достаточно большим, чтобы морские животные и суда могли без проблем в нем сосуществовать. К сожалению, суда ежегодно ранят множество животных. Алгоритм машинного обучения может позволить судам избегать животных, изучив издаваемые ими звуки и другие характеристики животных и судов. (Корабль будет полагаться на подводное акустическое оборудование, чтобы отслеживать издаваемые животными звуки, слышимые на большом расстоянии от корабля.)
- **Предсказание времени ожидания.** Большинству людей не нравится ждать, когда они понятия не имеют, как долго придется это делать. Машинное обучение позволяет приложению определить время ожидания на основании уровня персонала, их загрузки, сложности решаемых ими проблем, доступности ресурсов и т.д.

Переход от машинного обучения к глубокому обучению

Как упоминалось ранее, глубокое обучение — это подмножество машинного обучения. В обоих случаях алгоритмы, казалось бы, учатся в ходе анализа огромных объемов данных (но в некоторых случаях обучение может происходить даже на крошечных наборах данных). Тем не менее, глубокое обучение различается в зависимости от глубины анализа и вида автоматизации. Вы можете резюмировать различия между ними следующим образом.

- **Совершенно другая парадигма**. Машинное обучение представляет собой набор из множества различных методов, позволяющих компьютеру учиться на данных и использовать изученное, чтобы дать ответ, зачастую в форме предсказания. Машинное обучение опирается на различные парадигмы, такие как использование статистического анализа, поиск аналогий в данных, использование логики и работа с символами. Сравните множество методов, используемых в машинном обучении, с единственной техникой, используемой

для глубокого обучения, которая имитирует функционирование человеческого мозга. Она обрабатывает данные, используя такие вычислительные блоки как *нейроны*, расположенные в упорядоченных секциях — *слоях* (layer). Методика, лежащая в основе глубокого обучения, — это *нейронная сеть* (neural network).

- **Гибкие архитектуры**. Решения машинного обучения предлагают множество настроек, *гиперпараметров* (hyperparameter), которые вы задаете для оптимизации алгоритма обучения на основе данных. Решения глубокого обучения также используют гиперпараметры, но они используют несколько настраиваемых пользователем слоев (тип и количество задает пользователь). Фактически, в зависимости от получаемой нейронной сети, количество слоев может быть довольно большим и формировать уникальные нейронные сети, способные к специализированному обучению: некоторые могут научиться распознавать образы, а другие — распознавать и анализировать голосовые команды. Дело в том, что термин *глубокое* (deep) вполне оправдан; он относится к большому количеству слоев, потенциально используемых при анализе. Архитектура в решении глубокого обучения состоит из множества различных нейронов и их расположения в слоях.
- **Автономное создание признаков**. Для достижения успеха, решения машинного обучения требуют вмешательства человека. Для правильной обработки данных ученые и аналитики используют свои собственные знания при разработке алгоритмов. Например, в решении машинного обучения по определению стоимости дома на основании данных, содержащих показатели о стенах разных комнат, алгоритм машинного обучения не сможет рассчитать площадь дома, если аналитик не укажет заранее, как это сделать. *Создание признаков* (feature creation) — это создание правильной информации для алгоритма машинного обучения, оно занимает много времени. Глубокое обучение не требует, чтобы люди выполняли какие-либо действия по созданию признаков, поскольку благодаря своим многочисленным слоям оно определяет свои признаки лучше. Именно поэтому глубокое обучение превосходит машинное обучение в некоторых очень сложных задачах, таких как распознавание голоса и образов, понимание текста или победа чемпиона-человека в игре Го (цифровая форма настольной игры, в которой вы захватываете территорию противника).


ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Вам необходимо понять ряд проблем, связанных с решениями глубокого обучения, наиболее важной из которых являются то, что компьютер все еще ничего не понимает и не знает о предложенном им решении. Он просто предоставляет форму петли обратной связи и

автоматизации, объединенных для получения желаемых результатов за меньшее время, чем мог бы получить человек вручную (точно такой же результат), используя решение для машинного обучения.

Вторая проблема заключается в том, что некоторые невежественные люди настаивают на том, что слои глубокого обучения скрыты и недоступны для анализа. Это не так. Все, что может создать компьютер, в конечном итоге отслеживается человеком. Фактически, Общий регламент по защите данных (General Data Protection Regulation — GDPR) (`https://eugdpr.org/`) требует, чтобы такой анализ выполняли люди (см. подробнее на `https://www.pcmag.com/commentary/361258/how-gdpr-will-impact-the-ai-industry`). Требование выполнить этот анализ является спорным, но действующий закон гласит, что кто-то должен это делать.

Третья проблема заключается в том, что самонастройка происходит лишь в ограниченной мере. Глубокое обучение не всегда обеспечивает надежный или правильный результат. На самом деле, решения глубокого обучения могут сработать не так, как надо (см. подробнее в статье по адресу `https://www.theverge.com/2016/3/24/11297050/tay-microsoft-chatbot-racist`). Даже если код приложения не ошибается, используемые для поддержки глубокого обучения устройства вполне могут ошибаться (см. подробнее на `https://www.pcmag.com/commentary/361918/learning-from-alexas-mistakes?source=SectionArticles`). Тем не менее, учитывая эти проблемы, вы можете увидеть использование глубокого обучения для ряда чрезвычайно популярных приложений, как описано на `https://medium.com/@vratulmittal/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-andbeyond-7c6130c43b01`.

Использование глубокого обучения в реальном мире

Не заблуждайтесь: люди используют глубокое обучение в реальном мире для решения широкого круга задач. Например, многие автомобили сегодня используют голосовой интерфейс. Даже с самого начала голосовой интерфейс способен выполнять простые задачи. Но чем больше вы говорите с ним, тем лучше он работает. Когда вы с ним разговариваете, интерфейс учится, и не только тому, как вы говорите, но и вашим личным предпочтениям. Следующие разделы содержат некоторую информацию о том, как глубокое обучение работает в реальном мире.

Концепция обучения

Когда люди учатся, они полагаются не только на данные. У людей есть интуиция и невероятное понимание того, что будет, а что нет. Частью этого врожденного знания является инстинкт, который передается из поколения в поколение через ДНК. Взаимодействие людей с вводом, также отличается от того, как поступает компьютер. Для компьютера, обучение — это создание базы данных, состоящей из нейронной сети, в которую встроены весовые коэффициенты и смещения для обеспечения надлежащей обработки данных. Затем нейронная сеть обрабатывает данные, но сосем не так, как это делает человек.

Решение задач глубокого обучения

Люди и компьютеры лучше всего справляются с разными задачами. Люди лучше всего рассуждают, продумывают этические решения, они эмоциональны. Компьютер предназначен для очень быстрой обработки большого количества данных. Глубокое обучение обычно используют для решения задач требующих поиска шаблонов в огромных объемах данных, т.е. задач, решение которых не интуитивно понятно и не сразу заметно. В статье на `http://www.yaronhadad.com/deep-learning-most-amazing-applications/` рассказывается о 30 различных способах, которыми люди в настоящее время используют глубокое обучение. Практически в каждом случае вы можете свести задачу и ее решение к быстрой обработке огромных объемов данных, поиску шаблонов и последующему использованию этих шаблонов для обнаружения чего-то нового или для создания определенного вида вывода.

Использование глубокого обучения в приложениях

Глубокое обучение может быть автономным решением, как показано в этой книге, но зачастую оно используется как часть более крупного решения и совместно с другими технологиями. Например, сочетание глубокого обучения с экспертными системами не является редкостью. Это сочетание, в некоторой степени, описывает статья `https://www.sciencedirect.com/science/article/pii/0167923694900213`. Однако реальные приложения — это не просто числа, полученные из некого туманного источника. При работе в реальном мире следует также учитывать различные виды источников данных и понимать, как эти источники работают. Для получения информации от камеры может потребоваться одно решение глубокого обучения, в то время как термометр или датчик приближения могут выдавать простые числа (или аналоговые данные, требующие некой обработки). Реальные решения непредсказуемы, поэтому в вашем наборе инструментов должно быть заготовлено более одного решения задачи.

Среда программирования для глубокого обучения

Вы, вероятно, предполагаете, что для погружения в глубокое обучение вам придется прыгать сквозь ужасный набор обручей и выучить навыки эзотерического программирования. Это правда, что вы получаете гибкость, когда пишете приложения на одном из языков программирования, хорошо подходящих для потребностей глубокого обучения. Однако, Deep Learning Studio (см. подробнее на `https://towardsdatascience.com/is-deep-learning-without-programming-possible-be1312df9b4a`) и другие подобные продукты позволяют создавать решения глубокого обучения и без программирования. По сути, такие решения включают описание того, что вы хотите получить в ходе графического определения модели. Такие решения хорошо подходят для простых задач, которые уже приходилось решать другим, но им не хватает гибкости, чтобы сделать нечто совершенно другое, решить задачу, требующую чего-то большего, чем простой анализ.

Решения глубокого обучения в облаке, например, предоставляемые Amazon Web Services (AWS) (`https://aws.amazon.com/deep-learning/`), вполне могут обеспечить вам дополнительную гибкость. Эти среды стремятся также упростить среду разработки, обеспечивая столько поддержки, сколько вам нужно. Фактически, AWS предоставляет поддержку для различных видов бессерверных вычислений (`https://aws.amazon.com/serverless/`), при которых вы не беспокоитесь о какой-либо инфраструктуре. Но эти решения могут быть довольно дороги. Несмотря на то, что они обеспечивают вам бо́льшую гибкость, чем использование готового решения, они все же не так гибки, как использование реальной среды разработки.

У вас есть и другие, не подразумевающие программирования решения. Например, если вам нужны мощь и гибкость, но вы не хотите программировать, чтобы получить их, вы можете положиться на такой продукт, как MATLAB (`https://www.mathworks.com/help/deeplearning/ug/deep-learning-in-matlab.html`), предоставляющий инструменты для глубокого обучения. В MATLAB и некоторых других средах больше внимания уделяется алгоритмам, которые вы хотите использовать, но чтобы получить от них полную функциональность, вам нужно как минимум писать сценарии, а значит, вам в какой-то степени придется программировать. Проблема этих сред заключается в том, что им может недоставать мощи, поэтому некоторые решения могут занять больше времени, чем вы ожидаете.

ЗАПОМНИ!

В какой-то момент, независимо от количества других опробованных вами решений, серьезные проблемы глубокого обучения потребуют программирования. Когда вы просматриваете варианты в Интернете, вы часто видите, что искусственный интеллект, машинное обучение и глубокое обучение смешаны вместе. Однако, как и эти три технологии работают на разных уровнях, так и языки программирования вам нужны разные. Хорошее решение глубокого обучения потребует использования многопроцессорной обработки, предпочтительно с использованием *графического процессора* (Graphics Processing Unit — GPU), обладающего большим количеством ядер. Выбранный вами язык должен также поддерживать графический процессор через совместимую библиотеку или пакет. Так что простого выбора языка обычно недостаточно; вам необходимо продолжить исследование, чтобы убедиться, что этот язык действительно соответствует вашим потребностям. С учетом этого, вот основные языки (в порядке популярности, на момент написания этой статьи), подходящие для глубокого обучения (как определено на `https://www.datasciencecentral.com/profiles/blogs/which-programming-language-is-considered-to-be-best-for-machine`):

- Python
- R
- MATLAB (язык сценариев, а не продукт)
- Octave

Единственная проблема с этим списком в том, что у всех разработчиков свои мнения. Языки Python и R обычно присутствуют в верхней части всех списков, но после них вы можете найти всевозможные мнения. Статья на `https://www.geeksforgeeks.org/top-5-best-programming-languages-for-artificial-intelligence-field/` дает вам несколько альтернативных идей. При выборе языка вы обычно должны учитывать следующие вопросы.

- **Кривая обучения**. Ваш опыт может многое сказать о том, что вам легче всего изучить. Вероятно, Python — лучший выбор для тех, кто программировал в течение нескольких лет, а язык R может быть лучшим выбором для тех, кто уже испытал функциональное программирование. MATLAB или Octave лучше всего подойдут для математика.
- **Скорость**. Любое решение глубокого обучения потребует больших вычислительных ресурсов. Многие люди говорят, что, поскольку язык R является статистическим языком, он предлагает больше

возможностей для поддержки статистики и обычно обеспечивает более быстрый результат. На самом деле, Python великолепно поддерживает параллельное программирование, что вероятно, компенсирует это преимущество, когда у вас есть необходимое оборудование.

- **Поддержка сообщества**. Существует множество форм поддержки сообщества, но две из них, наиболее важны для глубокого обучения, — это помощь в определении решения и доступ к множеству готовых программных средств. Язык Octave, вероятно, обладает наименьшей поддержкой сообщества, а Python — наибольшей.
- **Стоимость**. Стоимость языка зависит от того, какое решение вы выберете и где вы его используете. Например, MATLAB — это лицензионный продукт, который требует покупки, поэтому при его использовании вы платите сразу. Но несмотря на то, что другие языки изначально бесплатны, вы можете найти скрытые затраты, такие как запуск вашего кода в облаке, чтобы получить доступ к поддержке GPU.
- **Поддержка инфраструктуры DNN**. Инфраструктура может значительно облегчить работу с вашим языком. Однако у вас должна быть структура, которая хорошо работает со всеми остальными частями вашего решения. Две самые популярные инфраструктуры — это TensorFlow и PyTorch. Как ни странно, Python является единственным языком, который поддерживает обе, поэтому он предлагает вам наибольшую гибкость. С MATLAB вы используете Caffe и TensorFlow с языком R.
- **Готовность к выпуску**. Язык должен поддерживать тот вид вывода, который необходим для вашего проекта. В этом отношении язык Python великолепен, поскольку это язык общего назначения. Он позволяет создать любое необходимое приложение. Однако более конкретные инфраструктуры, предоставляемые другими языками, могут быть невероятно полезны для некоторых проектов, поэтому вам необходимо рассмотреть их все.

Преодоление заблуждений, связанных с глубоким обучением

В предыдущих частях этой главы обсуждались некоторые проблемы с восприятием глубокого обучения, например, убеждение некоторых людей, что оно используется везде и делает все. Проблема глубокого обучения заключается в том, что оно стало жертвой собственной медиа-кампании. Глубокое обучение

решает конкретные задачи. Следующие разделы помогут вам избежать заблуждений, связанных с глубоким обучением.

Экосистема запуска

Использование решения глубокого обучения сильно отличается от создания собственного решения глубокого обучения. Инфографика по адресу `https://www.analyticsvidhya.com/blog/2018/08/infographic-complete-deep-learning-path/` дает вам несколько советов о том, как начать работу с языком Python (процесс, который эта книга упрощает для вас). Только изучение может занять некоторое время. Однако после того как вы самостоятельно поработаете над несколькими проектами, вы начинаете понимать, что мифы вокруг глубокого обучения распространяется вплоть до начала установки. Глубокое обучение не является зрелой технологией, поэтому попытки использовать ее сродни строительству деревни на Луне или глубокому погружению в Марианскую впадину. Вы столкнетесь с проблемами, и технология будет постоянно преподносить сюрпризы.

Некоторые методы, используемые для создания решений глубокого обучения, тоже должны сработать. Концепция компьютера, действительно изучающего что-либо, является ложной, как и идея, что компьютеры вообще имеют какую-либо форму восприятия. Причина, по которой у Microsoft, Amazon и других поставщиков возникают проблемы с глубоким обучением, заключается в том, что даже у их инженеров нереальные ожидания. Глубокое обучение сводится к математике и сопоставлению с шаблоном, что, конечно, действительно интересно, но мнение, что это нечто большее, просто ошибочно.

Когда глубокое обучение неприменимо

Глубокое обучение — это только один из способов проведения анализа, причем не всегда наилучший. Например, несмотря на то, что экспертные системы считаются устаревшей технологией, вы не можете создать беспилотный автомобиль без таковой по причинам, описанным по адресу `https://aitrends.com/ai-insider/expert-systems-ai-self-driving-cars-crucial-innovative-techniques/`. Решение глубокого обучения оказывается слишком медленным для этой конкретной задачи. Ваш автомобиль, вероятно, будет содержать решение глубокого обучения, но вы, скорее всего, будете использовать его как часть голосового интерфейса.

Искусственный интеллект в целом и глубокое обучение в частности могут стать главными, когда технология не оправдает ожиданий. Например, статья `https://www.techrepublic.com/article/top-10-ai-failures-of-2016/` содержит список отказов искусственного интеллекта, некоторые из которых

основаны также на глубоком обучении. Ошибочно полагать, что глубокое обучение может каким-то образом принимать этические решения или что оно выберет правильный курс действий на основании чувств (которых нет ни у одной машины). Антропоморфизация использования глубокого обучения всегда будет ошибкой. Некоторые задачи просто требуют человека.

Скорость и способность мыслить как человек являются главными вопросами для глубокого обучения, но их слишком много. Например, вы не можете использовать глубокое обучение, если у вас недостаточно данных для обучения. На самом деле, статья `https://www.sas.com/en_us/insights/articles/big-data/5-machine-learning-mistakes.html` предлагает список из пяти наиболее распространенных ошибок, совершаемых людьми в отношении машинного и глубокого обучения. Если у вас нет нужных ресурсов, глубокое обучение никогда не сработает.

Глава 2

Знакомство с принципами машинного обучения

В ЭТОЙ ГЛАВЕ...

- Что включает в себя машинное обучение
- Методы машинного обучения
- Причины использования машинного обучения

Как обсуждалось в главе 1, концепция обучения для компьютера отличается от концепции обучения для людей. Тем не менее, глава 1 на самом деле не описывает машинное обучение, то есть обучение, которое использует компьютер, на любой глубине. В конце концов, то, на что вы действительно обращаете внимание, — это совершенно другой вид обучения, который некоторые люди рассматривают как комбинацию математики, поиска соответствия шаблону и хранения данных. Эта глава начинается с указания пути к более глубокому пониманию того, как работает машинное обучение.

Тем не менее, объяснение принципов машинного обучения не поможет вам полностью понять происходящее, когда вы работаете с ним. Важно также понять, как машинное обучение работает, что и является предметом следующего раздела главы. В этом разделе вы узнаете, что не существует совершенных методов проведения анализа. Возможно, чтобы получить ожидаемый результат, вам придется поэкспериментировать с вашим анализом. Кроме того,

существуют различные подходы к машинному обучению, каждый из которых имеет свои преимущества и недостатки.

Третья часть главы посвящена применению того, что вы изучили в двух предыдущих частях. Независимо от того, как вы формируете свои данные и выполняете их анализ, машинное обучение в некоторых случаях является неправильным подходом и никогда не даст вам полезного результата. Знание правильных областей применения машинного обучения очень важно, если вы хотите получать необходимый результат, который поможет вам выполнять интересные задачи. Цель машинного обучения заключается в том, чтобы узнать из данных нечто интересное, а затем сделать с ними что-то интересное.

Определение машинного обучения

Вот краткое определение *машинного обучения* (machine learning): это приложение искусственного интеллекта, способное автоматически учиться и совершенствоваться на опыте, не будучи явно запрограммированным для этого. Обучение происходит в результате анализа постоянно растущих объемов данных, поэтому основные алгоритмы не меняются, но внутренние веса и смещения, используемые в коде для выбора конкретного ответа, меняются. Конечно, все не так просто. В следующих разделах машинное обучение обсуждается подробней, чтобы вы могли понять его место в мире искусственного интеллекта и то, что глубокое обучение получает от него.

ЗАПОМНИ!

Аналитики данных нередко упоминают технологию, используемую для реализации машинного обучения, как алгоритм. *Алгоритм* (algorithm) — это последовательность пошаговых операций, обычно вычислений, которые могут решить определенную задачу за конечное количество шагов. В машинном обучении алгоритмы используют конечную серию шагов для решения задачи в ходе изучения данных.

Как работает машинное обучение

Алгоритмы машинного обучения учатся, но зачастую трудно найти точное значение термина *обучение* (learning), поскольку существуют разные способы извлечения информации из данных в зависимости от того, как построен алгоритм машинного обучения. Как правило, учебный процесс требует огромных объемов данных, которые обеспечивают ожидаемый ответ с учетом конкретных входных данных. Каждая *пара ввод/ответ* (input/response pair) представляет пример, и чем больше примеров, тем легче обучение алгоритма. Дело

в том, что каждая пара ввод/ответ вписывается в строку, кластер или другое статистическое представление, определяющее предметную область. Обучение — это процесс оптимизации модели, представляющей собой математическое обобщенное самих данных, так что она может прогнозировать или иным образом определить соответствующий ответ, даже когда она получает те входные данные, которых раньше не видела. Чем точнее модель может давать правильные ответы, тем лучше она извлекала уроки из предоставленных входных данных. Алгоритм *подбирает* (fit) модель к данным, и этот *процесс подбора* (fitting process) является обучением.

На рис. 2.1 показан чрезвычайно простой график, имитирующий происходящее при машинном обучении. В данном случае, начиная с входных значений 1, 4, 5, 8 и 10 и сопоставляя их с соответствующими выходными значениями 7, 13, 15, 21 и 25, алгоритм машинного обучения определяет, что наилучшим способом представления взаимосвязи между вводом и выводом является формула $2x + 5$. Эта формула определяет модель, используемую для обработки входных данных (даже новых, не встречавшихся ранее) и расчета соответствующего выходного значения. Линия тенденции (модель) демонстрирует шаблон, сформированный данным алгоритмом, так что новый ввод 3 даст прогнозируемый результат 11. Хотя большинство сценариев машинного обучения намного сложнее данного (и алгоритм не может создать правила, точно соотносящие каждый ввод с выводом), этот пример дает упрощенное представление о происходящем. Вместо того чтобы индивидуально программировать ответ для входного значения 3, модель может вычислить правильный ответ на основе пар ввод/ответ, которые она изучила.

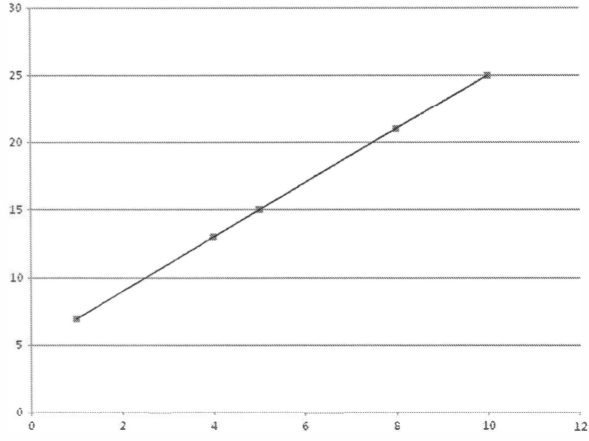


Рис. 2.1. Визуализация простого сценария машинного обучения

Это все чистая математика

Основная идея машинного обучения заключается в том, что вы можете представлять реальность, используя математическую функцию, которую алгоритм не знает заранее, но которую он может вывести после изучения некоторых данных (всегда в форме пар вводов и выводов). Вы можете выразить реальность и всю ее сложность в терминах неизвестных математических функций, которые алгоритмы машинного обучения находят и делают доступными в качестве модификации своих внутренних математических функций. То есть каждый алгоритм машинного обучения построен на основе модифицируемой математической функции. Функция может быть изменена, поскольку для этого она имеет внутренние параметры или веса. В результате алгоритм может адаптировать функцию к конкретной информации, взятой из данных. Эта концепция является основной идеей для всех видов алгоритмов машинного обучения.

Обучение в машинном обучении носит чисто математический характер и завершается тем, что определенные входные данные связываются с определенными выходными данными. Это не имеет ничего общего с пониманием того, что изучил алгоритм. (Когда люди анализируют данные, мы в определенной степени строим понимание этих данных.) Этот процесс зачастую описывается как обучение, поскольку алгоритм учится сопоставлять правильный ответ (выходные данные) с каждым предложенным вопросом (входными данными). (Этот процесс подробно описан в книге *Machine Learning For Dummies* Джона Пола Мюллера и Луки Массарон (издательство Wiley).)

Несмотря на отсутствие осознанного понимания и сугубо математический процесс, машинное обучение может оказаться полезным при решении многих задач. Оно позволяет многим приложениям искусственного интеллекта имитировать рациональное мышление в определенном контексте, когда обучение происходит с использованием правильных данных.

Разные стратегии обучения

Машинное обучение использует несколько разных способов обучения на основе данных. В зависимости от ожидаемого результата и типа вводимых вами данных, вы можете классифицировать алгоритмы по стилю обучения. Выбираемый вами стиль зависит от типа имеющихся данных и ожидаемого результата. Вот четыре стиля обучения, используемых для создания алгоритмов.

- С учителем (supervised).
- Без учителя (unsupervised).

- Самообучение (self-supervised).
- С подкреплением (reinforcement).

Подробней стили обучения обсуждаются в следующих разделах.

С учителем

При работе алгоритма обучения с учителем входные данные помечаются и имеют определенный ожидаемый результат. Вы используете обучение для создания модели, которую алгоритм подбирает к данным. По мере обучения прогнозы или классификации становятся все более точными. Вот несколько примеров алгоритмов обучения с учителем.

- Линейная или логистическая регрессия.
- Метод опорных векторов (Support Vector Machine — SVM).
- Наивный Байесовский классификатор.
- K-ближайшие соседи (K-Nearest Neighbors — KNN).

Необходимо различать задачи регрессии, целью которых является числовое значение, и задачи классификации, целью которых является качественная переменная, такая как класс или тег. Задача регрессии может определить средние цены на дома в районе Бостона, в то время как примером задачи классификации является различие между видами цветов ириса на основе их чашелистика и лепестков. Вот несколько примеров обучения с учителем.

Ввод данных (X)	Вывод данных (Y)	Реальное применение
История покупок клиентов	Список товаров, которые клиенты никогда не покупали	Система рекомендаций
Образы	Список коробок, помеченных неким названием	Обнаружение и распознавание образов
Английский текст в виде вопросов	Английский текст в виде ответов	Чатбот, приложение, способное беседовать
Английский текст	Немецкий текст	Машинный перевод
Аудио	Текстовая транскрипция	Распознавание речи
Образ, полученный сенсором	Поворот, торможение или ускорение	Планирование поведения при автономном вождении

Без учителя

При работе с алгоритмами обучения без учителя входные данные не помечаются, а результаты не известны. В этом случае необходимую модель дает анализ структур в данных. Структурный анализ может иметь несколько целей, таких как уменьшение избыточности или группирование похожих данных. Примеры обучения без учителя.

- Кластеризация.
- Обнаружение аномалий.
- Нейронные сети.

Самообучение

В Интернете вы найдете описания всевозможных видов обучения, но *самообучение* (self-supervised) — это отдельная категория. Некоторые люди описывают это как автономное обучение без учителя, которое обладает преимуществами обучения с учителем, но без всей работы, необходимой для маркировки данных.

ЗАПОМНИ!

Теоретически самообучение может решить проблемы с другими видами обучения, доступными для использования в настоящее время. В следующем списке самообучение сравнивается с другими используемыми видами обучения.

- **Обучение с учителем**. Самой близкой формой обучения, связанной с самообучением, является обучение с учителем, поскольку оба вида обучения основаны на парах входных и помеченных выходных данных. Кроме того, обе формы обучения связаны с регрессией и классификацией. Тем не менее, разница в том, что самообучение не требует от человека маркировки результатов. Вместо этого для контекстного обнаружения выходной метки оно использует корреляции, встроенные метаданные или знания предметной области, встроенные во входные данные.
- **Обучение без учителя**. Подобно обучению без учителя, самообучение не требует маркировки данных. Следовательно, обучение без учителя фокусируется на структуре данных, то есть на шаблонах в данных. Поэтому вы не используете самообучение для таких задач, как кластеризация, группировка, уменьшение размерности, механизмы рекомендаций и тому подобное.
- **Обучение с частичным привлечением учителя**. Решение *обучения с частичным привлечением учителя* (semi-supervised learning) работает как решение без учителя в том смысле, что оно ищет

шаблоны в данных. Тем не менее, обучение с частичным привлечением учителя опирается на сочетание маркированных и немаркированных данных для выполнения задач быстрее, чем это возможно при использовании строго немаркированных данных. Для выполнения своей задачи самообучение никогда не требует меток и использует контекст, поэтому изначально оно фактически игнорирует метки.

С подкреплением

Вы можете рассматривать обучение с подкреплением как расширение самообучения, поскольку обе формы используют одинаковый подход к обучению с немаркированными данными для достижения схожих целей. Однако в обучении с подкреплением добавлена петля обратной связи. Когда решение для обучения с подкреплением выполняет задачу правильно, оно получает положительную обратную связь, которая усиливает модель в связях целевых вводов и выводов. Кроме того, она может получить отрицательную обратную связь при неправильных решениях. В некоторых отношениях система работает почти так же, как дрессировка собаки, основываясь на системе поощрений.

Обучение, проверка и тестирование данных

Машинное обучение — это процесс, так же как и все в мире компьютеров. Чтобы создать успешное решение машинного обучения, вы выполняете следующие задачи по мере необходимости и так часто, как это необходимо.

- **Обучение**. Машинное обучение начинается, когда вы обучаете модель с использованием определенного алгоритма на основе конкретных данных. Учебные данные отделены от любых других данных, но они также должны быть репрезентативны. Если учебные данные не будут представлять предметную область, итоговая модель не сможет предоставить полезные результаты. В процессе обучения вы видите, как модель реагирует на учебные данные, и вносите необходимые изменения в используемые алгоритмы и способ, которым вы обрабатываете данные перед вводом в алгоритм.
- **Проверка**. Многие наборы данных достаточно велики, чтобы разделить их на учебную и тестовую части. Сначала вы обучаете модель, используя учебные данные, а затем проверяете ее, используя тестовые данные. Конечно, тестовые данные также должны точно представлять предметную область. Они также должны быть статистически совместимы с учебными данными. В противном случае результаты не будут отражать работу модели в реальных условиях.

» **Тестирование**. После обучения и проверки модели, вам все еще нужно протестировать ее с использованием реальных данных. Этот шаг важен, поскольку вам нужно убедиться, что модель действительно будет работать с большим набором данных, который вы не использовали ни для обучения, ни для проверки. Как и в случае этапов обучения и проверки, любые данные, которые вы используете на этом этапе, должны отражать предметную область, с которой вы хотите взаимодействовать, используя модель машинного обучения.

Обучение подразумевает алгоритм машинного обучения со всеми видами примеров необходимых вводов и выводов, ожидаемых от этих вводов. Впоследствии алгоритм машинного обучения использует этот ввод для создания математической функции. Другими словами, обучение — это процесс, в ходе которого алгоритм определяет, как адаптировать функцию к данным. Вывод такой функции обычно является вероятностью определенного вывода или просто числовым значением.

Чтобы получить представление о происходящем в процессе обучения, вообразите, что ребенок учится отличать деревья от предметов, животных и людей. Прежде чем ребенок сможет сделать это самостоятельно, учитель демонстрирует ему определенное количество образов деревьев, в комплекте со всеми фактами, которые отличают дерево от других объектов. К этим фактам могут относиться такие признаки как материал дерева (древесина), его части (ствол, ветви, листья или иголки, корни) и местоположение (в почве). Ребенок вырабатывает понимание того, как выглядит дерево, противопоставляя образы элементов дерева образам других примеров, таких как предметы мебели, которые сделаны из древесины, но не имеют других схожих с деревом характеристик.

Классификатор машинного обучения работает точно так же. Алгоритм классификатора предоставляет вам в качестве вывода класс. Например, он может сказать, что фотография, которую вы предоставили в качестве входных данных, соответствует классу дерева (а не животному или человеку). Для этого он формирует свои когнитивные способности, создавая математическую формулу, включающую все заданные входные признаки таким образом, что создается функция, способная отличать один класс от другого.

Поиск обобщения

Чтобы быть полезной, модель машинного обучения должна представлять обобщенный вид полученных данных. Если модель недостаточно точно соответствует данным, она *недообучена* (underfitted), т.е. она недостаточно подобрана из-за недостаточного обучения. С другой стороны, если модель подобрана слишком близко к данным, она *переобучена* (overfitted), и из-за *слишком большого* обучения облегает точки данных плотно как перчатка. Недообучение

и переобучение вызывают проблемы, поскольку модель недостаточно обобщена для получения полезных результатов. С учетом неизвестных входных данных, результирующие прогнозы или классификации будут содержать значения с большими ошибками. Только когда модель правильно подобрана к данным, она дает результаты в разумном диапазоне ошибок.

Вопрос обобщения также важен при принятии решения, использовать ли машинное обучение. Решение машинного обучения всегда обобщает конкретные примеры в общие примеры того же рода. То, как оно выполняет эту задачу, зависит от ориентации решения машинного обучения и алгоритмов, используемых для его работы.


ЗАПОМНИ!

Проблема для аналитиков данных и других специалистов, использующих методы машинного и глубокого обучения, заключается в том, что компьютер не будет отображать знак, указывающий, что модель правильно соответствует данным. Нередко именно человеческая интуиция решает, когда модель достаточно подготовлена, чтобы обеспечить хороший обобщенный результат. Кроме того, создатель решения должен выбрать правильный алгоритм из тысяч существующих. Без правильного алгоритма для подбора модели к данным результаты будут удручающим. Чтобы процесс отбора работал, аналитик должен обладать следующим.

- Глубокое знание доступных алгоритмов.
- Опыт работы с такими данными.
- Понимание желаемого результата.
- Желание экспериментировать с различными алгоритмами.

Последнее требование является наиболее важным, поскольку не существует жестких правил, согласно которым конкретный алгоритм будет работать с любыми типами данных в любой возможной ситуации. Если бы это было так, многие алгоритмы были бы не нужны. Чтобы найти лучший алгоритм, аналитик данных нередко прибегает к экспериментам с несколькими алгоритмами и сравнивает результаты.

Знакомство с предубеждением

Ваш компьютер не имеет *предубеждений* (bias) или *смещения*. У него нет цели мирового господства или усложнения вашей жизни. На самом деле, у компьютеров нет никаких целей. Единственное, что может компьютер, — это обеспечить вывод, основанный на вводе и методе обработки. Но предубеждение все же проникает в компьютер некоторыми способами и портит результаты, которые он дает.

- **Данные**. Сами данные могут содержать неверные или просто искаженные сведения. Например, если некое значение появляется в данных в два раза чаще, чем в реальном мире, результаты решения машинного обучения будут неверны, даже если сами данные верны.
- **Алгоритм**. Использование неподходящего алгоритма машинного обучения приведет к некорректному подбору модели к данным.
- **Обучение**. Слишком большое или слишком малое обучение влияет на степень соответствия модели данным, а, следовательно, и на результат.
- **Человеческая интерпретация**. Даже когда решение машинного обучения выдает правильный результат, использующий этот вывод, человек может неправильно его интерпретировать. Результат может быть ни чуть не лучше, а возможно и хуже, чем, если решение машинного обучения вообще не работает, как ожидалось. (Эта проблема рассматривается в статье по адресу `https://thenextweb.com/artificial-intelligence/2018/04/10/human-bias-huge-problem-ai-heres-going-fix/`)

Вы должны учитывать влияние предубеждения независимо от того, какое решение машинного обучения вы создаете. Важно знать, какие ограничения оно накладывает на ваше решение, и является ли решение достаточно надежным для обеспечения полезного результата.

Сложность модели

Когда дело доходит до машинного обучения, чем проще, тем всегда лучше. Полезный вывод для вашего решения машинного обучения способно обеспечить множество различных алгоритмов. Но лучший алгоритм, который и нужно использовать, это алгоритм, дающий самые простые ответы, и который проще понять. Бритва Оккама (`http://math.ucr.edu/home/baez/physics/General/occam.html`) обычно считается лучшей стратегией для подражания. По сути, этот принцип предлагает использовать простейшее решение, способное справиться с конкретной задачей. По мере увеличения сложности увеличивается и вероятность ошибок.

Разнообразие способов обучения

Часть обучения в машинном обучении делает его динамичным, т.е. способным изменять себя при получении дополнительных данных. Способность к обучению отличает машинное обучение от других видов искусственного интеллекта, таких как графы знаний и экспертные системы. Это не делает машинное

обучение лучше, чем другие типы искусственного интеллекта (как описано в главе 1), оно просто полезней для определенного набора задач. Конечно, проблема количественного определения того, что влечет за собой обучение, заключается в том, что люди и компьютеры смотрят на обучение по-разному. Кроме того, компьютеры используют не такие методы обучения, как люди, и некоторые люди могут вообще не воспринимать обучающую часть машинного обучения как обучение. В следующих разделах обсуждаются методы, используемые алгоритмами машинного обучения для обучения, чтобы вы могли лучше понять, что машинное обучение и человеческое по своей сути различны.

Бесплатных обедов не бывает

Возможно, вы слышали распространенный миф о том, что, имея все, что нужно для вывода в компьютер, можно получить решение, не прилагая особых усилий. К сожалению, не существует абсолютного решения какой-либо задачи, и лучшие ответы зачастую бывают весьма дорогостоящими. Работая с алгоритмами, вы быстро обнаруживаете, что некоторые из них работают лучше других при решении определенных задач, но также не существует единственного алгоритма, который лучше всего подходит для каждой задачи. Все дело в математическом механизме алгоритмов. Одни математические функции хороши для представления неких задач, но они могут стать причиной неких других проблем. У каждого алгоритма есть своя специализация.

Пять основных подходов

Алгоритмы бывают разными и выполняют они разные задачи. Одним из способов классификации алгоритмов является классификация по школам — методам, которые, по мнению группы единомышленников, решат конкретную задачу. Конечно, существуют и другие способы категоризации алгоритмов, но этот подход имеет то преимущество, что он помогает вам лучше понять использование и ориентацию алгоритмов. В следующих разделах представлен обзор пяти основных алгоритмических методов.

Символическое рассуждение

Группа, называемая *символистами* (symbolists), опирается при поиске решения задачи на алгоритмы, использующие *символическое рассуждение* (symbolic reasoning). Термин *обратная дедукция* (inverse deduction) обычно интерпретируется как *индукция* (induction). В символических рассуждениях *дедукция* расширяет сферу человеческого знания, в то время как *индукция* повышает уровень знания. Индукция обычно открывает новые области исследования, а дедукция исследует эти области. Тем не менее, наиболее важным

соображением является то, что индукция — это научная часть рассуждений такого типа, а дедукция — инженерная. Две стратегии работают рука об руку, чтобы решать задачи, сначала открывая область потенциального исследования, а затем, исследуя данную область, чтобы определить, действительно ли это решает задачу.

Вот пример этой стратегии: дедукция сказала бы, что если дерево зеленое и что если зеленые деревья живы, то дерево должно быть живым. Индукция скажет, что дерево зеленое и что дерево тоже живое; следовательно, зеленые деревья живы. Индукция дает ответ на то, на что не хватает знания с учетом известного ввода и вывода.

Нейронные сети

Нейронные сети (neural network) — это детище группы *коннекционистов* (connectionists). Эта группа алгоритмов стремится воспроизвести функции мозга, используя кремний вместо нейронов. По сути, каждый из нейронов (созданный в виде алгоритма, моделирующего реальный аналог) решает небольшую часть задачи, а параллельное использование множества нейронов решает задаче в целом.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Нейронная сеть может обеспечить метод коррекции ошибочных данных, и наиболее популярным из них является *обратное распространение* (backpropagation). (В состоящей из двух частей статье, по адресу `http://www.breloff.com/no-backprop/` и `http://www.breloff.com/no-backprop-part2/`, рассматриваются альтернативы обратного распространения.) При использовании обратного распространения (или обратного распространения ошибок) стремятся определить условия, при которых ошибки удаляются из сетей построенных подобно нейронам человека. Для этого изменяют *веса* (weight) (количество участия в результате конкретного входного значения) и *смещения* (bias) (какие признаки выбраны) сети. Задача в том, чтобы продолжать изменять весовые коэффициенты и смещения до тех пор, пока фактический результат не будет соответствовать целевому результату.

Теперь искусственный нейрон срабатывает и передает свое решение следующему нейрону в серии. Решение, созданное каждым отдельным нейроном, является лишь частью всего решения. Каждый нейрон продолжает передавать информацию следующему нейрону в серии, пока группа нейронов не создаст окончательный результат.

Эволюционные алгоритмы

Группа, называемая *эволюционистами* (evolutionaries), полагается при поиске решения задачи на принципы эволюции. Эта стратегия основана на выживании наиболее приспособленных решений и удалении любых других, которые не соответствуют желаемому результату. *Функция выживания* (fitness function) определяет жизнеспособность каждой функции в решении задачи.

Используя древовидную структуру, метод решения ищет наилучшее решение на основании вывода функции. Для построения функции следующего уровня выбирается победитель текущего уровня развития. Идея в том, что каждый следующий уровень будет ближе к решению задачи, и если решение еще не полное, то просто необходим следующий уровень. Для решения задачи данный конкретный тип алгоритма в значительной степени полагается на рекурсию (см. объяснение рекурсии на `https://www.cs.cmu.edu/~adamchik/15-121/lectures/Recursions/recursions.html`) и поддерживающие ее языки. Интересным результатом этой стратегии стали самостоятельно развивающиеся алгоритмы: одно поколение алгоритмов фактически создает следующее поколение.

Байесовский вывод

Байесианцы (bayesians) используют при поиске решения задачи различные статистические методы. Поскольку статистические методы способны создавать более одного очевидно правильного решения, одним из определяющих решений становится выбор функции, способной обеспечить наибольшую вероятность успеха. При использовании этих методов, например, в качестве входных данных вы можете получить набор симптомов. В качестве результата, алгоритм вычислит вероятность того, что эти симптомы являются результатом конкретного заболевания. Принимая во внимание, что множество заболеваний имеют одинаковые симптомы, вероятность важна, поскольку в некоторой ситуации вывод с более низкой вероятностью может фактически оказаться правильным при данных обстоятельствах.

СОВЕТ

В конечном счете, байесовские алгоритмы основаны на идее, никогда полностью не доверять какой-либо гипотезе (результат, который кто-то дал вам), не видя доказательств, использованных для ее создания (ввод, который некто другой использовал для создания гипотезы). Гипотезу подтверждает или опровергает анализ доказательств. Следовательно, вы не можете диагностировать заболевание, пока не проверите все симптомы. Одним из наиболее узнаваемых результатов этой группы алгоритмов является фильтр спама.

Системы, учащиеся на аналогиях

Аналогисты (analogizers) используют многоядерные машины для распознавания шаблонов в данных. Распознав шаблон одного набора входных данных, и сравнив его с шаблоном известного результата, вы можете выработать решение задачи. Цель в том, чтобы использовать сходство для определения наилучшего решения задачи. Это тот вид рассуждения, при котором полагают, что если в прошлом данное конкретное решение сработало при неких обстоятельствах, то при подобных же стечениях обстоятельств оно также должно сработать. Одним из наиболее известных результатов этой научной школы являются системы рекомендаций. Например, когда вы покупаете товар на Amazon, система рекомендаций предлагает другой связанный с этим товар, которые вы также могли бы захотеть купить.

Несколько разных подходов

Имеет смысл получить некоторое представление об алгоритмах, чтобы вы понимали, что они делают и почему. В предыдущем разделе алгоритмы рассматривались согласно создавшим их научным школам. Но есть и другие подходы классификации алгоритмов. Следующий список классифицирует некоторые популярные алгоритмы по сходству.

- **Искусственная нейронная сеть**. Моделирует структуру или функцию биологических нейронных сетей (иногда обе). Цель заключается в поиске соответствия шаблону для задач регрессии или классификации. Тем не менее, метод имитирует подход, используемый живыми организмами, а не строго полагается на исключительно математический подход. Вот примеры алгоритмов искусственной нейронной сети.
  - Перцептрон.
  - Нейронная сеть с прямой связью.
  - Нейронная сеть Хопфилда.
  - Сеть радиально-базисных функций (Radial Basis Function Network — RBFN).
  - Самоорганизующаяся карта (Self-Organizing Map — SOM).
- **Ассоциативные правила**. Извлекает правила, помогающие объяснить отношения между переменными в данных. Вы можете использовать эти правила для выявления полезных ассоциаций в огромных наборах данных, которые обычно легко пропустить. Вот наиболее популярные алгоритмы ассоциативных правил.
  - Алгоритм Apriori.
  - Алгоритм ECLAT.

- **Байесовский классификатор**. Применяет к вероятностным задачам теорему Байеса. Этот алгоритм применяется для задач классификации и регрессии. Вот примеры байесовских алгоритмов.
  - Наивный байесовский классификатор.
  - Гауссовский наивный байесовский классификатор.
  - Полиномиальный наивный байесовский классификатор.
  - Байесовская сеть доверия (Bayesian Belief Network — BBN).
  - Байесовская сеть (Bayesian Network — BN).
- **Кластеризация**. Описывает модель организации данных по классу или другим критериям. Результаты зачастую имеют центроидный или иерархический характер. То, что вы видите, — это отношения данных, помогающие понять их смысл, т.е. как значения влияют друг на друга. Вот примеры алгоритмов кластеризации.
  - Метод k-средних.
  - Метод k-медиан.
  - Алгоритм EM (Expectation Maximisation).
  - Иерархическая кластеризация.
- **Древо решений**. Создает модель решений на основе найденных в данных фактических значений. Полученная древовидная структура позволяет очень быстро выполнить сравнение между новыми данными и существующими данными. Эта форма алгоритма нередко находит применение для задач классификации и регрессии. В следующем списке показаны некоторые из наиболее распространенных алгоритмов дерева решений.
  - Дерево классификации и регрессии (Classification and Regression Tree — CART).
  - Алгоритм ID3 (Iterative Dichotomiser 3).
  - C4.5 и C5.0 (улучшенные версии алгоритма ID3).
  - Автоматическое обнаружение взаимодействий на основе статистик хи-квадрат (Chi-squared Automatic Interaction Detection — CHAID).
- **Глубокое обучение**. Модификация искусственных нейронных сетей, использующих несколько слоев для построения более сложных нейронных сетей для еще больших наборов данных. Данная конкретная группа алгоритмов хорошо работает с задачами обучения с частичным привлечением учителя, в которых количество помеченных данных минимально. Вот несколько примеров алгоритмов глубокого обучения.
  - Глубокая машина Больцмана (Deep Boltzmann Machine — DBM).

- Глубокая сеть доверия (Deep Belief Network — DBN).
- Сверточная нейронная сеть (Convolutional Neural Network — CNN).
- Рекуррентная нейронная сеть (Recurrent Neural Network — RNN).
- Многоярусный автокодировщик.

» **Уменьшение размерности**. Поиск и использование сходств в структуре данных аналогично алгоритмам кластеризации, но с использованием методов обучения без учителя. Цель заключается в суммировании или описании данных с использованием меньшего количества информации, чтобы набор данных становился меньше и проще в управлении. В некоторых случаях эти алгоритмы используют для задач классификации или регрессии. Вот список наиболее распространенных алгоритмов уменьшения размерности.

- Анализ основных компонентов (Principal Component Analysis — PCA).
- Факторный анализ (Factor Analysis — FA).
- Многомерное шкалирование (Multidimensional Scaling — MDS).
- Стохастическое вложение соседей с t-распределением (t-Distributed Stochastic Neighbor Embedding — t-SNE).

» **Ансамбли**. Объединяет группу из нескольких слабых моделей в единое целое, чьи индивидуальные прогнозы каким-то образом объединяются для формирования общего прогноза. Использование ансамбля может решать некоторые задачи быстрее, эффективнее или с меньшим количеством ошибок. Вот несколько популярных алгоритмов ансамбля.

- Бустинг.
- Бутстрэп-агрегирование (бэггинг).
- AdaBoost.
- Случайный лес.
- Метод градиентного бустинга (Gradient Boosting Machine — GBM).

» **Обучение на примерах**. Модель для решения задач, в которой учебные данные состоят из примеров, которые впоследствии используются для сравнения. Мера сходства помогает определить, когда новые примеры выгодно отличаются от примеров существующих в базе данных. Некоторые упоминают эти алгоритмы как "победитель получает все" или обучение на примерах, из-за способа их работы. В следующем списке приведены некоторые из наиболее популярных алгоритмов, связанных с этой категорией.

- K-ближайшие соседи (K-Nearest Neighbors — KNN).
- Квантизация векторов при обучении (Learning Vector Quantization — LVQ).

» **Регрессия**. Моделирует отношения между переменными. Эти отношения последовательно уточняются с использованием меры ошибки. Данная категория широко используется в статистическом машинном обучении. Следующий список содержит алгоритмы, обычно связываемые с этим типом.

- Регрессия обычных наименьших квадратов (Ordinary Least Squares Regression — OLSR).
- Логистическая регрессия.

» **Регуляризация**. Регулирует другие алгоритмы, штрафуя сложные решения и предпочитая более простые. Этот вид алгоритма обычно применяется с методами регрессии. Цель в том, чтобы гарантировать, что решение не потеряется в своей собственной сложности и предоставит результат в течение определенного периода времени, используя наименьшее количество ресурсов. Вот примеры алгоритмов регуляризации.

- Гребневая регрессия.
- Регрессия LASSO (Least Absolute Shrinkage and Selection Operator).
- Эластичная сеть.
- Метод наименьших углов (Least-Angle Regression — LARS).

» **Метод опорных векторов (SVM)**. Алгоритмы обучения с учителем, решающие задачи классификации и регрессии, отделяя лишь несколько примеров данных (называемых *опорами* (support), отсюда и название алгоритма) от остальных данных с помощью функции. После разделения этих опор прогноз становится проще. Форма анализа зависит от типа функции (*ядра* (kernel)): линейного, полиномиального или радиального. Вот примеры алгоритмов SVM.

- Линейные методы опорных векторов.
- Метод опорных векторов с радиальной базисной функцией.
- Метод опорных векторов для одного класса (для обучения без учителя).

» **Другие**. Есть также много других алгоритмов на выбор. Этот список содержит основные категории алгоритмов. Некоторые из категорий в этом списке отсутствуют, включая используемые для выбора признака, точности алгоритма, показателей производительности и специализированных подполей машинного обучения. Например, целые категории алгоритмов посвящены темам компьютерного

зрения (Computer Vision — CV) и обработки текстов на естественном языке (Natural Language Processing — NLP). Читая эту книгу, вы обнаружите множество других категорий алгоритмов и можете задаться вопросом, как аналитик может сделать какой либо выбор вообще, не говоря уже о правильном.

В ожидании следующего прорыва

Прорывы требуют терпения, поскольку в основе компьютеров, по сути, лежит математика. Вы можете не видеть ее как таковой при работе с такими высокоуровневыми языками как Python, но все происходящее внутри требует хорошего понимания математики и данных, которыми язык манипулирует. Следовательно, в будущем вы можете ожидать появления новых применений для машинного и глубокого обучения, поскольку ученые продолжают находить все новые способы обработки данных, создания алгоритмов и их использования для определения моделей данных.

ЗАПОМНИ!

К сожалению, работы с тем, что доступно сегодня, недостаточно для создания приложений завтрашнего дня (несмотря на то, в чем пытаются убедить кинофильмы). В будущем можно ожидать, что достижения в области аппаратного обеспечения сделают вполне реальными приложения, которые сегодня неосуществимы. Это не просто вопрос дополнительной вычислительной мощности или большего объема памяти. Завтрашний компьютер будет иметь доступ к таким датчикам, которые сегодня недоступны; процессоры, которые делают то, чего не могут современные процессоры; и новые взгляды на то, как компьютеры думают. Сейчас миру больше всего нужен опыт, а накопление опыта всегда требует времени.

Размышления об истинном использовании машинного обучения

Тот факт, что у вас есть несколько вариантов выбора, когда дело доходит до искусственного интеллекта, означает, что машинное обучение — это не единственная технология, которую вы должны рассматривать для решения любой задачи. Машинное обучение действительно позволяет решать задачи, относящиеся к конкретным категориям. Чтобы узнать, где машинное обучение работает лучше всего, следует начать с рассмотрения того, как обучается алгоритм, а затем применить эти знания к классам задач, которые необходимо

решить. Помните, что машинное обучение связано с обобщением, поэтому оно не очень хорошо работает в следующих случаях.

- В результате должен быть дан точный ответ, такой как расчет полета на Марс.
- Вы можете решить задачу с помощью обобщения, но другие методы проще, например, разработка программного обеспечения для вычисления факториала числа.
- У вас нет хорошего обобщения задачи, поскольку проблема неправильно понята, не существует конкретной связи между входными данными и результатами, или предметная область слишком сложна.

В следующих разделах обсуждается истинное применение машинного обучения с точки зрения того, как оно осуществляется с последующим определением его преимуществ с учетом конкретных предметных областей.

Преимущества машинного обучения

То, как вы можете извлечь пользу из машинного обучения, зависит отчасти от вашей среды и от того, чего вы от нее ожидаете. Например, если вы тратите время на покупку продуктов Amazon, вы можете ожидать, что в какой-то момент машинное обучение даст полезные рекомендации, основанные на прошлых покупках. Эти рекомендации предназначены для продуктов, о которых вы, возможно, иначе и не знали бы. Рекомендовать продукты, которые вы уже используете или в которых не нуждаетесь, не особенно полезны, и именно здесь в игру вступает машинное обучение. По мере того как Amazon собирает больше данных о ваших привычках в покупках, рекомендации должны стать более полезными, хотя даже самый лучший алгоритм машинного обучения никогда не будет правильно угадывать ваши потребности каждый раз.


ЗАПОМНИ!

Конечно, машинное обучение способно принести вам пользу во многих других отношениях. Разработчик может использовать машинное обучение для придания приложению возможности NLP. Исследователь может использовать его для поиска следующего лекарства от рака. Вы уже можете использовать его для фильтрации спама в своей электронной почте или полагаться на него как на часть голосового интерфейса, когда садитесь в машину. С учетом этого, следующие преимущества, вероятно, в большей степени соответствуют бизнес-перспективам эффективного использования машинного обучения, но следует иметь в виду, что существует и множество других способов.

- **Упростить маркетинг товаров**. Одной из проблем любой торговой организации является определение, что и когда продавать, исходя из предпочтений клиента. Рекламные кампании стоят дорого, поэтому даже один провал — это обычно проблема. Кроме того, организация может найти странную информацию: покупателям могут понравиться товары красного цвета, но не зеленого. Узнать, чего хочет клиент, невероятно сложно, если вы не можете проанализировать огромные объемы данных о покупках. Именно это хорошо подходит для машинного обучения.
- **Точно прогнозировать будущие продажи**. Бизнес может показаться азартной игрой, поскольку вы не можете быть уверены, окупятся ли ваши ставки. Решение машинного обучения способно отслеживать продажи каждую минуту, а также отслеживать тенденции, прежде чем они станут очевидными. Возможность такого рода отслеживания означает, что вы можете точнее настроить каналы продаж и обеспечить оптимальные результаты, гарантируя, что в магазинах достаточно товаров, нужных для продажи. Это не совсем похоже на хрустальный шар, но все же близко.
- **Прогноз простоя медицинского или другого работника**. Как ни странно, в некоторых организациях возникают проблемы, когда их сотрудники выбирают наихудшее время для отсутствия на работе. В некоторых случаях эти пропуски кажутся непредсказуемыми, например, медицинские нужды, в то время как другие вы могли бы предсказать, например, внезапную потребность в личном времени. Отслеживая различные тенденции из легко доступных источников данных, вы можете отслеживать пропуски, как по медицинским, так и личным причинам для вашей отрасли в целом, местоположению в целом и вашей организации в частности. Это позволит убедиться, что у вас достаточно людей для выполнения работ в любой момент времени.
- **Снижение количества ошибок при вводе данных**. Некоторых видов ошибок при вводе данных относительно легко избежать, если правильно использовать возможности форм или включить в свое приложение проверку правописания. Кроме того, добавление определенных типов распознавания шаблонов может помочь уменьшить количество ошибок в заглавных буквах или неправильных номерах телефонов. Машинное обучение может поднять снижение количества ошибок на другой уровень, правильно идентифицируя сложные шаблоны, которые пропустят другие методы. Например, заказ клиента может требовать одну часть A и две части B для создания целого блока. Соответствие шаблону при продажах такого типа может быть неуловимым, но машинное обучение может сделать это

возможным, удаляя такие ошибки, которые особенно трудно найти и устранить.

- **Улучшение финансовых правил и точности модели**. Поддержание финансов в порядке может оказаться затруднительным в организации любого размера. Машинное обучение позволяет с большей точностью решать такие задачи, как управление портфелем, алгоритмическая торговля, оценка кредитоспособности и обнаружение мошенничества. Вы не можете исключить участие человека в таких ситуациях, но человек и машина, работающие вместе, могут стать невероятно эффективной комбинацией, которая позволит выявить множество ошибок.
- **Прогнозирование потребности в обслуживании**. Любая система, которая состоит из чего-то физического, вероятно, требует обслуживания различного рода. Например, на основании производительности в прошлом и мониторинга окружения машинное обучение может помочь предсказать, когда системе потребуется чистка. Вы также можете осуществлять такие действия, как планирование замены или ремонта определенного оборудования на основе прошлых ремонтов и статистики оборудования. Решение машинного обучения может даже помочь определить, является ли замена или ремонт лучшим вариантом.
- **Улучшение взаимодействия с клиентом**. Клиенты любят чувствовать себя особенными; практически все. Однако попытка создать индивидуальный план для каждого клиента вручную окажется невозможной. Через источники в Интернете вы можете найти множество информации о клиентах, включая все, от недавних покупок до постоянных покупательских привычек. Объединив все эти данные с хорошим решением машинного обучения и обслуживающим персоналом с проницательным взглядом, вы можете создать специальное решение для каждого клиента, хотя необходимое для этого время минимально.

Границы машинного обучения

Границы технологии зачастую трудно определить точно, поскольку эти ограничения нередко являются результатом недостатка воображения со стороны ее создателя или потребителя. Тем не менее, машинное обучение имеет определенные ограничения, которые необходимо учитывать, прежде чем использовать эту технологию для выполнения какой-либо задачи. Следующий список не является полным. На самом деле, вы можете даже не согласиться с ним, но он является хорошей отправной точкой.

- **Требуются огромные объемы учебных данных**. В отличие от программных решений прошлого, решение машинного обучения опирается на огромные объемы учебных данных. По мере увеличения сложности задачи увеличивается количество точек данных, необходимых для моделирования конкретной задачи, что требует еще больше данных. Хотя люди создают все большие объемы данных в конкретных предметных областях, и необходимая для обработки этих данных вычислительная мощь также ежедневно увеличивается, некоторым предметным областям просто не хватает данных или достаточной вычислительной мощности, чтобы сделать машинное обучение эффективным.
- **Маркировка данных утомительна и подвержена ошибкам**. При использовании методики обучения с учителем (см. ранее в этой главе), кто-то должен пометить данные, чтобы обеспечить выходные значения. Процесс маркировки огромных объемов данных утомителен и отнимает много времени, что иногда затрудняет машинное обучение. Проблема в том, что человек может просмотреть любое количество неких примеров, вроде знака STOP, и узнать, что это все знаки STOP, но для компьютера каждый такой знак должен быть помечен индивидуально.

ЗАПОМНИ!

- **Машины не могут себя объяснить**. По мере роста гибкости и способностей решения машинного обучения, количество скрытых функций также растет. Фактически, при работе с решениями глубокого обучения, вы обнаружите, что решение содержит один или, как правило, несколько скрытых слоев, которые создает решение, но люди не тратят время на их изучение. Следовательно, как машинное обучение (в некоторой степени), так и глубокое обучение (в большей степени) встречаются с задачами, в которых ценится прозрачность, и встречаются некоторые противоречия законам, таким как Общий регламент по защите данных (GDPR) (`https://eugdpr.org/`). Поскольку процесс становится непрозрачным, человек вынужден теперь анализировать процесс, который должен был быть автоматическим. Потенциальным решением этой проблемы могут быть новые стратегии, такие как локально интерпретируемые моделе-агностические объяснения (Local Interpretable Model-Agnostic Explanations — LIME) (см. подробнее на `https://homes.cs.washington.edu/~marcotcr/blog/lime/`).
- **Недостоверность делает результаты менее полезными**. Алгоритм не может определить, когда в данных содержатся разные недостоверности (Подробно эта проблема рассматривается в книге *Искусственный интеллект для чайников* Джона Пола Мюллера и Луки Массарона). Следовательно, он считает все данные объективными и

абсолютно правдивыми. В результате, любой анализ, выполненный алгоритмом, обученным с использованием этих данных, является подозрительным. Проблема становится еще больше, когда недостоверен сам алгоритм. Вы можете найти бесчисленные примеры сетевых алгоритмов, неправильно определяющих такие обычные объекты как знаки STOP, из-за комбинации данных, содержащих неверные и необъективные алгоритмы.

- **Решения машинного обучения не могут сотрудничать**. Одним из важнейших преимуществ людей является способность сотрудничать друг с другом. Потенциал знаний растет в геометрической прогрессии, поскольку каждая сторона потенциального решения предоставляет свою часть знаний, чтобы создать общее, намного большее, чем сумма всех его частей. Единственное решение машинного обучения остается единым решением, поскольку оно не может обобщать знания и тем самым вносить вклад в комплексное решение с участием нескольких сотрудничающих сторон.

Глава 3

Получение и использование языка Python

В ЭТОЙ ГЛАВЕ...

- Получение экземпляра Python
- Взаимодействие с Jupyter Notebook
- Создание простого кода Python
- Работа в облаке

Глубокое обучение требует использования кода, и у вас есть множество вариантов при выборе языка. Тем не менее, эта книга опирается на язык Python, поскольку он работает на множестве разных инфраструктур и пользуется значительной поддержкой в сообществе разработчиков. Фактически, согласно индексу Tiobe (`https://www.tiobe.com/tiobe-index/`), доступному на момент написания, Python является четвертым по счету языком в мире и лучше всего подходит для глубокого обучения, согласно нескольким источникам (см. `https://www.analyticsindiamag.com/top-10-programming-languages-data-scientists-learn-2018/`).

Прежде чем вы сможете сделать достаточно много в языке Python или использовать его для решения глубоких проблем, вам нужна работоспособная версия. Вам также необходим доступ к наборам данных и коду, используемому в этой книге. Загрузка примеров кода и его установка в вашей системе — лучший способ получить опыт из изучения этой книги. Данная глава поможет

настроить вашу систему так, чтобы вы могли легко следовать примерам в остальной части книги. Здесь также рассматриваются потенциальные альтернативы, такие как Google Colaboratory (`https://colab.research.google.com/notebooks/welcome.ipynb`), иногда называемый также просто Colab, на случай, если вы захотите работать на альтернативном устройстве, например на планшете.


ЗАПОМНИ!

Использование загружаемого исходного кода ничуть не мешает вам набирать примеры самостоятельно, исследовать их с помощью отладчика, дополнять их или работать с кодом всевозможными способами. Загружаемый исходный код облегчит вам начало изучения глубокого обучения и получения опыта работы с языком Python. Увидев, как работает правильно набранный и отлаженный код, вы можете попробовать создать примеры самостоятельно. Если вы допустили ошибку, вы можете сравнить то, что вы ввели, с загруженным кодом и точно определить, где допущена ошибка. Вы можете найти загружаемый исходный код примеров из этой главы в файлах `DL4D_03_Sample.ipynb`, `DL4D_03_Dataset_Load.ipynb`, `DL4D_03_Indentation.ipynb` и `DL4D_03_Comments.ipynb`. (Где можно скачать исходный код этой книги, сказано во введении.)

Работа с языком Python в этой книге

Среда Python постоянно меняется. По мере того, как сообщество Python продолжает совершенствовать язык Python, он претерпевает *серьезные изменения*, обуславливающие новое поведение при одновременном снижении обратной совместимости. Эти изменения могут и не быть значительными, но они отвлекают внимание, снижая ваши возможности по изучению программных методов глубокого обучения. Вполне очевидно, что вы хотите изучать глубокое обучение с как можно меньшим количеством отвлекающих факторов, поэтому наличие правильной среды очень важно. Вот что вам нужно для использования языка Python с этой книгой.

- Jupyter Notebook версии 5.5.0.
- Среда Anaconda 3 версии 5.2.0.
- Python версии 3.6.7.


СОВЕТ

Если у вас нет этих версий, вы можете обнаружить, что примеры работают не так, как задумано. Копии экрана, скорее всего, будут отличаться, и процедуры могут работать не так, как запланировано.

ВНИМАНИЕ!

Когда вы читаете книгу, вам нужно установить различные пакеты Python, чтобы код работал. Как и среда Python, которую вы настраиваете в данной главе, эти пакеты имеют конкретные номера версий. Если вы используете другую версию пакета, примеры могут вообще не выполняться. Кроме того, вас могут расстроить сообщения об ошибках, которые не имеют ничего общего с кодом книги, а являются результатом использования неправильной версии. Обязательно соблюдайте осторожность при установке Anaconda, Jupyter Notebook, Python и всех пакетов, необходимых для того, чтобы сделать процесс глубокого обучения максимально плавным.

Получение собственного экземпляра Anaconda

Прежде чем двигаться вперед, вам необходимо получить и установить экземпляр Anaconda. Да, вы можете получить и установить Jupyter Notebook отдельно, но тогда у вас не будет других приложений, которые поставляются вместе с Anaconda, таких как Anaconda Prompt, упоминаемых в различных частях книги. Лучше всего установить Anaconda, следуя инструкциям, приведенным в следующих разделах согласно вашей конкретной платформе (Linux, MacOS или Windows).

Получение пакета Anaconda от Continuum Analytics

Базовый пакет Anaconda можно бесплатно загрузить с сайта `https://repo.anaconda.com/archive/`, чтобы получить версию 5.2.0, используемую в этой книге. Чтобы получить доступ к бесплатному продукту, просто перейдите по одной из ссылок Python 3.6 Version. Искомое имя файла начинается с `Anaconda3-5.2.0-`, за которым следует платформа и версия (32- или 64-разрядная), например `Anaconda3-5.2.0-Windows-x86_64.exe` для 64-разрядной версии Windows. Пакет Anaconda поддерживает следующие платформы.

- 32- и 64-разрядную версию Windows (программа установки может предложить вам только 64- или 32-разрядную версию, в зависимости от обнаруженной им версии Windows).
- 32- и 64-разрядную версии Linux.
- 64-разрядную версию Mac OS X.

Бесплатный продукт — это все, что вам нужно для этой книги. Но при просмотре сайта вы увидите, что доступно много других дополнительных

продуктов. Эти продукты могут помочь вам создавать надежные приложения. Например, когда вы добавляете в комплект Accelerate, вы получаете возможность выполнять операции на нескольких ядрах и GPU. Использование этих дополнительных продуктов выходит за рамки данной книги, но сайт Anaconda предоставляет подробные сведения об их применении.

Установка Anaconda на Linux

Для установки Anaconda на Linux используется командная строка, поскольку графический интерфейс отсутствует. Прежде чем выполнить установку, вы должны загрузить экземпляр программного обеспечения для Linux с сайта Continuum Analytics. Необходимую информацию для загрузки см. выше, в разделе "Получение пакета Anaconda от Continuum Analytics". Следующая процедура должна нормально работать на любой системе Linux, независимо от того, используете ли вы 32- или 64-разрядную версию Anaconda.

1. **Откройте экземпляр терминала.**

   Появится окно Terminal.

2. **Измените каталоги для загруженного экземпляра Anaconda в вашей системе.**

   Эти файлы имеют разные имена, но обычно они выглядят как `Anaconda3-5.2.0-Linux-x86.sh` для 32-разрядных систем и `Anaconda3-5.2.0-Linuxx86_64.sh` для 64-разрядных. Номер версии встроен в имя файла. В данном случае имя файла относится к версии 5.2.0, используемой в этой книге. Если вы используете какую-то другую версию, у вас могут возникнуть проблемы с исходным кодом и вам придется вносить изменения при работе с ним.

3. **Введите `bash Anaconda3-5.2.0-Linux-x86` (для 32-разрядной версии) или `Anaconda3-5.2.0-Linux-x86_64.sh` (для 64-разрядной версии) и нажмите клавишу <Enter>.**

   Запустится мастер установки, который попросит вас принять условия лицензии для использования Anaconda.

4. **Прочитайте лицензионное соглашение и примите условия, используя метод, необходимый для вашей версии Linux.**

   Мастер попросит вас указать место установки Anaconda. В книге предполагается, что вы используете стандартное местоположение `~/anaconda`. Если вы выберете другое место, впоследствии вам, возможно, придется изменить некоторые процедуры, чтобы работать с вашими настройками.

5. **Укажите место установки (при необходимости) и нажмите клавишу <Enter> (или щелкните на кнопке Next (Далее)).**

Вы видите, что начинается процесс извлечения приложения. По окончании извлечения вы увидите сообщение о завершении.

6. **Добавьте путь установки в инструкцию `PATH`, используя метод, необходимый для вашей версии Linux.**

   Вы готовы начать использовать Anaconda.

Установка Anaconda на MacOS

Установка на Mac OS X возможна только в одной форме: 64-разрядной. Прежде чем выполнить установку, вы должны загрузить экземпляр программного обеспечения для Mac с сайта Continuum Analytics. Необходимую информацию для загрузки см. в разделе "Получение пакета Anaconda от Continuum Analytics". Следующие шаги помогут установить Anaconda 64-разряда на системе Mac.

1. **Найдите загруженный экземпляр Anaconda в вашей системе.**

   Эти файлы имеют разные имена, но обычно они выглядят как `Anaconda3-5.2.0-MacOSX-x86_64.pkg`. Номер версии встроен в имя файла. В данном случае имя файла относится к версии 5.2.0, используемой в этой книге. Если вы используете какую-то другую версию, у вас могут возникнуть проблемы с исходным кодом и вам придется вносить изменения при работе с ним.

2. **Дважды щелкните на установочном файле.**

   Откроется начальное диалоговое окно.

3. **Щелкните на кнопке Continue (Продолжить).**

   Мастер спросит, хотите ли вы просмотреть материалы Read Me. Поскольку вы можете прочитать эти материалы позже, пропустите эту информацию.

4. **Щелкните на кнопке Continue (Продолжить).**

   Мастер отобразит лицензионное соглашение. Обязательно ознакомьтесь с ним, чтобы знать условия использования.

5. **Щелкните на кнопке I Agree (Я согласен), если вы согласны с лицензионным соглашением.**

   Мастер попросит вас указать место для установки. Это определяет, предназначена ли установка для отдельного пользователя или группы.

ВНИМАНИЕ!

Может появиться сообщение об ошибке, в котором говорится, что вы не можете установить Anaconda в систему. Причиной сообщения является ошибка в установщике, не имеющая ничего общего с вашей системой. Чтобы избавиться от сообщения об ошибке, выберите вариант Install Only for Me (Установить только для меня). Вы не можете установить Anaconda для группы пользователей в системе Mac.

6. **Щелкните на кнопке Continue (Продолжить).**

   Установщик отобразит диалоговое окно, содержащее возможности для изменения типа установки. Щелкните на Change Install Location (Изменить место установки), если хотите изменить место установки Anaconda в вашей системе (в книге предполагается, что вы используете стандартный путь `~/anaconda`). Щелкните на кнопке Customize (Настроить), если хотите изменить работу установщика. Например, вы можете не добавлять Anaconda в свою инструкцию `PATH`. Однако в книге предполагается, что вы выбрали стандартные параметры установки, и нет веских причин менять их, если у вас не установлен другой экземпляр Python 2.7 в другом месте.

7. **Щелкните на кнопке Install (Установить).**

   Вы видите начало установки. Индикатор выполнения показывает, как проходит этот процесс. После завершения установки откроется диалоговое окно завершения.

8. **Щелкните на кнопке Continue (Продолжить).**

   Вы готовы начать использовать Anaconda.

Установка Anaconda на Windows

Для Windows Anaconda поставляется с графическим приложением установки, поэтому установка подразумевает использование мастера, как и при любой другой установке. Конечно, вам нужен экземпляр установочного файла, прежде чем вы начнете (см. выше раздел "Получение пакета Anaconda от Continuum Analytics"). Следующая процедура должна нормально сработать в любой системе Windows, независимо от того, используете ли вы 32- или 64-разрядную версию Anaconda.

1. **Найдите загруженный экземпляр Anaconda в вашей системе.**

   Эти файлы имеют разные имена, но обычно они выглядят как `Anaconda3-5.2.0-Windows-x86.exe` для 32-разрядных систем и `Anaconda3-5.2.0-Windowsx86_64.exe` для 64-разрядных. Номер версии встроен в имя файла. В данном случае имя файла относится к версии 5.2.0, используемой в этой книге. Если вы используете какую-то другую версию, у вас могут возникнуть проблемы с исходным кодом и вам придется вносить изменения при работе с ним.

2. **Дважды щелкните на установочном файле.**

   Может появиться диалоговое окно Open File – Security Warning, спрашивающее, хотите ли вы запустить этот файл. Щелкните Run (Выполнить), если появится это диалоговое окно. Вы увидите диалоговое окно установки Anaconda 5.2.0, аналогичное показанному на рис. 3.1. Конкретное диалоговое окно, которое вы увидите, зависит от скачанной версии программы установки

Anaconda. Если у вас 64-разрядная операционная система, всегда лучше использовать 64-разрядную версию Anaconda, чтобы добиться максимальной производительности. В этом первом диалоговом окне сообщается версия продукта (64-разрядная).

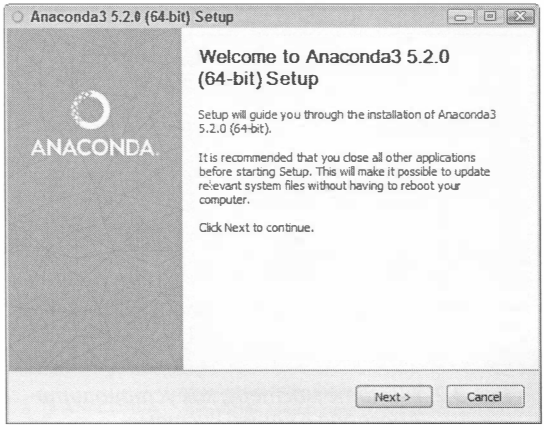


Рис. 3.1. Процесс установки начинается с сообщения о наличии 64-разрядной версии

3. **Щелкните на кнопке Next (Далее)**.

   Мастер отобразит лицензионное соглашение. Обязательно ознакомьтесь с ним, чтобы знать условия использования.

4. **Щелкните на кнопке I Agree (Я согласен), если вы согласны с лицензионным соглашением**.

   Вас спросят, какой тип установки выполнять (рис. 3.2). Как правило, вы хотите установить продукт только для себя. Исключение составляют случаи, когда вашу систему используют несколько человек и им всем нужен доступ к Anaconda.

5. **Выберите один из типов установки и щелкните на кнопке Next (Далее)**.

   Мастер спросит, где установить Anaconda на диске, как показано на рис. 3.3. В книге предполагается, что вы используете стандартное расположение. Если вы выберете другое место, впоследствии вам, возможно, придется изменить некоторые процедуры.

6. **Выберите место установки (при необходимости) и щелкните на кнопке Next (Далее)**.

   Вы увидите дополнительные параметры установки, показанные на рис. 3.4. Эти параметры выбраны стандартно, и в большинстве случаев нет веских причин для их изменения. Возможно, вам придется изменить их, если Anaconda не предоставит настройки Python 3.6 стандартно. Однако в книге

предполагается, что вы настроили Anaconda с использованием стандартных параметров.

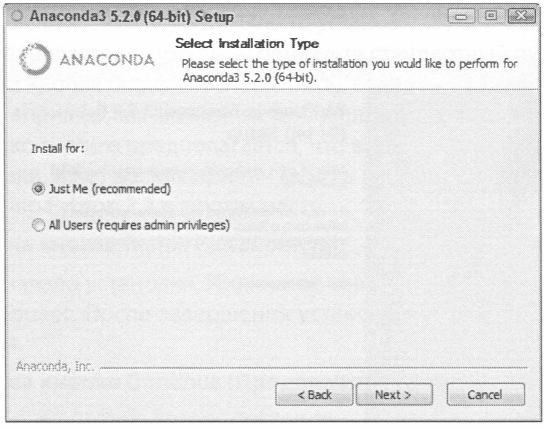


Рис. 3.2. Укажите мастеру, как установить Anaconda в вашей системе


СОВЕТ

Флажок Add Anaconda to My PATH Environment Variable (Добавить Anaconda в мою переменную среды `PATH`) стандартно сброшен, и вы должны оставить его без изменения. Добавление в переменную среды `PATH` позволяет находить файлы Anaconda с помощью стандартной командной строки, но если у вас установлено несколько версий Anaconda, то доступна будет только первая установленная версия. Вместо этого лучше использовать Anaconda Prompt, чтобы получать доступ к ожидаемой версии.

7. **Измените дополнительные параметры установки (при необходимости) и щелкните на кнопке Install (Установить).**

   Откроется диалоговое окно установки с индикатором выполнения. Процесс установки может занять несколько минут, поэтому возьмите себе чашку кофе и немного почитайте комиксы. Когда процесс установки завершится, вы увидите кнопку Next (Далее).

8. **Щелкните на кнопке Next (Далее).**

   Мастер сообщит, что установка завершена.

9. **Щелкните на кнопке Next (Далее).**

   Anaconda предложит интегрировать поддержку кода Visual Studio. Для данной книги эта поддержка не нужна, и ее добавление может потенциально изменить работу инструментов Anaconda. Если вам абсолютно не нужна поддержка Visual Studio, имеет смысл поддержать чистоту среды Anaconda.

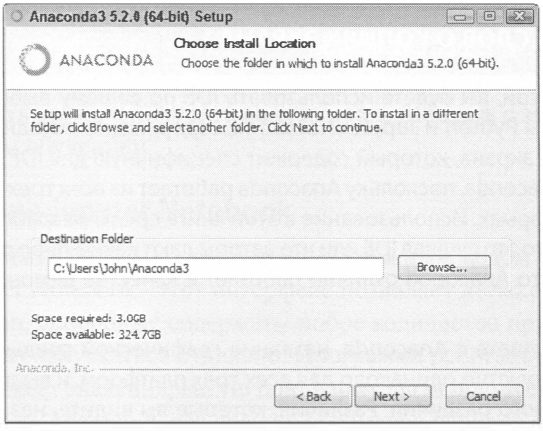


Рис. 3.3. Укажите место установки

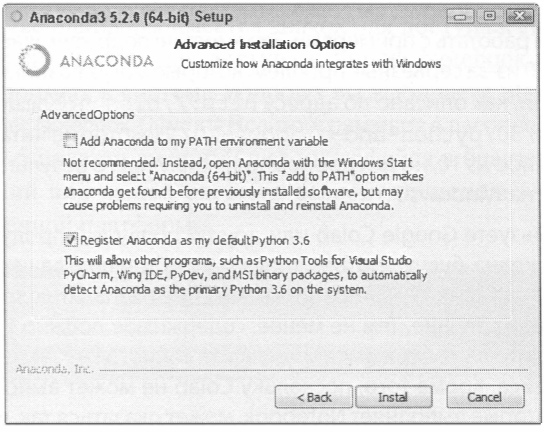


Рис. 3.4. Задайте дополнительные параметры установки

10. **Щелкните на кнопке Skip (Пропустить).**

    Вы увидите экран завершения. Этот экран содержит параметры, позволяющие узнать больше об Anaconda Cloud и получить информацию о запуске вашего первого проекта Anaconda. Установка этих флажков (или их сброс) зависит от того, что вы хотите сделать далее; флажки не влияют на настройки Anaconda.

11. **Установите все необходимые параметры. Щелкните на кнопке Finish (Готово).**

    Вы готовы начать использовать Anaconda.

НЕСКОЛЬКО СЛОВ О КОПИЯХ ЭКРАНА

Работая с книгой, вы будете использовать IDE по вашему выбору, чтобы открывать файлы Python и Jupyter Notebook, содержащие исходный код книги. Каждая копия экрана, который содержит специфичную для IDE информацию, основан на Anaconda, поскольку Anaconda работает на всех трех упоминаемых в книге платформах. Использование в этой книге среды разработки Anaconda не означает, что это лучшая IDE или что авторы дают какие-либо рекомендации для нее, просто Anaconda отлично работает в качестве демонстрационного продукта.

Когда вы работаете с Anaconda, название графической среды (GUI), Jupyter Notebook, абсолютно одинаково для всех трех платформ, и вы даже не увидите существенного различия. Различия, которые вы видите, незначительны, и вы должны игнорировать их во время работы с книгой. С учетом этого, книга существенно зависит от сценариев Windows 7. Работая на Linux, Mac OS X или другой платформе Windows, вы должны ожидать некоторых отличий, но они не помешают вам работать с примерами. Эта книга не подразумевает использования Windows 10 из-за серьезных проблем, которые могут возникнуть при работе языка Python, как описано по адресу `http://blog.johnmuellerbooks.com/2015/10/30/python-and-windows-10/`. Некоторые читатели успешно используют Windows 10, но для достижения наилучшего результата продолжают полагаться на Windows 7.

Если вы используете Google Colab или другой облачный продукт, отображаемые копии экрана будут соответствовать комбинации вашего браузера и облачной среды. Копии экранов в книге могут не совпадать с тем, что вы фактически видите на экране. Тем не менее, содержимое должно быть таким же, поэтому смотрите на содержимое и не ищите точного соответствия графического интерфейса. Кроме того, поскольку Colab не может выполнять некоторые задачи, которые выполняет Notebook, может оказаться так, что некоторое содержимое отсутствует или вместо него появляется сообщение об ошибке.

Загрузка наборов данных и примеров кода

Эта книга об использовании языка Python для решения задач глубокого обучения. Конечно, вы можете потратить все свое время на создание примера кода с нуля, на его отладку и только потом узнать, как он связан с глубоким обучением, или можете пойти простым путем и скачать уже написанный код, чтобы сразу приступить к работе. Аналогично создание наборов данных, достаточно больших для задач глубокого обучения, займет довольно много времени. К счастью, вы можете легко получить доступ к стандартизированным,

предварительно созданным наборам данных, используя функции, предоставляемые в некоторых библиотеках науки о данных. В следующих разделах вы узнаете, как загрузить и использовать примеры кода, а также наборы данных, чтобы сэкономить время и получить возможность работать с задачами, связанными с глубоким обучением.

Использование Jupyter Notebook

Чтобы упростить работу с относительно сложным кодом этой книги, используйте Jupyter Notebook. Этот интерфейс позволяет легко создавать файлы блокнотов Python, способные содержать любое количество примеров, каждый из которых может запускаться отдельно. Программа работает в вашем браузере, поэтому не имеет значения, какую платформу вы используете для разработки. Пока в ней есть браузер, у вас должно быть все в порядке.

Начинаем работу с Jupyter Notebook

На большинстве платформ имеется значок для доступа к Jupyter Notebook. Все, что вам нужно сделать для доступа к Jupyter Notebook, — это открыть этот значок. Например, в системе Windows вы выбираете Start⇨All Programs⇨Anaconda3⇨Jupyter Notebook (Пуск⇨Все программы⇨Anaconda3⇨Jupyter Notebook). На рис. 3.5 показан внешний вид интерфейса в браузере Firefox. Конкретный внешний вид вашей системы зависит от используемого браузера и типа установленной платформы.

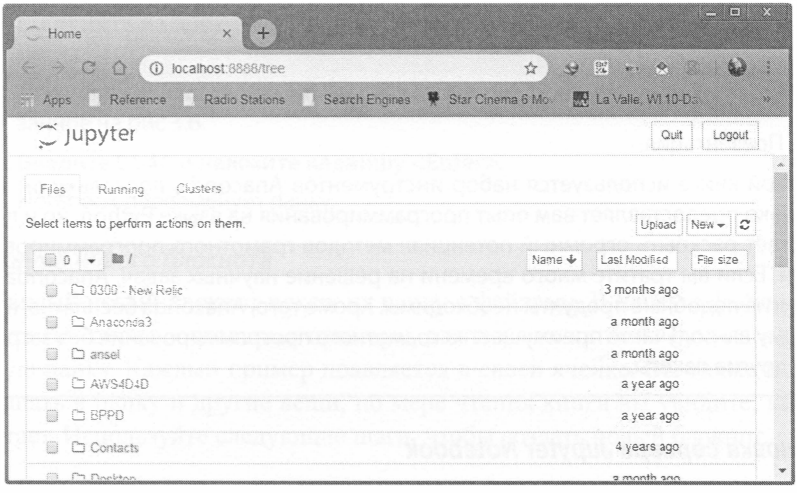


Рис. 3.5. Jupyter Notebook предоставляет простой способ создания примеров науки о данных

Если ваша платформа предлагает простой доступ с помощью значка, используйте следующие этапы для доступа к Jupyter Notebook.

1. **Откройте Anaconda Prompt, Command Prompt или окно Window в вашей системе.**

   Откроется окно, в котором можно вводить команды.

2. **Перейдите в каталог `\Anaconda3\Scripts` на вашем компьютере.**

   Для этого большинство систем позволяют использовать команду `CD`.

3. **Введите `..\python Jupyter-script.py notebook` и нажмите клавишу <Enter>.**

   В браузере откроется страница Jupyter Notebook.

РАЗЛИЧИЕ МЕЖДУ БЛОКНОТОМ И IDE

Блокнот (notebook) отличается от текстового редактора тем, что он фокусируется на разработанной Дональдом Кнутом (Donald Knuth) методике *грамотного программирования* (literate programming), используемой для создания своего рода представления кода, заметок, математических уравнений и графики. Короче говоря, вы попадаете в блокнот ученого, содержащий все необходимое для полного понимания кода. Вы встречаете методы грамотного программирования и в дорогих пакетах, таких как Mathematica и MATLAB. Разработка в Notebook отличается следующим.

- Демонстрация.
- Сотрудничество.
- Исследование.
- Учебные цели.
- Презентация.

В этой книге используется набор инструментов Anaconda, поскольку он не только предоставляет вам опыт программирования на языке Python, но и помогает раскрыть огромный потенциал методов грамотного программирования. Если вы тратите много времени на решение научных задач, Anaconda и другие подобные продукты необходимы. Кроме того, Anaconda бесплатна, поэтому вы получаете преимущества грамотного программирования без затрат на другие пакеты.

Остановка сервера Jupyter Notebook

Независимо от того, как вы запускаете Jupyter Notebook (или просто Notebook, как он упоминается в оставшейся части книги), система обычно открывает командную строку или окно терминала для его размещения. Это окно

содержит сервер, позволяющий приложению работать. После закрытия окна браузера и завершения сеанса перейдите в окно сервера и нажмите комбинацию клавиш <Ctrl + C> или <Ctrl + Break>, чтобы остановить сервер.

Определение хранилища кода

Код, который вы создаете и используете в этой книге, будет находиться в хранилище на вашем жестком диске. Считайте это *хранилище* неким шкафом для хранения документов, куда вы помещаете свой код. Jupyter Notebook открывает ящик, вынимает папку и показывает код вам. Вы можете изменять его, запускать в папке отдельные примеры, добавлять новые примеры и просто взаимодействовать со своим кодом естественным образом. В следующих разделах вы познакомитесь с Notebook лучше и узнаете, как работает концепция хранилища.

Создание новой папки

Вы используете папки для хранения файлов кода конкретного проекта. Проект этой книги называется `DL4D` (что означает *Deep Learning For Dummies*). Следующие шаги помогут вам создать новую папку для этой книги.

1. **Выберите пункт меню New⇨Folder (Новая⇨Папка).**

   Блокнот создает новую папку. Имя папки может отличаться, но для пользователей Windows она указана как Untitled Folder (Папка без названия). Возможно, вам придется прокрутить вниз список доступных папок, чтобы найти нужную.

2. **Установите флажок в поле рядом с Untitled Folder (Папка без названия).**

3. **Щелкните на элементе Rename (Переименовать) в верхней части страницы.**

   Откроется диалоговое окно Rename Directory (Переименовать каталог), показанное на рис 3.6.

4. **Введите `DL4D` и нажмите клавишу <Enter>.**

   Notebook переименует папку.

Создание нового блокнота

Каждый новый блокнот похож на папку с файлами. В папку с файлами вы можете поместить отдельные примеры, так же, как и листы бумаги в физическую папку. Каждый пример появляется в своей ячейке. Вы также можете помещать в папку и другие вещи, по мере чтения книги вы увидите, как это работает. Используйте следующие шаги, чтобы создать новый блокнот.

1. **Щелкните на элементе `DL4D` на домашней странице.**

   Вы увидите содержимое папки проекта этой книги, которая будет пустой, если вы выполняете это упражнение впервые.

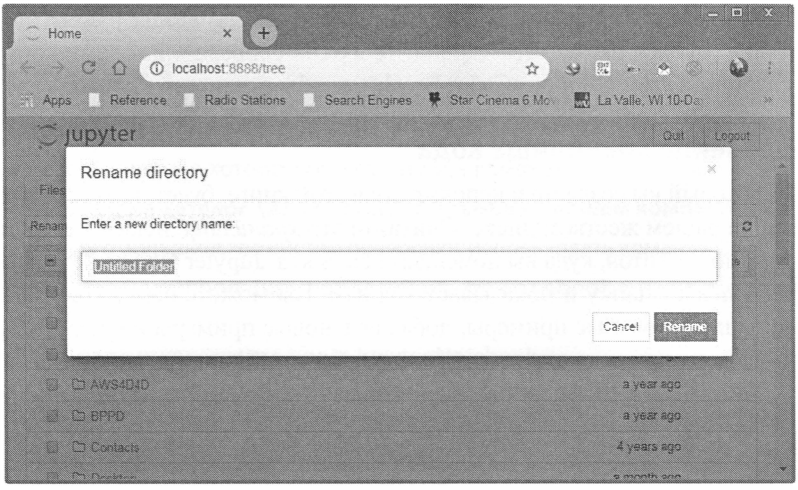


Рис. 3.6. Создание папки для хранения кода книги

2. **Выберите пункт меню New⇨Python 3.**

В браузере откроется новая вкладка с новым блокнотом, как показано на рис. 3.7. Обратите внимание, что блокнот содержит ячейку, и эта ячейка выделена, чтобы вы могли вводить в нее код. Название блокнота сейчас Untitled (Без названия), поэтому его нужно изменить.

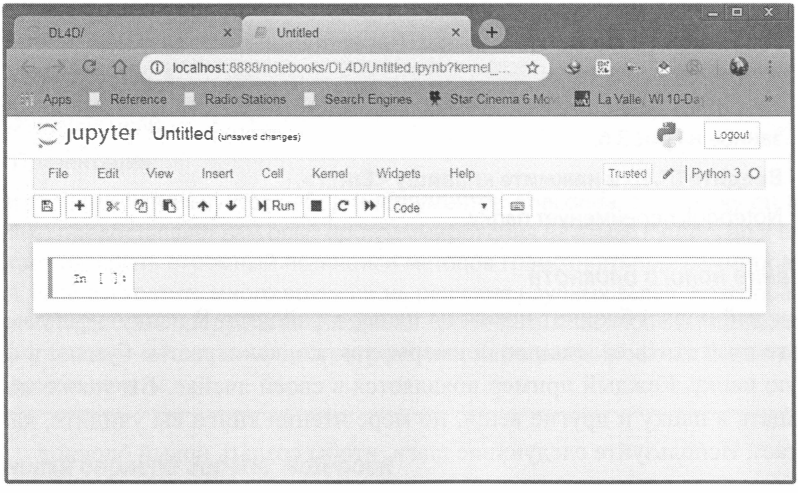


Рис. 3.7. Блокнот содержит ячейки, используемые для хранения кода

3. **Щелкните на странице Untitled (Без названия).**

   Notebook спросит, хотите ли вы использовать новое имя (рис. 3.8).

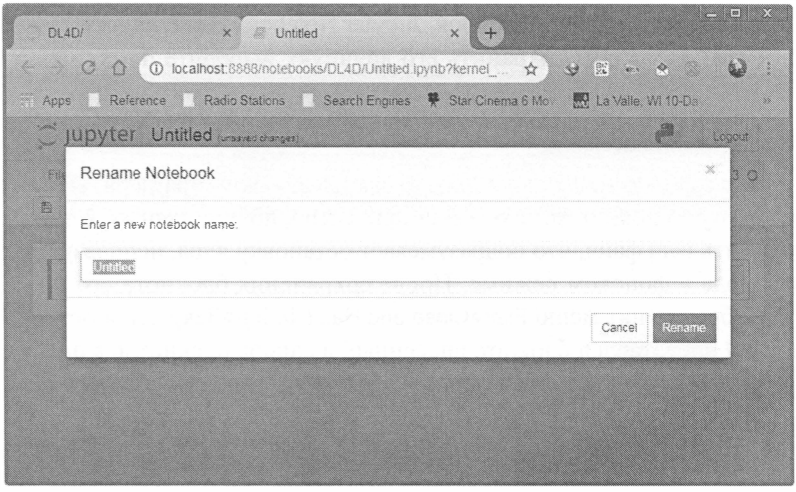


Рис. 3.8. Укажите новое имя блокнота

4. **Введите `DL4D_03_Sample` и нажмите клавишу <Enter>.**

   Новое имя говорит вам, что это файл для примера главы 3 этой книги. Использование этого соглашения об именах позволит вам легко отличать эти файлы от других файлов в вашем хранилище.

Экспорт блокнота

Создавать блокноты и хранить их при себе не очень весело. В какой-то момент вы захотите поделиться блокнотами с другими людьми. Для этого вы должны экспортировать свой блокнот из хранилища в файл. Затем можете отправить файл кому-то другому, кто импортирует его в свое хранилище.

В предыдущем разделе было показано, как создать блокнот `DL4D_03_Sample`. Вы можете открыть этот блокнот, щелкнув на его пункте в списке хранилища. Файл откроется, чтобы вы снова увидели свой код. Чтобы экспортировать этот код, выберите пункт меню File⇨Download As⇨Notebook (.ipynb) (Файл⇨Загрузить как⇨Блокнот (.ipynb)). То, что вы увидите далее, зависит от вашего браузера, но обычно это какое-то диалоговое окно для сохранения блокнота в виде файла. Для сохранения файла Notebook используйте тот же способ, что и для любого другого файла, сохраняемого с помощью браузера.

Сохранение блокнота

Затем вы захотите сохранить свой блокнот, чтобы впоследствии просмотреть его или произвести впечатление на своих друзей, предварительно убедившись в отсутствии в нем ошибок. Jupyter Notebook периодически сохраняет блокнот сам. Но чтобы сохранить его вручную, выберите пункт меню File⇨Save and Checkpoint (Файл⇨Сохранить и установить контрольную точку).

Закрытие блокнота

Вы определенно не должны просто закрывать окно браузера, когда завершаете работу с блокнотом. Это может привести к потере данных. Сначала следует закрыть ваш файл, что подразумевает остановку ядра, используемого для запуска кода в фоновом режиме. После сохранения блокнота его можно закрыть, выбрав пункт меню File⇨Close and Halt (Файл⇨Закрыть и остановить). На рис. 3.9 представлен блокнот, внесенный в список блокнотов в папке вашего проекта.

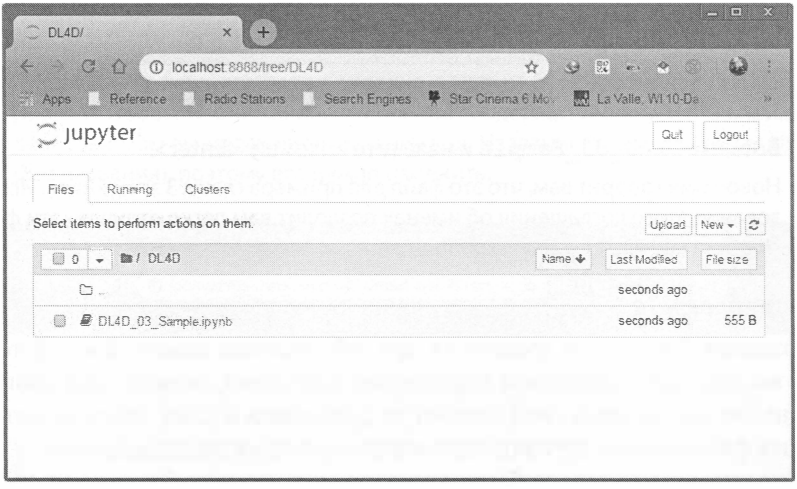


Рис. 3.9. Сохраненные блокноты появятся в списке папки проекта

Удаление блокнота

Иногда блокноты устаревают, или вам просто больше не нужно с ними работать. Чтобы не засорять хранилище ненужными файлами, вы можете удалить ненужные блокноты из списка. Используйте следующие шаги, чтобы удалить файл.

1. **Установите флажок рядом с элементом `DL4D_03_Sample.ipynb`.**
2. **Щелкните на значке Delete (Удалить) (корзина).**

   Появится предупреждающее сообщение об удалении блокнота, подобное показанному на рис. 3.10.

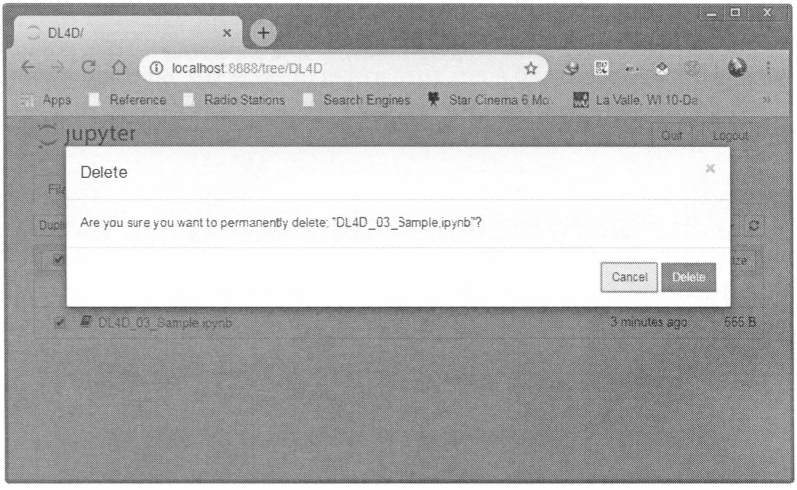


Рис. 3.10. Notebook предупреждает перед удалением любых файлов из хранилища

3. **Щелкните на кнопке Delete (Удалить).**

   Notebook удаляет файл блокнота из списка.

Импорт блокнота

Чтобы использовать исходный код этой книги, следует импортировать загруженные файлы в свое хранилище. Исходный код поставляется в файле архива, который вы распаковываете в папку на жестком диске. Архив содержит набор файлов `.ipynb` (IPython Notebook), содержащих исходный код этой книги (подробности о загрузке исходного кода см. во Введении). Используйте следующие шаги, чтобы импортировать эти файлы в свое хранилище.

1. **На странице блокнота DL4D щелкните на кнопке Upload (Выгрузить).**

   То, что вы видите, зависит от вашего браузера. В большинстве случаев вы видите диалоговое окно загрузки файлов, предоставляющее доступ к файлам на жестком диске.
2. **Перейдите в каталог, содержащий файлы, подлежащие импорту в Notebook.**

3. **Выделите один или несколько файлов для импорта, а затем щелкните на кнопке Open (Открыть) (или другой аналогичной кнопке), чтобы начать процесс загрузки.**

   Вы увидите, что файл добавлен в список загрузки, как показано на рис. 3.11. Файл еще не является частью хранилища, вы просто выбрали его для загрузки.

4. **Щелкните на кнопке Upload (Выгрузить).**

   Notebook помещает файл в хранилище, чтобы вы могли начать его использовать.

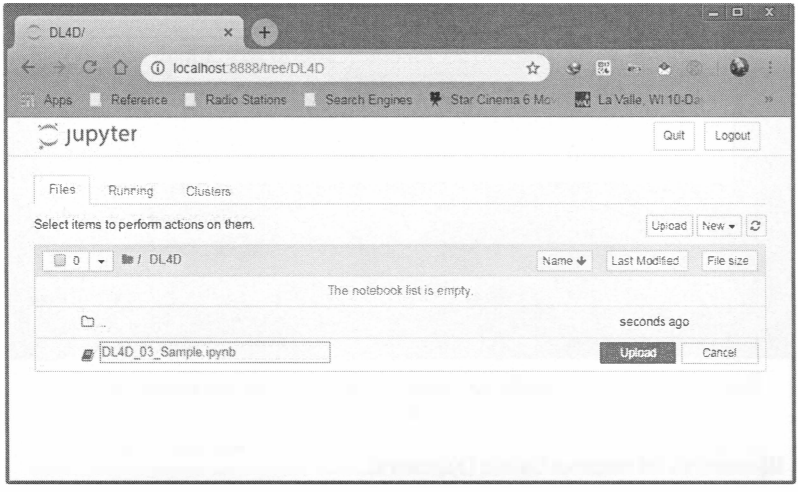


Рис. 3.11. Подлежащие добавлению в хранилище файлы появляются в списке выгрузки

Получение и использование наборов данных

В этой книге используется несколько наборов данных, некоторые из которых вы загружаете непосредственно из Интернета, а другие содержатся в пакетах Python, таких как библиотека Scikit-learn. Эти наборы данных демонстрируют различные способы взаимодействия с данными, и вы будете использовать их в примерах для решения различных задач. Ниже представлен краткий обзор функций, используемых для импорта каждого из наборов данных библиотеки Scikit-learn в код Python.

- `load_boston()`. Регрессионный анализ с набором данных по ценам на жилье в Бостоне.
- `load_iris()`. Классификация с набором данных Iris.

- `load_digits([n_class])`. Классификация с набором данных digits.
- `fetch_20newsgroups(subset='train')`. Данные из 20 групп новостей.

Техника загрузки каждого из этих наборов данных одинакова для всех примеров. В следующем примере показано, как загрузить набор данных о ценах на жилье в Бостоне. Вы можете найти этот код в блокноте `DL4D_03_Dataset_Load.ipynb`.

```
from sklearn.datasets import load_boston

Boston = load_boston()

print(Boston.data.shape)
```

Чтобы увидеть, как работает код, щелкните на кнопке Run Cell (Запустить ячейку). Вывод функции `print` составляет `(506, 13)`. Результат приведен на рис. 3.12. (Будьте терпеливы, загрузка набора данных может занять несколько секунд.)

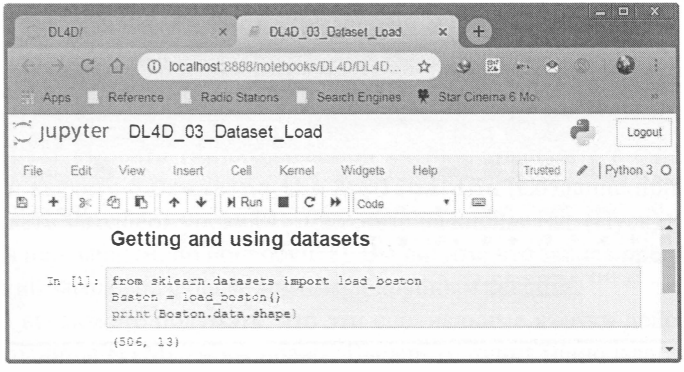


Рис. 3.12. Объект `Boston` содержит загруженный набор данных

Создание приложения

В разделе "Создание нового блокнота" показано, как создать пустой блокнот, что приятно, но не полезно. Вы хотите использовать блокнот для хранения приложения, которое вы можете использовать для исследования работы глубокого обучения. В следующих разделах показано, как создать в блокноте простое приложение для любых целей. Тем не менее, прежде чем начать,

убедитесь, что у вас есть файл DL4D_03_Sample.ipynb и он открыт для использования, поскольку он нужен для изучения Notebook.

Понятие ячеек

Если бы Notebook был стандартной средой IDE, у вас не было бы ячеек. То, что у вас есть, — это документ, содержащий одну непрерывную серию утверждений. Чтобы разделить элементы кода, вам нужны отдельные файлы. Ячейки разные, поскольку каждая ячейка отдельна. Да, результаты того, что вы делали в предыдущих ячейках, имеют значение, но если ячейка предназначена для работы в одиночку, вы можете просто перейти к этой ячейке и запустить ее. Чтобы увидеть, как это работает, введите следующий код в первую ячейку файла DL4D_03_Sample.

```
myVar = 3 + 4
print(myVar)
```

Теперь щелкните на кнопке Run (Пуск) (стрелка вправо). Код выполняется, и вы видите вывод, как показано на рис. 3.13. Вывод, как и ожидалось, 7. Но обратите внимание на надпись In [1]:. Эта уведомление о том, что выполняется первая ячейка.

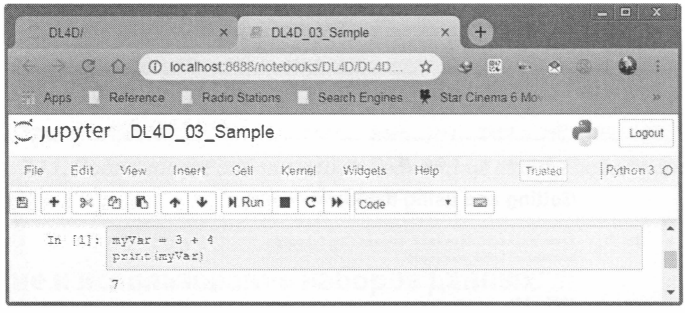


Рис. 3.13. В Notebook ячейки выполняются индивидуально

Теперь поместите курсор во вторую ячейку, она в настоящее время пуста, и введите **print("This is myVar: ", myVar)**. Щелкните на кнопке Run (Пуск). Вывод на рис. 3.14 показывает, что ячейки выполнялись индивидуально (поскольку надпись In [2]: показывает отдельное выполнение), а также то, что переменная myVar является глобальной для блокнота. То, что вы делаете в других ячейках с данными, влияет на все остальные ячейки, независимо от порядка выполнения.

Рис. 3.14. Изменения данных влияют на каждую ячейку, использующую измененную переменную

Добавление ячеек документации

Ячейки бывают разных типов. Эта книга использует не все из них. Однако знание того, как использовать ячейки документации, может пригодиться. Выберите первую ячейку (ту, которая в данный момент помечена 1). Выберите пункт меню Insert⇨Insert Cell Above (Вставка⇨Вставить ячейку выше).

Вы видите новую добавленную в блокнот ячейку. Обратите внимание на раскрывающийся список, где в данный момент указано Code (Код). Этот список позволяет выбрать тип создаваемой ячейки. Выберите в списке пункт Markdown и введите **`# Creating the Application`** (Создание приложения), чтобы создать заголовок уровня 1. Щелкните на кнопке Run (Пуск) (что может показаться необычным, но попробуйте). Вы видите, что запись превратилась в фактический заголовок с более темным и крупным текстом.

Сейчас вы можете подумать, что эти специальные ячейки действуют так же, как страницы HTML, и вы правы. Выберите пункт меню Insert⇨Insert Cell Above (Вставка⇨Вставить ячейку выше), выберите в раскрывающемся списке Markdown и введите **`## Understanding cells`** (Понятие ячеек), чтобы создать заголовок уровня 2. Щелкните на кнопке Run (Пуск). Как вы можете заметить на рис. 3.15, количество знаков #, добавляемых к тексту, влияет на уровень заголовка, но знаки # не отображаются в фактическом заголовке. (Вы можете найти полную документацию по Markdown для Notebook по адресу `https://www.ibm.com/support/knowledgecenter/en/SSGNPV_2.0.0/dsx/markd-jupyter.html`, а также в других местах в Интернете.)

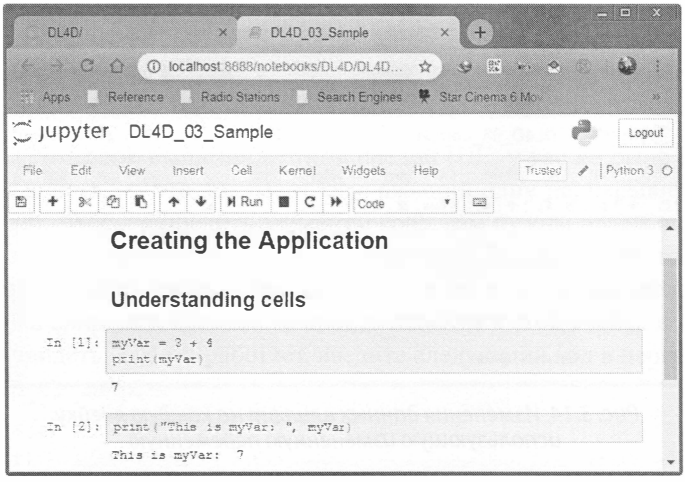


Рис. 3.15. Применение заголовков различных уровней подчеркивает содержание ячеек

Использование ячеек других типов

Эта глава (и книга) демонстрирует не все виды ячеек, которые можно просматривать с помощью Notebook. Но вы можете добавлять в свои блокноты и другие элементы, например графику. Когда придет время, вы сможете вывести (распечатать) свой блокнот в виде отчета и использовать его в презентациях любого рода. Техника грамотного программирования отличается от того, что вы, возможно, использовали в прошлом, и она имеет определенные преимущества, как вы увидите в следующих главах.

Использование отступов

Работая с примерами из этой книги, вы видите, что некоторые строки имеют отступ. Фактически, примеры демонстрируют также достаточное количество пустых пространств (таких как дополнительные строки между строками кода). Python игнорирует любые отступы в вашем приложении. Основной причиной добавления отступов является предоставление визуальных подсказок о вашем коде. Как и в книгах, отступы в коде показывают отношения между различными элементами.

Различные способы применения отступов станут понятней, когда вы будете разбирать примеры из книги. Тем не менее, вы должны с самого начала знать, как и почему применяется отступ. Поэтому пришло время для другого

примера. Следующие шаги помогут вам создать новый пример, в котором используются отступы, чтобы сделать связь между элементами приложения намного наглядней.

1. **Выберите пункт меню New⇨Python3 (Новый⇨Python3).**

   Jupyter Notebook создает новый блокнот. Файл загруженного исходного кода имеет имя `DL4D_03_Indentation.ipynb`, но вы можете использовать любое имя.

2. **Введите `print("This is a really long line of text that will " +`.**

   Вы видите текст, нормально отображаемый на экране, как и ожидалось. Знак плюс (+) сообщает Python о наличии дополнительного текста для отображения. Суммирование текста из нескольких строк в один длинный фрагмент текста называется *конкатенацией* (concatenation). Позже в книге вы узнаете больше об использовании этой функции, поэтому сейчас вам не нужно беспокоиться об этом.

3. **Нажмите клавишу <Enter>.**

   Точка вставки не возвращается к началу строки, как можно было бы ожидать. Вместо этого она перемещается прямо под первую двойную кавычку. Эта функция, называемая автоматическим отступом и является одной из функций, отличающей обычный текстовый редактор от редактора, предназначенного для написания кода.

4. **Введите `"appear on multiple lines in the source code file.")` и нажмите клавишу <Enter>.**

   Обратите внимание, что точка вставки возвращается к началу строки. Когда Notebook обнаружит, что вы достигли конца кода, он автоматически переместит текст в исходное положение.

5. **Щелкните на кнопке Run (Пуск).**

   Результат показан на рис. 3.16. Несмотря на то, что текст в файле с исходным кодом содержится в нескольких строках, в выводе находится только одна строка. Строка разделена, поскольку она длиннее размера окна, но на самом деле это всего лишь одна строка.

Добавление комментариев

Люди создают для себя заметки все время. Когда вам нужно купить продукты, вы просматриваете свои запасы, определяете, что вам нужно, и заносите это в список или в приложение на своем телефоне. Попав в магазин, вы просматриваете свой список, чтобы вспомнить, что вам нужно. Использование заметок удобно для решения самых разных задач, таких как отслеживание

хода переговоров между деловыми партнерами или запоминания основных моментов лекции. Людям нужны заметки, чтобы освежить их воспоминания. Комментарии в исходном коде — это просто еще одна форма заметки. Вы добавляете их в код, чтобы впоследствии помнить, какую задачу выполняет код. Следующие разделы описывают комментарии более подробно. Вы можете найти эти примеры в загружаемом файле `DL4D_03_Comments.ipynb`.

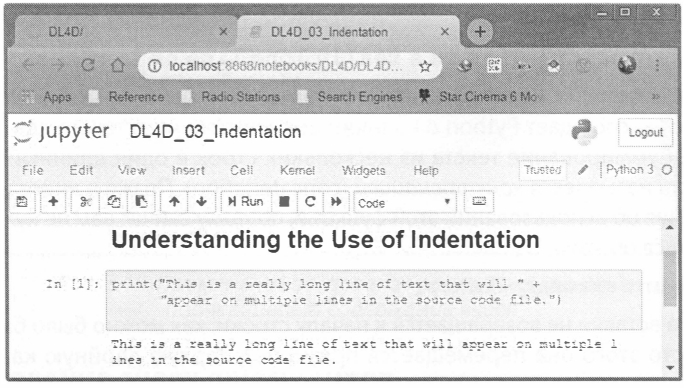


Рис. 3.16. Конкатенация приводит к слиянию нескольких строк кода в одну строку вывода

ЗАГОЛОВКИ ИЛИ КОММЕНТАРИИ

Поначалу заголовки и комментарии могут показаться немного непонятными. Заголовки располагаются в отдельных ячейках; комментарии располагаются в исходном коде. Они служат разным целям. Заголовки способны поведать о группировке всего кода, а комментарии — об отдельных шагах кода или даже строках кода. Хотя оба используются для документации, каждый служит уникальной цели. Комментарии, как правило, подробней заголовков.

Понятие комментариев

Компьютерам нужен некий особый способ определить, что написанный вами текст является комментарием, а не кодом для выполнения. Язык Python предоставляет два способа определения текста как комментария, а не кода. Первый способ — однострочный комментарий. Для него используется знак #, например.

```
# Это комментарий.
print("Hello from Python!") #Это тоже комментарий.
```

ЗАПОМНИ!

Однострочный комментарий может располагаться в строке сам по себе или после исполняемого кода. Он отображается только в одной строке. Обычно вы используете однострочный комментарий для короткого описательного текста, такого как объяснение определенного фрагмента кода. Notebook выделяет комментарии цветом (обычно синим) и курсивом.

Python фактически не поддерживает многострочные комментарии непосредственно, но вы можете создать его, используя строку в тройных кавычках. Многострочный комментарий начинается и завершается тремя двойными кавычками (`"""`) или тремя одинарными кавычками (`'''`) следующим образом.

```
"""
    Приложение: Comments.py
    Автор: Джон
    Цель: демонстрирует использование комментариев.
"""
```

ЗАПОМНИ!

Эти строки не выполняются. Встретив их в коде, Python не будет отображать сообщения об ошибке. Но Notebook отображает их иначе, как показано на рис. 3.17. Обратите внимание, что реальные комментарии Python, которым предшествует знак # в ячейке 1, не создают никакого вывода. Однако строки с тройными кавычками выходные данные создают. Кроме того, в отличие от стандартных комментариев, текст в тройных кавычках отображается красным (в зависимости от редактора), а не синим цветом, и текст не выделен курсивом. Если вы планируете выводить блокнот в виде отчета, вам следует избегать использования строк в тройных кавычках. (Некоторые IDE, такие как IDLE, полностью игнорируют строки в тройных кавычках.)

Многострочные комментарии обычно используются для подробного описания того, кто создал приложение, почему оно было создано и какие задачи оно решает. Конечно, жестких правил использования комментариев нет. Главное, сообщить компьютеру, что является комментарием, а что не является, чтобы он не пытался обрабатывать комментарий как код.

Применение комментариев для напоминания

Многие люди не совсем понимают назначение комментариев и не знают, что делать с заметками в коде. Имейте в виду, что вы можете написать фрагмент кода сегодня, а потом не трогать его годами. Вам нужны заметки для напоминания, чтобы вы могли вспомнить, какую задачу решает код и почему вы

его написали. Фактически, вот несколько распространенных причин использования комментариев в вашем коде.

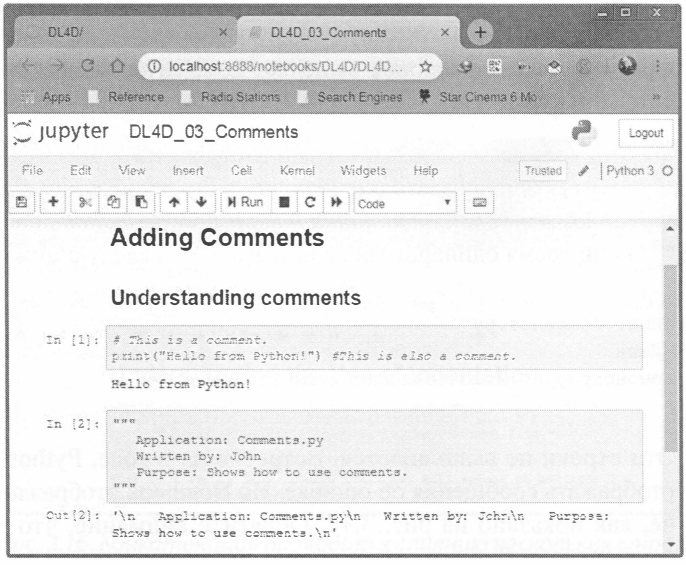


Рис. 3.17. Многострочные комментарии работают, но они также создают вывод

- Напомнить себе, что делает код и почему вы его написали.
- Рассказать другим, как поддерживать ваш код.
- Сделать свой код понятным для других разработчиков.
- Список идей для будущих обновлений.
- Список источников документации, использованной для написания кода.
- Список сделанных улучшений.

Комментарии можно также использовать и другими способами, но эти способы являются наиболее популярными. Посмотрите, как используются комментарии в примерах этой книги, особенно в последующих главах, где код становится более сложным. По мере того, как ваш код становится все более сложным, вам нужно добавлять больше комментариев и относиться они должны к тому, что вам нужно запомнить.

Применение комментариев для предотвращения выполнения кода

Разработчики иногда используют комментарии для предотвращения выполнения строк кода (это *комментирование* (commenting out)). Возможно, вам придется делать это, чтобы убедиться, не эта ли строка кода вызывает проблемы в вашем приложении. Подобно любым другим комментариям, вы можете использовать либо однострочное, либо многострочное комментирование. Но при использовании многострочного комментирования в выводе вы увидите не выполняющийся код (и на самом деле, увидеть, какой именно код влияет на вывод, может быть очень полезно).

Получение справки по языку Python

Эта книга не учит вас языку Python, что потребовало бы отдельной книги. Конечно, для этого вы всегда можете использовать книгу Джона Пола Мюллера *Python для чайников, 2-е издание* (пер. с англ., изд. "Диалектика"). Но у вас есть много других вариантов получения справки по языку Python. На самом деле доступно так много вариантов, что охватить их все в этой главе невозможно. Вот лучшие методы для получения справки.

- Выберите один из параметров в меню Help (Справка) приложения Notebook.
- Откройте приглашение Anaconda, запустите экземпляр Python и используйте текстовые команды для нахождения справки.
- Загрузите документацию по Python с сайта `https://docs.python.org/3.6/download.html`.
- Просмотрите сетевую документацию по адресу `https://docs.python.org/3.6/`.
- Используйте любое из следующих руководств.
  - Официальный учебник. `https://docs.python.org/3.6/`.
  - TutorialsPoint. `https://www.tutorialspoint.com/python/`.
  - W3Schools. `https://www.w3schools.com/python/`.
  - learnpython.org. `https://www.learnpython.org/`.
  - Codecademy. `https://www.codecademy.com/learn/learn-python`.

ЗАПОМНИ!

Дело в том, что эта книга подразумевает, что вы уже умеете программировать на языке Python. Данная глава предоставляет вам некоторую помощь по сопутствующим инструментам, чтобы облегчить переход от любых других инструментов, которые вы использовали в прошлом, к инструментам, использованным в этой книге.

Работа в облаке

Несмотря на то, что в этой главе представлен подход локальной обработки, для выполнения определенных задач может возникнуть необходимость взаимодействия с облачными ресурсами. В следующих разделах обсуждаются два связанных с облаком действия, которые вы можете выполнять при использовании этой книги. Первый — это доступ к облачным ресурсам для различных нужд. Второй — использование Google Colab Laboratory для работы с примерами на планшете, а не на настольной системе.

Использование наборов данных Kaggle и ядер

Kaggle (`https://www.kaggle.com/`) — это огромное сообщество аналитиков данных и других специалистов, которым приходится работать с большими наборами данных для получения информации, необходимой для достижения различных целей. В Kaggle вы можете создавать новые проекты, просматривать работу, проделанную другими или участвовать в одном из текущих конкурсов. Но Kaggle — это куда больше, чем просто сообщество действительно умных людей, которые любят играть с данными; это также место, где вы можете получить ресурсы, необходимые для изучения всех вопросов глубокого обучения и создания собственных проектов.

СОВЕТ

Наилучшее место, где можно узнать, чем Kaggle может помочь вам в изучении глубокого обучения — это `https://www.kaggle.com/m2skills/datasets-and-tutorial-kernels-for-beginners`. На этом сайте перечислены различные наборы данных и учебные ядра, предоставляемые Kaggle. *Набор данных* (dataset) — это просто некая база данных, используемая для выполнения стандартных тестов кода приложения. *Учебное ядро* (tutorial kernel) — это своего рода проект, который вы используете, чтобы научиться анализировать данные различными способами. Например, вы можете найти учебное ядро о классификации грибов по адресу `https://www.kaggle.com/uciml/mushroom-classification`.

Использование Google Colaboratory

Colab Laboratory (`https://colab.research.google.com/notebooks/welcome.ipynb`), или кратко Colab, представляет собой облачный сервис Google, реплицирующий Jupyter Notebook в облаке. Это пользовательская реализация, поэтому вы можете встретить случаи, когда Colab и Notebook не синхронизированы — функции одного из них могут не всегда работать в другом. Для его использования не нужно ничего устанавливать в своей системе. В большинстве случаев Colab используется так же, как и настольная версия Jupyter Notebook. Узнать больше о Colab имеет смысл в случае, если для работы с примерами этой книги вы хотите использовать устройство отличное от стандартной конфигурации рабочего стола. Если вы хотите получить более полное руководство по Colab, его можно найти в главе 4 книги *Python и наука о данных для чайников, 2-е издание*, написанной Джоном Полом Мюллером и Лукой Массароном (пер. с англ., изд. "Диалектика"). На данный момент в этом разделе излагаются основы использования существующих файлов. Открытый экран Colab показан на рис. 3.18.

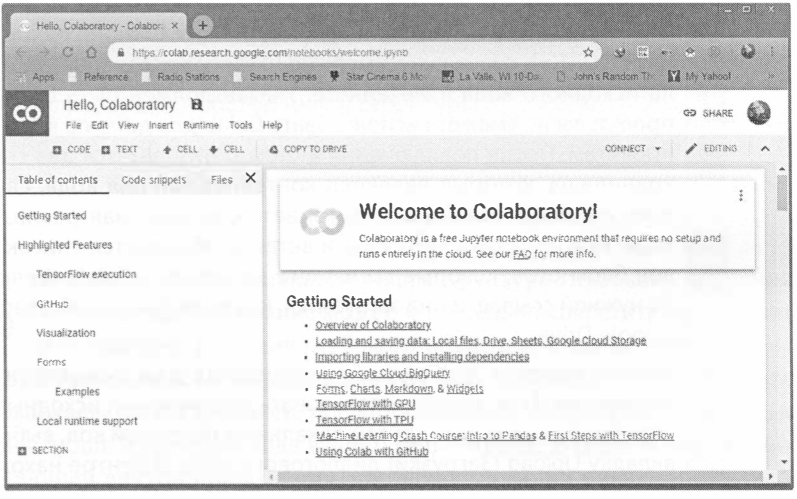


Рис. 3.18. Colab упрощает использование проектов Python на планшете

Вы можете открывать существующие блокноты, находящиеся в локальном хранилище, на Google Drive или на GitHub. Вы также можете открыть любой из примеров Colab или загрузить файлы из таких источников, к которым у вас есть доступ, включая сетевой диск в вашей системе. В любом случае сначала выберите пункт меню File⇨Open Notebook (Файл⇨Открыть блокнот). В стандартном представлении отображаются все файлы, которые вы недавно

открывали, независимо от их местоположения. Файлы отображаются в алфавитном порядке. Вы можете отфильтровать отображаемые элементы, введя соответствующую строку в поле Filter Notebooks (Фильтровать блокноты). Вверху находятся другие варианты открытия блокнотов.


СОВЕТ

Даже если вы не вошли в систему, вы все равно можете получить доступ к примерам проектов Colab. Эти проекты помогут вам изучить Colab, но не позволят что-либо делать со своими проектами. Тем не менее, вы можете по-прежнему экспериментировать с Colab, не входя первоначально в Google. В следующих разделах эти варианты обсуждаются более подробно.

- **Использование Google Drive для существующих блокнотов**. Google Drive является стандартным местоположением для многих операций в Colab, и вы всегда можете выбрать его в качестве пункта назначения. При работе с Google Drive вы увидите список файлов. Чтобы открыть тот или иной файл, щелкните на его ссылке в диалоговом окне. Файл откроется в текущей вкладке вашего браузера.
- **Использование GitHub для существующих блокнотов**. При работе с GitHub сначала необходимо указать местоположение файла исходного кода в Интернете. Указан должен быть открытый проект; вы не сможете использовать Colab для доступа к частным проектам. После подключения к GitHub отобразятся два списка: хранилища, которые являются контейнерами для кода, связанного с конкретным проектом; и ветви, конкретная реализация кода. При выборе хранилища и ветви отображается список файлов блокнотов, которые вы можете загрузить в Colab. Щелкните на нужной ссылке, и она загрузится, как если бы вы использовали Google Drive.
- **Использование локального хранилища для существующих блокнотов**. Если хотите использовать загружаемый исходный код этой книги или любой другой локальный исходный код, выберите вкладку Upload (Загрузки) диалогового окна. В центре находится одна кнопка Choose File (Выбрать файл), после щелчка на которой откроется диалоговое окно File Open (Открыть файл) вашего браузера. Необходимый для загрузки файл вы находите как обычно при открытии любого файла. Выбрав файл и щелкнув на кнопке Open (Открыть), вы загружаете файл на Google Drive. Если вы вносите изменения в файл, то они появятся на Google Drive, а не на локальном диске.

Глава 4

Использование инфраструктуры глубокого обучения

В ЭТОЙ ГЛАВЕ...

- Понятие инфраструктуры выполнения
- Использование простой инфраструктуры
- Работа с TensorFlow

В этой главе рассматривается инфраструктура глубокого обучения, поскольку ее использование может значительно сократить время и затраты на разработку решения глубокого обучения, а также упростить ее. Конечно, начать следует с определения термина *инфраструктура* (framework), который является абстракцией, предоставляющей общие функциональные возможности для изменения кода вашего приложения. В отличие от библиотеки, работающей внутри приложения, при использовании инфраструктуры, ваше приложение работает внутри нее. Вы не можете изменять базовые функции инфраструктуры, а значит, при работе она стабильна, но большинство инфраструктур допускают некоторый уровень расширяемости. Инфраструктуры обычно специфичны для конкретных нужд, веб-инфраструктуры, например, используются для создания сетевых приложений. Следовательно, хотя инфраструктуры глубокого обучения имеют много общих характеристик, они также предоставляют специфические функциональные возможности, рассматриваемые в этой главе.

Не все используют для запуска приложений глубокого обучения одинаковые идеи и концепции. Кроме того, не каждая организация хочет платить за

сложную инфраструктуру глубокого обучения, когда подойдут и менее дорогие. Следовательно, вы найдете множество инфраструктур глубокого обучения, способных обеспечить вам базовую функциональность, которую вы можете использовать для экспериментов и простых приложений. Некоторые из этих простых инфраструктур рассматриваются в этой главе в сравнении, чтобы вы получили лучшее представление о том, что доступно.

Для обеспечения наилучшей возможной инфраструктуры обучения, для примеров этой книги используется инфраструктура TensorFlow. В ситуациях, рассматриваемых этой книгой, TensorFlow работает лучше, чем другие решения, и в этой главе объясняется, почему. Здесь также точно указано, почему TensorFlow является хорошим общим решением для многих сценариев глубокого обучения.

ЗАПОМНИ!

Вам не нужно вводить исходный код примеров из этой главы вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файлах `DL4D_03_Comments.ipynb`, `DL4D_03_Dataset_Load.ipynb`, `DL4D_03_Indentation.ipynb` и `DL4D_03_Sample.ipynb`.

Знакомство с инфраструктурой

Как упоминалось во введении, ваш код работает в инфраструктуре. В окружении инфраструктуры, ваш код запрашивает инфраструктуру, которая затем и выполняет запрос сама. Следовательно, инфраструктура предоставляет некую структуру для разработки приложений. Из-за этого инфраструктуры являются предметно-ориентированными и отвечают специфическим требованиям разработчиков приложений. В следующих разделах инфраструктуры обсуждаются как решение глубокого обучения. Обратите внимание, что эти разделы не предоставляют полную информацию об инфраструктурах, но они помогут вам достаточно хорошо понять инфраструктуру глубокого обучения, чтобы принимать правильные решения.

Определение различий

Предметная область специфична по природе, а, следовательно, необходимо найти инфраструктуру, подходящую именно для ваших нужд. *Предметная область* (problem domain) — это описание опыта и ресурсов, необходимых для решения задачи. Например, вы не идете к врачу, чтобы решить свои проблемы с сантехникой, вместо этого вы идете к сантехнику. Просто выясните, какая

инфраструктура обеспечит наибольший выигрыш. Вот несколько примеров типов инфраструктур, каждая из которых имеет специфические характеристики для удовлетворения потребностей своей предметной области.

- Инфраструктура приложения (используется для создания приложений конечного пользователя).
- Художественная (рисунки, музыка и другие формы творчества).
- Инфраструктура Cactus (высокопроизводительные научные вычисления).
- Система поддержки принятия решений.
- Моделирование системы Земли.
- Финансовое моделирование.
- Веб-инфраструктура (включая инфраструктуры специфические для таких языков, как AJAX и JavaScript).

ЗАПОМНИ!

Разнообразие программных инфраструктур поражает воображение, и вряд ли они вам когда-нибудь понадобятся все. У них есть две важные общие черты. В любом случае, инфраструктура имеет набор *холодных точек* (frozen spot), определяющих характеристики приложения и которые разработчик не может изменить. Кроме того, инфраструктура определяет *горячие точки* (hot spot), которые разработчик использует для определения специфики целевого программного обеспечения. Например, холодная точка в веб-приложении может определять интерфейс, на который пользователь полагается при выполнении запросов, а горячая точка может определять способ выполнения этого запроса. Тот, кто разрабатывает приложение для поиска книг, сосредоточится на специфике поиска книг, игнорируя при этом требования к управлению состоянием и обработке запросов.

Популярность инфраструктур

Размышляя о программном обеспечении, вы можете легко увидеть развитие используемых для его создания инструментов. В свое время разработчикам приходилось вводить свой код при помощи перфокарт, что было чрезвычайно трудоемким и подверженным ошибкам делом. Редакторы облегчают работу, поскольку теперь вы можете вводить то, что хотите. *Интегрированная среда разработки* (Integrated Development Environment — IDE) стала следующим шагом. Использование IDE позволяет моделировать, компилировать и тестировать код в одной среде. Использование библиотек позволяет быстро создавать большие и сложные приложения. Таким образом, инфраструктура,

представляющая собой среду, в которой разработчик должен учитывать только особенности конкретного приложения, — это просто следующий шаг в повышении производительности труда разработчиков, а также в создании приложений, более устойчивых и менее подверженных ошибкам. Отсюда и популярность инфраструктур у разработчиков.

ЗАПОМНИ!

Тем не менее, инфраструктура — это гораздо больше, чем просто средство для быстрого создания кода с меньшими усилиями и меньшим количеством ошибок. Инфраструктура позволяет создать стандартизированную среду, в которой все используют одни и те же библиотеки, инструменты, интерфейсы прикладного программирования (API) и другие средства. Использование стандартизированной среды позволяет переносить код между системами, не опасаясь появления странных проблем из-за несоответствий среды. Кроме того, меньше проблем коллективной разработки, поскольку среда совместной разработки упрощена.

Поскольку инфраструктура сама выполняет все действия низкого уровня, вам остается учесть состав команды разработчиков приложения. В прошлом команде могли бы понадобиться люди, обладающие опытом во взаимодействии с аппаратным обеспечением или создании основ пользовательского интерфейса. Использование инфраструктуры означает, что все эти задачи уже выполнены, поэтому в состав группы входят эксперты в предметной области, способные эффективно общаться друг с другом, что делает возможным согласованный подход к разработке приложений.

Самая важная причина нынешней популярности инфраструктур, связана со способом современного программирования. Когда-то разработчикам нужно было знать, как взаимодействовать с аппаратным и программным обеспечением на крайне низком уровне. Сегодня инфраструктуры облегчают программирование в среде, в которой.

- Большинство приложений состоят в основном из вызовов функций API, соединенных вместе для достижения определенной цели.
- Люди должны понимать, как работают функции API, а не то, что они делают или как они это делают. Разработчик должен учитывать, какие структуры данных получает функция API и насколько хорошо она обрабатывает их.
- Установлена огромная база существующего программного обеспечения, требующая сохранения этого кода, а также поиска быстрых и эффективных методов взаимодействия с ним.

- Основное внимание уделяется архитектуре, а не деталям. Поскольку большинство новых приложений в значительной степени зависят от существующего кода, доступ к которому осуществляется через библиотеки или интерфейсы API, разработчики не тратят много времени на изучение особенностей языка; лучше выяснить, какой код из этого вороха способен выполнить данную задачу и не писать какой-либо код самостоятельно.
- Выбор правильного алгоритма важнее всего.
- Инструменты стали настолько умными, что они часто исправляют мелкие ошибки программирования и правильно интерпретируют неоднозначности в коде разработчика, поэтому акцент делается на выработке идеи, а не на написании идеального кода.
- Визуальные языки, в которых вы перетаскиваете объекты в графической среде, становятся все более распространенными. В какой-то момент код может фактически исчезнуть (по крайней мере, для большинства разработчиков приложений).
- Знания об одной инфраструктуре недостаточно. Большинство приложений сегодня должны безупречно работать на Windows, Linux, OS X, Android, большинстве смартфонов и множестве других инфраструктур, поскольку пользователи хотят, чтобы программное обеспечение находилось в понятной им форме.

УЧЕТ НЕДОСТАТКОВ ИНФРАСТРУКТУР

Инфраструктурное решение не всегда является панацеей, каковой ее считают сторонники. Одна из самых больших проблем при использовании инфраструктур заключается в том, что инфраструктура сама по себе является приложением. Команда разработчиков должна изучить как саму структуру, так и все инструменты, используемые для написания приложения. Следовательно, если большинство занятых разработкой членов команды ранее не использовали эту инфраструктуру, им потребуется дополнительное время для обучения. Тем не менее, после того, как они научатся использовать инфраструктуру, они легко наверстают часть потраченного времени за счет более высокой производительности в целом.

Другая проблема с инфраструктурами — их тенденция неэффективно использовать ресурсы. Размер написанного на инфраструктуре приложения, включая саму инфраструктуру, обычно куда больше, чем у приложения, разработанного с использованием библиотек. Конечно, монолитные приложения, как правило, эффективней, поскольку они могут использовать только те ресурсы, которые необходимы только для этого приложения. Весь рас-

положенный в инфраструктурах код является следствием попыток создать универсальное решение.

Обсуждаемые в этой книге инфраструктуры являются открытыми предложениями. На самом деле, большинство из них также разработано с открытым исходным кодом. Однако некоторые сторонники инфраструктур считают, что у каждого предприятия должна быть своя собственная инфраструктура, разработанная с использованием общего кода из приложений данного предприятия. При таком подходе итоговая инфраструктура имеет согласованный вид, соответствующий прежним (до инфраструктуры) приложениям, которые необходимо поддерживать предприятию. Однако разработка пользовательской инфраструктуры для конкретного предприятия занимает много времени. Поэтому многие люди отмечают, что решение на основе инфраструктуры не так полезно и не так легко понятно, как не инфраструктурные решения.

Определение инфраструктуры глубокого обучения

Когда вы думаете об инфраструктуре глубокого обучения, в действительности вы задаетесь вопросом о том, как она управляет холодными и горячими точками. В большинстве случаев инфраструктура глубокого обучения предоставляет холодные и горячие точки в следующих областях.

- Доступ к аппаратному обеспечению (например, простота использования графического процессора).
- Стандартный доступ к слоям нейронной сети.
- Простой доступ к глубокому обучению.
- Управление вычислительными графами.
- Обучение модели.
- Развертывание модели.
- Тестирование модели.
- Построение графиков и презентаций.
- Вывод (прямое распространение).
- Автоматическое дифференцирование (обратное распространение).

Инфраструктуры решают и другие проблемы, а фокус на определенных задачах определяет применимость конкретной инфраструктуры для конкретной цели. Как и обычно, при разработке программного обеспечения, вам нужно тщательно выбирать используемую инфраструктуру.

Выбор конкретной инфраструктуры

В предыдущих разделах этой главы обсуждалась привлекательность инфраструктур в целом, а также то, как инфраструктуры могут создать значительно лучшую рабочую среду для разработчиков. Также рассматривались признаки, делающие инфраструктуру глубокого обучения специфичной. Конечно, объем автоматизации, предоставляемой инфраструктурой, и количество поддерживаемой ей типовых функций, являются отправной точкой для поиска инфраструктуры, отвечающей вашим потребностям. Вы также должны рассмотреть такие вопросы, как скорость обучения и простота использования инфраструктуры.

СОВЕТ

Одним из наиболее важных соображений при выборе инфраструктуры является то, что они специфичны для предметной области, а значит, если вам нужно создать приложение, охватывающее несколько областей, например, приложение глубокого обучения, включающее в себя веб-интерфейс, вам понадобится несколько инфраструктур. Применение инфраструктур, которые хорошо работают друг с другом, может иметь решающее значение. Если вы размещаете свое приложение в облаке, вам необходимо продумать, какие инфраструктуры работают также с предложением поставщика облака. Например, если в качестве своей инфраструктуры вы решите использовать TensorFlow, то для размещения своего приложения вы также можете полагаться на Amazon Web Services (AWS) (см. `https://aws.amazon.com/tensorflow/`).

ЗАПОМНИ!

В качестве другого варианта при использовании TensorFlow вы можете перейти непосредственно в Google Cloud (см. подробнее на `https://cloud.google.com/tpu/`), где вы можете обучить свое решение глубокого обучения с использованием графических процессоров или модулей обработки Tensor Processing Units (TPU). Модули TPU были разработаны Google специально для машинного обучения нейронной сети с использованием TensorFlow. TPU — это специализированные интегральные схемы (Application-Specific Integrated Circuits — ASIC), оптимизированные для конкретного использования. В данном случае они предназначены для обработки нейронных сетей с использованием TensorFlow.

Размер и сложность приложения также играют роль при выборе среды глубокого обучения, поскольку для правильного взаимодействия с большими приложениями зачастую требуется более мощная инфраструктура. Необходимость

иметь дело с приложениями различного рода компенсируется обычными проблемами стоимости и доступности. Многие из бюджетных инфраструктур глубокого обучения в этой главе не будут стоить вам ничего, при опробовании, и могут предоставить все необходимое для начала.

Работа с бюджетными инфраструктурами

Бюджетные инфраструктуры глубокого обучения зачастую имеют внутренний компромисс. Вы должны выбирать между ценой и сложностью применения, а также возможностью поддержки больших приложений в сложных условиях. Компромиссы, с которыми вы готовы мириться, обычно отражают то, что вы можете использовать для завершения своего проекта. С учетом этого, в следующих разделах обсуждается ряд бюджетных инфраструктур, которые невероятно полезны и хорошо работают с небольшими и средними проектами.

Caffe2

Инфраструктура Caffe2 (`https://caffe2.ai/`) основана на инфраструктуре Caffe, которая первоначально была разработана в Калифорнийском университете в Беркли. Она написана на языке C++ с интерфейсом Python. Одна из причин, по которым людям действительно нравится Caffe2, заключается в том, что вы можете обучать и развертывать модель без написания кода. Вместо этого вы выбираете одну из заранее написанных моделей и добавляете ее в файл конфигурации (который удивительно похож на код JSON). Фактически, большой выбор предварительно обученных моделей предоставлен как часть Model Zoo (`https://github.com/BVLC/caffe/wiki/Model-Zoo`).

ПЕРЕХОД С CAFFE НА CAFFE2

Несмотря даже на то, что Caffe (`http://caffe.berkeleyvision.org/` и `https://github.com/BVLC/caffe`) все еще существует, и многие ее используют, вы можете обнаружить, что продукт, который вам действительно нужен — это Caffe2. Если у вас сейчас есть несколько приложений Caffe, вы можете перенести их на Caffe2, используя методы, которые можно найти по адресу `https://caffe2.ai/docs/caffe-migration.html`, поэтому любые инвестиции, сделанные вами в Caffe, по-прежнему полезны в Caffe2.

У оригинальной инфраструктуры Caffe было много проблем, которые делали ее менее привлекательной для аналитиков данных, чем Caffe2. Нынешняя версия Caffe по-прежнему популярна, но на самом деле вы не можете

использовать ее для чего-то сложного. Инфраструктура Caffe2 превосходит Caffe в следующем.

- Лучшая поддержка крупномасштабного распределенного обучения.
- Мобильная разработка.
- Добавлена поддержка CPU и поддержка графических процессоров через CUDA.

Chainer

Chainer (`https://chainer.org/`) — это библиотека, написанная исключительно на языке Python и основанная на библиотеках NumPy (`http://www.numpy.org/`) и CuPy (`https://cupy.chainer.org/`). Разработку этой библиотеки возглавляет Preferred Networks (`https://www.preferred-networks.jp/en/`), но IBM, Intel, Microsoft и NVIDIA также играют свою роль. Основным преимуществом этой библиотеки является возможность CUDA использовать GPU, добавив всего несколько строк кода. Другими словами, эта библиотека позволяет значительно повысить скорость вашего кода при работе с огромными наборами данных.

Многие современные библиотеки глубокого обучения, такие как Theano (обсуждается в главе 19) и TensorFlow (обсуждается далее в этой главе), используют статический подход глубокого обучения, известный как *определение и выполнение* (define-and-run), при котором вы определяете математические операции, а затем выполняете обучение на основе этих операций. В отличие от Theano и TensorFlow, Chainer использует подход *определения при выполнении* (define-by-run), основанный на динамическом подходе глубокого обучения, при котором код определяет математические операции в процессе обучения. Вот два основных преимущества этого подхода.

- **Интуитивно понятный и гибкий подход**. Подход определения при выполнении может опираться на собственные возможности языка, а не требовать от вас создания специальных операций для выполнения анализа.
- **Отладка**. Поскольку подход определения при выполнении определяет операции во время обучения, вы можете полагаться на средства внутренней отладки, чтобы найти источник ошибок в наборе данных или коде приложения.

СОВЕТ

TensorFlow 2.0 также может использовать определение при выполнении, полагаясь на Chainer для обеспечения быстрого выполнения.

PyTorch

Боблиотека PyTorch (https://pytorch.org/) является преемником Torch (http://torch.ch/), написанным на языке Lua (https://www.lua.org/). Одна из основных библиотек Torch (библиотека-предшественник PyTorch) изначально была ветвью Chainer, которая описана в предыдущем разделе. Изначально PyTorch разработал Facebook, но сегодня ее используют многие другие организации, включая Twitter, Salesforce и Оксфордский университет. Особенной PyTorch делают следующие признаки.

- Высокое удобство для пользователя.
- Эффективное использование памяти.
- Относительная быстрота.
- Обычно используется для исследований.

Некоторым PyTorch нравится, потому что ее легко читать, как Keras, но ученый не теряет способность использовать сложные нейронные сети. Кроме того, PyTorch напрямую поддерживает динамический расчет вычислительной модели (подробней эта проблема рассматривается в разделе "Понятно, почему TensorFlow так хорош" далее в этой главе), что придает ей большую гибкость, чем у TensorFlow без добавления TensorFlow Fold.

MXNet

Основная причина использования библиотеки MXNet — это скорость. Может быть трудно определить, быстрее ли работает MXNet (https://mxnet.apache.org/) или CNTK (https://www.microsoft.com/en-us/cognitive-toolkit/), но оба продукта работают довольно быстро и часто используются теми, кого не устраивает медлительность TensorFlow. (В официальном документе по адресу https://arxiv.org/pdf/1608.07249v7.pdf приведены некоторые подробности о сравнительном анализе кода глубокого обучения.)

Библиотека MXNet — это продукт Apache, поддерживающий множество языков, включая Python, Julia, C++, R и JavaScript. Ее используют многие крупные организации, в том числе Microsoft, Intel и Amazon Web Services. Вот отличительные черты MXNet.

- Расширенная поддержка графического процессора.
- Возможность запуска на любом устройстве.
- Предоставляет высокопроизводительные императивные API.
- Обладает простой моделью обслуживания.
- Обладает высокой масштабируемостью.

Это может казаться идеальным продуктом для ваших нужд, но MXNet имеет хоть и один, но серьезный недостаток — ей не хватает поддержки сообщества, в отличие от TensorFlow. Кроме того, большинство исследователей недолюбливают MXNet за способность создавать сложности, и нестабильность модели в большинстве случаев.

Microsoft Cognitive Toolkit/CNTK

Как упоминалось в предыдущем разделе, одной из причин использования Microsoft Cognitive Toolkit (CNTK) является ее скорость. Microsoft использует CNTK для действительно больших наборов данных. Как продукт, она поддерживает языки программирования Python, C++, C# и Java. Следовательно, если вы исследователь полагающийся на язык R, это не ваш продукт. Microsoft использовала этот продукт в Skype, Xbox и Cortana. Особенности этого продукта.

- Отличная производительность.
- Высокая масштабируемость.
- Высокая оптимизация компонентов.
- Поддержка Apache Spark.
- Поддержка Azure Cloud.

Как и в случае с MXNet, у CNTK есть особая проблема, связанная с отсутствием адекватной поддержки сообщества. Кроме того, она, как правило, не обладает значительной поддержкой сторонних разработчиков, поскольку, если пакет не содержит необходимых вам функций, вы можете вообще их не получить.

Инфраструктура TensorFlow

В настоящее время TensorFlow находится на вершине систем глубокого обучения (см. подробнее на диаграмме по адресу `https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a`). Успех TensorFlow обусловлен многими причинами, но в основном он связан с надежностью среды и относительной простотой использования пакета. Следующие разделы помогут вам понять, почему эта книга использует TensorFlow. Вы узнаете, что делает TensorFlow такой интересной инфраструктурой и как дополнения делают ее еще проще в использовании.

Понятно, почему TensorFlow так хорош

Продукт предлагает не так уж и много с точки зрения функциональности, простоты использования и надежности, чтобы именно это обусловило его

успех на рынке, когда у людей есть столько вариантов. Одной из причин успеха TensorFlow является поддержка ряда самых популярных языков: Python, Java, Go и JavaScript. Кроме того, TensorFlow довольно расширяем. Каждое расширение является *операцией*, о которой вы можете прочитать на `https://www.tensorflow.org/guide/extend/op`. Дело в том, что когда продукт имеет отличную поддержку нескольких языков и обеспечивает значительную расширяемость, он становится популярным, поскольку люди могут выполнять задачи такими способами, которые им больше подходит, а не такими, которые нужны пользователю, по мнению поставщика.

ПОДДЕРЖКА TENSORFLOW НА COLAB

Сегодня для решения задач многие разработчики полагаются на такие сетевые среды как Colab, поскольку установка и настройка TensorFlow на настольном компьютере может оказаться сложной задачей, а если хотите ускоренную обработку, у вас должен быть графический процессор, поддерживаемый TensorFlow (`https://developer.nvidia.com/cuda-gpus`). Кроме того, будет множество других вопросов, которые придется учесть (`https://www.tensorflow.org/install/gpu`).

Colab, казалось бы, облегчает жизнь. Чтобы получить поддержку процессора, вам достаточно открыть окно конфигурации. Чтобы обеспечить надлежащую поддержку, вы просто запускаете немного дополнительного кода, специфичного для Colab (`https://colab.research.google.com/notebooks/gpu.ipynb`). Однако реальность редко совпадает с теорией. Прежде всего, вам нужно переустанавливать все каждый раз, когда вы запускаете новый сеанс Colab, поскольку поддержка библиотеки не постоянна (`https://www.kdnuggets.com/2018/02/essential-google-colaboratory-tips-tricks.html`). Конечно, у вас может вообще не быть доступа к графическому процессору (это на усмотрение Google) или поддержка графического процессора может иметь ограничения (`https://stackoverflow.com/questions/48750199/google-colaboratory-misleading-information-about-its-gpu-only-5-ram-available`).

В этой книге используется чрезвычайно упрощенная настройка TensorFlow, позволяющая избежать многих подводных камней, с которыми сталкиваются другие инфраструктуры. Конфигурация в книге подойдет для пользователей с любым опытом, который вы, возможно, получили в школе, на небольших экспериментальных проектах или даже проектах для малых и средних предприятий, использующих наборы данных малого и среднего размера. Вы никогда не сможете использовать эту настройку для запуска проекта типа Facebook.

Способ, которым TensorFlow оценивает и выполняет код, также важен. Собственно, TensorFlow поддерживает только статические вычислительные графы. Однако расширение TensorFlow Fold (`https://github.com/tensorflow/fold`) поддерживает также динамические графы. У *динамического графа* (dynamic graph) структура вычислительного графа динамически изменяется как функция структуры входных данных по мере выполнения приложения. Используя *динамическое пакетирование* (dynamic batching), TensorFlow Fold способен создать статический граф из динамических графов, который затем может быть передан в TensorFlow. Этот статический граф представляет собой преобразование одного или нескольких динамических графов, моделирующих неопределенные данные. Конечно, вам даже может не понадобиться строить вычислительный граф, поскольку TensorFlow поддерживает также *немедленное выполнение* (eager execution) (немедленная оценка операций без построения вычислительного графа), так что он может немедленно оценить код Python (*динамическое выполнение* (dynamic execution)). Наличие этой динамической функциональности делает TensorFlow чрезвычайно гибким в работе с данными.


ЗАПОМНИ!

Кроме различных видов динамической поддержки, TensorFlow позволяет также использовать графический процессор для ускорения вычислений. На самом деле вы можете использовать несколько графических процессоров и распределить вычислительную модель по нескольким машинам в кластере. Возможность применить такую вычислительную мощь для решения задачи делает TensorFlow быстрее, чем большинство конкурентов. Скорость важна, поскольку ответы на вопросы зачастую следует получать быстро; получение ответа на вопрос завтра, когда ответ нужен сегодня, во многих случаях не сработает. Например, врач, полагающийся на услуги искусственного интеллекта для выяснения альтернатив во время операции, нуждается в немедленном ответе, иначе пациент может умереть.

Вычислительные возможности только помогают найти решение задачи. TensorFlow помогает также визуализировать решение различными способами, используя расширение TensorBoard (`https://www.tensorflow.org/guide/summaries_and_tensorboard`). Это расширение поможет

- визуализировать вычислительный граф
- вывести метрики выполнения графа
- показать дополнительные данные при необходимости.

Как и во многих продуктах, включающих множество функций, работе с TensorFlow придется учиться. Тем не менее, он также пользуется значительной поддержкой сообщества, предоставляет доступ к множеству практических руководств, имеет отличную стороннюю поддержку для сетевых курсов и предлагает множество других средств для ускорения обучения. Вы можете начать с учебника по адресу `https://www.tensorflow.org/tutorials/` и ознакомиться с руководством, предлагаемым по адресу `https://www.tensorflow.org/guide/`.

Упрощение TensorFlow с помощью TFLearn

Одна из основных претензий к TensorFlow заключается в том, что программирование одновременно является и низкоуровневым, и порой сложным. Компромисс, который вы получаете с TensorFlow, заключается в том, что вы получаете дополнительную гибкость и контроль, написав больше кода. Однако не всем нужна глубина, которую может обеспечить TensorFlow, поэтому такие пакеты, как TFLearn (`http://tflearn.org/`), что означает TensorFlow Learn, так важны. (На рынке вы можете найти несколько пакетов, пытающихся уменьшить сложность; TFLearn — только один из них.)


ЗАПОМНИ!

TFLearn облегчает работу с TensorFlow несколькими способами.

- Высокоуровневый *интерфейс прикладных программ* (Application Programming Interface — API) помогает получать результаты с меньшим количеством кода.
- Высокоуровневый API уменьшает объем написанного вами стандартного кода.
- Прототипирование происходит быстрее, схоже с функциональностью Caffe2 (описано ранее в этой главе).
- Прозрачность с TensorFlow означает, что вы можете видеть, как работают функции, и использовать их напрямую, не полагаясь на TFLearn.
- Использование вспомогательных функций автоматизирует многие задачи, которые обычно приходится выполнять вручную.
- Использование великолепной визуализации помогает вам с большей легкостью увидеть различные аспекты вашего приложения, включая вычислительную модель.

Вы получаете всю эту функциональность и многое другое, не отказываясь от аспектов, делающих TensorFlow таким замечательным продуктом. Например, у вас по-прежнему есть полный доступ к возможностям TensorFlow по

использованию процессоров, графических процессоров и даже нескольких систем для увеличения вычислительной мощности при решении любой задачи.

Использование Keras для большего упрощения

Keras — это не столько инфраструктура, сколько API (набор спецификаций интерфейса, который вы можете использовать с несколькими инфраструктурами в качестве объединителя). Тем не менее, это, как правило, основа для глубокого обучения, поскольку именно так люди ее используют. Чтобы использовать Keras, вы также должны иметь систему глубокого обучения, такую как TensorFlow, Theano, MXNet или CNTK. Keras на самом деле связан с TensorFlow, что также делает его простым решением для снижения сложности TensorFlow.

СОВЕТ

В этой книге предполагается, что вы используете Keras с Tensor Flow, но знание того, что вы можете использовать Keras с другими инфраструктурами глубокого обучения, является преимуществом. Вот почему эта книга не использует версию Keras, встроенную в TensorFlow, но устанавливает ее отдельно (см. подробнее на `https://medium.com/tensorflow/standardizing-on-kerasguidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a`). Вы можете использовать один и тот же интерфейс с несколькими инфраструктурами, что позволяет использовать нужную инфраструктуру без необходимости учиться работать с ней. Самым большим преимуществом Keras является то, что процесс создания приложений с использованием инфраструктуры глубокого обучения превращается в парадигму, которую большинство людей могут хорошо понять.

Вы не можете разработать приложение любого типа, которое было бы простым в использовании и способным справляться с действительно сложными ситуациями, а также быть при этом гибким. Поэтому Keras не обязательно хорошо справляется со всеми ситуациями. Например, это хороший продукт для использования, когда ваши потребности просты, но не лучший выбор, если вы планируете разрабатывать новый тип нейронной сети.

Сила Keras в способности быстро создавать прототипы без особых хлопот. API не мешает вам, пока он пытается обеспечить гибкость, которая вам может не понадобиться в текущем проекте. Кроме того, поскольку Keras упрощает выполнение задач, вы не можете расширять его так же, как другие продукты, что ограничивает ваши возможности по добавлению функциональности в существующую среду.

ВНИМАНИЕ!

Многие жалуются на неоднозначные иногда сообщения об ошибках Keras. Но Keras частично компенсирует эту проблему сильной поддержкой сообщества. Кроме того, многие из жаловавшиеся на сообщения об ошибках явно пытаются сделать что-то сложное. Помня о быстром прототипировании Keras, не стоит пытаться опробовать проекты, которые могут оказаться слишком сложными для этого продукта.

Получение экземпляра TensorFlow и Keras

Ваша версия Python, поставляемая с Anaconda, не включает экземпляр TensorFlow или Keras; эти продукты придется установить отдельно. Чтобы избежать проблем с интеграцией TensorFlow с инструментами Anaconda, не следуйте инструкциям на `https://www.tensorflow.org/install/pip` для установки продукта с помощью pip. Также не используйте инструкции по установке Keras по адресу `https://keras.io/#installation`. Чтобы убедиться, что ваш экземпляр TensorFlow и Keras доступны в Notebook, следует открыть приглашение Anaconda, а не стандартную командную строку или окно терминала. В противном случае вы не сможете гарантировать, что у вас установлены соответствующие пути. Следующие шаги помогут вам начать установку.

1. **В командной строке Anaconda введите `python --version` и нажмите клавишу <Enter>.**

   Вы видите текущую установленную версию Python, которая должна быть версией 3.6.5 для этой книги, как показано на рис. 4.1. Путь, который вы видите в окне, зависит от вашей операционной системы, в данном случае Windows, но вы можете увидеть другой путь при использовании приглашения Anaconda.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Следующим шагом является создание среды для выполнения кода, основанного на TensorFlow и Keras. Преимущество использования среды заключается в том, что вы поддерживаете первозданную среду для последующего использования с другими библиотеками. Для обеспечения интеграции программного обеспечения с инструментами Anaconda вы используете conda, а не другой продукт среды, такой как virtualenv. Если вы используете такой продукт, как virtualenv, итоговая конфигурация будет работать, но вам придется выполнить множество других шагов, чтобы получить к ней доступ, и эти шаги не рассматриваются в книге. Среды для этой книги — это `DL4Denv`.

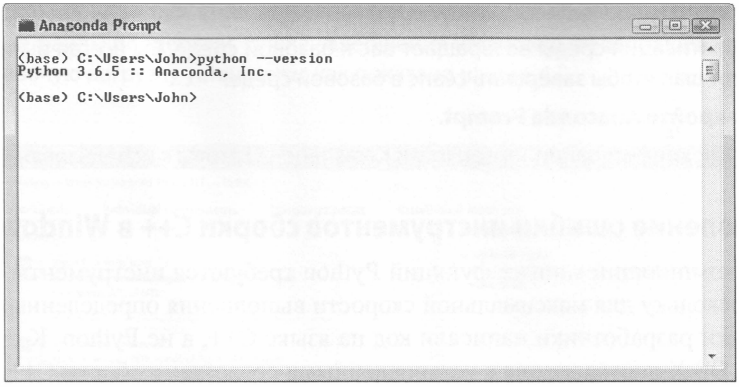


Рис. 4.1. Для установки и проверки версии Python обязательно используйте приглашение Anaconda

2. **Введите `conda create -n DL4Denv python=3 anaconda=5.3.0 tensorflow=1.11.0 keras=2.2.4 nb_conda` и нажмите клавишу <Enter>.**

   Для выполнения этого шага может потребоваться некоторое время, поскольку вашей системе придется загрузить TensorFlow 1.11.0 и Keras 2.2.4 из сетевого источника. После завершения загрузки должна быть создана полная конфигурация. После того, как все необходимые шаги будут выполнены, вы увидите приглашение Anaconda.


СОВЕТ

Может появиться предупреждающее сообщение о наличии более новой версии conda. Это сообщение можно игнорировать (или вы можете обновить conda, используя команду, показанную в предупреждении, если это необходимо). При необходимости введите `Y` и нажмите клавишу <Enter>, чтобы очистить сообщение и продолжить процесс создания.

3. **Введите `conda activate DL4Denv` и нажмите клавишу <Enter>.**

   Приглашение изменится, чтобы показать среду DL4Denv, а не базовую или корневую среду. Любые выполняемые вами задачи будут влиять на среду DL4D, а не на исходную базовую среду.

4. **Введите `python -m pip install --upgrade pip` и нажмите клавишу <Enter>.**

   Этот шаг потребует немного времени, но не так долго, как создание среды. Цель этого шага — убедиться, что у вас установлена самая последняя версия pip, чтобы сработали последующие команды (некоторые из которых указаны в коде книги).

5. **Введите `conda deactivate` и нажмите клавишу <Enter>.**

   Деактивация среды возвращает вас к базовой среде. Вы всегда выполняете этот шаг, чтобы завершать сеанс в базовой среде.

6. **Закройте Anaconda Prompt.**

   Ваши конфигурации TensorFlow и Keras теперь готовы к использованию.

Исправление ошибки инструментов сборки C++ в Windows

Для компиляции многих функций Python требуются инструменты сборки C++, поскольку для максимальной скорости выполнения определенных видов обработки разработчики написали код на языке C++, а не Python. К счастью, Linux и OS X поставляются с установленными средствами сборки C++. Таким образом, вам не нужно делать ничего особенного, чтобы заставить работать команды сборки Python.

Тем не менее, пользователям Windows необходимо установить экземпляр инструментов сборки C++ 14 или более поздней версии, если они еще не установлены. На самом деле среда Notebook довольно требовательна — вам нужен Visual C++ 14 или выше, а не какая-либо версия языка C++ (например, GCC, `https://www.gnu.org/software/gcc/`). Если вы недавно установили Visual Studio или другой продукт Microsoft для разработки, возможно, у вас установлены средства сборки, и вам не нужно устанавливать вторую копию.

В этой книге используются самые современные инструменты, доступные на момент написания, а именно C++ 17. Получение только инструментов сборки не будет стоить вам ничего. Следующие шаги показывают короткий и простой способ получения необходимых инструментов сборки, если у вас еще не установлен C++ 14 или выше.

1. **Загрузите автономный установщик инструментов сборки по адресу `https://aka.ms/vs/15/release/vs_buildtools.exe`.**

   Вы загрузите экземпляр приложения `vs_buildtools.exe`. Попытка использования сетевых инструментов сборки зачастую сопряжены со слишком большим количеством вариантов, и Microsoft, естественно, хочет, чтобы вы купили ее продукт.

2. **Найдите загруженный файл на жестком диске и дважды щелкните на файле `vs_buildtools.exe`.**

   Вы видите диалоговое окно установщика Visual Studio. Прежде чем вы сможете установить инструменты сборки, вы должны сообщить установщику, что именно вы хотите установить.

3. **Щелкните Continue (Продолжить).**

Установщик Visual Studio загружает и устанавливает некоторые дополнительные файлы поддержки. После завершения установки он запрашивает, какую рабочую нагрузку установить, как показано на рис. 4.2.

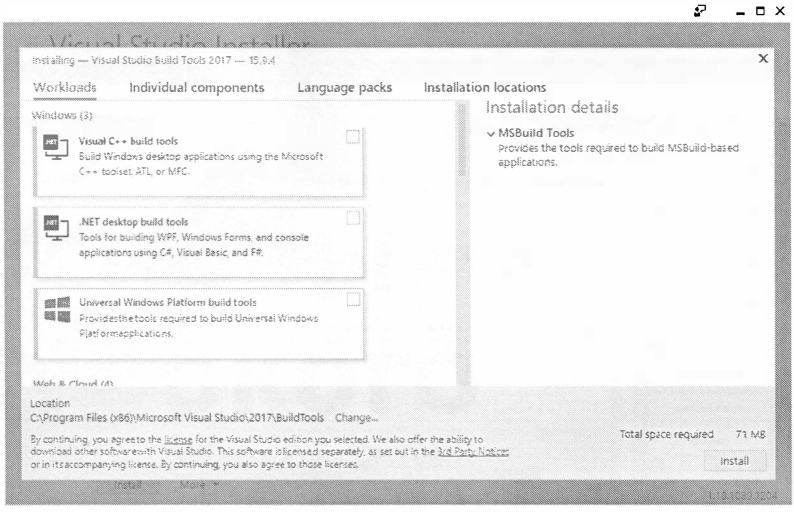


Рис. 4.2. Выберите рабочую нагрузку Visual C++ Build Tools для поддержки настроек Python

4. **Установите флажок Visual C++ Build Tools (Инструменты сборки Visual C++) и щелкните Install (Установить).**

   Вам не нужно устанавливать ничего, кроме стандартных функций. Панель сведений об установке в правой части окна установщика Visual Studio содержит запутанный набор параметров, которые вам не понадобятся для этой книги. Процесс загрузки размером около 1,1 Гб начинается немедленно. Вы можете взять чашку кофе, пока ждете. Окно установщика Visual Studio отображает ход загрузки и установки. В какой-то момент вы видите сообщение о том, что установка прошла успешно.

5. **Закройте окно установщика Visual Studio.**

   Ваш экземпляр Visual C++ Build Tools готов к использованию. После выполнения установки вам может потребоваться перезагрузить систему, особенно если у вас ранее был установлен Visual Studio.

Доступ к новой среде в Notebook

Когда вы открываете Notebook, он автоматически выбирает базовую или корневую среду — стандартную среду для инструментов Anaconda. Но для работы с кодом из этой книги вам нужно получить доступ к среде `DL4Denv`.

Чтобы это произошло, откройте Anaconda Navigator, а не Jupyter Notebook как обычно. В появившемся окне, показанном на рис. 4.3, вы увидите раскрывающийся список Applications On (Приложения на). Выберите DL4Denv. Теперь можно щелкнуть на кнопке Launch (Запуск) на панели Jupyter Notebook, чтобы запустить Notebook с помощью среды `DL4Denv`.

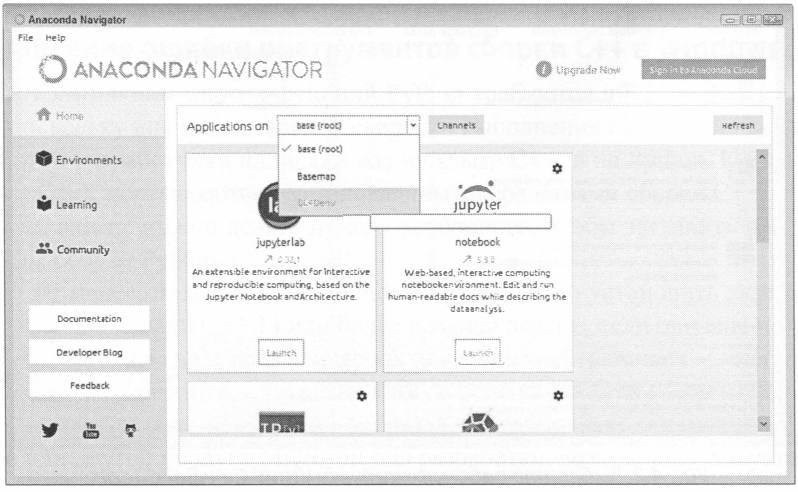


Рис. 4.3. Выберите в Anaconda Navigator среду для использования

2 Основы лубокого обучения

В ЭТОЙ ЧАСТИ...

- Основные операции матричной математики
- Линейная регрессия
- Основы нейронной сети
- Основы глубокого обучения
- Работа с CNN и RNN

Глава 5

Обзор матричной математики и оптимизации

В ЭТОЙ ГЛАВЕ...

- Математические требования для простого глубокого обучения
- Решение скалярных, векторных и матричных математических задач
- Приравнивание обучения к оптимизации

В главе 1 этой книги рассказывалось об основах глубокого обучения и о том, почему это важно сегодня. В главе 2 вы немного углубились в процесс изучения данных с помощью машинного обучения. Ключевым моментом обеих этих глав является то, что ваш компьютер ничего не понимает, но вы можете предоставить ему данные, и он, в свою очередь, может помочь вам понять что-то новое исходя из этих данных. Например, вы можете описать математическую операцию, которая поможет понять ваши данные или сделать из них вывод так, как вы не могли бы иначе. Компьютер становится инструментом для выполнения действительно передовой математики намного быстрее, чем вы могли бы сделать это вручную. Основой этих математических операций является использование определенных структур данных, включая матрицу.

Вы должны понимать, что скалярные, векторные и матричные операции в составе глубокого обучения способны существенно повлиять на то, как вы

просматриваете данные, описывающие мир сегодня. Объединение данных, находящихся в структурах определенных видов с алгоритмами, предназначенными для работы с этими структурами, является основным элементом глубокого обучения. Эта глава поможет вам понять структуры данных, используемые для хранения данных, и то, как вы можете выполнять простые задачи с этими структурами.

До сих пор вы действительно не видели ничего похожего на обучение. Для процесса обучения недостаточно просто иметь структуры данных и соответствующие операции для взаимодействия с ними. Последний раздел этой главы поможет вам установить связь между выполнением этих операций и их ускорением с помощью оптимизации. *Обучение* (learning) — это процесс оптимизации операций, выполняемых с данными: компьютер учится, чтобы избежать ненужных задержек в выполнении анализа, необходимого для решения ваших задач.

ЗАПОМНИ!

Вам не нужно вводить исходный код примеров этой главы вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файле `DL4D_05_Reviewing_Matrix_Math_and_Optimization.ipynb`.

В какой математике вы действительно нуждаетесь

Мир — невероятно сложное место, и попытка представить его с использованием данных и математики делает этот факт вполне очевидным. *Данные* (data) выражают реальный мир как абстракцию, используя числовые или другие значения в качестве средства количественной оценки абстракции. Например, синий цвет может стать значением 1. *Математика* (math) — это средство, с помощью которого вы манипулируете этими значениями, чтобы лучше понять их и распознать шаблоны, которые в противном случае могут быть неясны. Например, вы можете обнаружить, что большая часть людей, живущих в определенной области, предпочитает синий цвет по сравнению с любым другим цветом. Следующие разделы помогут вам понять и данные, и математику с точки зрения искусственного интеллекта, который позволяет вам автоматически взаимодействовать с миром (например, чистя ковер с помощью робота или запрашивая навигационную систему вашего автомобиля, чтобы помочь добраться в место, где вы не бывали ранее).

Работа с данными

Без данных невозможно представить объекты реального мира в форме, которую компьютер может помочь вам понять и контролировать. Компьютер не понимает данные; он просто хранит их и позволяет вам манипулировать ими с помощью математики. Вывод компьютер тоже не понимает. Чтобы результат манипуляции имел смысл, требуется интерпретация человеком. Таким образом, данные начинаются и заканчиваются человеческой интерпретацией реального мира, представленной как абстракция.

При создании данных вы должны обладать некой непротиворечивой мерой абстракции, иначе связь станет невозможной. Например, если один набор данных представляет синий цвет в виде целого числа 1, другой набор данных представляет синий цвет в виде действительного числа 2,0, а третий представляет синий цвет в виде строки `"blue"`, вы не сможете объединить информацию, если не создадите другой набор данных, содержащий одинаковые значения для каждой записи о синем цвете. Поскольку люди непоследовательны, данные также могут быть непоследовательными (даже если они верны). Преобразование значений между наборами данных не меняет того факта, что люди, интерпретирующие его, по-прежнему видят синий цвет, закодированный в абстракции, являющейся данными.

Собрав достаточно данных, вы сможете манипулировать ими таким образом, чтобы компьютер мог представить вам шаблоны, которые вы, возможно, не видели раньше. Как и всегда, компьютер не понимает данных или их интерпретации, или даже того, что он создал шаблон. Математика, определенная невероятно умными учеными, манипулируя данными, находит шаблон, используя математические выражения.

Что у вас есть с точки зрения глубокого обучения, так это человек-интерпретатор, представляющий абстракции данных объектов реального мира, компьютер, выполняющий одну или несколько манипуляций с этими данными, и вывод, который снова требует человеческой интерпретации. *Глубокое обучение*, с точки зрения данной главы, — это просто автоматизация процесса манипулирования данными с использованием тех же методов, которые может использовать и человек, но в сочетании со скоростью, которую может обеспечить только компьютер. Акт обучения означает выяснение, как успешно выполнять манипуляции, чтобы в качестве результатов получить полезные шаблоны.

ЗАПОМНИ!

Автоматизация бесполезна, если она не контролируется, а глубокое обучение обеспечивает этот контроль с помощью матричных вычислений. *Матричное вычисление* (matrix computation) — это серия умножений и суммирований упорядоченных наборов чисел. Вы

должны понимать, как глубокое обучение работает математически, чтобы вы могли

- Развеять любые фантазии о том, что глубокое обучение действует так же, как человеческий мозг
- Определить инструменты, необходимые для создания примеров глубокой нейронной сети.

Создание и работа с матрицей

Гарантии того, что все абстракции, используемые для конкретных объектов реального мира, были одинаковыми, недостаточно для создания содержательной модели. Простое решение о том, что целочисленное значение 1 представляет синий цвет, не обеспечивает необходимую структуру для выполнения математических манипуляций, если только такие манипуляции не относятся к одному значению (*скаляру*). Группа связанных значений может присутствовать в списке (*вектор*), но только если каждое из значений представляет объект одного и того же типа. Например, вы можете создать список цветов, каждый из которых имеет определенное значение. Чтобы быть действительно полезными, данные должны отображаться в форме, которая группирует подобные записи. Такая форма улучшает автоматизированную обработку. Обычно предпочтительной формой является таблица (*матрица*), содержащая значения объектов конкретных типов в столбце и отдельные записи в строках.

Матрицы часто используются в этой книге, поскольку они представляют собой удобное средство обработки сложных записей как единого целого. Матрица свойств в наборе данных Boston может включать в себя всевозможную связанную информацию, такую как цена, количество комнат и характеристики окружающей среды для каждого дома. Фактически, вы можете найти такой *набор данных* (dataset) (файл, содержащий необходимые данные для представления реальных записей) по адресу `https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`. Даже если вы получаете данные в другой форме, процесс импорта, который преобразует их в матрицу, является первым шагом в использовании набора данных для просмотра полезных шаблонов в ходе применения глубокого обучения.


ЗАПОМНИ!

Математика, которая вам нужна, сводится к следующему.

- Математический процесс преобразования всех элементов данных в одинаковую форму.
- Процесс, в том числе математический, для размещения элементов данных в структуре, такой как матрица, чтобы помочь в автоматической обработке данных.

- Математический процесс для манипулирования матрицей, чтобы проявились полезные шаблоны.
- Методология (включая математическую) предоставления результатов моделей для интерпретации человеком.

Скалярные, векторные и матричные операции

Чтобы выполнять полезную работу на языке Python, вам зачастую нужно работать с большими объемами данных, которые поступают в определенных формах. Эти формы имеют странные названия, но их имена довольно важны. Три термина, которые вы должны знать для изучения этой главы.

- **Скаляр** (scalar). Единичный базовый элемент данных. Например, число 2, показанное само по себе, является скаляром.
- **Вектор** (vector). Одномерный массив (по сути, список) элементов данных. Например, массив, содержащий числа 2, 3, 4 и 5, будет вектором. Для доступа к элементам вектора используется отсчитываемый от нуля *индекс*, указывающий на нужный элемент. Элемент с индексом 0 является первым элементом в векторе, и в данном случае равен 2.
- **Матрица** (matrix). Массив двух или более измерений (по сути, таблицы) элементов данных. Например, массив, содержащий числа 2, 3, 4 и 5 в первой строке и 6, 7, 8 и 9 во второй строке, является матрицей. Для доступа к элементам матрицы используется отсчитываемые от нуля индексы строки и столбца. Элемент в строке 0, столбце 0 является первым элементом в матрице, и в данном случае он равен 2.


ЗАПОМНИ!

Глубокое обучение опирается на матрицы. Используемые вами источники данных имеют формат строки и столбца для описания атрибутов конкретного элемента данных. Например, при описании человека, матрица может включать такие атрибуты, как имя, возраст, адрес и номер приобретаемого им каждый год определенного товара. Зная эти атрибуты, вы можете выполнить анализ, способный выдать новые типы информации и помочь вам обобщить информацию о конкретной популяции.

Язык Python сам по себе предоставляет интересный ассортимент функций, но вам все равно придется проделать большую работу для выполнения некоторых задач. Чтобы уменьшить объем выполняемых работ, вы можете

положиться на код, написанный другими людьми и расположенный в библиотеках. В следующих разделах описано, как использовать библиотеку NumPy для выполнения различных задач с матрицами.

Создание матрицы

Прежде чем вы сможете что-либо сделать с матрицей, вы должны создать ее и заполнить данными. Самый простой способ выполнить эту задачу — использовать библиотеку NumPy, которую вы импортируете как `np`, используя следующий код.

```
import numpy as np
```

Чтобы создать простую матрицу, вы просто используете функцию `array` NumPy, как если бы вы использовали вектор, но с определением дополнительных размерностей. *Размерность* — это направление в матрице. Например, двумерная матрица содержит строки (одно направление) и столбцы (второе направление). Вызов `myMatrix = np.array([[1,2,3], [4,5,6], [7,8,9]])` создает матрицу, содержащую три строки и три столбца.

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Обратите внимание, чтобы создать два измерения, вы встраиваете в список контейнеров три списка. Для доступа к конкретному элементу массива вы предоставляете значение индекса строки и столбца, например `myMatrix[0, 0]`, для доступа к первому значению `1`. Используя аналогичную технику, вы можете создавать матрицы с любым количеством измерений. Например, `myMatrix = np.array([[[1,2], [3,4]], [[5,6], [7,8]]])` создает трехмерную матрицу с осями x, y и z, которая выглядит так.

```
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

В данном случае вы встраиваете два списка в два списка контейнеров и в один список контейнеров, который содержит все вместе. В этом случае для доступа к определенному значению вы должны предоставить значения индексов x, y и z. Например, `myMatrix[0, 1, 1]` ссылается на значение `4`.


СОВЕТ

В некоторых случаях вам нужно создать матрицу, которая имеет определенные начальные значения. Например, если вам нужна матрица, с самого начала заполненная единицами, вы можете

использовать функцию ones. Вызов myMatrix = np.ones([4,4], dtype=np.int32) создает матрицу, содержащую четыре строки и четыре столбца, заполненные значениями типа int32, например.

```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

Аналогично, вызов myMatrix = np.ones([4,4,4], dtype=np.bool) создаст трехмерный массив. На этот раз матрица будет содержать логические значения True. Также доступны функции для создания матрицы, заполненной нулями, единичными матрицами и др. Вы можете найти полный список функций создания векторных и матричных массивов по адресу https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html.


ЗАПОМНИ!

Библиотека NumPy поддерживает класс matrix. Класс matrix имеет специальные функции, облегчающие выполнение задач, связанных с матрицей. Мы рассмотрим эти функции далее в этой главе. На данный момент все, что вам действительно нужно знать, это как создать матрицу типа данных matrix. Самый простой способ — сделать вызов, аналогичный тому, который вы используете для функции array, но вместо этого использовать функцию mat, например, myMatrix = np.mat([[1,2,3], [4,5,6], [7,8,9]]), которая производит следующую матрицу.

```
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
```

Вы также можете преобразовать существующий массив в матрицу, используя функцию asmatrix. Используйте функцию asarray для преобразования объекта matrix обратно в форму array.


ВНИМАНИЕ!

Единственная проблема с классом matrix заключается в том, что он работает только с двумерными матрицами. Если вы попытаетесь преобразовать в класс matrix трехмерную матрицу, вы увидите сообщение об ошибке, информирующее, что форма слишком велика, чтобы быть матрицей.

Умножение матриц

Два наиболее распространенных метода умножения матриц — это элемент за элементом и скалярное произведение. Поэлементный подход прост. Следующий код производит поэлементное умножение двух матриц.

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[1,2,3],[4,5,6]])

print(a*b)
```

В ответ вы получаете массив вида, показанного здесь.

```
[[ 1  4  9]
 [16 25 36]]
```

ВНИМАНИЕ!

Обратите внимание, что a и b имеют одинаковую форму: две строки и три столбца. Чтобы выполнить поэлементное умножение, две матрицы должны иметь одинаковую форму. В противном случае, вы увидите сообщение об ошибке, информирующее о том, что формы не являются правильными. Как и с векторами, функция `multiply` также дает поэлементный результат.

ВНИМАНИЕ!

К сожалению, поэлементное умножение может давать неверные результаты при работе с алгоритмами. Во многих случаях вам действительно нужно *скалярное произведение* (dot product), представляющее собой сумму произведений двух числовых последовательностей. Обсуждение на `https://www.mathsisfun.com/algebra/vectors-dot-product.html` рассматривает скалярное произведение и помогает понять, как они могут соответствовать алгоритмам. Вы можете узнать больше о функциях манипуляции линейной алгебры для `numpy` по адресу `https://docs.scipy.org/doc/numpy/reference/routines.linalg.html`.

При выполнении скалярного произведения с матрицей, количество столбцов в матрице a должно соответствовать количеству строк в матрице b. Однако количество строк в матрице a может быть любым, и количество столбцов в матрице b может быть любым, если вы создаете скалярное произведение a на b. Например, следующий код создает правильное скалярное произведение.

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[1,2,3],[3,4,5],[5,6,7]])

print(a.dot(b))
```

Вот что вы получите в этом случае.

```
[[22 28 34]
 [49 64 79]]
```

Обратите внимание, что выходные данные содержат количество строк как в матрице a, и количество столбцов как в матрице b. Так как же все это работает?

Чтобы получить значение, находящееся в выходном массиве по индексу `[0,0]` т.е. 22, вы суммируете значения `a[0,0] * b[0,0]` (что равно 1), `a[0,1] * b[1,0]` (что равно 6) и `a[0,2] * b[2,0]` (что равно 15), что дает значение 22. Другие записи работают точно так же.


СОВЕТ

Преимущество использования класса `numpy matrix` заключается в упрощении решения некоторых задач. Например, умножение работает точно так, как вы ожидаете. Следующий код создает скалярное произведение, используя класс `matrix`.

```
a = np.mat([[1,2,3],[4,5,6]])
b = np.mat([[1,2,3],[3,4,5],[5,6,7]])

print(a*b)
```

Вывод оператора `*` такой же, как при использовании функции `dot` с `array`. Однако, хотя вывод выглядит так же, как и при использовании функции `dot`, он не совсем такой же. Вывод предыдущего кода — это `array`, а вывод этого — `matrix`. В данном примере также указывается, что вы должны знать, используете ли вы объект `array` или `matrix` при выполнении таких задач, как умножение двух матриц.


СОВЕТ

Чтобы выполнить поэлементное умножение с использованием двух объектов `matrix`, необходимо использовать функцию `numpy multiply`.

Расширенные матричные операции

Эта книга знакомит вас со всевозможными интересными матричными операциями, но некоторые из них вы используете часто, именно поэтому они рассматриваются в этой главе подробней. При работе с массивами вы иногда получаете данные в форме, которая не работает с алгоритмом. К счастью, `numpy` поставляется со специальной функцией `reshape`, которая позволяет вам преобразовывать данные в любую нужную форму. Фактически, вы можете использовать ее для преобразования вектора в матрицу, как показано в следующем коде.

```
changeIt = np.array([1,2,3,4,5,6,7,8])

print(changeIt)

print(changeIt.reshape(2,4))

print(changeIt.reshape(2,2,2))
```

Этот код создает следующие выходные данные, которые показывают ход изменений, произведенных функцией `reshape`.

```
[1 2 3 4 5 6 7 8]

[[1 2 3 4]
 [5 6 7 8]]

[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

ЗАПОМНИ!

Начальная форма `changeIt` — это вектор, но использование функции `reshape` превращает его в матрицу. Кроме того, вы можете сформировать матрицу с любым количеством измерений, которые работают с данными. Однако вы должны предоставить форму, которая соответствует требуемому количеству элементов. Например, вызов `changeIt.reshape (2,3,2)` не удастся, поскольку не хватает элементов для представления матрицы этой размерности.

В некоторых формулировках алгоритма вы можете встретиться с двумя важными матричными операциями. Это транспонирование и обращение матрицы. *Транспонирование* (transposition) происходит, когда матрица формы $n \times m$ преобразуется в матрицу $m \times n$ в ходе замены строк столбцами. В большинстве текстов эту операцию обозначают, используя верхний индекс T, как A^T. Эта операция чаще всего используется для умножения, чтобы получить правильные размерности. При работе с `numpy` для выполнения необходимых действий вы используете функцию `transpose`. Например, когда вы начинаете с матрицы, которая имеет две строки и четыре столбца, вы можете транспонировать ее так, чтобы она содержала четыре строки с двумя столбцами в каждой, как показано в этом примере.

```
changeIt = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(np.transpose(changeIt))
```

Выходные данные показывают результат транспонирования.

```
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
```

Обращение матриц (matrix inversion) применяется к матрицам формы $m \times m$, представляющим собой квадратные матрицы с одинаковым количеством строк и столбцов. Эта операция очень важна, поскольку она позволяет немедленно разрешать уравнения, включающие умножение матриц, например, `y = bX`, где

вы должны обнаружить значения в векторе b. Поскольку большинство скалярных чисел (кроме нуля) имеют число, умножение на которое дает значение 1, идея заключается в том, чтобы найти обратную матрицу, умножение на которую приведет к специальной *единичной матрице* (identity matrix). Чтобы увидеть единичную матрицу в numpy, используйте функцию identity, например.

```
print(np.identity(4))
```

Вывод этой функции таков.

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Обратите внимание, что единичная матрица содержит по диагонали все единицы. Найти обратный скаляр довольно просто (скалярное число n имеет обратное значение n^{-1}, равное $1/n$). Что касается матриц, то это совсем другая история. Инверсия матрицы требует довольно большого количества вычислений. Обратная матрица A обозначается как A^{-1}. При работе с numpy для инверсии вы используете функцию linalg.inv. В следующем примере показано, как осуществить инверсию, использовать ее для получения скалярного произведения, а затем сравнить это скалярное произведение с матрицей идентификаторов, используя функцию allclose.

```
a = np.array([[1,2], [3,4]])
b = np.linalg.inv(a)

print(np.allclose(np.dot(a,b), np.identity(2)))
```

Вывод этого кода таков.

```
True
```


ЗАПОМНИ!

Иногда найти обратную матрицу невозможно. Когда матрица не может быть обращена, ее называют *сингулярной матрицей* (singular matrix) или *вырожденной матрицей* (degenerate matrix). Сингулярные матрицы не являются нормой; они довольно редки.

Расширение анализа до тензоров

Проще всего начать рассмотрение тензоров с утверждения, что они являются обобщенной матрицей, способной иметь любое количество измерений. Они могут быть 0-мерными (скаляры), 1-мерными (вектор) или 2-мерными (матрица). Фактически, тензоры могут иметь даже больше размерностей, чем можно себе представить. Тензор имеет такое количество измерений, которое

необходимо для передачи значения некого объекта с использованием данных. Хотя большинство людей рассматривают данные как двумерную матрицу, в которой есть строки, содержащие отдельные объекты, и столбцы, содержащие отдельные элементы данных, которые определяют эти объекты, во многих случаях двумерной матрицы недостаточно.

Например, вам может потребоваться обработать данные, которые содержат элемент времени, что создает двумерную матрицу для каждого наблюдаемого момента. Все эти последовательности двумерных матриц должны храниться в трехмерной структуре, поскольку третье измерение — это время.

ЗАПОМНИ!

Однако тензоры — это не просто причудливая матрица. Они представляют собой математический объект, который находится в структуре, заполненной другими математическими объектами. Все эти объекты взаимодействуют друг с другом так, что преобразование объектов в целом означает, что отдельные тензоры должны следовать определенному правилу преобразования. Динамическая природа тензоров отличает их от стандартных матриц. Каждый тензор в структуре реагирует на изменения любого другого тензора, которое происходит как часть преобразования.

Чтобы рассмотреть работу тензоров в отношении глубокого обучения, представьте, что алгоритму может потребоваться три ввода для работы, как выражается этим вектором.

```
inputs = np.array([5, 10, 15])
```

Это отдельные значения, основанные на отдельном событии. Возможно, они представляют запрос о том, какое моющее средство лучше всего на Amazon. Однако прежде, чем вы сможете ввести эти значения в алгоритм, вы должны взвесить их на основе обучения, выполненного на модели. Другими словами, с учетом моющих средств, купленных большой группой людей, матрица представляет, какое из них на самом деле лучше всего, учитывая конкретные входные данные. Дело не в том, что это моющее средство является лучшим в любой ситуации, просто дело в том, что оно представляет собой лучший вариант при определенных затратах.

Акт взвешивания значений помогает отразить то, что приложение глубокого обучения извлекло из анализа огромных наборов данных. В качестве аргумента вы можете увидеть весовые коэффициенты в матрице, которые следуют как изученные значения.

```
weights = np.array([[.5,.2,-1], [.3,.4,.1], [-.2,.1,.3]])
```

Теперь, когда для входных данных доступно взвешивание, вы можете преобразовать входные данные на основе обучения алгоритма, выполненного в прошлом.

```
result = np.dot(inputs, weights)
```

Вывод таков.

```
[2.5 6.5 0.5]
```

Вывод преобразует исходные данные так, чтобы они теперь отражали эффект обучения. Вектор, `inputs`, является скрытым слоем в нейронной сети, а выходные данные `result` являются следующим скрытым слоем в той же нейронной сети. Преобразования или другие действия, которые происходят на каждом уровне, определяют, как каждый скрытый слой вносит вклад во всю нейронную сеть, которая в этом случае была взвешенной. Последующие главы помогут понять концепции слоев, взвешивания и других действий в нейронной сети. А пока просто учтите, что каждый тензор взаимодействует со структурой, основываясь на действиях любого другого тензора.

Эффективное использование векторизации

Векторизация (vectorization) — это процесс, в котором приложение обрабатывает несколько скалярных значений одновременно, а не по одному. Основной причиной использования векторизации является экономия времени. Во многих случаях процессор включает в себя специальную инструкцию, связанную с векторизацией, например инструкцию SSE в системах x86 (`https://docs.oracle.com/cd/E26502_01/html/E28388/eojde.html`). Вместо выполнения отдельных инструкций в цикле, подход векторизации будет выполнять их как группу, что значительно ускоряет процесс.

При работе с огромными объемами данных векторизация становится важной, поскольку вы выполняете одну и ту же операцию много раз. Все, что вы можете сделать, чтобы организовать процесс вне цикла, заставит код в целом выполняться быстрее. Вот пример простой векторизации.

```
def doAdd(a, b):
    return a + b

vectAdd = np.vectorize(doAdd)

print(vectAdd([1, 2, 3, 4], [1, 2, 3, 4]))
```

Когда вы выполните этот код, вы получите следующий вывод.

```
[2 4 6 8]
```

Функция `vectAdd` обработала все значения одновременно, за один вызов. Следовательно, функция `doAdd`, которая допускает только два скалярных ввода, была расширена, чтобы разрешить четыре ввода одновременно. В общем, векторизация предлагает следующие преимущества.

- Код короче и легче для чтения.
- Сокращение времени отладки из-за меньшего количества строк кода.
- Средства для более точного представления математических выражений в коде.
- Уменьшение количества неэффективных циклов.

Интерпретация обучения как оптимизации

До сих пор в главе данные обсуждались как абстракция, преобразование данных в полезные формы, хранение данных в матрице и основы манипулирования этой матрицей после ее создания. Все это приводит к возможности автоматизировать обработку данных, чтобы вы могли найти полезные шаблоны. Например, пиксель, самый маленький элемент изображения, представляет собой просто последовательность чисел в матрице. Чтобы найти конкретное лицо на изображении, нужно манипулировать этими числами так, чтобы найти конкретные последовательности, которые соответствуют лицу.

ЗАПОМНИ!

Вскоре вы поймете, что нахождение шаблона и его правильная интерпретация требует времени, даже у компьютера и при любой точности. Конечно, время всегда является фактором. Обнаружение того, что преступник вошел в аэропорт через час после этого факта, совершенно бесполезно — раскрытие должно происходить как можно скорее. Чтобы это произошло, манипулирование данными и распознавание образов должно происходить как можно быстрее, что означает оптимизацию процесса. *Оптимизация* (optimization) — это просто поиск способов быстрее выполнить задачу, не теряя или почти не теряя точности.

Обучение (learning) с точки зрения компьютера происходит, когда приложение находит средства для успешного выполнения оптимизации. Вы должны иметь в виду, что компьютерное обучение отличается от обучения человека тем, что в процессе обучения компьютер на самом деле не понимает ничего нового. Компьютер просто может манипулировать данными с большей скоростью и точностью, чтобы определять интересующие модели. В остальной

части этой книги подробно рассматривается концепция оптимизации, но в следующих разделах вы получите краткий обзор того, что означает оптимизация манипуляции.

Функции стоимости

Люди понимают идею стоимости довольно хорошо. Вы идете в один магазин и обнаруживаете, что продукт стоит определенную сумму. Однако вы знаете, что другой магазин продает точно такой же товар за меньшую цену. В обоих случаях товары одинаковы, поэтому вы покупаете товар в том магазине, который продает его дешевле. Тот же принцип стоимости применим к компьютерному обучению. Компьютер может предоставить несколько методов поиска нужных шаблонов, но только один из этих методов будет производить вывод с требуемой точностью в требуемый период времени. Наиболее эффективный метод, который вы в конечном итоге будете использовать, имеет наименьшую *стоимость* (cost).

Например, для решения задачи вам может потребоваться прогнозировать число или класс. Каждую из этих задач можно превратить в стоимость, которую алгоритм глубокого обучения может использовать для определения правильности своего прогноза. Эта задача решается с использованием *функции стоимости* (cost function) (или *функции потерь* (loss function)), которая измеряет разницу между правильным ответом и ответом, предоставленным алгоритмом глубокого обучения. Результатом функции стоимости является разница между правильным значением и прогнозируемым значением в виде числа. Функция стоимости — это то, что действительно движет успехом глубокого обучения, поскольку она определяет то, что изучает алгоритм. Вы должны правильно выбрать функцию стоимости для своей задачи. Вот функции стоимости, которые вы часто используете при глубоком обучении.

- **Среднеквадратичная ошибка.** Возвращает квадрат разности между правильным значением и значением, представленным алгоритмом. Когда разница велика, значение в квадрате еще больше, что подчеркивает ошибку алгоритма.
- **Кросс-энтропийные или логистические потери.** Оценивает ошибки прогнозирования, используя логарифм. Для представления ответов алгоритмы глубокого обучения используют вероятности. (Они выводят не вероятность, но вывод имеет определенную вероятность.) Вероятности основаны на правильности и преобразуются в числовую меру, которая представляет ошибку.

Знание стоимости алгоритма глубокого обучения при прогнозировании результатов является лишь одной частью процесса. Точно так же, как люди

учатся на ошибках, когда узнают о них, глубокое обучение учится с помощью функции стоимости. Стоимость подразумевает нахождение метода, который выполняет задачи оптимальным образом. Слово *оптимум* (optimum) намеренно неточно, поскольку то, что может показаться оптимальным в одной ситуации, может не быть оптимальным в другой. *Оптимальное решение* (optimal solution) — это то решение, которое будет находить требуемые шаблоны за минимальное время с заданной точностью для большого количества элементов данных. Создание метода, работающего с данными, о которых вы знаете, не сложно. То, что вы хотите, — это оптимальный метод для работы с данными, о которых вы не знаете на настоящий момент.

Нисходящая кривая ошибок

Когда человек совершает ошибку, и другой человек замечает ее, он предоставляет обратную связь, чтобы помочь первому человеку понять природу ошибки и правильное ее решение. Одного сеанса обратной связи может быть недостаточно, чтобы помочь человеку исправить ошибку; поэтому может потребоваться повторение обратной связи. Аналогично, автоматизация, обеспечиваемая глубоким обучением, требует коррекции последующими исправлениями.

После обнаружения ошибки автоматизация обеспечивает исправление алгоритмов, выполняющих обработку. С течением времени этот цикл обратной связи улучшает ответы, которые дает решение глубокого обучения, что делает решение более точным в поиске правильных шаблонов. По ходу этого процесса уровень ошибки, измеряемый функцией стоимости, уменьшается, что приводит к построению *нисходящей кривой* (descending curve). Только что описанным процессом управляет функция стоимости, но для внесения реальных изменений нужны другие алгоритмы, такие как оптимизация и исправление ошибок. Функция стоимости сообщает об уровне ошибки только тогда, когда модель глубокого обучения выводит прогноз.

ЗАПОМНИ!

Описанные в этой книге алгоритмы достигают разных видов оптимизации. Градиентный спуск, стохастический градиентный спуск, Momentum, Adagrad, RMSProp, Adadelta и Adam — это все варианты той же концепции оптимизации, которую книга исследует позже. Для исправления ошибок используется другой алгоритм, *обратное распространение* (backpropagation). Функция ошибки осуществляет обратную связь по нейронной сети в виде возвращаемых весов, которые влияют на то, как решение преобразует входные данные для обеспечения правильного вывода.

Обучение в правильном направлении

Градиентный спуск (gradient descent) — это широко используемый подход, позволяющий определить, какие исправления необходимы, чтобы модель глубокого обучения работала лучше при определенной ошибке. Он всегда начинается с начальной конфигурации сети глубокого обучения и передает обратную связь функции стоимости в общую коррекцию для распределения по узлам сети глубокого обучения. Этот процесс требует ряда повторений, пока выходная функция стоимости не окажется в желаемом диапазоне.

Образно говоря, вы можете рассматривать градиентный спуск как капитана корабля, который должен пройти по фарватеру и избежать многочисленных препятствий, таких как рифы или айсберги. Поскольку капитан видит опасность (ошибка, сообщаемая функцией стоимости), он обеспечивает коррекцию курса корабля, что позволяет избежать столкновения. Естественно, капитан передает коррекции экипажу. Экипаж использует эту информацию для управления машиной и рулями корабля, что и является частью роли, которую играет алгоритм обратного распространения (см. главу 7, в которой также подробно рассматриваются внутренние детали сети глубокого обучения).

ЗАПОМНИ!

Основываясь на функции стоимости, сеть также требует оптимизации, чтобы минимизировать ошибку. Однако оптимизация происходит только на учебных данных. К сожалению, идеальная оптимизация учебных данных может привести к переобучению. Признание таких проблем, как переобучение, — вот где возникает творческий аспект глубокого обучения; вы должны оптимизировать, используя учебные данные, но не оптимизировать полностью (переобучение), чтобы результирующая модель также хорошо работала с тестовыми данными. Этот сбалансированный поиск правильного уровня оптимизации является *обобщением* (generalization). Общими стратегиями достижения глубокой оптимизации обучения являются фиксация количества циклов оптимизации или остановка оптимизации, когда вы замечаете, что модель начинает плохо работать с тестовыми данными, которые вы отделяете от обучающих данных (ранняя остановка).

Один интересный момент заключается в том, что серия поправок, обеспечиваемых алгоритмом градиентного спуска, может быть в конце не оптимальной. Определить, как успешно исправить одну ошибку, довольно просто; исправление многих ошибок одновременно может оказаться затруднительным. Как показано на рис. 5.1, во многих случаях алгоритм оптимизации застревает в тупике и не может найти правильный способ улучшения производительности нейронной сети. Эта ситуация является *локальным минимумом* (local minima),

при котором решение, казалось бы, работает оптимально, хотя на самом деле это не так, поскольку дальнейшие коррекции могут продолжать повышать производительность.

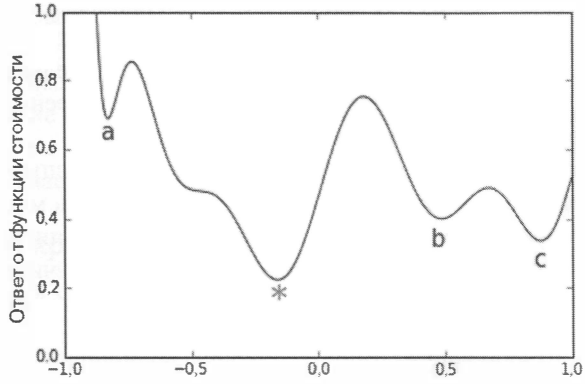


Рис. 5.1. Оптимизация стремится к глобальному минимуму

На рис. 5.1 показан пример процесса оптимизации со многими локальными минимумами (минимальные точки на кривой отмечены буквами), способными сорвать процесс оптимизации и не продолжать снижение до глубокого минимума, отмеченного звездочкой. В процессе оптимизации модели глубокого обучения вы различаете разные результаты оптимизации. У вас может быть *глобальный минимум* (global minimum), хорошая модель, выводящая прогнозы с наименьшей возможной ошибкой, и множество *локальных минимумов* (local minima), решения, казалось бы, обеспечивающие лучшее исправление ошибок, но на самом деле не делающие этого.


ЗАПОМНИ!

Другой проблемой, помимо локальных минимумов, являются седловины, с которыми вы можете встретиться при оптимизации. В *седловинах* (saddle point) у вас нет минимума, но ваша оптимизация резко замедляется, заставляя поверить, что алгоритм достиг минимума. В действительности, седловины представляют собой только паузу в оптимизации. Настаивая на том, чтобы алгоритм двигался в определенном направлении при оптимизации, вы гарантируете, что он может легко избежать седловин и приступить к уменьшению ошибок. Вот способы повышения шансов на получение алгоритмов, которые оптимизированы и работают хорошо.

- Хорошо готовьте учебные данные, чтобы они отражали проблему.
- Выберите разные варианты оптимизации и настраивайте их по мере необходимости.
- Установите другие ключевые характеристики сети глубокого обучения.

Обновление

Обновление нейронной сети новыми весами может принимать одну из двух форм: стохастическая и пакетная. При выполнении *стохастических обновлений* (stochastic updating) каждый ввод генерирует корректировку веса индивидуально. Преимущество этого подхода в уменьшении риска того, что алгоритм может застрять в локальных минимумах. При выполнении *пакетного обновления* (batch updating) ошибка накапливается некоторым образом, и корректировка веса происходит, когда пакет завершен. Преимущество этого подхода в том, что обучение происходит быстрее, поскольку влияние поправок на вес больше.

СОВЕТ

Лучший способ обучения глубокой нейронной сети — попытаться свести к минимуму ошибки всех примеров за один раз. Эта цель не всегда возможна, поскольку данные могут быть слишком большими, чтобы поместиться в память. Пакетные обновления — это лучшая стратегия, возможная во многих случаях, причем размеры пакетов являются максимально возможными для используемого вами оборудования.

Глава 6

Основы линейной регрессии

В ЭТОЙ ГЛАВЕ...

- Выполнение различных действий с переменными
- Работа с вероятностями
- Какие функции использовать
- Обучение с использованием стохастического градиентного спуска (SGD)

Термин *линейная регрессия* (linear regression) может показаться сложным, но это не так, как вы увидите в этой главе. *Линейная регрессия* — это, по сути, прямая линия, проведенная через серию координат *x*/*y*, определяющих местоположение точки данных. Точки данных не всегда могут лежать непосредственно на линии, но линия показывает, где точки данных расположены в идеальном мире линейных координат. Используя линию, вы можете прогнозировать значение *y* (переменная *критерия* (criterion)), учитывая значение *x* (переменная *предиктора* (predictor)). Когда у вас есть только одна переменная предиктора, у вас есть *простая линейная регрессия* (simple linear regression). Когда у вас много предикторов, это *множественная линейная регрессия* (multiple linear regression), которая основана не на прямой, а на плоскости, проходящей через несколько измерений. Глубокое обучение использует входные данные, чтобы найти нелинейную плоскость, которая наиболее правильно пройдет через середину набора точек данных более изощренным способом, чем линейная регрессия. Она действительно имеет некоторые общие ключевые характеристики с линейной регрессией, которая и является

основной темой этой главы. Изучив линейную регрессию и получив полезные идеи, вы сможете впоследствии перенести их на глубокое обучение. В первой части этой главы рассматриваются переменные и методы работы с ними для создания линейной регрессии.

Далее, предположим, что вы создали модель линейной регрессии, но линия разделяет две категории. Точки данных на одной стороне линии — это нечто одно, а точки данных на другой стороне — нечто другое. Нейронная сеть может использовать линейную регрессию для определения вероятности нахождения точки данных на одной стороне линии или на другой. Зная, с каким *объектом* (выраженным точкой данных) вы имеете дело, вы можете *классифицировать* объект, то есть определить, к какой группе объектов он принадлежит.

Суть выполнения всей этой работы заключается в выработке решения задачи. Например, у вас может быть полный список точек данных, и вам нужно знать, к какой группе принадлежит каждая точка данных, что было бы трудной задачей без какой-либо автоматизации. Тем не менее, для создания правильного решения любой конкретной задачи, у вас должны быть правильные данные, что означает определение правильных входных данных или *признаков* (feature) для использования. В третьей части этой главы обсуждается, как выбрать признаки, которые лучше всего ответят на подлежащие рассмотрению вопросы.

Наконец, эта глава использует то, что вы уже применяли для решения простой задачи — стохастический градиентный спуск (Stochastic Gradient Descent — SGD). Объединение всей информации вместе сделает яснее использование линейной регрессии при решении задач.

Объединение переменных

Регрессия имеет долгую историю в различных областях: статистика, экономика, психология, социальные науки и политические науки. Помимо способности выполнять широкий диапазон прогнозов, включающих числовые значения, бинарные и множественные классы, вероятности и данные подсчета, линейная регрессия помогает также понимать групповые различия, моделировать предпочтения потребителей и количественно определять важность признака в модели.

Линейная регрессия, лишенная большинства своих статистических свойств, остается простым, понятным, но эффективным алгоритмом для прогнозирования значений и классов. Быстрая в обучении, простая для объяснения нетехническим людям и простая в реализации на любом языке программирования линейная и логистическая регрессия — это первый выбор большинства практиков с глубоким обучением при построении моделей по сравнению с более

сложными решениями (базовая модель). Их также используют для определения ключевых признаков в задаче, проведении экспериментов и получении представления о создании признаков.

Простая линейная регрессия

Вы должны различать статистические подходы линейной регрессии, подразумевающие построение координат и построение проходящих через них линий на основании алгоритма, который используется для глубокого обучения, чтобы прогнозировать местоположение этой линии на графике. Линейная регрессия работает в ходе объединения числовых признаков при суммировании. Суммирование завершает добавление постоянного числа, *смещения* (bias). Смещение представляет базовую линию прогноза, когда все объекты имеют нулевые значения. Смещение может сыграть важную роль в создании стандартных прогнозов, особенно когда некоторые из ваших признаков отсутствуют (и поэтому имеют нулевое значение). Вот общая формула линейной регрессии.

$$y = \beta X + \alpha$$

В этом выражении `y` является вектором значений ответов. Возможными векторами ответов являются цены на дома в городе или продажи продукта, то есть просто любой числовой ответ, например, мера или количество. `X` — это матрица объектов, используемых для прогнозирования вектора `y`. Матрица `X` содержит только числовые значения. Греческая буква альфа (α) представляет собой смещение, которое является константой, тогда как буква бета (β) представляет собой вектор коэффициентов, который модель линейной регрессии использует со смещением для создания прогноза.

ЗАПОМНИ!

Использование греческих букв альфа и бета в регрессии распространено настолько широко, что большинство практиков называют вектор коэффициентов *бета-регрессией* (regression beta).

Вы можете понимать это выражение по-разному. Для простоты вы можете представить, что `X` на самом деле состоит из одного признака (описанного в статистической практике как *предиктор*), поэтому вы можете представить его как вектор по имени `x`. Когда доступен только один предиктор, вычисление представляет собой *простую линейную регрессию*. Теперь, когда у вас есть более простая формулировка, ваши знания алгебры и геометрии со средней школы говорят вам, что формула `y = bx + a` — это линия на координатной плоскости, состоящей из оси `x` (*абсцисса*) и оси `y` (*ордината*).

Переход к множественной линейной регрессии

Мир редко предлагает задачи, имеющие только один признак. При прогнозировании цен на жилье, вы должны учитывать все виды запросов, таких как соседство и количество комнат в доме. В противном случае люди могут решать большинство задач и без использования автоматизации, такой как глубокое обучение. Когда у вас есть более одного объекта (*множественная линейная регрессия*), вы больше не можете использовать простую координатную плоскость, состоящую из осей x и y. Пространство теперь охватывает несколько измерений, причем каждое измерение является элементом. Теперь ваша формула стала сложней. Она состоит из нескольких значений x, каждое из которых имеет свою собственную бета. Например, если у вас есть четыре объекта (т.е. пространство является четырехмерным), формула регрессии, как показано в матричной форме, будет такова.

$$y = x_1b_1+x_2b_2+x_3b_3+x_4b_4+a$$

Эта сложная формула для многомерного пространства, является уже не линией, а плоскостью, имеющей столько же измерений, сколько и пространство. Это *гиперплоскость*, и ее поверхность определяет значения ответа для каждой возможной комбинации значений в измерениях элемента.

Это обсуждение рассматривает регрессию в ее геометрическом значении, но вы также можете рассматривать ее как большое взвешенное суммирование. Вы можете разделить ответ на множество частей, каждая из которых относится к признаку и вносит свой вклад в определенную часть. Геометрический смысл особенно полезен для обсуждения свойств регрессии, но взвешенное значение суммирования помогает вам лучше понять практические примеры. Например, если необходимо прогнозировать расходы на рекламу, вы можете использовать регрессию и создать такую модель.

$$\texttt{sales} = \texttt{advertising}*b_{adv} + \texttt{shops}*b_{shop} + \texttt{price}*b_{price} + a$$

В этой формуле продажи (sales) — это сумма расходов на рекламу, количество распространяющих продукт магазинов и цена продукта. Вы можете быстро прояснить линейную регрессию, объяснив ее компоненты. Во-первых, у вас есть смещение (константа a), действующее как отправная точка. Во-вторых, у вас есть три значения признаков, каждое из которых выражено в разном масштабе (реклама — это большие деньги, цена — достаточно доступные средства, а магазины — положительное число), и каждое из которых пересчитывается с помощью соответствующего коэффициента бета.

Каждый коэффициент бета представляет числовое значение, описывающее степень отношения к ответу. У него также есть знак, демонстрирующий влияние изменений признаков. Когда коэффициент бета близок к нулю, влияние

признака на ответ является слабым, но если его значение далеко от нуля, положительное или отрицательное, эффект будет значительным, и признак важен для регрессионной модели.


СОВЕТ

Чтобы получить оценку целевого значения, вы масштабируете каждый коэффициент бета до меры признака. Высокое значение бета в большей или меньшей степени влияет на ответ, в зависимости от масштаба признака. Хорошей привычкой является стандартизация признаков (в результате вычитания среднего значения и деления на стандартное отклонение), чтобы избежать заблуждений при высоких значениях бета для мелкомасштабных объектов и сравнения различных коэффициентов бета. Полученные значения бета сопоставимы, что позволяет определить, какие из них оказывают наибольшее влияние на ответ (те, которые имеют наибольшее абсолютное значение).

Если бета положительна, увеличение признака увеличит ответ, а уменьшение признака уменьшит ответ. И наоборот, если бета отрицательна, ответ будет действовать противоположно признаку: когда один увеличивается, другой уменьшается. Каждая бета в регрессии представляет влияние.

Включение градиентного спуска

Используя алгоритм градиентного спуска, рассмотренный далее в этой главе, линейная регрессия может найти лучший набор коэффициентов бета (и смещения), чтобы минимизировать функцию стоимости, заданную квадратом разницы между прогнозами и действительными значениями.

$$J(w) = \frac{1}{2n}\sum(Xw - y)^2$$

Эта формула сообщает вам стоимость J как функцию от w, вектора коэффициентов линейной модели. Стоимость — это сумма квадратов разностей значений ответа и прогнозируемых значений (произведение Xw), деленная два раза на количество наблюдений (n). Алгоритм стремится найти минимально возможные значения решения для разницы между реальными целевыми значениями и прогнозами, полученными из линейной регрессии.

Графически, результат оптимизации можно выразить в виде вертикальных расстояний между точками данных и линией регрессии. Линия регрессии хорошо представляет переменную ответа, когда расстояния малы, как показано на рис. 6.1 (у простой линейной регрессией слева и множественной линейной регрессией справа). Если вы суммируете квадраты расстояний (длина линии, соединяющей точку данных с линией регрессии на рисунке), сумма всегда

будет минимально возможной при правильном расчете линии регрессии. (Никакая другая комбинация бета не приведет к меньшей ошибке.)

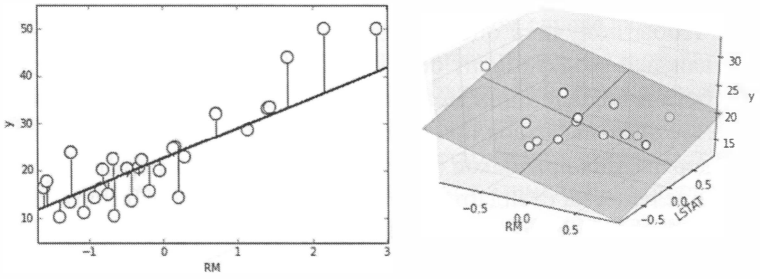


Рис. 6.1. Пример визуализации ошибок для линии и плоскости регрессии

В статистике практики зачастую указывают оценку решения линейной регрессии на основе матричного исчисления (это *решение в аналитическом виде*). Использование такого подхода не всегда возможно, и вычисления очень медленны, когда матрица ввода велика. При глубоком обучении вы получаете те же результаты, используя *оптимизацию градиентного спуска* (Gradient Descent Optimization — GDO), обрабатывающую большие объемы данных проще и быстрее, оценивая, таким образом, решение из любой входной матрицы.

Линейная регрессия в действии

В следующем примере кода Python используется набор данных Boston из библиотеки Scikit-learn, чтобы попытаться сделать прогноз цен на жилье в Бостоне с помощью линейной регрессии. Пример также пытается определить, какие переменные больше влияют на результат. Помимо проблем вычисления, стандартизация предикторов оказывается весьма полезной, если вы хотите определить влиятельные переменных.

```
from sklearn.datasets import load_boston
from sklearn.preprocessing import scale
boston = load_boston()
X, y = scale(boston.data), boston.target
```

Класс регрессии библиотеки Scikit-learn является частью модуля `linear_model`. Поскольку вы ранее масштабировали переменные X, вам не нужно каких либо других приготовлений или специальных параметров при использовании этого алгоритма.

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X, y)
```

Теперь, когда алгоритм подобран, вы можете использовать метод `score`, чтобы сообщить о метрике R^2.

```
print('R2 %0.3f' % regression.score(X, y))
```

```
R2 0.741
```

НЕМНОГО ПОДРОБНЕЙ ОБ R^2

Метрика R^2, известная также как *коэффициент детерминации* (coefficient of determination), является мерой в диапазоне от 0 до 1. Она показывает, что использование модели регрессии лучше в прогнозировании ответа, чем использование простого среднего. Коэффициент детерминации получен из статистической практики и напрямую связан с суммой квадратов ошибок. Вы также можете понимать R^2 как количество информации, объясняемой моделью (такое же, как квадратичная корреляция), поэтому получение значения близкого к 1 означает возможность объяснить большую часть данных с использованием модели.

Расчет R^2 по тому же набору данных, который использовался для обучения, является обычным в статистике. В науке о данных и глубоком обучении вам всегда лучше тестировать результаты на данных, которые не используются для обучения. Сложные алгоритмы способны запоминать данные, а не учиться на них. В определенных обстоятельствах эта проблема также может возникать при использовании более простых моделей, таких как линейная регрессия.

Чтобы понять, что движет оценками в модели множественной регрессии, вам нужно взглянуть на атрибут `coefficients_`, представляющий собой массив, содержащий коэффициенты бета регрессии. Выведя атрибут `boston.DESCR`, вы сможете понять ссылку на переменную.

```
print([a + ':' + str(round(b, 1)) for a, b in
       zip(boston.feature_names, regression.coef_)])
```

```
['CRIM:-0.9', 'ZN:1.1', 'INDUS:0.1', 'CHAS:0.7',
 'NOX:-2.1', 'RM:2.7', 'AGE:0.0', 'DIS:-3.1',
 'RAD:2.7', 'TAX:-2.1', 'PTRATIO:-2.1',
 'B:0.9', 'LSTAT:-3.7']
```

Переменная `DIS`, содержащая взвешенные расстояния до пяти центров занятости, имеет наибольшее абсолютное изменение единиц. В сфере недвижимости, дом находящийся слишком далеко от интересов людей (например, работа),

дешевле. Переменные AGE или INDUS, представляющие собой пропорции, описывающие возраст здания и наличие в этом районе нетрадиционных видов деятельности, напротив, уже не так влияют на результат; абсолютное значение их коэффициентов бета намного ниже.

ЗАПОМНИ!

Вы можете удивиться, почему в примерах главы не используются Keras и TensorFlow. Использование этих библиотек возможно, но пакеты глубокого обучения лучше подходят для решений глубокого обучения. Использование их для более простых моделей означает слишком сложное решение. Библиотека Scikit-learn предлагает простые и понятные реализации моделей линейной регрессии, которые помогут вам лучше понять, как работают эти алгоритмы.

Смешивание типов переменных

С эффективным, но простым инструментом линейной регрессии возникает довольно много проблем. Иногда, в зависимости от используемых данных, проблем больше, чем. Лучший способ определить, будет ли работать линейная регрессия, — это использовать алгоритм и проверить его эффективность на ваших данных.

Моделирование ответов

Линейная регрессия может моделировать ответы только в виде количественных данных. Когда вам нужно моделировать в качестве ответа категории, приходится обращаться к логистической регрессии. При работе с предикторами лучше использовать непрерывные числовые переменные; хотя к качественным категориям вы можете подобрать как порядковые числа, так и некоторые преобразования.

Качественная переменная может выражать цветовую характеристику, такую как цвет продукта или признак, указывающий на профессию человека. У вас есть несколько вариантов преобразования качественной переменной с использованием такой техники, как двоичное кодирование (наиболее распространенный подход). При создании качественной бинарной переменной вы создаете столько признаков, сколько в ней классов. Каждый признак содержит нулевые значения, если только его класс не отображается в данных, тогда он принимает значение, равное единице. Это *унитарное кодирование* (one-hot encoding). Простой пример на языке Python с использованием модуля предварительной обработки Scikit-learn показывает, как выполнить унитарное кодирование.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
lbl = LabelEncoder()
enc = OneHotEncoder()
qualitative = ['red', 'red', 'green', 'blue',
               'red', 'blue', 'blue', 'green']
labels = lbl.fit_transform(qualitative).reshape(8,1)
print(enc.fit_transform(labels).toarray())
```

```
[[ 0. 0. 1.]
 [ 0. 0. 1.]
 [ 0. 1. 0.]
 [ 1. 0. 0.]
 [ 0. 0. 1.]
 [ 1. 0. 0.]
 [ 1. 0. 0.]
 [ 0. 1. 0.]]
```

В этом случае вы видите три столбца: синий (`blue`), зеленый (`green`) и красный (`red`). Обратите, например, внимание на то, что в элементе массива `[0, 2]` вы видите значение `1.`, что соответствует значению красного цвета в этой позиции. Теперь посмотрите на исходный массив, где вы видите, что `qualitative[0]` действительно `'red'`.

Моделирование признаков

Когда в статистике, при решении линейной регрессии с использованием в аналитического вида, вы хотите получить двоичную переменную из категориальной, вы преобразуете все уровни, кроме одного, поскольку вы используете формулу вычисления обратной матрицы, имеющую довольно много ограничений. В глубоком обучении вы используете градиентный спуск, поэтому вместо этого вы преобразуете все уровни.

Если в матрице данных отсутствуют данные, и вы не исправите это должным образом, модель перестанет работать. Следовательно, вам необходимо вычислить отсутствующие значения (например, заменив их средним значением, вычисленным из самого признака). Другое решение заключается в использовании для пропущенного случая нулевого значения или создании дополнительной двоичной переменной, единичные значения которой указывают на пропущенные значения признака. Кроме того, *выбросы* (значения вне нормального диапазона) нарушают линейную регрессию, поскольку модель пытается минимизировать квадратичное значение ошибок (*остатки* (residual)). Выбросы имеют большие остатки, что заставляет алгоритм сосредоточиться на них больше, чем на обычных точках.

Работа со сложными отношениями

Наибольшее ограничение линейной регрессии заключается в том, что модель является суммой независимых членов, поскольку каждый признак стоит в суммировании отдельно, умноженный только на свой собственный коэффициент бета. Эта математическая форма идеально подходит для выражения ситуации, в которой признаки независимы. Например, возраст человека и цвет глаз не связаны между собой, поскольку они не влияют друг на друга. Таким образом, вы можете считать их независимыми членами, и в сумме регрессии их разделение имеет смысл.

Сравните несвязанные члены со связанными членами. Например, возраст человека и цвет волос связаны, поскольку у стариков волосы седые. Когда вы помещаете эти признаки в регрессионное суммирование, это похоже на суммирование одной и той же информации. Из-за этого ограничения вы не можете определить, как представить влияние комбинаций переменных на результат. Другими словами, вы не можете представлять сложные ситуации со своими данными. Поскольку модель состоит из простых комбинаций взвешенных признаков, на ее прогнозы больше влияет смещение, чем дисперсия. Фактически, после подбора наблюдаемых значений, решение, предлагаемое линейными моделями, всегда представляет собой пропорционально измененный набор признаков.

К сожалению, вы не можете точно представить некоторые отношения между ответом и признаком, используя пропорционально измененный набор признаков. Во многих случаях ответ зависит от признаков нелинейным образом: некоторые значения признаков действуют как препятствия, после чего ответ внезапно увеличивается или уменьшается, усиливается или ослабляется или даже обращается. В качестве примера рассмотрим, как люди увеличиваются в росте с детства. Если наблюдать в определенном возрастном диапазоне, соотношение между возрастом и ростом будет близким к линейному: рост ребенка пропорционален возрасту. Тем не менее, некоторые дети растут больше (общий рост), а некоторые растут быстрее (рост за определенное время). Это наблюдение имеет место, когда вы ожидаете, что линейная модель найдет средний ответ. Однако после определенного возраста дети перестают расти, и рост остается постоянным на протяжении всей жизни, медленно уменьшаясь в старшем возрасте. Очевидно, что линейная регрессия не может охватить такие нелинейные отношения. (В конце концов, вы можете представить его как своего рода параболу.)

Поскольку отношение между целью и каждой переменной-предиктором основано на одном коэффициенте, у вас нет способа представить сложные отношения, такие как парабола (уникальное значение x, максимизирующее или

минимизирующее ответ), экспоненциальный рост или более сложная нелинейная кривая, если вы не улучшите признаки.

Самый простой способ моделирования сложных отношений — применение математических преобразований предикторов с использованием полиномиального разложения. *Полиномиальное разложение* (polynomial expansion), когда дана определенная степень *d*, создает степени каждого признака вплоть до степени *d* и *d* комбинаций всех членов. Например, если вы начинаете с простой линейной модели, такой как следующая

```
y = b1x1 + b2x2 + a
```

а затем используете полиномиальное разложение второй степени, и эта модель становится такой

```
y = b1x1 + b2x2 + a + b3x1**2+b4x2**2+b5x1x2
```

Вы делаете дополнение к исходной формуле (разложение), используя степени и комбинации существующих предикторов. По мере роста степени разложения полинома растет количество производных членов.


ЗАПОМНИ!

При использовании полиномиального разложения вы начинаете размещать переменные по отношению друг к другу. Это именно то, что нейронные сети и глубокое обучение делают в другом масштабе; они связывают каждую переменную друг с другом.

В следующем примере кода Python для проверки эффективности метода используется набор данных Boston. В случае успеха, разложение полинома будет отслеживать нелинейные отношения в данных, для которых требуется кривая, а не линия, для правильного прогнозирования и преодоления любых трудностей в прогнозировании за счет увеличения количества предикторов.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

pf = PolynomialFeatures(degree=2)
poly_X = pf.fit_transform(X)
X_train, X_test, y_train, y_test = (
    train_test_split(poly_X, y, test_size=0.33, random_state=42))

from sklearn.linear_model import Ridge
reg_regression = Ridge(alpha=0.1, normalize=True)
reg_regression.fit(X_train,y_train)
print ('R2: %0.3f'
    % r2_score(y_test,reg_regression.predict(X_test)))

R2: 0.819
```

СОВЕТ

Поскольку масштабы признаков увеличиваются при увеличении степени, хорошей практикой является стандартизация данных после полиномиального разложения.

Полиномиальное разложение не всегда обеспечивает преимущества, продемонстрированные в предыдущем примере. Расширяя количество признаков, вы уменьшаете смещение прогнозов за счет возможного переобучения.

Переход на вероятности

До настоящего времени в этой главе рассматривались только регрессионные модели, которые выражают числовые значения в качестве результатов обучения на данных. Но большинство задач требуют классификации. В следующих разделах обсуждается, как можно получать и числовые, и классификационные выводы.

Обеспечение бинарного ответа

Решение проблемы, связанной с бинарным ответом (модель должна выбирать из двух возможных классов), заключалось бы в кодировании вектора ответа в виде последовательности единиц и нулей (или положительных и отрицательных значений). Следующий код Python доказывает как выполнимость, так и пределы использования двоичного ответа.

```
import numpy as np

a = np.array([0, 0, 0, 0, 1, 1, 1, 1])
b = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(8,1)
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(b,a)
print (regression.predict(b)>0.5)
```

```
[False False False False True True True True]
```

В статистике линейная регрессия не может решить проблемы классификации, поскольку ее осуществление нарушает ряд статистических допущений. Таким образом, для статистики использование регрессионных моделей при классификации является в основном теоретической, а не практической задачей. В глубоком обучении проблема с линейной регрессией заключается в том, что она является линейной функцией, пытающейся минимизировать ошибки прогнозирования; поэтому, в зависимости от наклона вычисляемой линии, она может и не решить задачу с данными.

Когда линейная регрессия используется для прогнозирования двух представляющих классы значений, таких как 0 и +1, она пытается вычислить линию, которая представляет результаты, близкие к целевым значениям. В некоторых случаях, даже если результаты являются точными, выходные данные слишком далеки от целевых значений, что вынуждает корректировать линию регрессии для минимизации суммарных ошибок. Коррекция приводит к меньшему количеству ошибок отклонения, но к большему количеству ошибочно классифицированных случаев.

Линейная регрессия не дает приемлемых результатов, когда приоритетом является точность классификации, как показано на рис. 6.2 слева. Поэтому она не будет удовлетворительно работать во многих задачах классификации. Линейная регрессия лучше всего работает на последовательности числовых оценок. Но для задач классификации вам понадобится более подходящая мера, например, вероятность принадлежности к классу.

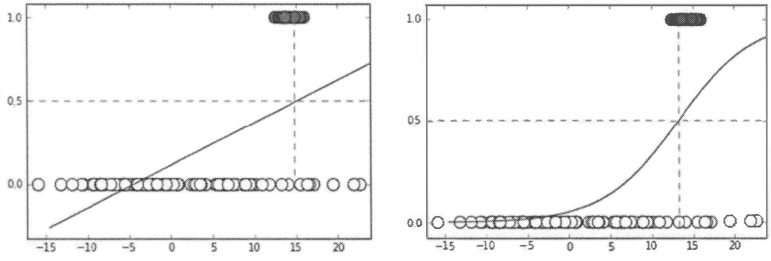


Рис. 6.2. Вероятности не так хорошо работают с прямой линией, как с сигмоидной кривой

Преобразование числовых оценок в вероятности

Благодаря следующей формуле вы можете преобразовать числовые оценки линейной регрессии в вероятности, более подходящие для описания соответствия класса наблюдению.

$$p(y = 1) = \frac{\exp(r)}{(1 + \exp(r))}$$

В этой формуле целью является вероятность того, что ответ y будет соответствовать классу 1. Переменная r — это *результат регрессии* (regression result), сумма переменных, взвешенных по их коэффициентам. Экспоненциальная функция $\exp(r)$ соответствует числу Эйлера e, возведенному в степень r. Линейная регрессия, использующая эту формулу преобразования (*связующую функцию* (link function)) для преобразования ее результатов в вероятности, является логистической регрессией.

Логистическая регрессия (logistic regression) (показанная справа на рис. 6.2) такая же, как и линейная регрессия, за исключением того, что данные y содержат целые числа, указывающие класс относительно наблюдения. Итак, используя набор данных Boston из модуля `datasets` библиотеки Scikit-learn, вы можете попытаться выяснить, что делает дома в некой области чрезмерно дорогими (медианные значения >= 40).

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

binary_y = np.array(y >= 40).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X,
           binary_y, test_size=0.33, random_state=5)
logistic = LogisticRegression()
logistic.fit(X_train,y_train)
from sklearn.metrics import accuracy_score
print('In-sample accuracy: %0.3f' %
      accuracy_score(y_train, logistic.predict(X_train)))
print('Out-of-sample accuracy: %0.3f' %
      accuracy_score(y_test, logistic.predict(X_test)))

In-sample accuracy: 0.973
Out-of-sample accuracy: 0.958
```

В этом примере данные разделяются на наборы для обучения и тестирования, что позволяет проверить эффективность модели логистической регрессии для данных, которые модель не использовала для обучения. Полученные коэффициенты сообщают вам вероятность того, что определенный класс окажется в целевом классе (который является любым классом, закодированным с использованием значения 1). Если коэффициент увеличивает вероятность, он будет иметь положительный коэффициент; в противном случае коэффициент будет отрицательным.

```
for var,coef in zip(boston.feature_names,
                    logistic.coef_[0]):
        print ("%7s : %7.3f" %(var, coef))

   CRIM :  -0.006
     ZN :   0.197
  INDUS :   0.580
   CHAS :  -0.023
    NOX :  -0.236
     RM :   1.426
    AGE :  -0.048
    DIS :  -0.365
    RAD :   0.645
    TAX :  -0.220
```

```
PTRATIO :  -0.554
      B :   0.049
  LSTAT :  -0.803
```

Читая результаты на экране, вы можете увидеть, что в Бостоне преступность (CRIM) оказывает некоторое влияние на цены. Тем не менее, уровень бедности (LSTAT), удаленность от работы (DIS) и загрязнение окружающей среды (NOX) оказывают гораздо большее влияние. Более того, в отличие от линейной регрессии, логистическая регрессия не просто выводит результирующий класс (в данном случае 1 или 0), но также оценивает вероятность того, что наблюдение является частью одного из двух классов.

```
print('\nclasses:',logistic.classes_)
print('\nProbs:\n',logistic.predict_proba(X_test)[:3,:])

classes: [0 1]

Probs:
 [[ 0.39022779 0.60977221]
  [ 0.93856655 0.06143345]
  [ 0.98425623 0.01574377]]
```

В этой небольшой выборке только в первом случае вероятность жилья оказаться дорогим составляет 61%. Когда вы выполняете прогнозы с использованием этого подхода, вы также знаете вероятность того, что ваш прогноз является точным, и действуете соответствующим образом, выбирая прогнозы только с подходящим уровнем точности. (Например, вы можете выбрать только те прогнозы, вероятность которых превышает 80%.)

Прогноз правильных признаков

Может показаться, что наличие множества признаков решает задачу глубокого обучения для полного понимания проблемы. Тем не менее, просто наличие признаков ничего не решает; для решения задачи вам нужны правильные признаки. В следующих разделах обсуждается, как правильно выбрать признаки при выполнении задач глубокого обучения.

Определение результата несовместимых признаков

Если вы не используете перекрестную проверку, меры ошибок, такие как R^2, могут вводить в заблуждение, поскольку количество признаков может легко их увеличить, даже если признак не содержит соответствующей информации.

В следующем примере показано, что происходит с метрикой R^2, когда вы добавляете только случайные признаки.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
X_train, X_test, y_train, y_test = train_test_split(X,
               y, test_size=0.33, random_state=42)
check = [2**i for i in range(8)]
for i in range(2**7+1):
    X_train = np.column_stack((X_train,np.random.random(
        X_train.shape[0])))
    X_test = np.column_stack((X_test,np.random.random(
        X_test.shape[0])))
    regression.fit(X_train, y_train)
    if i in check:
        print ("Random features: %i -> R2: %0.3f" % (i,
            r2_score(y_train,regression.predict(X_train))))
```

```
Random features: 1 -> R2: 0.739
Random features: 2 -> R2: 0.740
Random features: 4 -> R2: 0.740
Random features: 8 -> R2: 0.743
Random features: 16 -> R2: 0.746
Random features: 32 -> R2: 0.762
Random features: 64 -> R2: 0.797
Random features: 128 -> R2: 0.859
```

То, что кажется улучшенной способностью к прогнозированию, на самом деле просто иллюзия. Вы можете узнать о произошедшем, выполнив набор тестов и обнаружив, что производительность модели снизилась.

```
regression.fit(X_train, y_train)
print ('R2 %0.3f'
    % r2_score(y_test,regression.predict(X_test)))
# Обратите внимание, что результат R2 может меняться от запуска
# к запуску из-за случайного характера эксперимента
R2 0.474
```

Избежание переобучения с использованием выборки и регуляризации

Регуляризация (regularization) — это эффективное, быстрое и простое в реализации решение, когда у вас много признаков, и вы хотите уменьшить дисперсию оценок из-за мультиколлинеарности между предикторами. Это также может помочь, если в ваших данных есть выбросы и шум. Регуляризация работает за счет добавления штрафа к функции стоимости. Штраф представляет собой суммирование коэффициентов. Если коэффициенты возводятся в квадрат

(чтобы положительные и отрицательные значения не могли взаимно компенсировать друг друга), это *регуляризация L2* (известная также как *Ридж* (Ridge)). Когда вы используете абсолютное значение коэффициента, это *регуляризация L1* (известная также как *Лассо* (Lasso)).

Однако регуляризация не всегда работает идеально. Регуляризация *L2* сохраняет все признаки в модели и уравновешивает вклад каждого из них. В решении *L2*, если две переменные хорошо коррелируют, каждая вносит одинаковый вклад в решение частично, тогда как без регуляризации их общий вклад был бы распределен неравномерно.

В качестве альтернативы, *L1* выводит сильно коррелированные признаки из модели, обнуляя их коэффициент и предлагая реальный выбор среди признаков. Фактически, обнуление коэффициента аналогично исключению признака из модели. Когда мультиколлинеарность высока, выбор предиктора для обнуления становится немного случайным, и, в зависимости от вашего выбора, вы можете получить различные решения, характеризующиеся различными исключенными признаками. Такая нестабильность решения может создавать неудобства, делая решение *L1* не идеальным.

СОВЕТ

Выход был найден за счет создания разных решений, основанных на регуляризации *L1*, и последующего сравнения поведения коэффициентов между решениями. В этом случае алгоритм выбирает только стабильные коэффициенты (те, которые редко обнуляются). Вы можете прочитать больше об этой технике на веб-сайте Scikit-learn по адресу `https://scikit-learn.org/0.15/auto_examples/linear_model/plot_sparse_recovery.html`. Следующий пример — модификация примера полиномиального разложения с использованием регуляризации *L2* (регрессия Ридж) и уменьшения влияния избыточных коэффициентов, создаваемых процедурой разложения.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

pf = PolynomialFeatures(degree=2)
poly_X = pf.fit_transform(X)
X_train, X_test, y_train, y_test =
    train_test_split(poly_X,
                     y, test_size=0.33, random_state=42)

from sklearn.linear_model import Ridge
reg_regression = Ridge(alpha=0.1, normalize=True)
reg_regression.fit(X_train,y_train)
```

```
print ('R2: %0.3f'
    % r2_score(y_test,reg_regression.predict(X_test)))
```

```
R2: 0.819
```

Изучение по одному примеру за раз

Поиск правильных коэффициентов для линейной модели — это вопрос времени и памяти. Однако системе иногда не хватает памяти для хранения огромного набора данных. В этом случае вы должны прибегнуть к другим средствам, таким как обучение по одному примеру за раз, а не загрузка в память их всех. Этот подход к обучению рассматривается в следующих разделах.

Использование градиентного спуска

Градиентный спуск находит правильный способ минимизировать функцию стоимости по одной итерации за раз. После каждого шага он проверяет все суммированные ошибки модели и обновляет коэффициенты, чтобы сделать ошибку еще меньше во время следующей итерации. Эффективность этого подхода основывается на рассмотрении всех примеров в выборке. Недостатком этого подхода является то, что вы должны загрузить все данные в память.

К сожалению, вы не всегда можете хранить все данные в памяти, поскольку некоторые наборы данных огромны. Кроме того, для построения эффективных моделей обучение с использованием простых учеников требует больших объемов данных (большее количество данных помогает правильно устранить неоднозначность мультиколлинеарности). Получение и хранение фрагментов данных на жестком диске всегда возможно, но это неосуществимо из-за необходимости выполнять матричное умножение, требующее обмена данными с диском для выборки строк и столбцов. Ученые, работавшие над этой проблемой, нашли эффективное решение. Вместо обучения на всех данных после их просмотра целиком (*итерации*), алгоритм изучает за раз по одному примеру, выбранному из хранилища с использованием последовательного доступа, а затем переходит к обучению на следующем примере. Когда алгоритм изучил все примеры, он начинает с начала, если только он не удовлетворяет некоторому критерию остановки (например, выполнению заранее определенного количества итераций).

Чем отличается SGD

Стохастический градиентный спуск (Stochastic gradient descent — SGD) представляет собой небольшое изменение в алгоритме градиентного спуска.

Он предоставляет процедуру обновления для оценки коэффициентов бета. Линейные модели прекрасно справляются с этим подходом.

В SGD формула остается такой же, как и в стандартной версии градиентного спуска (называемой пакетной версией, в отличие от сетевой версии), за исключением обновления. В SGD обновление выполняется по одному экземпляру за раз, что позволяет алгоритму оставлять ядро данных в хранилище и помещать в память только одно наблюдение, необходимое для изменения вектора коэффициентов.

$$w_j = w_j - \alpha(wx - y)x_j$$

Как и в случае алгоритма градиентного спуска, алгоритм обновляет коэффициент `w` признака `j`, вычитая разницу между прогнозом и реальным ответом. Затем он умножает разницу на значение признака `j` и коэффициента обучения альфа (способного уменьшить или увеличить влияние обновления на коэффициент).

SGD имеет и другие небольшие отличия. Наиболее важным из них является термин "стохастический" в названии этого алгоритма дистанционного обучения. Фактически, SGD ожидает за раз по одному примеру, взятому случайным образом из доступных примеров (случайная выборка). Проблема с дистанционным обучением заключается в том, что упорядочивание примеров изменяет способ, которым алгоритм определяет коэффициенты бета. При частичной оптимизации один пример может изменить способ, которым алгоритм достигает оптимального значения, создавая другой набор коэффициентов, чем это было бы без этого примера. В качестве практического примера, SGD может изучить порядок, в котором он видит примеры. Если алгоритм выполняет какое либо упорядочивание (историческое, алфавитное или, что еще хуже, связанное с переменной ответа), он неизменно запоминает его. Только случайная выборка позволяет получить надежную сетевую модель, которая эффективно работает с новыми данными. При потоковой передаче данных вам нужно случайным образом изменить порядок ваших данных (перетасовка данных).

Алгоритм SGD, в отличие от пакетного обучения, требует гораздо большего количества итераций для получения правильного глобального направления, несмотря на противоположные указания, приходящие из отдельных примеров. Фактически, алгоритм обновляется после каждого нового примера, и последующее движение к оптимальному набору параметров является более склонным к ошибкам по сравнению с пакетной оптимизацией, которая стремится сразу же получить правильное направление, поскольку оно получается из данных в целом, как показано на рис. 6.3.

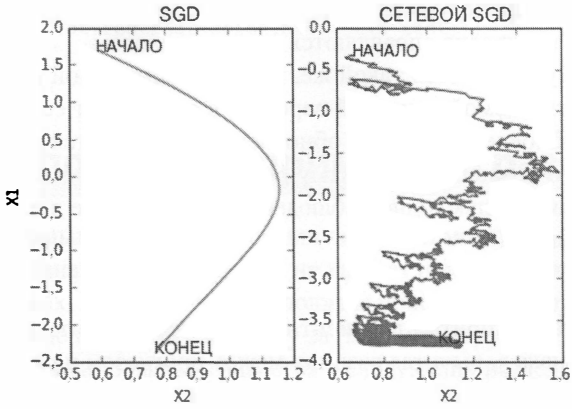


Рис. 6.3. Визуализация различных путей оптимизации для одной и той же задачи данных

В этом случае скорость обучения имеет еще большее значение, поскольку она определяет, как процедура оптимизации SGD может противостоять плохим примерам. На самом деле, если скорость обучения высока, отдаленный пример может полностью сорвать работу алгоритма, помешав ему достичь хорошего результата. С другой стороны, высокая скорость обучения помогает поддержать алгоритм обучения на примерах. Хорошая стратегия состоит в том, чтобы использовать гибкую скорость обучения, то есть начинать с гибкой скорости обучения и делать ее более жесткой по мере роста количества поступивших примеров.

И реализация классификации SGD, и регрессия в библиотеке Scikit-learn имеют разные функции потерь, которые можно применять для оптимизации стохастического градиентного спуска. Только две из этих функций ссылаются на методы, рассматриваемые в этой главе.

- `loss='squared_loss'`. Обычный метод наименьших квадратов (Ordinary Least Squares — OLS) для линейной регрессии.
- `loss='log'`. Классическая логистическая регрессия.

Для демонстрации эффективности *обучения вне ядра* (out-core learning), в следующем примере приведен краткий эксперимент на языке Python с использованием регрессии и `squared_loss` в качестве функции стоимости. Он опирается на набор данных Boston после его перетасовки и разделения на наборы для обучения и тестирования. Пример демонстрирует, как изменяются коэффициенты бета, по мере того, как алгоритм получает больше примеров. Код также передает одни и те же данные несколько раз, чтобы усилить изучение шаблона

данных. Использование тестового набора гарантирует справедливую оценку, предоставляя показатели способности алгоритма обобщать данные за пределы выборки. Выходные данные показывают, сколько требуется времени, прежде чем возрастет R^2 и стабилизируется значение коэффициентов.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDRegressor

X_train, X_test, y_train, y_test = train_test_split(X,
                    y, test_size=0.33, random_state=42)
SGD = SGDRegressor(penalty=None,
                   learning_rate='invscaling',
                   eta0=0.01, power_t=0.25,
                   max_iter=5, tol=None)

power = 17
check = [2**i for i in range(power+1)]
for i in range(400):
    for j in range(X_train.shape[0]):
        SGD.partial_fit(X_train[j,:].reshape(1,13),
                        y_train[j].reshape(1,))
        count = (j+1) + X_train.shape[0] * i
        if count in check:
            R2 = r2_score(y_test,SGD.predict(X_test))
            print ('Example %6i R2 %0.3f coef: %s' %
           (count, R2, ' '.join(map(
             lambda x:'%0.3f' %x, SGD.coef_))))

Example 131072 R2 0.724 coef: -1.098 0.891 0.374 0.849
        -1.905 2.752 -0.371 -3.005 2.026 -1.396 -2.011
        1.102 -3.956
```

ЗАПОМНИ!

Независимо от объема данных, вы всегда можете подобрать простую, но эффективную модель линейной регрессии, используя возможности дистанционного обучения SGD.

Куплено в рескладчину на опенхайде

Глава 7

Введение в нейронные сети

В ЭТОЙ ГЛАВЕ...

- Перцептрон
- Работа со сложными данными
- Разработка стратегий предотвращения переобучения

Возможно, вы уже слышали термин *нейронная сеть* (neural network) в контексте искусственного интеллекта. В первую очередь вам следует уяснить, что правильным термином является *искусственная нейронная сеть* (Artificial Neural Network — ANN), поскольку никто не нашел никакого способа воссоздания реального мозга, от которого происходит понятие нейронной сети. В главе 2 этой книги описываются различные подходы к глубокому обучению, одним из которых является ANN. В этой книге данный термин используется в сокращении, поскольку так его используют все, и вы должны знать, что ANN на самом деле является правильным термином и что это работа школы коннекционистов. (Обсуждение пяти школ машинного обучения и разработанных ими подходов к решению задач содержится в главе 2.)

После того, как вы откажетесь от идеи, что вашему компьютеру не хватает мозга, по крайней мере, *настоящего* мозга, вы сможете оценить перцептрон, являющийся самым простым типом нейронной сети. *Перцептрон* (perceptron) — это фокус многих видов нейронной сети, которые вы видите, но не все нейронные сети имитируют перцептрон.

Нейронная сеть способна работать со сложными данными благодаря тому, что она позволяет нескольким вводам проходить несколько слоев обработки

для получения множества выводов. (На самом деле перцептрон может выбирать только между двумя выходами.) Идея в том, что каждый из путей срабатывает только тогда, когда у него есть шанс ответить на любой вопрос, заданный вводами, на основе выбранных вами алгоритмов. Некоторые из этих методов работы со сложными данными рассматриваются в следующем разделе главы.

Поскольку нейронные сети способны моделировать невероятно сложные данные таким образом, который удивляет некоторых людей, вы можете подумать, что они способны исправить ошибки в обработке, такие как переобучение (см. подробности в главе 2). К сожалению, у компьютеров действительно нет настоящего мозга, поэтому переобучение — это проблема, которую вам нужно решить. В последнем разделе этой главы рассматриваются некоторые решения проблемы переобучения, и обсуждается, почему эта проблема так велика.

Знакомство с невероятным перцептроном

Несмотря на то, что эта книга о глубоком обучении, вам все равно нужно знать о предыдущих уровнях реализации машинного обучения и искусственного интеллекта. Перцептрон на самом деле является типом (реализацией) машинного обучения, для большинства людей, но другие источники скажут вам, что это истинная форма глубокого обучения. Вы можете начать путь к изучению работы алгоритмов машинного обучения, изучив модели, которые вычисляют свои ответы, используя линии и поверхности, чтобы разделить примеры на классы или оценить прогнозируемые значения. Это *линейные модели*, и в этой главе представлен один из самых ранних линейных алгоритмов, используемых в машинном обучении: перцептрон. Последующие главы помогут познакомиться с другими видами моделирования, значительно более передовыми, чем перцептрон. Однако прежде, чем вы сможете перейти к этим другим темам, вы должны узнать интересную историю перцептрона.

Функциональность перцептрона

Фрэнк Розенблатт (Frank Rosenblatt) из Корнелльской лаборатории аэронавтики разработал перцептрон в 1957 году под эгидой Военно-морских исследований США. Розенблатт был психологом и пионером в области искусственного интеллекта. Он был знатоком когнитивных наук, и его идея заключалась в том, чтобы создать компьютер, способный учиться методом проб и ошибок, как это делает человек.

Идея была успешно реализована, и в начале перцептрон не был задуман как просто часть другого программного обеспечения; он был создан как отдельное

программное обеспечение, работающее на специальном оборудовании. Вы можете увидеть его по адресу `https://blogs.umass.edu/comphon/2017/06/15/did-frank-rosenblatt-invent-deep-learning-in-1962/`. Использование этой комбинации позволило быстрее и точнее распознавать сложные изображения, чем любой другой компьютер в то время. Новая технология породила большие ожидания и вызвала огромное противоречие, когда Розенблатт подтвердил, что перцептрон является зародышем нового типа компьютера, который сможет ходить, разговаривать, видеть, писать и даже воспроизводить себя и осознавать свое существование. Если это правда, то получился бы мощный инструмент, и он познакомил мир с искусственным интеллектом.

Излишне говорить, что перцептрон не оправдал ожиданий своего создателя. Вскоре он продемонстрировал ограниченные возможности, даже в своей специализации — распознавании образов. Общее разочарование зажгло первую *зиму искусственного интеллекта* (период сокращения финансирования и интереса по большей части из-за чрезмерных ожиданий) и временного отказа от коннекционизма до 1980-х годов.

ЗАПОМНИ!

Коннекционизм (connectionism) — это подход к машинному обучению, основанный на нейробиологии, а также на примере биологически взаимосвязанных сетей. Вы можете проследить путь коннекционизма до перцептрона.

Перцептрон — это итеративный алгоритм, стремящийся с помощью последовательных повторяющихся приближений определить наилучший набор значений для *вектора коэффициентов* (coefficient vector) `w`. Когда перцептрон достигает подходящего вектора коэффициентов, он может прогнозировать, является ли пример частью класса. Например, одна из задач, которые первоначально выполнял перцептрон, заключалась в определении, напоминало ли изображение, полученное с визуальных датчиков, корабль (пример распознавания образов, требуемый Управлением военно-морских исследований Соединенных Штатов, спонсором исследований на перцептроне). Когда перцептрон видел изображение и относил его к классу кораблей, это означало, что он классифицировал изображение как корабль.

Вектор `w` может помочь прогнозировать класс примера, когда вы умножите его на матрицу объектов `X`, содержащую числовую информацию, выраженную в числовых значениях относительно вашего примера, а затем добавите результат умножения в постоянный член, смещение `b`. Если результат суммы равен нулю или положителен, перцептрон классифицирует пример как часть класса. Когда сумма отрицательна, пример не является частью класса. Вот формула перцептрона, в которой функция `sign` выдает 1 (если пример является частью

класса), когда значение в скобках равно или больше нуля; в противном случае выводится 0.

```
y = sign(Xw + b)
```

Обратите внимание, что этот алгоритм содержит все элементы, которые характеризуют глубокую нейронную сеть, а значит все строительные блоки, обеспечивающие технологию, присутствовали с самого начала.

- **Числовая обработка ввода**. `X` содержит числа, в качестве входных данных не используют ни каких символических значений, пока вы не обработаете его как число. Например, вы не можете вводить символическую информацию, такую как красный, зеленый или синий, пока не преобразуете эти значения цветов в числа.
- **Веса и смещения**. Перцептрон преобразует `X` в ходе умножения на веса и добавления смещения.
- **Суммирование результатов**. Использует матричное умножение при умножении `X` на вектор `w` (матричное умножение рассматривается в главе 5).
- **Функция активации**. Когда сумма превышает пороговое значение, перцептрон активирует результат и указывает, что входные данные являются частью некого класса.
- **Итеративное получение наилучшего набора значений для вектора `w`**. Решение основано на последовательных приближениях в результате сравнения вывода перцептрона с ожидаемым результатом.

Достижение предела неразделяемости

Секрет вычислений перцептрона в том, как алгоритм обновляет значения вектора `w`. Такие обновления происходят в результате случайного выбора одного из неправильно классифицированных примеров. Неправильно классифицированный пример — это когда перцептрон определяет, что пример является частью класса, но это не так, или когда перцептрон определяет, что пример не является частью класса, но это так. Перцептрон обрабатывает один ошибочно классифицированный пример за раз (назовем его x_t) и работает, изменяя вектор `w` при помощи простого взвешенного сложения.

$$w = w + \eta(x_t * y_t)$$

Эта формула называется стратегией обновления перцептрона, а буквы обозначают различные числовые элементы.

- Буква `w` — это *векторы коэффициентов*, которые обновляются, чтобы правильно показать, является ли неправильно классифицированный пример `t` частью класса или нет.
- Греческая буква эта (`η`) — *скорость обучения*. Это число с плавающей запятой от 0 до 1. Когда вы устанавливаете это значение близким к нулю, это может ограничить возможность формулы обновлять вектор `w` почти полностью, тогда как установка значения близкого к единице делает процесс обновления полностью влияющим на значения вектора `w`. Установка разных скоростей обучения может ускорить или замедлить процесс обучения. Многие другие алгоритмы применяют эту стратегию, и более низкое значение эта используется для улучшения процесса оптимизации в результате уменьшения количества внезапных скачков значений `w` после обновления. Компромисс заключается в том, что вам нужно подождать подольше, прежде чем вы получите окончательные результаты.
- Переменная x_t относится к вектору числовых объектов для примера `t`.
- Переменная y_t относится к истинной правде о том, является ли пример `t` частью класса или нет. Для алгоритма перцептрона y_t численно выражается как +1, когда пример является частью класса, и –1, когда пример не является частью класса.

Стратегия обновления дает интуитивное представление о том, что происходит при использовании перцептрона для изучения классов. Если вы представите примеры, спроецированные на декартову плоскость, то перцептрон — это не более чем линия, пытающаяся отделить положительный класс от отрицательного. Как вы помните из линейной алгебры, все, что выражается в виде `y = xb + a`, на самом деле является прямой на плоскости. Перцептрон использует формулу `y = xw + b`, которая использует другие буквы, но выражает ту же формулу — линию на декартовой плоскости.

Первоначально, когда `w` установлена в ноль или случайное значение, разделительная линия — это только одна линия из бесконечного количества возможных линий на плоскости. Это показано на рис. 7.1. Фаза обновления изменяет линию, заставляя ее приблизиться к ошибочно классифицированной точке. По мере прохода алгоритма по неправильно классифицированным примерам, он применяет ряд исправлений. В конце, используя несколько итераций для определения ошибок, алгоритм размещает разделительную линию на точной границе между двумя классами.

Несмотря на такой умный алгоритм, перцептрон довольно скоро продемонстрировал свои пределы. Помимо способности разделять два класса, используя только количественные признаки, у него было важное ограничение: если два

класса не имели границ из-за смешения, алгоритм не мог найти решение и продолжал бесконечно обновляться.

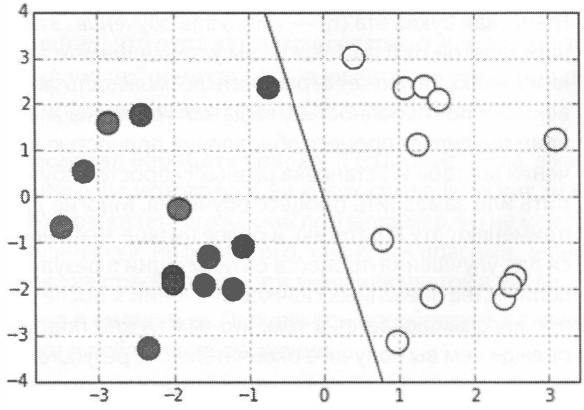


Рис. 7.1. Разделительная линия перцептрона между двумя классами

ЗАПОМНИ!

Если вы не можете разделить два распределенных по двум или более измерениям класса, ни по любой линии или плоскости, они будут *линейно неразделяемы* (nonlinearly separable). Решение проблемы линейной неразделяемости данных является одной из задач, которые необходимо преодолеть машинному обучению, чтобы стать эффективными решением сложных задач, основанных на реальных данных, а не только на искусственных данных, созданных для академических целей.

Когда вопрос линейной неразделяемости оказался под пристальным вниманием, и практики начали терять интерес к перцептрону, эксперты быстро предположили, что могут решить проблему, создав новое пространство признаков, в котором ранее неразделяемые классы были представлены так, чтобы стать разделяемыми. Таким образом, перцептрон будет так же хорош, как и раньше. К сожалению, создание новых пространств признаков является сложной задачей, поскольку требует вычислительных мощностей, которые сегодня доступны публике лишь частично. Создание нового пространства признаков — это сложная тема, которая обсуждается далее в книге при изучении стратегий обучения алгоритмов, таких как нейронные сети и метод опорных векторов.

Благодаря большим данным, в последние годы этот алгоритм претерпел возрождение: фактически, перцептрону не нужно работать со всеми данными в памяти, но он может прекрасно работать с отдельными примерами (обновляя

свой вектор коэффициентов только тогда, когда неправильно классифицированный случай делает это необходимым). Таким образом, это идеальный алгоритм для дистанционного обучения, например, обучение на больших данных за один раз.

Уменьшение сложности нейронными сетями

Предыдущий раздел главы помог вам познакомиться с нейронной сетью с точки зрения перцептрона. Конечно, в нейронных сетях есть нечто большее, чем это простое начало. Объемы и другие мешающие перцептрону проблемы, по крайней мере частично, решены в новых алгоритмах. Следующие разделы помогут вам понять, какие нейронные сети существуют сегодня.

Нейрон

Основным компонентом нейронной сети является *нейрон* (neuron) (или *блок* (unit)). Множество нейронов, расположенных во взаимосвязанной структуре, образуют нейронную сеть, где каждый нейрон связан с вводами и выводами других нейронов. Таким образом, нейрон может получать признаки из примеров или результатов работы других нейронов, в зависимости от их расположения в нейронной сети.

Когда психолог Розенблатт задумывал перцептрон, он считал его упрощенной математической версией нейрона мозга. В качестве входных данных перцептрон получает значения из близлежащего окружения (набора данных), взвешивает их (как это делают клетки мозга на основе силы связующих соединений), суммирует все взвешенные значения и активирует, когда сумма превышает пороговое значение. В результате выводится значение 1; в противном случае его прогноз равен 0. К сожалению, перцептрон не может распознать случаи, когда классы, которые он пытается обработать, не являются линейно разделяемыми. Однако ученые обнаружили, что хотя один перцептрон не может выучить логическую операцию XOR, показанную на рис. 7.2 (исключающее ИЛИ, возвращающее true только в случае различий между вводами), два перцептрона, работающие вместе, вполне могут.

Нейроны в нейронной сети представляют собой дальнейшую эволюцию перцептрона: в качестве входных данных они получают множество взвешенных значений, суммируют их и обеспечивают суммирование в результате, как это делает перцептрон. Тем не менее, они также обеспечивают более сложную трансформацию суммирования, чего не может сделать перцептрон. Наблюдая за природой, ученые заметили, что нейроны получают сигналы, но не всегда испускают собственный сигнал. Это зависит от количества полученного

сигнала. Когда нейрон получает достаточно стимулов, он дает ответ, а в противном случае молчит. Аналогичным образом алгоритмические нейроны после получения взвешенных значений суммируют их и используют функцию активации для оценки результата, который затем преобразует нелинейным образом. Например, *функция активации* (activation function) может испускать нулевое значение, если ввод не достиг определенного порогового значения, либо она может ослаблять или усиливать значение в ходе нелинейного масштабирования, передавая, таким образом, измененный сигнал.

Рис. 7.2. Получение логического XOR с использованием одной разделительной линии невозможно

Нейронная сеть имеет различные функции активации, как показано на рис. 7.3. Линейная функция не применяет никаких преобразований, и ее редко используют, поскольку она сводит нейронную сеть к регрессии с полиномиальными преобразованиями. Нейронные сети обычно используют сигмоидные или гиперболические касательные (tanh) функции, или функции активации

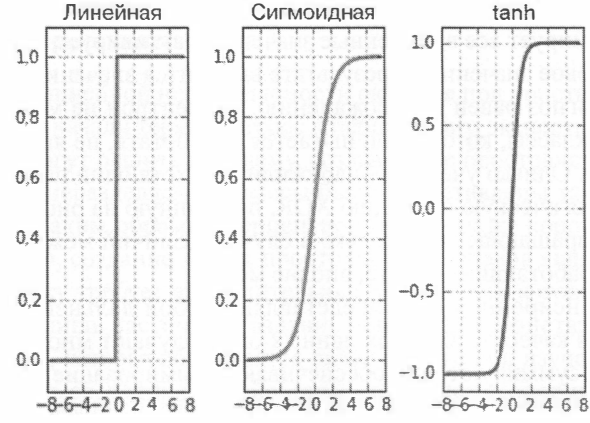


Рис. 7.3. Графики разных функций активации

ReLU (которые сегодня являются наиболее распространенными). (В главе 8 функции активации описаны более подробно.)

На рисунке показано, как ввод (выраженный на горизонтальной оси) может преобразовать вывод в нечто другое (выраженное на вертикальной оси). Примеры демонстрируют линейную, сигмоидную (или логистическую) и касательную гиперболическую функции активации (или tanh).

СОВЕТ

Более подробно функции активации рассматриваются далее в этой главе, а пока отметьте, что функции активации хорошо работают в определенных диапазонах значений x. Поэтому вы всегда должны масштабировать входные данные нейронной сети, используя статистическую стандартизацию (нулевое среднее и единичное отклонение) или нормализовать входные данные в диапазоне от 0 до 1 или от –1 до 1.

ЗАПОМНИ!

Функции активации — это то, что заставляет нейронную сеть выполнять классификацию или регрессию; тем не менее, первоначальный выбор сигмоидной активации или tanh для большинства сетей представляет собой критическое ограничение при использовании более сложных сетей, поскольку обе функции активации работают оптимально для очень ограниченного диапазона значений.

Прямая передача данных

В нейронной сети следует учитывать архитектуру, заключающуюся в расположении компонентов нейронной сети. В отличие от других алгоритмов, имеющих фиксированный конвейер, который определяет, как алгоритмы получают и обрабатывают данные, нейронные сети требуют, чтобы вы решали, как передается информация, фиксируя количество блоков (нейронов) и их распределение по слоям, как показано на рис. 7.4.

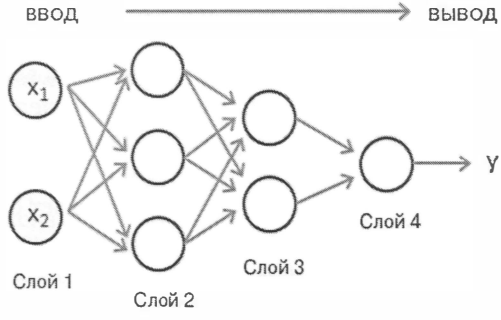


Рис. 7.4. Пример архитектуры нейронной сети

На рисунке показана простая нейронная архитектура. Обратите внимание, что слои фильтруют информацию постепенно. Это прямая передача, поскольку данные поступают в сеть с одной стороны. Соединения связывают блоки исключительно с блоками в одном следующем слое (информация передается слева направо). Нет никаких соединений между блоками в одном слое или с блоками вне следующего слоя. Более того, информация продвигается только вперед (слева направо). Обработанные данные никогда не возвращаются к предыдущим слоям нейронов.

Использование нейронной сети похоже на использование прямой системы фильтрации воды: вы наливаете воду сверху, и она, после фильтрации вытекает снизу. У воды нет пути назад; она идет только вперед (прямо вниз), и никогда вбок. Точно так же нейронные сети заставляют признаки данных проходить через сеть и смешиваться друг с другом только в соответствии с архитектурой сети. Используя лучшую архитектуру для смешивания признаков, нейронная сеть создает новые составные признаки на каждом слое и помогает достигать лучших прогнозов. К сожалению, невозможно определить лучшую архитектуру без эмпирического опробования различных решений и проверки того, помогают ли выходные данные прогнозировать ваши целевые значения после прохождения через сеть.

Первый и последний слои играют важную роль. Первый слой, *входной слой*, выбирает элементы из каждого примера данных, обработанного сетью. Последний слой, *выходной слой*, испускает результаты.

Нейронная сеть может обрабатывать только числовую непрерывную информацию; ее нельзя ограничивать работой с качественными переменными (например, качественными надписями, например красный, синий или зеленый на изображении). Вы можете обработать качественные переменные, преобразовав их в непрерывные числовые значения, такие как серия двоичных значений. Когда нейронная сеть обрабатывает двоичную переменную, нейрон рассматривает ее как общее число и в ходе обработки в блоках превращает двоичные значения в другие значения, даже отрицательные.

Обратите внимание на ограничение работы только с числовыми значениями, поскольку вы не можете ожидать, что последний слой выведет прогноз нечисловой метки. Когда речь идет о задаче регрессии, последний слой является одним блоком. Аналогично, когда вы работаете с классификацией и у вас есть выходные данные, которые должны быть выбором из n классов, у вас должно быть n конечных блоков, каждый из которых представляет оценку, связанную с вероятностью представляемого класса. Следовательно, при множественной классификации, такой как виды ирисов, конечный слой имеет столько же блоков, сколько есть видов. Например, в примере классификации архетипа Iris, созданном известным статистиком Фишером, у вас есть три класса: `setosa`,

versicolor и virginica. В нейронной сети, основанной на наборе данных Iris, у вас есть три блока, представляющих один из трех видов ирисов. Для каждого примера прогнозируемый класс — это тот, который в конце получает более высокий балл.

СОВЕТ

Некоторые нейронные сети имеют специальные конечные слои, совокупно называемые softmax, способные регулировать вероятность каждого класса на основе значений, полученных от предыдущего слоя. В классификации, благодаря softmax, последний слой может представлять как раздел вероятностей (задача множественной классификации, в которой суммарные вероятности составляют 100%), так и независимый прогноз оценки (поскольку в примере может быть больше классов, что является задачей с несколькими метками, в которой суммарные вероятности могут быть более 100%). Когда задача классификации представляет собой двоичную классификацию, достаточно одного выхода. Кроме того, в регрессии вы можете иметь несколько выходных блоков, каждый из которых представляет свою задачу регрессии. (Например, при прогнозировании у вас могут быть разные прогнозы на следующий день, неделю, месяц и т. д.)

Еще глубже в кроличью нору

Нейронные сети имеют разные слои, каждый из которых имеет свой вес. Поскольку нейронная сеть разделяет вычисления по слоям, важно знать *эталонный слой* (reference layer), поскольку вы можете учитывать определенные блоки и соединения. Вы можете обратиться к каждому слою, используя определенное число, и, в общем, говорить о каждом слое, используя букву l.

Каждый слой может иметь различное количество блоков, а количество блоков, расположенных между двумя слоями, определяет количество соединений. Умножив количество блоков в начальном слое на количество в следующем слое, вы можете определить общее количество соединений между ними:

$$количество\ соединений^{(l)} = блоков^{(l)} * блоков^{(l+1)}.$$

Матрица весов, обычно именуемая заглавной греческой буквой тета (θ), представляет связи. Для удобства чтения в книге используется заглавная буква W, что является хорошим выбором, поскольку она представляет матрицу или многомерный массив. Таким образом, вы можете использовать W^1 для указания веса соединений от слоя 1 до слоя 2, W^2 для соединений от слоя 2 до слоя 3 и так далее.

Веса представляют силу связи между нейронами в сети. Когда вес соединения между двумя слоями невелик, это означает, что сеть практически

игнорирует проходящие между ними значения, и выбирающие этот маршрут сигналы вряд ли повлияют на окончательный прогноз. В качестве альтернативы, большое положительное или отрицательное значение влияет на значения, которые получает следующий слой, изменяя, таким образом, некоторые прогнозы. Этот подход аналогичен клеткам мозга, которые не стоят отдельно, а соединяются с другими клетками. По мере того, как у кого-то накапливается опыт, связи между нейронами имеют тенденцию ослабевать или усиливаться, чтобы активировать или деактивировать определенные области клеток сети мозга, вызывая другую обработку или активность (например, реакцию на опасность, если обработанная информация сигнализирует об угрожающей жизни ситуации).

СКРЫТЫЕ СЛОИ

Слои между вводом и выводом иногда (но не в этой книге) называют *скрытыми слоями* (hidden layer), и количество слоев начинается с первого скрытого слоя. Это просто соглашение, отличное от того, которое используется в этой книге. Примеры в книге всегда начинают отсчет с входного слоя, поэтому первый скрытый слой — это слой номер 2.

Теперь, когда вы знаете некоторые соглашения, относящиеся к слоям, блокам и соединениям, вы можете приступить к подробному изучению операций, выполняемых нейронными сетями. Для начала вы можете называть вводы и выводы разными способами.

- **a**. Результат сохраняется в блоке нейронной сети после обработки функцией активации (называемой *g*). Это окончательный результат, который отправляется дальше по сети.
- **z**. Умножение между `a` и весами из матрицы `W`. `z` представляет сигнал, проходящий через соединения, аналогично воде в трубах, которая течет при более высоком или более низком давлении в зависимости от диаметра трубы. Таким же образом, значения, полученные от предыдущего слоя, получают более высокие или более низкие значения из-за весов соединений, используемых для их передачи.

Каждый последующий слой блоков в нейронной сети постепенно обрабатывает значения, взятые из признаков. Когда данные передаются по сети, они поступают в каждый блок как значение, полученное в ходе суммирования значений, присутствующих на предыдущем слое и взвешенных по соединениям, представленным в матрице `W`. Когда данные с добавленным смещением превышают определенный порог, функция активации увеличивает значение,

хранящееся в блоке; в противном случае она гасит сигнал, уменьшая его. После обработки функцией активации результат готов к отправке на соединение, связанное со следующим слоем. Эти шаги повторяются для каждого слоя, пока значения не достигнут конца, и вы получите результат, как показано на рис. 7.5.

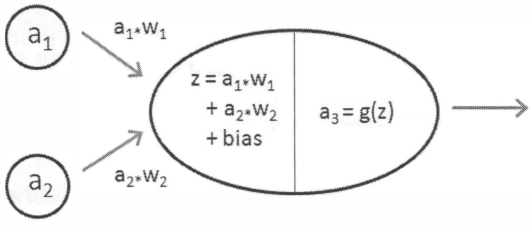


Рис. 7.5. Процесс прямой связи в нейронной сети

На рисунке показаны детали процесса, в ходе которого два блока передают свои результаты другому блоку. Это событие происходит в каждой части сети. Когда вы понимаете переход от двух нейронов к одному, вы можете понять весь процесс обратной связи, даже когда задействовано больше слоев и нейронов. Вот семь шагов, используемых для создания прогноза в нейронной сети, состоящей из четырех слоев (см. рис. 7.4).

1. Первый слой (обратите внимание на верхний индекс 1 у a) загружает значение каждого признака в отдельный блок.

   $a^{(1)} = X$

2. Веса соединений входного слоя со вторым слоем, умножаются на значения блока в первом слое. Умножение матриц взвешивает и суммирует входные данные для второго слоя.

   $z^{(2)} = W^{(1)} a^{(1)}$

3. Алгоритм добавляет константу смещения ко второму слою перед запуском функции активации. Функция активации преобразует входные данные второго слоя. Полученные значения готовы к передаче в соединения.

   $a^{(2)} = g(z^{(2)} + bias^{(2)})$

4. Соединения третьего слоя взвешивают и суммируют выходные данные второго слоя.

   $z^{(3)} = W^{(2)} a^{(2)}$

5. Алгоритм добавляет константу смещения к третьему слою перед запуском функции активации. Функция активации преобразует вводы третьего слоя.

   $a^{(3)} = g(z^{(3)} + bias^{(3)})$

6. Выходы третьего слоя взвешиваются и суммируются соединениями с выходным слоем.

$$z^{(4)} = W^{(3)}a^{(3)}$$

7. Наконец, алгоритм добавляет константу смещения к четвертому слою перед запуском функции активации. Выходные блоки получают свои вводы и преобразуют ввод, используя функцию активации. После этого окончательного преобразования выходные блоки готовы выдать итоговые прогнозы нейронной сети.

$$a^{(4)} = g(z^{(4)} + bias^{(4)})$$

Функция активации играет роль фильтра сигналов, помогая выбирать релевантные сигналы и избегать слабых и зашумленных (поскольку отбрасывает значения ниже определенного порога). Функции активации обеспечивают также нелинейность вывода, поскольку они усиливают или ослабляют значения, проходящие через них непропорциональным образом.


ЗАПОМНИ!

Веса соединений позволяют смешивать и составлять признаки по-новому, создавая новые признаки способом, не слишком отличающимся от полиномиального разложения. Активация дает нелинейную результирующую рекомбинацию объектов связями. Оба эти компонента нейронной сети позволяют алгоритму изучать сложные целевые функции, которые представляют взаимосвязь между входными признаками и целевым результатом.

Использование обратного распространения для настройки обучения

С архитектурной точки зрения нейронная сеть отлично справляется со смешиванием сигналов из примеров и превращением их в новые признаки для достижения приближения сложных нелинейных функций (функций, которые вы не можете представить в виде прямой линии в пространстве объектов). Для обеспечения этой возможности, нейронные сети работают как *универсальные аппроксиматоры* (universal approximator) (более подробная информация приведена по адресу `https://www.techleer.com/articles/449-the-universal-approximation-theorem-for-neural-networks/`), а значит, они могут прогнозировать любую целевую функцию. Однако необходимо учитывать, что одним из аспектов этого признака является способность моделировать сложные функции (*возможность представления* (representation capability)), а другим аспектом является способность эффективно изучать данные. В мозге обучение происходит из-за формирования и модификации синапсов между нейронами,

основанных на стимулах, полученных методом проб и ошибок. Нейронные сети предоставляют способ воспроизведения этого процесса в виде математической формулы — *обратного распространения* (backpropagation).

С момента своего появления в 1970-х годах в алгоритм обратного распространения внесено множество исправлений. Каждое усовершенствование процесса обучения нейронной сети приводило к появлению новых приложений и возобновлению интереса к этой технике. Кроме того, нынешняя революция глубокого обучения, возрождение нейронных сетей, от которых отказались в начале 1990-х годов, обусловлена ключевыми достижениями в том, как нейронные сети учатся на своих ошибках. Как видно из других алгоритмов, функция стоимости активирует необходимость лучшего изучения определенных примеров (большие ошибки соответствуют высоким стоимостям). Когда встречается пример с большой ошибкой, функция стоимости выводит высокое значение, которое минимизируется в ходе изменения параметров в алгоритме. Алгоритм оптимизации определяет наилучшее действие для уменьшения высоких выводов из функции стоимости.

В линейной регрессии найти правило обновления для применения к каждому параметру (вектору коэффициентов бета) довольно просто. Однако в нейронной сети все немного сложнее. Архитектура является переменной, и коэффициенты параметров (соединения) связаны друг с другом, поскольку соединения в слое зависят от того, как соединения в предыдущих слоях рекомбинировали вводы. Решением этой проблемы является алгоритм обратного распространения. Обратное распространение — это интеллектуальный способ передачи ошибки обратно в сеть и соответствующей настройке весов каждого соединения. Если вы изначально передали информацию в сеть, пришло время вернуться назад и дать отзыв о том, что пошло не так на этапе прямой передачи.

ЗАПОМНИ!

Обратное распространение — это механизм распространения по нейронной сети корректировок, необходимых алгоритму оптимизации. Важно отметить различие между оптимизацией и обратным распространением. Фактически, все нейронные сети используют обратное распространение, а в следующей главе обсуждается множество различных алгоритмов оптимизации.

Узнать, как работает обратное распространение, не сложно, хотя демонстрация того, как это работает с использованием формул и математики, требует производных и проверки некоторых формул, что довольно сложно и выходит за рамки этой книги. Чтобы понять, как работает обратное распространение, начните с конца сети, как раз в тот момент, когда пример был обработан, и у вас есть прогноз в качестве вывода. На этом этапе вы можете сравнить его с реальным результатом, а их разница позволит получить смещение, которое

и является ошибкой. Теперь, когда вы знаете величину несоответствия результатов на выходном слое, вы можете двинуться назад, чтобы распределить его по всем блокам в сети.

СОВЕТ

Функция стоимости нейронной сети для классификации основана на кросс-энтропии (как видно из логистической регрессии).

$$Cost = y * \log(h_W(X)) + (1 - y)*\log(1 - h_W(X))$$

Эта формула включает логарифмы. Она относится к прогнозу, созданному нейронной сетью и выраженному как $h_W(X)$ (что является результатом сети с заданными соединениями W и X в качестве ввода). Чтобы упростить задачу, когда вы думаете о стоимости, следует подумать о том, как вычислить смещение между ожидаемыми результатами и выводом нейронной сети.

Первый шаг в передаче ошибки обратно в сеть основан на обратном умножении. Поскольку значения, подаваемые в выходной слой, состоят из вкладов всех блоков, пропорциональных весу их соединений, вы можете перераспределить ошибку в соответствии с каждым вкладом. Например, вектор ошибок слоя n в сети (вектор, обозначенный греческой буквой дельта (δ)), является результатом следующей формулы.

$$\delta^{(n)} = W^{(n)T} * \delta^{(n+1)}$$

Эта формула говорит, что, начиная с последней дельты, вы можете продолжить перераспределение дельты, возвращаясь по сети и используя веса, которые были использованы при прямом проходе, чтобы распределить ошибку между различными блоками. Таким образом, чтобы минимизировать ошибку, вы можете перераспределить конечную ошибку на каждый нейронный блок и использовать ее для пересчета более подходящего веса для каждого сетевого соединения. Чтобы обновить веса W слоя l, вы просто применяете следующую формулу.

$$W^{(l)} = W^{(l)} + \eta * \delta^{(l)} * g'(z^{(l)}) * a^{(l)}$$

На первый взгляд формула может показаться загадочной, но это сумма, и вы можете узнать, как она работает, осмотрев ее элементы. Сначала рассмотрим функцию g'. Это первая производная функции активации g, оцененная по входным значениям z. Фактически это метод градиентного спуска. Градиентный спуск определяет, как уменьшить меру ошибки, найдя среди возможных комбинаций такие значения весов, которые больше всего уменьшают ошибку.

Скорость обучения обозначается греческой буквой эта (η), но в некоторых учебниках для ее обозначения используют также буквы альфа (α) или эпсилон (ε). Как и в других алгоритмах, скорость обучения уменьшает эффект обновления, предложенного производной градиентного спуска. Фактически, указанное

направление может быть только частично правильным или только приблизительно правильным. Делая несколько небольших шагов спуска, алгоритм может более точно указать направление к глобальному минимуму ошибки, что и является вашей целью (то есть нейронная сеть, обладающая наименьшей возможной ошибкой прогнозирования).

Для установки правильного значения эта доступны разные методы, поскольку от нее во многом зависит оптимизация. Один метод устанавливает значение эта начиная с самого высокого, и уменьшает его в процессе оптимизации. Другой метод попеременно увеличивает или уменьшает значение эта на основе улучшений, полученных с помощью алгоритма: большие улучшения приводят к большему значению эта (поскольку спуск легкий и прямой); меньшие улучшения дают меньшие значения эта, так что оптимизация будет продвигаться медленнее, ища наилучшие возможности для снижения. Это как на извилистой тропинке в горах: вы замедляетесь, стараясь не сбиться с дороги или не упасть при спуске.

СОВЕТ

Большинство реализаций предлагают автоматическую настройку правильного значения эта. При обучении нейронной сети необходимо учитывать актуальность данного параметра, поскольку это один из важнейших параметров, который необходимо настроить для получения более точных прогнозов наравне с архитектурой слоя. Обновления весов могут происходить по-разному в зависимости от учебного набора примеров.

- **Сетевой режим**. Обновление веса происходит после того, как каждый пример проходит через сеть. Таким образом, алгоритм обрабатывает обучающие примеры как поток, на котором можно учиться в режиме реального времени. Этот режим идеален, когда вам нужно обучение вне ядра, то есть когда обучающий набор не может поместиться в оперативную память. Но этот метод чувствителен к выбросам, поэтому вы должны поддерживать низкую скорость обучения. (Следовательно, алгоритм медленно приходит к решению.)
- **Пакетный режим**. Обновление веса происходит после обработки всех примеров в учебном наборе. Этот метод ускоряет оптимизацию и менее подвержен влиянию дисперсии в потоке примеров. В пакетном режиме обратное распространение учитывает суммированные градиенты всех примеров.
- **Мини-пакетный (или стохастический) режим**. Обновление веса происходит после того, как сеть обработала подвыборку случайно выбранных примеров из обучающего набора. Этот подход сочетает преимущества сетевого режима (низкое использование памяти)

и пакетного режима (быстрая сходимость), в то же время, вводя случайный элемент (подвыборку), чтобы градиентный спуск не застревал в локальных минимумах (снижение значения, не являющегося истинным минимум).

Борьба с переобучением

Учитывая архитектуру нейронной сети, вы можете себе представить, как легко алгоритм может извлечь из данных практически все, особенно если вы добавили слишком много слоев. На самом деле, алгоритм работает так хорошо, что на его прогнозы зачастую влияет высокая оценочная дисперсия, называемая *переобучением* (overfitting). Переобучение заставляет нейронную сеть изучать каждую деталь обучающих примеров, что делает возможным их копирование на этапе прогнозирования. Но что-то другое, кроме учебного набора, сеть никогда не сможет прогнозировать правильно. В следующих разделах более подробно обсуждаются некоторые проблемы переобучения.

Понятие проблемы

Когда вы используете нейронную сеть для реальной задачи, вы становитесь строже и осторожнее в реализации, чем с другими алгоритмами. Нейронные сети хрупки и более подвержены ошибкам, чем другие решения машинного обучения.

Вы тщательно разделяете свои данные на обучающие, проверочные и тестовые наборы. Прежде чем алгоритм изучит данные, вы должны оценить качество ваших параметров.

- Архитектура (количество слоев и узлов в них).
- Функции активации.
- Параметр обучения.
- Количество итераций.

В частности, архитектура предлагает большие возможности для создания мощных прогностических моделей с высоким риском переобучения. Параметр обучения управляет скоростью обучения сети на основе данных, но этого может быть недостаточно для предотвращения переобучения на обучающих данных. (Более подробная информация о причинах переобучения приведена в главе 2.)

Открываем черный ящик

У вас есть два возможных решения проблемы переобучения. Первое — это регуляризация, как в случае линейной и логистической регрессии. Вы можете суммировать все коэффициенты соединения в квадрате или в абсолютном значении, чтобы оштрафовать модели со слишком большим количеством коэффициентов с высокими значениями (обеспечивается регуляризацией *L2*) или со значениями, отличными от нуля (обеспечивается регуляризацией *L1*). Второе решение также эффективно, поскольку оно контролирует начало переобучения. Он называется *ранней остановкой* (early stop) и работает за счет проверки функции стоимости на проверочном наборе, когда алгоритм учится на обучающем наборе. (Более подробная информация о ранней остановке приведена в главе 5.)

СОВЕТ

Вы можете не понять, когда ваша модель начинает переобучаться. Функция стоимости, рассчитанная с использованием обучающего набора, продолжает улучшаться по мере оптимизации. Однако, как только вы начнете извлекать из данных шум и перестанете изучать общие правила, вы сможете проверить функцию стоимости на данных вне выборки (проверочная выборка). В какой-то момент вы заметите, что она перестает улучшаться и начинает ухудшаться, что означает, что ваша модель достигла своего предела обучения.

Глава 8

Построение простой нейронной сети

В ЭТОЙ ГЛАВЕ...

- Базовая архитектура
- Определение задачи
- Процесс решения

В главе 7 представлены нейронные сети в самой простой базовой форме — в виде перцептрона. Однако нейронные сети бывают разных форм, каждая из которых имеет свои преимущества. К счастью, для достижения своей цели все формы нейронных сетей следуют базовой архитектуре и полагаются на определенные стратегии. Если вы узнаете, как работает базовая нейронная сеть, вы сможете понять, как работают более сложные архитектуры. В первой части этой главы рассматриваются основы работы нейронной сети — то есть то, что вам нужно знать, чтобы понять, как нейронная сеть выполняет свою работу. Далее объясняются функции нейронной сети на примере базовой нейронной сети, которую вы можете построить с нуля, используя язык Python.

Вторая часть главы посвящена некоторым различиям между нейронными сетями. Например, в главе 7 упоминалось, что отдельные нейроны срабатывают после достижения определенного порога. Функция активации определяет, когда входной сигнал достаточен для срабатывания нейрона, поэтому знание, какие функции активатора доступны, важно для различения нейронных сетей. Кроме того, вам необходимо знать об оптимизаторе, используемом для быстрого получения результатов, которые фактически моделируют решаемую задачу. Наконец, вам нужно решить, как быстро учится ваша нейронная сеть.

ЗАПОМНИ!

Вам не нужно вводить исходный код вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файле `DL4D_08_NN_From_Scratch.ipynb`.

Понятие нейронных сетей

В Интернете можно найти много дискуссий об архитектуре нейронной сети (например, на `https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html`). Проблема, однако, в том, что все они быстро становятся безумно сложными. Некоторые неписаные законы, казалось бы, гласят, что математика мгновенно должна стать абстрактной и настолько сложной, что ни один простой смертный не сможет ее понять, однако нейронную сеть сможет понять любой. Хорошее начало дает материал главы 7. Несмотря на то, что глава 7 немного полагается на математику, чтобы донести свою мысль, эта математика относительно проста. Теперь, в этой главе вы узнаете, как внедрить в код Python все основные функции нейронной сети.

То, что на самом деле представляет собой нейронная сеть, является своего рода фильтром. Вы заливаете данные в верхнюю часть, эти данные просачиваются через различные слои, которые вы создаете, а выходные данные вытекают снизу. То, что отличает нейронные сети, — это те же самые элементы, которые вы можете найти в фильтре. Например, тип выбранного вами алгоритма определяет тип фильтрации, который будет выполнять нейронная сеть. Возможно, вы захотите отфильтровать из воды свинец, но оставить при этом кальций и другие полезные минералы, что означает выбор такого типа фильтра, который подходит именно для этого.

Однако фильтры могут быть контролируемыми. Например, вы можете выбрать фильтрацию частиц одного размера, но пропускать частицы другого размера. Использование весов и смещений в нейронной сети — это просто своего рода контроль. Вы настраиваете его так, чтобы обеспечить точную фильтрацию, которая вам необходима. В этом случае, поскольку вы используете электрические сигналы, смоделированные по аналогии с сигналами, обнаруженными в мозге, сигнал может пройти, когда он удовлетворяет некому условию — порогу, определяемому функцией активации. Для простоты считайте пока это как бы настройкой работы любого фильтра.

Вы можете отслеживать активность вашего фильтра. Но если вы не хотите стоять там весь день, глядя на работу фильтра, вы, вероятно, полагаетесь на какую-то автоматизацию, чтобы гарантировать, что вывод фильтра остается

постоянным. Вот где в игру вступает оптимизатор. Оптимизируя вывод нейронной сети, вы получаете нужные результаты, избежав настройки вручную.

Наконец, вы хотите, чтобы фильтр работал с той скоростью и емкостью, которые позволят ему правильно выполнять свои задачи. Слишком быстрая заливка воды или другого вещества в фильтр может привести к его переполнению. Если вы наливаете не достаточно быстро, фильтр может засориться или работать беспорядочно. Регулировка скорости обучения оптимизатора нейронной сети позволяет вам гарантировать, что нейронная сеть выдаст желаемый результат. Это похоже на регулировку скорости заливки фильтра.

Нейронные сети могут показаться трудными для понимания. Тот факт, что многое из того, что они делают, объясняет довольно сложная математика, отнюдь не поможет. Но вам не нужно быть ученым, чтобы понять, что такое нейронные сети. Все, что вам действительно нужно сделать, это разделить их на управляемые части и использовать правильную точку зрения, чтобы взглянуть на них. В следующих разделах показано, как создать код каждой части простой нейронной сети с самого начала.

Базовая архитектура

Нейронная сеть опирается на многочисленные вычислительные блоки, *нейроны*, организованные в иерархические слои. Каждый нейрон получает входные данные от всех своих предшественников и предоставляет выходные данные своим преемникам, пока вся нейронная сеть в целом не станет удовлетворять требованиям. В этот момент работа сети заканчивается, и вы получаете вывод.

Все эти вычисления происходят исключительно в нейронной сети. Сеть проходит по каждому из них, используя для итераций циклы. Вы также можете использовать тот факт, что большинство из этих операций представляют собой простые умножения с последующим сложением и использовать преимущества матричных вычислений, показанных в главе 5.

В примере этого раздела создается сеть с входным слоем (размеры которого определяются входными данными), скрытым слоем с тремя нейронами и одним выходным слоем, который сообщает, принадлежит ли входной элемент некому классу (проще говоря, ответ двоичный 0 или 1). Эта архитектура подразумевает создание двух наборов весов, представленных двумя матрицами (вот когда вы фактически используете матрицы).

- Первая матрица имеет размер, определяемый количеством входов × 3, и представляет веса, на которые умножаются вводы, и суммирует их в три нейрона.
- Вторая матрица имеет размер 3 × 1, она собирает все выходные данные со скрытого слоя и объединяет этот слой в выходные данные.

Вот необходимый код на языке Python (выполнение которого может занять некоторое время, в зависимости от скорости вашей системы).

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline

def init(inp, out):
    return np.random.randn(inp, out) / np.sqrt(inp)

def create_architecture(input_layer, first_layer,
                        output_layer, random_seed=0):
    np.random.seed(random_seed)
    layers = X.shape[1], 3 , 1
    arch = list(zip(layers[:-1], layers[1:]))
    weights = [init(inp, out) for inp, out in arch]
    return weights
```

Интересным моментом этой инициализации является то, что для автоматизации сетевых расчетов она использует последовательность матриц. То, как код инициализирует их, имеет значение, поскольку вы не можете использовать слишком малые числа — сигнал будет слишком слабым для работы сети. Однако вы также должны избегать слишком больших чисел, поскольку вычисления становятся слишком громоздкими для обработки. Иногда они терпят неудачу, что приводит к *проблеме взрывающегося градиента* (exploding gradient problem) или иначе *насыщение нейронов* (saturation of the neuron), что означает, что вы не можете правильно обучить сеть, поскольку все нейроны всегда активированы.


ЗАПОМНИ!

Инициализация сети с использованием всех нулей всегда плохая идея, поскольку если все нейроны имеют одинаковое значение, они будут одинаково реагировать на ввод обучения. Независимо от того, сколько нейронов содержится в архитектуре, они работают как один нейрон.

Более простое решение заключается в том, чтобы начать со случайных весов, находящихся в диапазоне, необходимом для *функций активации*, которые, по сути, являются функциями преобразования, что добавляет гибкость к решению задач с использованием сети. Возможное простое решение заключается в том, чтобы установить для весов нулевое среднее и единичное стандартное отклонение, которое в статистике называется *стандартным нормальным распределением* (standard normal distribution), а в коде отображается как команда `np.random.radn`.

СОВЕТ

Тем не менее, есть более разумные инициализации весов для более сложных сетей, таких как описаны в этой статье: `https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78`.

Более того, поскольку каждый нейрон получает входные данные всех предыдущих нейронов, код изменяет масштаб случайных нормально распределенных весов, используя, квадратный корень из количества входных данных. Следовательно, нейроны и их функции активации всегда вычисляют правильный размер, чтобы все работало гладко.

Документирование основных модулей

Архитектура является лишь частью нейронной сети. Вы можете представить ее как структуру сети. Архитектура объясняет, как сеть обрабатывает данные и предоставляет результаты. Тем не менее, для любой обработки, вам также необходимо реализовать в коде основные функции нейронной сети.

Первым строительным блоком сети является функция активации. В главе 7 подробно описаны некоторые функции активации, используемые в нейронных сетях, без подробного их объяснения. Пример в этом разделе предоставляет код для сигмоидной функции, одной из основных функций активации нейронной сети. *Сигмоидная функция* (sigmoid function) — это следующий шаг по сравнению со *ступенчатой функцией Хевисайда* (Heaviside step function), которая действует как переключатель, активирующийся при определенном пороге. Ступенчатая функция Хевисайда возвращает значение 1 для вводов превышающих порог и 0 для вводов ниже него.

Сигмоидные функции выводят 0 или 1 соответственно для малых входных значений ниже нуля или высоких значений выше нуля. Для входных значений в диапазоне от –5 до +5 функция выводит значения в диапазоне 0–1, медленно увеличивая выводимые значения до тех пор, пока оно не достигнет примерно 0,2, а затем линейно увеличивая его до значения 0,8. Затем она снова уменьшает вывод по мере приближения к 1. Такое поведение представляет собой логистическую кривую, применимую при описании многих природных явлений, таких как рост популяции, которая начинает рост медленно, а затем линейно развивается, пока ее рост не замедлится при почти полном исчерпании ресурсов (например, доступное жилое пространство или еда).

В нейронных сетях сигмоидная функция особенно полезна для моделирования входных данных, которые похожи на вероятности, и она *дифференцируема*, что является математическим аспектом, помогающим обратить вспять ее эффекты и выработать лучшую фазу обратного распространения, упомянутую в главе 7.

```
def sigmoid(z):
    return 1/(1 + np.exp(-z))

def sigmoid_prime(s):
    return s * (1 -s)
```

Получив функцию активации, вы можете создать *процедуру передачи* (forward procedure), представляющую собой матричное умножение между входными данными для каждого слоя и весами соединения. После завершения умножения код применяет к результатам функцию активации, чтобы преобразовать их нелинейным способом. Следующий код встраивает сигмоидную функцию в код прямой передачи по сети. Конечно, при желании вы можете использовать любую другую функцию активации.

```
def feed_forward(X, weights):
    a = X.copy()
    out = list()
    for W in weights:
        z = np.dot(a, W)
        a = sigmoid(z)
        out.append(a)
    return out
```

Применяя прямую передачу ко всей сети, вы, наконец, получите результат в выходном слое. Теперь вы можете сравнить выходные данные с реальными значениями, которые необходимо получить в сети. Сравнивая количество правильных предположений с общим количеством предоставленных прогнозов, функция точности определяет, хорошо ли нейронная сеть осуществляет прогнозирование.

```
def accuracy(true_label, predicted):
    correct_preds = np.ravel(predicted)==true_label
    return np.sum(correct_preds) / len(true_label)
```

Далее следует функция обратного распространения, поскольку сеть работает, но все или некоторые прогнозы неверны. Исправление прогнозов во время обучения позволяет создать нейронную сеть, способную получать новые примеры и предоставлять хорошие прогнозы. Обучение включается в весовые коэффициенты соединения в виде шаблонов, представленных в данных, и способные помочь правильно предсказать результаты.

Для выполнения обратного распространения, вы сначала вычисляете ошибку в конце каждого слоя (в этой архитектуре их два). Вы умножаете ошибку на производную функции активации. Результат дает вам градиент, то есть изменение весов, необходимое для более правильного вычисления прогнозов. Код начинается со сравнения выходных данных с правильными ответами (`l2_error`),

а затем вычисляет градиенты, являющиеся необходимыми поправками веса (`l2_delta`). Затем код переходит к умножению градиентов на веса, которые код должен исправить. Операция распространяет ошибку из выходного слоя на промежуточный (`l1_error`). Новое вычисление градиента (`l1_delta`) также предоставляет поправки веса для применения к входному слою, который завершает процесс для сети с входным слоем, скрытым слоем и выходным слоем.

```
def backpropagation(l1, l2, weights, y):
    l2_error = y.reshape(-1, 1) - l2
    l2_delta = l2_error * sigmoid_prime(l2)
    l1_error = l2_delta.dot(weights[1].T)
    l1_delta = l1_error * sigmoid_prime(l1)
    return l2_error, l1_delta, l2_delta
```

ЗАПОМНИ!

Это перевод в код Python, в упрощенной форме, формул главы 7. Функция стоимости — это разница между выводом сети и правильными ответами. В этом примере не добавляются смещения фазы прямой передачи, что уменьшает сложность процесса обратного распространения и облегчает понимание.

После того, как обратное распространение назначит каждому соединению свою часть коррекции, которая должна применяться ко всей сети, вы корректируете начальные веса так, чтобы представить обновленную нейронную сеть. Это осуществляется в результате добавления к весам каждого слоя, умножения входных данных для этого слоя и поправок дельта для слоя в целом. Это этап метода градиентного спуска, при котором вы подходите к решению, предпринимая многократные небольшие шаги в правильном направлении, поэтому вам может потребоваться отрегулировать размер шага, используемого для решения задачи. Изменение размера шага позволяют осуществить параметры альфа. Значения 1 не влияет на результат предыдущей коррекции веса, но значения меньше 1 эффективно уменьшат его.

```
def update_weights(X, l1, l1_delta, l2_delta, weights,
    alpha=1.0):
        weights[1] = weights[1] + (alpha * l1.T.dot(l2_delta))
        weights[0] = weights[0] + (alpha * X.T.dot(l1_delta))
        return weights
```

Нейронная сеть не является полной, если она может только учиться на основе данных, но не прогнозировать. Последняя функция, `predict`, выдает новые данные с использованием прямой связи, читает последний выходной слой и преобразует его значения в прогноз задачи. Поскольку сигмоидная функция активации настолько хороша для моделирования вероятности, код использует значение на полпути между 0 и 1, то есть 0,5, в качестве порога для получения

положительного или отрицательного вывода. Такой двоичный вывод может помочь при классификации двух классов или одного класса по отношению ко всем остальным, если набор данных имеет три или более типов результатов для классификации.

```
def predict(X, weights):
    _, l2 = feed_forward(X, weights)
    preds = np.ravel((l2 > 0.5).astype(int))
    return preds
```

На данный момент в примере есть все части, обеспечивающие работу нейронной сети. Вам просто нужна задача, демонстрирующая работу нейронной сети.

Решение простой задачи

В этом разделе вы проверите написанный вами код нейронной сети, попросив его решить простую, но не банальную задачу с данными. В коде для создания двух перемежающихся окружностей точек в форме двух полумесяцев используется функция `make_moons` из пакета Scikit-learn. Разделение этих двух окружностей требует алгоритма, способного определять нелинейную функцию разделения, которая обобщает и новые случаи того же рода. Нейронная сеть, такая как представленная ранее в этой главе, может легко справиться с этой задачей.

```
np.random.seed(0)

coord, cl = make_moons(300, noise=0.05)
X, Xt, y, yt = train_test_split(coord, cl,
                                test_size=0.30,
                                random_state=0)

plt.scatter(X[:,0], X[:,1], s=25, c=y, cmap=plt.cm.Set1)
plt.show()
```

Сначала код устанавливает случайное начальное число для получения одинакового результата при каждом запуске примера. Следующим шагом является создание 300 примеров данных и разделение их на наборы проверочных и тестовых данных. (Набор тестовых данных составляет 30% от общего количества.) Данные состоят из двух переменных, представляющих координаты x и y точек на декартовой плоскости. На рис. 8.1 показаны результаты этого процесса.

Поскольку обучение нейронной сети осуществляется в ходе последовательных итераций (называемых *эпохами*), после создания и инициализации набора весов код проводит 30 000 итераций для данных двух полумесяцев (каждый

проход является эпохой). На каждой итерации код вызывает некоторые из ранее подготовленных базовых функций нейронной сети.

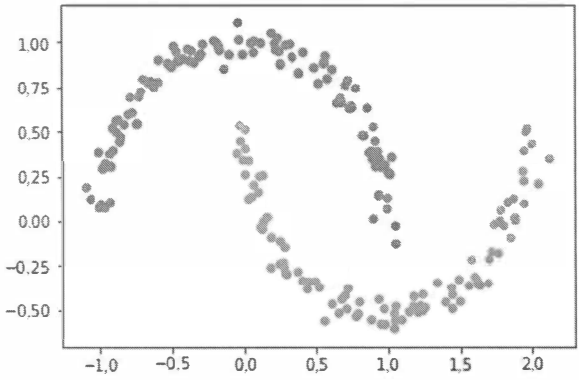


Рис. 8.1. Два чередующихся облака точек данных в форме полумесяца

- Передача данных через всю сеть.
- Обратное распространение ошибки по сети.
- Обновление веса каждого слоя сети, на основании обратного распространения ошибки.
- Вычисление ошибки обучения и проверки.

Следующий код использует комментарии для детализации работы каждой функции.

```
weights = create_architecture(X, 3, 1)

for j in range(30000 + 1):

    # Сначала прямая передача через скрытый слой
    l1, l2 = feed_forward(X, weights)

    # Затем обратное распространение ошибки от выхода к входу
    l2_error, l1_delta, l2_delta = backpropagation(l1,
                                        l2, weights, y)

    # Наконец, обновление весов сети
    weights = update_weights(X, l1, l1_delta, l2_delta,
                             weights, alpha=0.05)

    # Время от времени, сообщать о результатах
    if (j % 5000) == 0:
```

```
train_error = np.mean(np.abs(l2_error))
print('Epoch {:5}'.format(j), end=' - ')
print('error: {:0.4f}'.format(train_error),
      end= ' - ')
train_accuracy = accuracy(true_label=y,
                          predicted=(l2 > 0.5))
test_preds = predict(Xt, weights)
test_accuracy = accuracy(true_label=yt,
                         predicted=test_preds)
print('acc: train {:0.3f}'.format(train_accuracy),
      end= ' | ')
print('test {:0.3f}'.format(test_accuracy))
```

Переменная `j` считает итерации. На каждой итерации код пытается разделить `j` на 5000 по модулю и проверяет результат. Когда модуль равен нулю, код понимает, что с момента предыдущей проверки прошло 5000 эпох. А значит, возможно суммирование ошибки нейронной сети в ходе проверки ее точности (сколько раз прогноз является правильным по отношению к общему количеству прогнозов) на учебном и тестовом наборах. Точность учебного набора показывает, насколько хорошо нейронная сеть подбирает данные, адаптируя их параметры в процессе обратного распространения. Точность тестового набора дает представление о том, насколько хорошо решение обобщено для новых данных, а, следовательно, можно ли использовать его повторно.

СОВЕТ

Точность тестового набора должна иметь наибольшее значение, поскольку она демонстрирует потенциальную применимость нейронной сети с другими данными. Точность обучающего набора просто говорит о том, как сеть оценивает используемые данные.

Внутренняя работа нейронных сетей

Узнав, как нейронные сети работают в основном, вам нужно лучше понять, что их отличает. Существуют различия и помимо архитектур, выбора функций активации, оптимизаторов и скорости обучения нейронной сети. Знания основных операций недостаточно, поскольку вы не получите желаемых результатов. Взгляд изнутри поможет понять, как можно настроить решение нейронной сети для моделирования конкретных задач. Кроме того, понимание различных алгоритмов, используемых для создания нейронной сети, поможет вам быстрее получить лучшие результаты и с меньшими усилиями. Следующие разделы посвящены трем областям отличий нейронных сетей.

Выбор правильной функции активации

Функция активации просто определяет, когда сработает нейрон. Считайте, что это своего рода переломный момент: ввод определенного значения не вызовет срабатывания нейрона, поскольку этого недостаточно, но чуть большее значение ввода может вызвать срабатывание нейрона. Нейрон в простой форме определяется следующим образом.

```
y = Σ (weight * input) + bias
```

Вывод `y` может быть любым значением между + и – бесконечностью. Таким образом, задача заключается в том, чтобы решить, какое значение `y` является значением срабатывания, и где в игру вступает функция активации. Функция активации определяет, какое значение является достаточно высоким или низким, чтобы отразить точку принятия решения в нейронной сети для конкретного нейрона или группы нейронов.

Как и все остальное в нейронных сетях, у вас нет только одной функции активации. Вы используете ту функцию активации, которая лучше всего работает в конкретном случае. С учетом этого, функции активации можно отнести к следующим категориям.

- **Шаговая**. *Шаговая функция* (step function) (или двоичная функция) полагается на определенный порог для принятия решения об активации. Использование шаговой функции означает, что вы знаете, какое конкретное значение вызовет активацию. Однако шаговые функции ограничены тем, что они либо полностью активированы, либо полностью деактивированы — оттенки отсутствуют. Следовательно, при попытке определить на основании заданного ввода, какой класс, скорее всего, правильный, пошаговая функция не будет работать.
- **Линейная**. *Линейная функция* (linear function) ($A = cx$) обеспечивает простое линейное определение активации на основе входных данных. Использование линейной функции помогает определить, какой выход активировать, на основании того, какой выход является наиболее правильным (как указано весами). Однако линейные функции работают только как один слой. Если вы хотите собирать несколько слоев линейных функций, результат будет как при использовании одного слоя, что противоречит цели использования нейронных сетей. Следовательно, линейная функция может использовать один слой, но не несколько.
- **Сигмоидная**. *Сигмоидная функция* (sigmoid function) ($A = 1 / 1 + e^{-x}$), создающая кривую в форме буквы C или S, является нелинейной. Сначала она выглядит как шаговая функция, за исключением того, что значения между двумя точками фактически расположены на

кривой, а значит сигмоидные функции можно располагать слоями для выполнения классификации с несколькими выходами. Диапазон сигмоидной функции распространяется от 0 до 1, а не от – бесконечности до + бесконечности, как с линейной функцией, поэтому активации ограничены в определенном диапазоне. Однако сигмоидная функция страдает от проблемы *исчезающего градиента* (vanishing gradient), когда функция отказывается учиться после определенной точки, поскольку распространяемая ошибка уменьшается до нуля при приближении к удаленным слоям.

- **Tanh**. *Функция tanh* (`A = (2 / 1 + e^-2x) - 1`) на самом деле является масштабной сигмоидной функцией. Она имеет диапазон от –1 до 1, поэтому это точный метод активации нейронов. Основная разница между сигмоидными функциями и функциями tanh в том, что градиент функции tanh сильнее, а значит обнаружение небольших различий легче, что делает классификацию более чувствительной. Подобно сигмоидной функции, функция tanh страдают от исчезающего градиента.
- **ReLU**. *Функция ReLU* или *линейный выпрямитель* (Rectified Linear Units) (`A(x) = max(0, x)`) обеспечивает вывод в диапазоне от 0 до бесконечности, поэтому она похожа на линейную функцию, за исключением того, что она также нелинейна, что позволяет наслаивать функции ReLU. Преимущество ReLU в том, что она требует меньше вычислительной мощности, потому что срабатывает меньше нейронов. Отсутствие активности, когда нейрон приближается к нулевой части линии, означает, что слишком мало потенциальных выходов для рассмотрения. Тем не менее, это преимущество также может стать недостатком, когда у вас есть проблема *затухающего ReLU* (dying ReLU). Через некоторое время веса нейронной сети уже не дают желаемого эффекта (она просто прекращает обучение), и пораженные нейроны умирают — они не реагируют ни на какие входные данные.
- **ELU**. *Экспоненциальная линейная функция* (Exponential Linear Unit). Отличается от ReLU, когда вводы отрицательны. В этом случае выходные данные не стремятся к нулю, а постепенно экспоненциально уменьшаются до –1.
- **PReLU**. *Параметрический линейный выпрямитель* (Parametric Rectified Linear Unit). Отличается от ReLU, когда вводы отрицательны. В этом случае вывод является линейной функцией, параметры которой изучаются с использованием той же методики, что и любой другой параметр сети.
- **LeakyReLU**. Аналогичен PReLU, но параметр для линейной стороны фиксирован.

Полагаясь на умный оптимизатор

Оптимизатор обеспечивает быстрое и правильное моделирование нейронной сетью любой задачи, которую вы хотите решить, изменяя смещения и веса нейронной сети. Оказывается, что эту задачу выполняет алгоритм, но вы должны выбрать правильный алгоритм, чтобы получить ожидаемые результаты. Как и во всех случаях нейронной сети, у вас есть несколько необязательных типов алгоритмов, из которых можно выбирать (см. `https://keras.io/optimizers/`).

- Стохастический градиентный спуск (SGD).
- RMSProp.
- AdaGrad.
- AdaDelta.
- AMSGrad.
- Adam и его варианты, Adamax и Nadam.

Оптимизатор работает минимизируя или максимизируя выходные данные целевой функции (функции ошибок), представленной как E(x). Эта функция зависит от внутренних обучаемых параметров модели, используемых для вычисления целевых значений (Y) из предикторов (X). Двумя внутренними обучаемыми параметрами являются веса (W) и смещения (b). Различные алгоритмы имеют разные методы работы с целевой функцией.

Вы можете классифицировать функции оптимизатора по тому, как они работают с производной (dy/dx), которая представляет собой мгновенное изменение y относительно x. Вот два уровня обработки производных.

- **Первый порядок**. Эти алгоритмы минимизируют или максимизируют целевую функцию, используя значения градиента по отношению к параметрам.
- **Второй порядок**. Эти алгоритмы минимизируют или максимизируют объектную функцию, используя значения производных второго порядка по параметрам. Производная второго порядка может дать подсказку о том, увеличивается или уменьшается производная первого порядка, что дает информацию о кривизне линии.

Вы обычно используете методы оптимизации первого порядка, такие как Gradient Descent, поскольку они требуют меньше вычислений и имеют тенденцию относительно быстро сходиться к хорошему решению при работе с большими наборами данных.

Установка рабочей скорости обучения

Каждый оптимизатор имеет совершенно разные параметры для настройки. Одной из констант является фиксация скорости обучения, представляющей собой скорость, с которой код обновляет веса в сети (например, параметр `alpha`, используемый в примере этой главы). Скорость обучения может влиять как на время, необходимое нейронной сети для получения хорошего решения (количество эпох), так и на результат. На самом деле, если скорость обучения слишком низка, сеть будет учиться вечно. Установка слишком высокого значения приводит к нестабильности при обновлении весов, и сеть никогда не сможет найти хорошее решение.

Выбор рабочей скорости обучения является пугающим потому, что вы можете опробовать значения в диапазоне от 0,000001 до 100. Лучшее значение варьируется от оптимизатора к оптимизатору. Значение, которое вы выбираете, зависит от того, какой тип данных у вас есть. Теория может здесь мало помочь; вы должны проверить различные комбинации, прежде чем найдете наиболее подходящую скорость для успешного обучения вашей нейронной сети.

ЗАПОМНИ!

Несмотря на всю окружающую их математику, настройка нейронных сетей и обеспечение их наилучшей работы — это в основном вопрос эмпирических усилий при опробовании различных комбинаций архитектур и параметров.

Глава 9

Переход к глубокому обучению

В ЭТОЙ ГЛАВЕ...

- Понятие источников и использования данных
- Ускорение обработки данных
- Разница в глубоком обучении
- Выработка наилучших решений глубокого обучения

В главах 7 и 8 искусственный интеллект рассматривается с точки зрения машинного обучения, и с небольшой дополнительной информацией о глубоком обучении. Эта глава посвящена исключительно глубокому обучению, поскольку на самом деле вам нужны решения глубокого обучения, чтобы справиться с современным переизбытком данных. Хотя машинное обучение добавляет в арсенал искусственного интеллекта возможность обучения, важно с самого начала осознать, что у компьютеров есть ограничения — на самом деле они не понимают, что делают люди. Все контролируют алгоритмы, являющиеся математическим представлением различных процессов интерпретации данных. Итак, первая часть этой главы рассматривает данные с точки зрения глубокого обучения, поскольку для эффективного поиска шаблонов вам нужны огромные объемы данных.

По мере перехода от искусственного интеллекта к машинному обучению и глубокому обучению вычислительные запросы возрастают. Фактически, одной из основных причин зим искусственного интеллекта в прошлом была нехватка вычислительной мощности. Сегодня вы можете использовать графические

процессоры, такие как NVIDIA Titan V (`https://www.nvidia.com/en-us/titan/titan-v/`), с 5120 ядрами Compute Unified Device Architecture (CUDA) для обработки данных такими способами, которые были невозможны даже несколько лет назад. Поэтому во второй части этой главы обсуждается, как вы можете улучшить свой опыт глубокого обучения, используя больше оборудования или другие стратегии, применяемые в настоящее время аналитиками данных (среди многих других).

Третья часть главы посвящена именно тому, насколько глубокое обучение отличается от машинного обучения — эта разница является постоянным источником проблем для многих людей. Найти точное определение, с которым каждый может согласиться, практически невозможно, поэтому, если вы уже являетесь экспертом по глубокому обучению, вы можете не полностью согласиться со всем, что изложено в этой главе. Несмотря на это, данная книга для представления принципов и примеров глубокого обучения опирается на свое определение, поэтому вам необходимо знать способ определения глубокого обучения в этой книге.

Наконец, в четвертой части главы все основные элементы, упомянутые в первых трех частях, рассматриваются более подробно. Вы начинаете понимать, что глубокое обучение бывает во многих формах и что некоторые формы особенно подходят для решения конкретных проблем. В настоящее время не существует единого решения для любой задачи, поэтому знание правильного набора возможных решений для конкретной задачи может сэкономить вам много времени и позволит избежать разочарований.

Данные везде

Большие данные (big data) — это не просто модное слово, используемое производителями для предложения новых способов хранения данных и их анализа. Революция больших данных — это повседневная реальность и движущая сила нашего времени. Возможно, вы слышали о больших данных, упоминаемых во многих специализированных научных и деловых изданиях, и задавались вопросом, что на самом деле означает этот термин. С технической точки зрения *большие данные* относятся к большим и сложным объемам компьютерных данных, настолько большим и сложным, что приложения не могут обрабатывать их, даже используя дополнительное хранилище или увеличивая мощность компьютера. Следующие разделы помогут вам понять, что сегодня делает данные универсальным ресурсом.

Учет влияния структуры

Большие данные подразумевают революцию в хранении и манипулировании данными. Это влияет на то, что вы можете достичь с помощью данных в более качественном выражении (это значит, что, помимо выполнения б*о*льших задач, вы можете выполнять и лучше). Компьютеры хранят большие данные в разных форматах с человеческой точки зрения, но компьютер видит данные как поток единиц и нулей (внутренний язык компьютеров). Вы можете просматривать данные в одной из двух форм, в зависимости от того, как вы их производите и используете.

- **Структурированные**. Вы точно знаете, что содержат данные и где найти каждый их фрагмент. Типичными примерами структурированных данных являются таблицы базы данных, в которых информация организована в столбцы, а каждый столбец содержит определенный тип информации. Как правило, данные структурированы по своей конструкции. Вы собираете их выборочно и записываете на правильное место. Например, вы можете захотеть поместить количество людей, покупающих определенный товар, в определенном столбце определенной таблицы конкретной базы данных. Как и в случае с библиотекой, если вы знаете, какие данные вам нужны, вы можете найти их немедленно.
- **Неструктурированные**. У вас есть представление о том, что содержат данные, но вы не знаете точно, как они устроены. Типичными примерами неструктурированных данных являются изображения, видео и звуковые записи. Вы можете использовать неструктурированную форму для текста, чтобы пометить ее такими характеристиками, как размер, дата или тип содержимого. Обычно вы не знаете точно, где расположены данные в неструктурированном наборе данных, поскольку данные представляют собой последовательности единиц и нулей, которые приложение должно интерпретировать или визуализировать.


ЗАПОМНИ!

Преобразование неструктурированных данных в структурированную форму может потребовать много времени и усилий и может потребовать труда многих людей. Б*о*льшая часть данных, созданных до революции больших данных неструктурированна и хранится в том виде, в каком она есть, если только кто-то не предоставит ее в структурированной форме.

Это огромное и сложное хранилище данных не появилось внезапно за одну ночь. Разработка технологии хранения такого количества данных потребовала времени. Распространение технологии, способной создавать и предоставлять

данные, а именно компьютеров, датчиков, смартфонов, Интернета и его сервисов World Wide Web, также потребовало времени.

Следствия закона Мура

В 1965 году Гордон Мур (Gordon Moore), соучредитель Intel и Fairchild Semiconductor, написал в статье *Cramming More Components Onto Integrated Circuits* (`https://ieeexplore.ieee.org/document/4785860/`), что количество компонентов в интегрированных схемах будут удваиваться каждый год в течение следующего десятилетия. На тот момент в электронике доминировали транзисторы. Возможность встраивать больше транзисторов в интегральную микросхему (Integrated Circuit — IC) означала возможность сделать электронные устройства более функциональными и полезными. Этот процесс называется *интеграцией* и подразумевает процесс сильный миниатюризации электроники (делающей ту же схему намного меньше). Современные компьютеры не намного меньше, чем компьютеры десятилетия назад, но все же они значительно мощнее. То же самое касается мобильных телефонов. Несмотря на то, что они того же размера, что и их предшественники, они способны выполнять больше задач.

То, что Мур заявил в той статье, действительно было правдой на протяжении многих лет. Полупроводниковая отрасль называет это законом Мура (см. подробнее на `http://www.mooreslaw.org/`). Удвоение происходило в течение первых десяти лет, как и предсказывалось. В 1975 году Мур исправил свое заявление, прогнозируя удвоение каждые два года. На рис. 9.1 показан эффект этого удвоения. Указанный коэффициент удвоения остается в силе, хотя в настоящее время распространено мнение, что оно не сохранится после конца нынешнего десятилетия (примерно до 2020 года). Начиная с 2012 года, между ожидаемым увеличением скорости и достижениями полупроводниковых компаний в отношении миниатюризации, началось несоответствие.

Существуют физические барьеры для интеграции большего количества схем в интегральные микросхемы с использованием существующих кремниевых компонентов, поскольку вы не можете сделать их очень маленькими. Однако инновации продолжаются, как описано на сайте `https://www.nature.com/news/the-chipsare-down-for-moores-law-1.19338`. В будущем закон Мура может не выполняться. Это произойдет потому, что отрасль перейдет на новую технологию, такую как создание компонентов с использованием оптических лазеров вместо транзисторов (см. информацию об оптических вычислениях на `https://www.extremetech.com/extreme/187746-by-2020-you-could-havean-exascale-speed-of-light-optical-computeron-your-desk`). В конце концов, закон Мура перестанет выполняться, поскольку промышленность не сможет идти в ногу с такими темпами, как это

было в прошлом (см. статью в MIT Technology Review по адресу `https://www.technologyreview.com/s/601441/mooreslaw-is-dead-now-what/`).

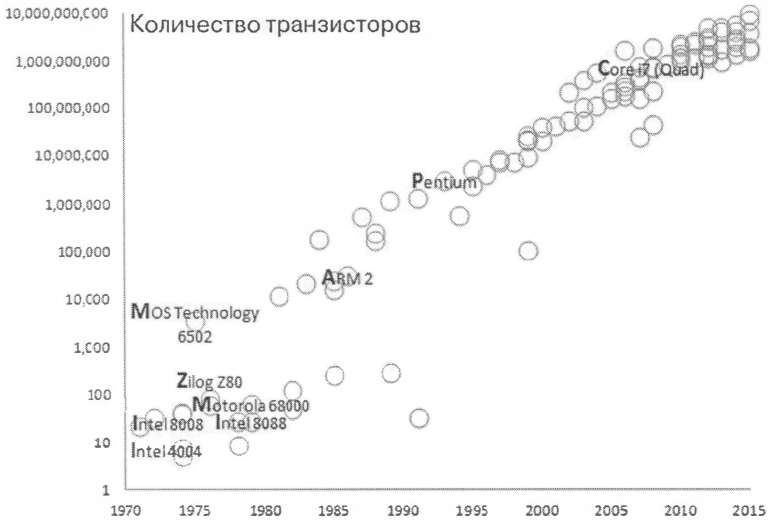


Рис. 9.1. Рост количества транзисторов в процессоре

Что меняет закон Мура

Что важно для аналитиков данных и других, заинтересованных в глубоком обучении, так это то, что с 1965 года удвоение компонентов каждые два года привело к значительным достижениям в области цифровой электроники, которые имели далеко идущие последствия в приобретении, хранении, манипулировании и управлении данными.

Закон Мура напрямую влияет на данные, начиная с более умных устройств. Чем умнее устройство, тем больше людей полагаются на него, при взаимодействии с данными новыми способами (о чем свидетельствует электроника, которая сегодня повсюду). Чем больше распространение вычислительной мощи, тем ниже становится ее цена, что создает бесконечный цикл, стимулирующий использование мощных вычислительных машин и небольших датчиков повсюду. В связи с наличием большого объема компьютерной памяти и большими дисками для хранения данных, последствием становится расширение доступности данных, таких как веб-сайты, записи транзакций, измерения, цифровые изображения и другие виды данных. Без этих достижений современный

Интернет не был бы возможен, поскольку его поток данных зависит от таких умных устройств.

Благодаря компьютерам, мобильным устройствам и связанным с ними датчиками, Интернет теперь генерирует и распространяет новые данные в больших количествах. Некоторые источники оценивают текущее ежедневное производство данных примерно в 2,5 квинтиллиона (число с 18 нулями), причем львиная доля приходится на неструктурированные данные, такие как видео и аудио (см. подробности на `https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/`). Большая часть этих данных относится к обычной человеческой деятельности, чувствам, опыту и отношениям, сопровождаемым растущей долей данных о функционировании подключенных машин, которые варьируются от сложных промышленных механизмов до простых умных домашних ламп (ламп, которыми вы можете управлять дистанционно с помощью Интернет).

Преимущества дополнительных данных

С ростом доступности данных на цифровых устройствах, данные приобретают все новые нюансы с точки зрения их ценности и полезности, выходящие за рамки первоначального объема информации и передачи знаний (передачи данных). Обилие данных, когда речь идет об их анализе, приобретает новое значение, отличное от просто информации.

- Данные лучше описывают мир, представляя множество фактов и более подробно описывая нюансы каждого факта. Они стали настолько обильными, что охватывают все аспекты реальности. Вы можете использовать их для выяснения того, как даже, казалось бы, явно не связанные вещи и факты на самом деле связаны друг с другом.
- Данные показывают, как факты связаны с событиями. Вы можете вывести общие правила и узнать, как будет меняться мир с учетом определенных предпосылок. Когда люди действуют определенным образом, данные также обеспечивают определенную возможность прогнозирования.

В следующих разделах обсуждается, почему лучше иметь больше данных. Имея больше данных для работы, ваш проект глубокого обучения может стать более точным, надежным и, в некоторых случаях, даже осуществимым.

Последствия больших данных

В некоторых отношениях данные предоставляют нам новые суперспособности. Крис Андерсон (Chris Anderson), предыдущий главный редактор журнала *Wired*, рассказывает, как большие объемы данных могут помочь научным открытиям вне научного метода (см. статью `https://www.wired.com/2008/06/pb-theory/`). Автор приводит пример достижений Google в бизнес-секторах рекламы и перевода, в которых Google добился известности не благодаря использованию конкретных моделей или теорий, а скорее благодаря применению алгоритмов для изучения данных.

Как и в рекламе, научные данные (например, из физики или биологии) могут поддерживать инновации, позволяющие ученым подходить к проблемам без гипотез, рассматривая вместо этого различия, обнаруженные в больших объемах данных, и используя алгоритмы их обнаружения. Галилео Галилей опирался на научный метод для создания основ современной физики и астрономии (см. `https://www.biography.com/people/galileo-9305220`). Самые ранние достижения основаны на наблюдениях и контролируемых экспериментах, которые определяют причины того, как и почему все происходит. Способность вводить новшества, используя одни только данные, является главным прорывом в нашем понимании мира.

В прошлом ученые проводили бесчисленные наблюдения и делали множество выводов, чтобы описать физику Вселенной. Этот ручной процесс позволил людям найти основополагающие законы мира, в котором мы живем. Анализ данных в результате объединения наблюдений, выраженных в виде входных и выходных данных, позволяет определить, как все работает, а благодаря глубокому обучению выявить приблизительные правила или законы, мира, не прибегая к использованию ручных наблюдений и выводов. Процесс теперь протекает быстрее и более автоматически.

Своевременность и качество данных

Данные не просто способствуют глубокому обучению, а делают его возможным. Некоторые люди скажут, что глубокое обучение является результатом сложных алгоритмов повышенной математической сложности, и это, безусловно, верно. Такие действия, как зрение и понимание человеческого языка, требуют таких алгоритмов, которые нелегко объяснить с точки зрения непрофессионала, и для работы требуются миллионы вычислений. (Здесь тоже играет роль аппаратное обеспечение.)

Глубокое обучение — это больше, чем алгоритмы. Доктор Александер Уисснер-Гросс (Alexander Wissner-Gross), американский исследователь, предприниматель и научный сотрудник Института прикладной информатики в

Гарварде, предлагает свои взгляды на глубокое обучение в недавнем интервью *Edge* (`https://www.edge.org/response-detail/26587`). Интервью рассказывает, почему технология глубокого обучения заняла так много времени. Уисснер-Гросс заключает, что ключевыми факторами могли быть качество и доступность данных, а не просто доступность алгоритмов. Другими словами, наличие мощных алгоритмов необходимо, но недостаточно, если у вас нет необходимых данных для их работы.

Уисснер-Гросс рассматривает время, необходимое для самых значительных достижений в области глубокого обучения за последние годы, и показывает, как данные и алгоритмы способствуют успеху каждого прорыва, и подчеркивает, что каждое из них было новостью на момент, когда сообщество искусственного интеллекта достигало определенного рубежа. Уисснер-Гросс показывает, что данные относительно новые и всегда обновляются, в то время как алгоритмы не являются новыми открытиями, а скорее полагаются на консолидацию старых технологий.

Например, если учесть недавние достижения в области глубокого обучения, почти нечеловеческая производительность сети GoogleLeNet по правильной классификации образов основывается на старом алгоритме, запущенном на последних данных. Для визуального распознавания она использует алгоритм Convolutional Neural Networks, разработанный в 1989 году, и который может показать свою реальную эффективность только после обучения с использованием корпуса ImageNet (`http://www.image-net.org/`) из более чем 1,5 миллиона популярных изображений более 1000 категорий (корпус ImageNet стал доступен в 2010 году).

Еще одно достижение, которое стоит рассмотреть, — это результат работы команды Google DeepMind. Команда развернула глубокую нейронную сеть, которая достигла в играх того же уровня мастерства, что и люди, играя в 29 различных игр Atari. Они опирались на алгоритм 1992 года Q-Learning, который они смогли применить к играм Atari только после 2013 года, когда сверточные нейронные сети стали более распространенными, и полный набор данных из 50 игр Atari 2600, называемый Arcade Learning Environment (`https://github.com/mgbellemare/Arcade-Learning-Environment`).

Уисснер-Гросс приводит и другие примеры такого же глубокого обучения, например, когда IBM Deep Blue одержала победу над Гарри Каспаровым и когда IBM Watson стала чемпионом мира по *Jeopardy!*. Во всех этих случаях Уисснер-Гросс приходит к выводу, что алгоритм в среднем обычно на 15 лет старше данных. Он указывает, что данные продвигают достижения глубокого обучения и заставляют читателя задуматься над тем, что может произойти, если получится снабдить имеющиеся в настоящее время алгоритмы лучшими данными с точки зрения качества и количества.

Ускорение обработки

Заглянув внутрь глубокого обучения, вы можете быть удивлены, обнаружив множество старых технологий, но удивительно, что все работает так, как никогда раньше. Поскольку исследователи наконец-то выяснили, как заставить работать несколько старых добрых решений, большие данные позволяют автоматически фильтровать, обрабатывать и преобразовывать данные. Например, новые функции активации, такие как ReLU, не так уж новы; они известны со времен перцептрона (датируемого 1957 годом; см. главу 7).

Возможности распознавания образов, которые изначально делали глубокое обучение столь популярным, также не новы. Первоначально, глубокое обучение достигло большого успеха благодаря *сверточным нейронным сетям* (Convolutional Neural Network — CNN). Изобретенные в 1980-х годах французским ученым Яном Лекуном (Yann LeCun) (чья личная домашняя страница находится по адресу `http://yann.lecun.com/`), такие сети в настоящее время дают удивительные результаты, поскольку используют много нейронных слоев и большое количество данных.

То же самое относится к технологии, позволяющей машине понимать человеческую речь или переводить с одного языка на другой. В любом случае, решение опирается на десятилетнюю технологию, к которой вновь обратились исследователи и приступили к работе в новой парадигме глубокого обучения. Единственная проблема заключается в том, что вся эта обработка данных требует большого количества циклов, поэтому в следующих разделах обсуждается, как повысить скорость обработки, чтобы вы могли увидеть результат анализа данных за разумное время.

Использование мощного оборудования

Использование невероятных объемов данных влияет на производительность алгоритма. Для обработки такого большого количества данных и ускорения получения ответов ученые разных школ полагаются на более широкое использование графических процессоров и компьютерных сетей. Наравне с параллелизмом (большое количество компьютеров размещается в кластерах и работает параллельно), графические процессоры позволяют создавать большие сети и успешно обучать их большему количеству данных. Фактически, графический процессор может выполнять определенные операции в 70 раз быстрее любого центрального процессора, что позволяет сократить время обучения нейронных сетей с недель до дней или даже часов. (В статье на `https://www.quora.com/Why-are-CPUs-still-being-made-when-GPUs-are-so-much-faster`

рассказывается, почему для создания эффективных систем глубокого обучения вам нужны как CPU, так и GPU.).

Графические процессоры — это мощные вычислительные блоки для матричного и векторного вычисления, необходимого для обратного распространения. Эти технологии делают обучение нейронных сетей осуществимым за более короткие сроки и доступным для большего количества людей. Исследования открыли также мир новых приложений. Нейронные сети могут извлекать уроки из огромных объемов данных и использовать преимущества больших данных (изображений, текста, транзакций и данных социальных сетей), создавая модели, которые неуклонно работают лучше, в зависимости от передаваемых им потоков данных.

Для получения дополнительной информации о том, насколько GPU может расширить возможности машинного обучения при использовании нейронной сети, ознакомьтесь с техническим документом по этой теме: `https://icml.cc/2009/papers/218.pdf`.

Другие инвестиции

Крупные игроки, такие как Google, Facebook, Microsoft и IBM, заметили новую тенденцию и с 2012 года начали приобретать компании и нанимать экспертов в новых областях глубокого обучения. Двое из этих экспертов — Джеффри Хинтон (Geoffrey Hinton), более известный своей работой по применению алгоритма обратного распространения для многослойных нейронных сетей и в настоящее время работающий в Google, и Ян Лекун, создатель сверточных нейронных сетей, который сейчас возглавляет исследование искусственного интеллекта в Facebook.

Сегодня каждый может получить доступ к сетям, и люди могут получить доступ к инструментам, которые также помогают создавать сети глубокого обучения. Это выходит за рамки чтения общедоступных научных работ, объясняющих, как работает глубокое обучение; это относится также к инструментам для программирования сетей.

В первые дни глубокого обучения ученые создавали каждую сеть с нуля, используя такие языки, как C++. К сожалению, разработка приложений на таком низкоуровневом языке ограничивает доступ к данным несколькими хорошо подготовленным специалистами. Возможности сценариев сегодня (например, с использованием Python `https://www.python.org`) куда выше благодаря большому массиву инфраструктур глубокого обучения с открытым исходным кодом, таких как TensorFlow от Google (`https://www.tensorflow.org/`) или PyTorch от Facebook (`https://pytorch.org/`). Эти инфраструктуры

позволяют воспроизводить самые последние достижения в области глубокого обучения с использованием простых команд.

Отличие глубокого обучения от других форм искусственного интеллекта

Учитывая разнообразие препятствий, связанных с искусственным интеллектом в целом, таких как большие объемы данных, новое мощное вычислительное оборудование, доступное каждому, и множество частных и государственных инвестиций, вы можете скептически относиться к технологии *глубокого обучения*, которая состоит из нейронных сетей, имеющих больше нейронов и скрытых слоев, чем в прошлом. Глубокие сети контрастируют с более простыми, меньшими сетями прошлого, имевшими, в лучшем случае, один или два скрытых слоя. Многие решения, делающие возможным глубокое обучение сегодня, вовсе не новы, но глубокое обучение использует их по-новому.

ЗАПОМНИ!

Глубокое обучение — это не просто ребрендинг старой технологии, перцептрона, открытого в 1957 году Фрэнком Розенблаттом в Корнелльской лаборатории аэронавтики (более подробную информацию о перцептроне см. в главе 7). Глубокое обучение работает лучше благодаря дополнительным усложнениям, полученным благодаря использованию более мощных компьютеров и доступности более качественных (а не просто больших) данных. Глубокое обучение также подразумевает глубокое качественное изменение возможностей, предлагаемых технологией, наряду с новыми и удивительными приложениями. Наличие этих возможностей модернизирует старые, но добрые нейронные сети, превращая их в нечто новое. В следующих разделах описывается, как глубокое обучение достигает своей цели.

Добавление большего количества слоев

Вы можете задаться вопросом, почему глубокое обучение расцвело только сейчас, когда использованная в его основе технология существовала уже давно. Как упоминалось ранее в этой главе, компьютеры сегодня мощнее, и глубокое обучение может получить доступ к огромным объемам данных. Но эти ответы указывают только на важные проблемы с глубоким обучением в прошлом, и более низкая вычислительная мощь наряду с меньшим объемом данных не были единственными непреодолимыми препятствиями. До недавнего времени

глубокое обучение страдало также от серьезной технической проблемы, которая не позволяла нейронным сетям иметь достаточно слоев для решения действительно сложных задач.

Поскольку глубокое обучение может использовать много слоев, оно способно решать проблемы, недоступные машинному обучению, такие как распознавание образов, машинный перевод и распознавание речи. При наличии всего нескольких слоев, нейронная сеть является идеальным *универсальным аппроксиматором функций* (universal function approximator), представляющим собой систему, способную воссоздать любую возможную математическую функцию. При наличии еще большего количества слоев нейронная сеть становится способной создавать внутри своей цепи умножения матриц сложную систему представлений для решения сложных задач. Чтобы понять, как выполняется такая сложная задача, как распознавание образов, рассмотрим этот процесс.

1. Система глубокого обучения, обученная распознавать образы (например, сеть, способная отличать фотографии собак от фотографий кошек), определяет внутренние веса, способные распознавать содержимое изображений.
2. После обнаружения каждого отдельного контура и угла на изображении, сеть глубокого обучения объединяет все такие основные черты в составные характерные признаки.
3. Сеть сопоставляет такие признаки с идеальным представлением, обеспечивающим ответ.

Другими словами, сеть глубокого обучения может отличать собак от кошек, используя ее внутренние веса для определения вида, на который в идеале должны походить собака и кошка. Затем он использует эти внутренние веса для выяснения соответствия любому новому изображению, которое вы ему предоставляете.

ЗАПОМНИ!

Одним из самых ранних достижений глубокого обучения, благодаря которому общественность осознала его потенциал, является *кошачий нейрон* (cat neuron). Команда Google Brain, возглавляемая в то время Эндрю Ыном (Andrew Ng) и Джеффом Дином (Jeff Dean), собрала 16 000 компьютеров, чтобы рассчитать сеть глубокого обучения с более чем миллиардом весов, что позволяет обучаться на видео YouTube без учителя. Компьютерная сеть может даже сама определить, без вмешательства человека, что такое кошка, и ученым Google удалось выяснить у сети ее представление о том, как по ее мнению должна выглядеть кошка (см. статью *Wired* на `https://www.wired.com/2012/06/google-x-neural-network/`).

В свое время, когда ученые не могли объединить достаточно слоев в нейронную сеть из-за ограничений компьютерного оборудования, потенциал технологии оставался скрытым, а ученые игнорировали нейронные сети. Отсутствие успеха добавило глубокого скептицизма, возникшего вокруг технологии в ходе последней зимы искусственного интеллекта. Но что действительно мешало ученым создать нечто более сложное, так это проблема исчезающего градиента.

Исчезающий градиент (vanishing gradient) возникает, когда вы пытаетесь передать сигнал через нейронную сеть, но сигнал быстро затухает до почти нулевых значений; он не может пройти через функции активации. Это происходит потому, что нейронные сети являются цепочками умножения. Каждое умножение на значение ниже нуля быстро уменьшает входящие значения, а функции активации должны иметь достаточно большие значения, чтобы сигнал проходил. Чем дальше от выхода находятся слои нейронов, тем выше вероятность того, что они будут заблокированы обновлениями, поскольку сигналы слишком малы и функции активации их остановят. Следовательно, ваша сеть будет обучается невероятно медленно или перестает учиться вообще.

Каждая попытка собрать и протестировать сложные сети заканчивалась неудачей потому, что алгоритм обратного распространения не мог обновлять слои ближе к входным, что делало практически невозможным любое обучение на сложных данных, даже когда такие данные были доступны в то время. Сегодня глубокие сети возможны благодаря исследованиям ученых из Университета Торонто в Канаде, таких как Джеффри Хинтон (Geoffrey Hinton) (`https://www.utoronto.ca/news/artificial-intelligence-u-t`), которые настояли на работе над нейронными сетями, даже когда они казались большинству устаревшим подходом машинного обучения.

Профессор Хинтон, ветеран области нейронных сетей (он участвовал в разработке алгоритма обратного распространения), и его команда в Торонто разработали несколько методов решения проблемы исчезающего градиента. Он открыл поле для переосмысления и новых решений, которые сделали нейронные сети важнейшим инструментом машинного обучения и искусственного интеллекта.


ЗАПОМНИ!

Профессор Хинтон и его команда также запомнились тем, что были одними из первых, кто проверил использование графического процессора для ускорения обучения глубокой нейронной сети. В 2012 году они выиграли открытый конкурс, организованный фармацевтической компанией Merck and Kaggle (последний веб-сайт для соревнований по науке о данных) с использованием своих последних открытий в области глубокого обучения. Это событие

привлекло большое внимание к их работе. Вы можете прочитать все детали революционного достижения команды Хинтона в его интервью на `http://blog.kaggle.com/2012/11/01/deep-learning-how-i-did-it-merck-1stplace-interview/`.

Изменение активаций

Команда Джеффри Хинтона (см. предыдущий раздел) смогла добавить больше слоев в нейронную архитектуру благодаря двум решениям, которые предотвратили проблемы с обратным распространением.


СОВЕТ

- Они решили проблему взрывающегося градиента с помощью более умной инициализации сети. *Взрывающийся градиент* (exploding gradient) отличается от исчезающего градиента тем, что он может взорвать сеть, поскольку градиент становится слишком большим для обработки.
- Ваша сеть может взорваться, если не инициализировать ее правильно и предотвратить вычисление больших весовых чисел. Затем вы решаете проблему исчезающего градиента, изменяя активацию сети.
- Команда поняла, что прохождение сигнала через различные слои активации имеет тенденцию ослаблять сигнал обратного распространения, пока он не станет слишком слабым и пропадет совсем после изучения сигмоидной функцией активации. В качестве решения этой проблемы они использовали новую функцию активации. Выбор используемого алгоритма пришелся на старый тип активации ReLU, или линейный выпрямитель (см. о ReLU подробнее в главе 7). Активация ReLU остановит принятый сигнал, если он ниже нуля, гарантируя нелинейную характеристику нейронных сетей и пропустит сигнал как есть, если он выше нуля. (Использование этого типа активации является примером объединения старой, но все еще хорошей технологии с современной технологией.) На рис. 9.2 показано, как работает этот процесс.

Функция ReLU работала невероятно хорошо и позволяла сигналу обратного распространения достигать самых начальных слоев сети. Когда сигнал положительный, его производная равна 1. Вы также можете найти подтверждение производной ReLU на рис. 9.2. Обратите внимание, что скорость изменения постоянна и эквивалентна единице, когда входной сигнал положительный (тогда как при отрицательном сигнале производная равна 0, что препятствует прохождению сигнала).

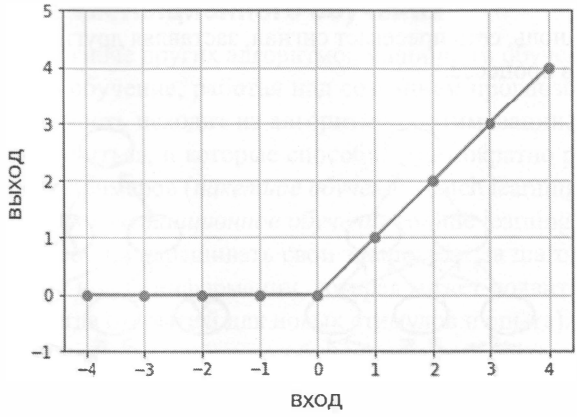


Рис. 9.2. Функция активации ReLU работает, получая и испуская сигналы

СОВЕТ

Вы можете рассчитать функцию ReLU, используя формулу `f(x)=max(0,x)`. Использование этого алгоритма значительно увеличило скорость обучения, позволив быстро обучаться даже очень глубоким сетям без каких-либо мертвых нейронов. *Мертвый нейрон* (dead neuron) — это нейрон, который сеть не может активировать, поскольку сигналы слишком слабы.

Добавление регуляризации в ходе отсева

Другое нововведение в глубокое обучение, сделанное командой Хинтона (см. предыдущие разделы в этой главе), заключалось в улучшении первоначального решения глубокого обучения за счет регуляризации сети. *Регуляризованная сеть* (regularized network) ограничивает веса сети, что не позволяет ей запоминать входные данные и обобщать очевидные шаблоны данных.

Предыдущие обсуждения в этой главе отмечают, что некоторые нейроны запоминают конкретную информацию и заставляют другие нейроны полагаться на этот более сильный нейрон, в результате чего слабые нейроны сами перестают учиться чему-либо полезному (ситуация, называемая *коадаптацией* (co-adaptation)). Для предотвращения коадаптации, код временно отключает активацию случайной части нейронов в сети.

Как видно на левой части рис. 9.3, для активаций веса обычно работают в ходе умножения входов на выходы. Чтобы отключить активацию, код умножает маску, составленную из случайного сочетания единиц и нулей, на результаты.

Если нейрон умножается на единицу, сеть передает свой сигнал. Когда нейрон умножается на ноль, сеть пресекает сигнал, заставляя другие нейроны не полагаться на него в процессе.

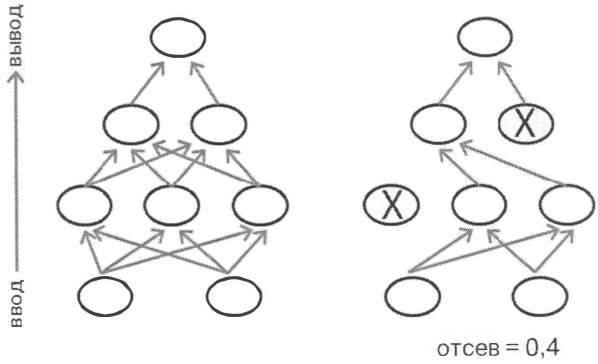


Рис. 9.3. Отсев временно исключает из обучения 40% нейронов


ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Отсев действует только во время обучения, и не воздействует ни на какие части весов. Он просто маскирует и скрывает часть сети, заставляя немаскированную часть играть более активную роль в изучении шаблонов данных. Во время прогнозирования отсев не работает, и веса пересчитываются численно, чтобы учесть тот факт, что они не работали все вместе во время обучения.

Поиск еще более разумных решений

Глубокое обучение существенно повлияло на эффективность искусственного интеллекта в решении проблем распознавания образов, машинного перевода и распознавания речи. Эти задачи были первоначально решены с помощью классического искусственного интеллекта и машинного обучения. Кроме того, глубокое обучение представляет новые выгодные решения в следующих областях.

- Непрерывное обучение с использованием дистанционного обучения.
- Многоразовые решения с использованием переноса обучения.
- Простые решения с использованием сквозного обучения.

Следующие разделы помогут вам понять, что такое дистанционное обучение, перенос обучения и сквозное обучение.

Использование дистанционного обучения

Нейронные сети гибче других алгоритмов машинного обучения, и они способны продолжать обучение, работая над созданием прогнозов и классификаций. Эта возможность исходит из алгоритмов оптимизации, позволяющих нейронным сетям учиться, и которые способны многократно работать на небольших выборках примеров (*пакетное обучение* (batch learning)) или даже на отдельных примерах (*дистанционное обучение* (online learning)). Сети глубокого обучения способны наращивать свои знания шаг за шагом и оставаться восприимчивыми к новой информации, которая может появиться (например, как ум ребенка, всегда открытый для новых стимулов и опыта).

Например, приложение глубокого обучения на веб-сайте социальных сетей можно обучить распознавать изображения кошек. Когда люди публикуют фотографии кошек, приложение распознает их и помечает соответствующей меткой. Когда люди начинают публиковать в социальной сети фотографии собак, нейронной сети не нужно начинать обучение с самого начала; она может продолжать учиться на изображениях собак. Эта возможность особенно полезна для борьбы с изменчивостью данных в Интернет. Сеть глубокого обучения может быть открыта для новизны, и адаптироваться к ней.

Перенос обучения

Гибкость удобна, даже когда сеть завершает свое обучение, но вы должны использовать ее для целей, отличных от первоначального обучения. Сети, которые различают объекты и правильно их классифицируют, требуют длительного времени обучения и больших вычислительных мощностей, чтобы научиться тому, как это делать. Расширение возможностей сети для новых видов изображений, которые не были частью предыдущего обучения, означает перенос знаний на эту новую задачу (*перенос обучения* (transfer learning)).

Например, вы можете преобразовать сеть, способную различать собак и кошек, для выполнения работ, подразумевающих определение блюд из макарон и сыра. Большинство слоев сети вы используете в том виде, в каком они есть, (вы их замораживаете), а затем работаете с конечными выходными слоями (тонкая настройка). В скором времени и с меньшим количеством примеров сеть применяет то, чему научилась при различении собак и кошек, для макарон и сыра. Он будет работать даже лучше, чем нейронная сеть, обученная распознавать только макароны и сыр.

Перенос обучения является чем-то новым для большинства алгоритмов машинного обучения и открывает возможный рынок для передачи знаний из одного приложения в другое, из одной компании в другую. Компания Google уже делает это, она на самом деле делится своим огромным хранилищем данных,

обнародовав сети, на которых оно построено (это подробно описано в статье https://techcrunch.com/2017/06/16/object-detection-api/). Это шаг к демократизации глубокого обучения, позволяющий каждому получить доступ к его возможностям.

Обучение от начала до конца

Наконец, глубокое обучение позволяет осуществлять сквозное обучение, а значит, оно решает задачи более прямым и простым способом, чем предыдущие решения глубокого обучения. Такая гибкость может оказать большое влияние при решении задач.

Возможно, вы захотите решить сложную задачу, например, заставить искусственный интеллект распознавать известные лица или вести машину. При использовании классического подхода искусственного интеллекта вам пришлось бы разделить проблему на более управляемые подзадачи, чтобы достичь приемлемого результата за приемлемое время. Например, если вы хотели распознать лица на фотографии, предыдущие системы искусственного интеллекта разделили бы задачи на части следующим образом.

1. Найти лица на фотографии.
2. Вырезать лица с фотографии.
3. Обработать обрезанные лица, чтобы получить позу, похожую на фотографию удостоверения личности.
4. Передать обработанные обрезанные лица в качестве обучающих примеров в нейронную сеть для распознавания изображений.

Сегодня вы можете передать фотографию в архитектуру глубокого обучения, настроить ее на обучение поиску лиц на изображениях, а затем использовать архитектуру глубокого обучения для их классификации. Вы можете использовать тот же подход для языкового перевода, распознавания речи или даже для беспилотных автомобилей. Во всех случаях вы просто передаете ввод в систему глубокого обучения и получаете желаемый результат.

Глава 10

Сверточные нейронные сети

В ЭТОЙ ГЛАВЕ...

- Знакомство с основами компьютерного зрения
- Работа сверточных нейронных сетей
- Воссоздание сети LeNet5 с использованием Keras
- Как свертки видят мир

Заглянув внутрь глубокого обучения, вы можете быть удивлены, обнаружив множество старых технологий, но куда удивительней то, что все работает так, как никогда раньше, поскольку исследователи, наконец, узнали, как заставить работать вместе несколько старых простых решений. В результате большие данные могут автоматически фильтровать, обрабатывать и преобразовывать данные.

Например, новые функции активации, такие как Rectified Linear Unit (ReLU), обсуждавшиеся в предыдущих главах, не новы, но вы видите, что они используются по-новому. ReLU — это функция нейронных сетей, которая оставляет нетронутыми положительные значения и превращает отрицательные в ноль; вы можете найти первую ссылку на ReLU в научной статье Ганлозера (Hahnloser) и др. от 2000 года. Кроме того, возможности распознавания образов, сделавшие глубокое обучение столь популярным несколько лет назад, также не новы.

В последние годы глубокое обучение достигло больших успехов благодаря способности кодировать определенные свойства в архитектуру

с использованием *сверточных нейронных сетей* (Convolutional Neural Network — CNN), или ConvNets. Французский ученый Ян Лекун и другие известные ученые выдвинули идею CNN в конце 1980-х годов, а полностью они разработали свою технологию в 1990-х годах. Но только сейчас, примерно через 25 лет, такие сети начинают давать удивительные результаты, достигая в конкретных задачах распознавания даже лучшей производительности, чем люди. Изменения произошли потому, что можно настроить такие сети на сложные архитектуры, способные улучшить их возможности по обучению на основе множества полезных данных.

CNN сильно подпитывают недавний ренессанс вокруг глубокого обучения. В следующих разделах обсуждается, как CNN помогают определять края и формы изображения для решения таких задач как расшифровка рукописного текста, точное определение местоположения определенного объекта на изображении или разделение различных частей сложной сцены изображения.

ЗАПОМНИ!

Вам не нужно вводить исходный код примеров из этой главы вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файле `DL4D_10_LeNet5.ipynb`.

Начнем обзор CNN с распознавания символов

Идея CNN не нова. Она появилась в конце 1980-х годов как решение проблем распознавания символов. Ян Лекун разработал CNN, когда работал в AT&T Labs Research вместе с другими учеными, Йошуа Бенгио, Леоном Боттоу и Патриком Хаффнером, над сетью LeNet5. Прежде чем углубиться в технологию данных специализированных нейронных сетей, мы уделим в этой главе время пониманию проблемы распознавания изображений.

Из-за повсеместного присутствия цифровых камер, веб-камер и мобильных телефонов с камерами, цифровые изображения сегодня повсюду. Поскольку получение изображений стало таким легким, они создают новый огромный поток данных. Возможность обработки изображений открывает двери для новых приложений в таких областях как робототехника, автономное вождение, медицина, безопасность и наблюдение.

Основы изображения

Обработка изображения для использования компьютером превращает его в данные. Компьютеры отправляют изображения на монитор в виде потока

данных, состоящего из пикселей, поэтому компьютерные изображения лучше всего представлять в виде матрицы значений пикселей, причем каждая позиция в матрице соответствует точке на изображении.

Современные компьютерные изображения представляют цвета, используя серию из 32 бит (по 8 бит для красного, синего, зеленого и альфа-канала прозрачности). Но для создания истинно цветного изображения вы можете использовать только 24 бита. Статья по адресу `http://www.rit-mcsl.org/fairchild/WhyIsColor/Questions/4-5.html` объясняет этот процесс более подробно. Компьютерные изображения представляют цвет с использованием трех перекрывающихся матриц, каждая из которых предоставляет информацию относительно одного из трех цветов: красного, зеленого или синего (Red, Green, Blue — RGB). Смешение различных количеств этих трех цветов позволяет представить любой стандартный цвет, видимый человеку, но не тот, который видят люди с необычайным восприятием. Большинство людей могут видеть максимум 1 000 000 цветов, находящихся в пределах цветового диапазона из 16 777 216 цветов, определяемых 24-битами. Тетрахроматы могут видеть 100 000 000 цветов, поэтому вы не можете использовать компьютер для анализа того, что они видят. Больше о тетрахроматах можно узнать в статье `http://nymag.com/scienceofus/2015/02/what-like-see-a-hundred-million-colors.html`.

Как правило, изображение обрабатывается компьютером как трехмерная матрица, состоящая из высоты, ширины и количества каналов — три для изображения RGB, но для черно-белого изображения может быть только один. (Оттенки серого — это особый вид изображения RGB, у которого каждый из трех каналов имеет одинаковое значение; обсуждение происходящего при преобразовании между цветом и оттенками серого см. на `https://introcomputing.org/image-6-grayscale.html`). При использовании изображения в оттенках серого одной матрицы может быть достаточно, если одно число представляет цвета в 256 градациях серого, как показано в примере на рис. 10.1. На этом рисунке значение каждого пикселя изображения количественно определяется значениями матрицы.

Учитывая тот факт, что изображения представляют собой пиксели (представленные в виде числовых вводов), специалисты по нейронным сетям первоначально достигли хороших результатов, подключив изображение непосредственно к нейронной сети. Каждый пиксель изображения подключен к входному узлу в сети. Затем один или несколько следующих скрытых слоев завершали сеть, в результате чего появлялся выходной слой. Этот подход работал приемлемо для небольших изображений и для решения небольших задач, уступая место другим подходам для решения задач распознавания изображений. В качестве альтернативы исследователи использовали другие алгоритмы

машинного обучения или интенсивное создание признаков для преобразования изображения во вновь обработанные данные, которые могли бы помочь алгоритмам лучше распознавать изображение. Примером создания объекта изображения являются *гистограммы направленных градиентов* (Histograms of Oriented Gradients — HOG), представляющие собой вычислительный способ обнаружения шаблонов в изображении и их преобразования в числовую матрицу. (О работе HOG можно узнать в учебнике для пакета Skimage: `http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html`.)

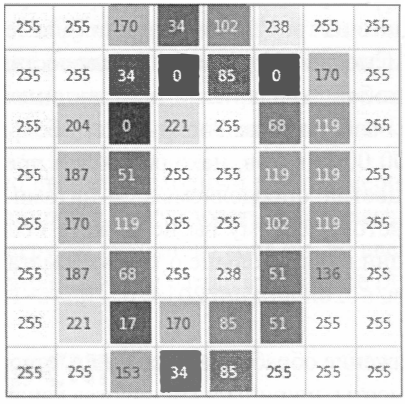


Рис. 10.1. Каждый пиксель читается компьютером как число в матрице

Специалисты по нейронным сетям сочли создание признаков изображения слишком требовательным в вычислительном отношении и зачастую непрактичным. Объединение пикселей изображения с нейронами было трудным, поскольку требовалось вычислять невероятно большое количество параметров, и сеть не могла достичь трансляционной инвариантности, то есть способности расшифровывать представленный объект при различных условиях размера, ракурса или положения в изображении, как показано на рис. 10.2.

Нейронная сеть, состоящая из плотных слоев, как описано в предыдущих главах, может выявлять только те изображения, которые похожи на использовавшиеся для обучения, то есть виденные раньше, поскольку она обучается выявляя шаблоны в определенных местах изображений. Кроме того, нейронная сеть может сделать много ошибок. Частично решить проблему может преобразование изображения перед его передачей в нейронную сеть (изменение размера, перемещение, очистка пикселей и создание специальных фрагментов информации для лучшей обработки сетью). Этот метод, называемый созданием

признаков, требует знаний о необходимых преобразованиях изображений, а также большого количества вычислений с точки зрения анализа данных. Из-за большого количества сложной обработки, задача распознавания образов — это скорее искусство, чем наука. Однако количество работ по настройке со временем уменьшилось, поскольку увеличилась база библиотек, автоматизирующих определенные задачи.

Рис. 10.2. Только по трансляционной инвариантности алгоритм может определить собаку и ее варианты

Как работает свертка

Свертка легко решают проблему трансляционной инвариантности, поскольку она предлагает другой подход к обработке изображений внутри нейронной сети. Идея началась с биологических исследований происходящего в зрительной коре человека.

Эксперимент, проведенный в 1962 году лауреатами Нобелевской премии Дэвидом Хантером Хьюбелом (David Hunter Hubel) и Торстеном Визелем

(Torsten Wiesel), показал, что в мозгу активируются только определенные нейроны, когда глаз видит определенные шаблоны, такие как горизонтальные, вертикальные или диагональные края. Кроме того, эти ученые обнаружили, что нейроны организованы в иерархии вертикально и предположили, что зрительное восприятие зависит от организованного вклада многих отдельных специализированных нейронов. (Больше об этом эксперименте можно узнать в статье на `https://knowingneurons.com/2014/10/29/hubel-and-wiesel-the-neural-basis-of-visual-perception/`.) Свертки просто используют эту идею математически, применяя ее к обработке изображений для расширения возможности нейронной сети по точному распознаванию различных образов.

Свертка

Чтобы понять, как работает свертка, начнем с ввода. Входными данными является изображение, состоящее из одного или нескольких слоев пикселей, называемых каналами. При этом изображение использует значения от 0 (пиксель полностью отключен) до 255 (пиксель полностью включен). (Обычно, для экономии памяти, значения хранятся как целые числа.) Как упоминалось в предыдущем разделе этой главы, изображения RGB имеют отдельные каналы для красного, зеленого и синего цветов. Смешение этих каналов создает палитру цветов, видимых вами на экране.

Свертка работает, обрабатывая небольшие фрагменты изображения по всем каналам изображения одновременно. (Представьте себе кусок слоеного пирога, где каждый кусочек демонстрирует все слои). Фрагменты изображения — это просто движущиеся по изображению окно: окно свертки может быть квадратным или прямоугольным, оно начинает движение слева вверху и перемещается слева направо и сверху вниз. Полный проход окна по изображению называется *фильтром* (filter) и подразумевает полное преобразование изображения. Следует также отметить, что новый фрагмент ограничивается окном, затем окно сдвигается на определенное количество пикселей; размер сдвига — это *шаг* (stride). Шаг 1 означает, что окно перемещается на один пиксель вправо или вниз; шаг 2 подразумевает движение на два пикселя; и так далее.

Каждый раз, когда окно свертки перемещается в новую позицию, происходит процесс фильтрации, для создания части фильтра, описанного в предыдущем абзаце. В этом процессе значения в окне свертки умножаются на значения в *ядре* (kernel) (небольшая матрица, используемая для размытия, повышения резкости, тиснения, обнаружения краев и т.д. — вы сами выбираете ядро, необходимое для рассматриваемой задачи). (Статья на `http://setosa.io/ev/image-kernels/` рассказывает больше о различных типах ядер.) Размер ядра совпадает с размером окна свертки. Умножение каждой части изображения на

ядро создает новое значение для каждого пикселя, что в некотором смысле является новым обработанным признаком изображения. Свертка выводит значение пикселя, и когда скользящее окно завершает свой проход по изображению, изображение считается *отфильтрованным*. В результате свертки вы получите новое изображение, имеющее следующие характеристики.

- Если вы используете один процесс фильтрации, результатом будет преобразованное изображение одного канала.
- Если вы используете несколько ядер, новое изображение имеет столько же каналов, сколько и фильтров, каждый из которых содержит специально обработанные новые значения функций. Количество фильтров — это *глубина фильтра* свертки.
- Если вы используете шаг 1, вы получите изображение того же размера, что и оригинал.
- Если вы используете шаги размером более 1, результирующее сверточное изображение будет меньше оригинала (шаг размером 2 подразумевает вдвое меньший размер изображения).
- Результирующее изображение может быть меньше в зависимости от размера ядра, поскольку ядро должно начать и закончить свой проход на границах изображения. При обработке изображения ядро уменьшит его размер. Например, ядро размером 3 × 3 пикселя, обрабатывающее изображение размером 7 × 7 пикселей, уменьшит его на 2 пикселя по высоте и ширине, и результатом свертки будет вывод размером 5 × 5 пикселей. Изображение можно дополнить нулями по границе (то есть, по сути, поставить черную рамку на изображение), чтобы процесс свертки не уменьшал конечный размер вывода. Эта стратегия называется *дополнением Same* (Same padding). Если вы просто позволите ядру уменьшить размер вашего начального изображения, это *дополнение Valid* (Valid padding).

Обработка изображений долгое время опиралась на процесс свертки. Сверточные фильтры могут определять края или улучшать определенные характеристики изображения. На рис. 10.3 приведен пример некоторых сверток, преобразующих изображение.

Проблема использования свертки заключается в том, что она требует участия человека и усилий с его стороны. При использовании свертки нейронной сети вы задаете следующее.

- Количество фильтров (количество ядер, обрабатывающих изображение, являющееся его выходными каналами).
- Размер ядра (установите только одну сторону для квадрата; установите ширину и высоту для прямоугольника).

» Шаги (обычно 1- или 2-пиксельные шаги).

» Хотите ли вы, чтобы изображение было окаймлено черным цветом (выберите допустимый или тот же отступ).

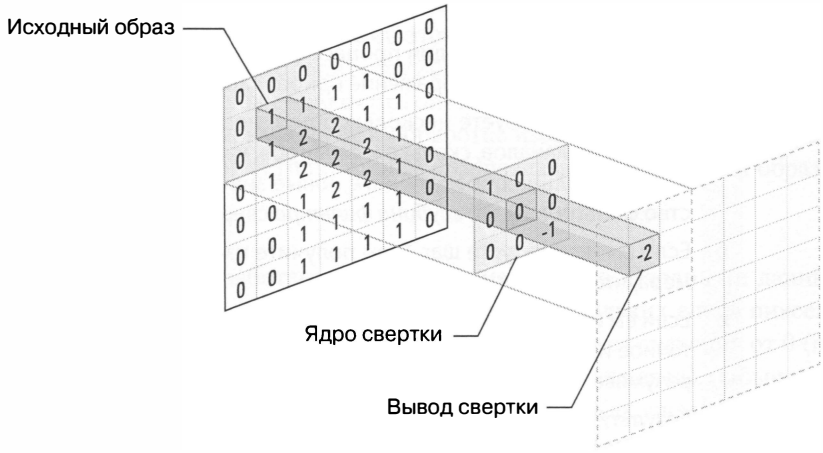


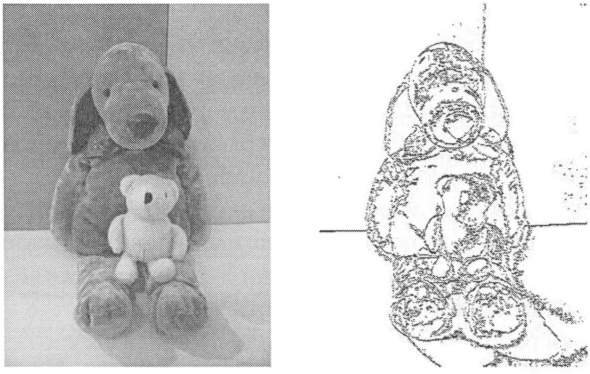
Рис. 10.3. Свертка обрабатывает фрагмент изображения с помощью матричного умножения

После определения параметров обработки изображения процесс оптимизации определяет значения ядра, используемые для обработки изображения, таким образом, чтобы обеспечить наилучшую классификацию конечного выходного слоя. Таким образом, каждый элемент матрицы ядра является нейроном нейронной сети и модифицируется во время обучения с использованием обратного распространения для обеспечения наилучшей производительности самой сети.

Еще один интересный аспект этого процесса заключается в том, что каждое ядро специализируется на поиске конкретных аспектов изображения. Например, ядро, специализирующееся на признаках фильтрации, типичных для кошек, может найти кошку независимо от того, где она находится на изображении, и, если вы используете достаточно ядер, обнаруживается каждый возможный вариант изображения такого типа (измененный размер, поворот, отражение изображения), что делает вашу нейронную сеть эффективным инструментом для классификации и распознавания образов.

Границы изображения легко обнаруживаются после применения ядра размером 3 × 3 пикселя. Это ядро специализируется на поиске краев, но другое ядро может обнаружить другие признаки изображения. Изменяя значения в ядре, как это делает нейронная сеть во время обратного распространения, сеть

находит лучший способ обработки изображений в целях регрессии или классификации.



Оригинальное изображение Ядро краев

Рис. 10.4. Границы изображения определяются после применения ядра 3 × 3 пикселя

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Ядро (kernel) — это матрица, значения которой определяются оптимизацией нейронной сети, и умножаются на небольшой участок такого же размера, перемещающийся по изображению, но оно может быть задано как нейронный слой, вес которого распределяется между различными входными нейронами. Вы можете рассматривать фрагмент как неподвижный нейронный слой, соединенный со многими частями изображения, всегда использующий одинаковый набор весов. Это точно такой же результат.

Keras предлагает готовый сверточный слой `Conv2D`. Этот слой Keras может получать входные данные как непосредственно из изображения (вы должны установить в кортеже `input_shape` ширину, высоту и количество каналов вашего изображения), так и из другого слоя (например, другой свертки). Вы также можете установить фильтры, `kernel_size`, `strides` и `padding`, являющиеся основными параметрами для любых сверточных слоев, как описано ранее в этой главе.

СОВЕТ

При настройке слоя `Conv2D` вы также можете установить множество других параметров, которые на самом деле чисто технические и, возможно, не нужны для ваших первых экспериментов с CNN. Единственные другие параметры, которые вы можете найти полезными сейчас, — это `activation`, добавляющий активацию по вашему выбору, и `name`, задающий имя для слоя.

Упрощение использования подвыборки

Сверточные слои преобразуют исходное изображение, используя различные виды фильтрации. Каждый слой находит определенные рисунки на изображении (определенные наборы форм и цветов, делающие изображение узнаваемым). По мере этого процесса сложность нейронной сети возрастает, поскольку количество параметров увеличивается по мере того, как сеть получает больше фильтров. Для снижения сложности управления вам необходимо ускорить фильтрацию и сократить количество операций.

Слой подвыборки может упростить вывод, полученный от сверточных слоев, уменьшая тем самым количество выполняемых последовательных операций и используя меньше операций свертки для фильтрации. Работая так же, как свертка (используя размер окна для фильтра и шаг для его перемещения), слой подвыборки работает с фрагментами полученного вывода, и уменьшают фрагмент до единого числа, эффективно сокращая, таким образом, объем данных, проходящих через нейронную сеть.

На рис. 10.5 представлены операции, выполняемые слоем подвыборки, получающим в качестве ввода отфильтрованные данные, представленные левой матрицей 4 × 4: они обрабатываются окном размером 2 пикселя и перемещающимся с шагом 2 пикселя. В результате слой подвыборки выдает правильный результат: матрицу 2 × 2. Сеть применяет операцию подвыборки к четырем фрагментам, представленным четырьмя частями матрицы разного цвета. Для каждого фрагмента слой подвыборки вычисляет максимальное значение и сохраняет его как вывод.

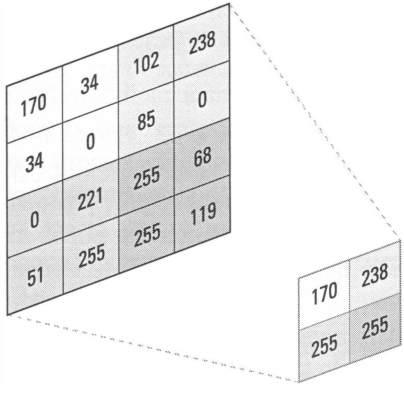


Рис. 10.5. Слой подвыборки максимума, работающий с фрагментами уменьшенного изображения

Текущий пример основан на слое подвыборки максимума, поскольку в своем скользящем окне он использует преобразование max. На самом деле у вас есть доступ к четырем основным типам слоев подвыборки.

- Подвыборка максимума (Max).
- Подвыборка среднего (Average).
- Подвыборка глобального максимума (Global max).
- Подвыборка глобального среднего (Global average).

Кроме того, эти четыре типа слоев подвыборки имеют разные версии, в зависимости от размерности входных данных, которые они могут обрабатывать.

- **Одномерная подвыборка**. Работает с векторами. Таким образом, одномерная подвыборка идеальна для данных последовательности, таких как временные данные (данные, представляющие события, следующие друг за другом во времени) или текст (представленные в виде последовательностей букв или слов). Возвращает максимальное или среднее значение смежных частей последовательности.
- **Двумерная подвыборка**. Подходит для пространственных данных, соответствующих матрице. Вы можете использовать двумерную подвыборку для изображения в градациях серого или для каждого канала изображения RBG по отдельности. Возвращает максимальное или среднее значение небольших пятен (квадратов) данных.
- **Трехмерная подвыборка**. Подходит для пространственных данных, представленных в виде пространственно-временных данных. Вы можете использовать трехмерную подвыборку для изображений, снятых во времени. Типичным примером является использование магнитно-резонансной томографии (МРТ) при медицинском обследовании. Радиологи используют МРТ для исследования тканей организма с помощью магнитных полей и радиоволн. (Чтобы узнать больше о вкладе глубокого обучения в здравоохранение, см. статью Stanford AI по адресу `https://medium.com/stanford-ai-forhealthcare/dont-just-scan-this-deep-learning-techniques-formri-52610e9b7a85`.) Этот тип подвыборки возвращает максимальное или среднее значение небольших кусков (кубов) данных.

Описание всех этих слоев можно найти в документации Keras, вместе со всеми их параметрами, по адресу `https://keras.io/layers/pooling/`.

Архитектура LeNet

Возможно, вы были поражены описанием CNN в предыдущем разделе и тем, как работают его слои (свертка и подвыборка максимума), но вы можете быть еще более удивлены, обнаружив, что это не новая технология; она появился в 1990-х годах. Следующие разделы описывают архитектуру LeNet более подробно.

Основные функции

Ключевым лицом этой инновации был Ян Лекун, работавший в AT&T Labs Research руководителем отдела исследований по обработке изображений. Лекун специализируется на оптическом распознавании символов и компьютерном зрении. Ян Лекун — французский ученый, создавший сверточные нейронные сети с Леоном Боттоу (Leon Bottou), Йошуа Бенгио (Yoshua Bengio) и Патриком Хаффнером (Patrick Haffner). В настоящее время он является главным научным сотрудником по искусственному интеллекту в Facebook AI Research (FAIR) и профессором в Нью-Йоркском университете (главным образом, он связан с Центром науки о данных NYU). Его личная домашняя страница находится по адресу `http://yann.lecun.com/`.

В конце 90-х годов AT&T внедрила сеть Лекуна LeNet 5 для чтения почтовых индексов почтовой службой США. Компания также использовала LeNet5 для считывателей чеков банкоматов, способных автоматически считывать сумму чека. Система отлично работает, как сообщает Лекун на `https://pafnuty.wordpress.com/2009/06/13/yann-lecun/`.

Тем не менее, в то время успех LeNet прошел почти незамеченным, поскольку отрасль искусственного интеллекта переживала *зиму искусственного интеллекта* (AI winter): и общественность, и инвесторы были значительно менее заинтересованы и внимательны к улучшениям нейронных технологий, чем сейчас.

ЗАПОМНИ!

Одна из причин зимы искусственного интеллекта заключается в том, что многие исследователи и инвесторы потеряли веру в идею, что нейронные сети произведут революцию в искусственном интеллекте. Данные того времени не обладали достаточной сложностью, чтобы такая сеть работала хорошо. (Банкоматы и USPS были заметными исключениями из-за большого количества данных, с которыми они работали.) При недостатке данных, свертки лишь незначительно превосходят обычные нейронные сети, состоящие из соединенных слоев. Кроме того, многие исследователи достигли результатов, сравнимых с LeNet5, используя совершенно новые алгоритмы машинного обучения, такие как метод опорных векторов

(SVM) и случайные леса, основанные на математических принципах, отличных от используемых для нейронных сетей.

Вы можете увидеть сеть в действии по адресу `http://yann.lecun.com/exdb/lenet/` или на видео, в котором младший Лекун демонстрирует более раннюю версию сети: `https://www.youtube.com/watch?v=FwFduRA_L6Q`. В то время наличие машины, способной расшифровывать как машинописные, так и рукописные числа, было настоящим подвигом.

Как показано на рис. 10.6, архитектура LeNet5 состоит из двух последовательностей сверточных слоев и слоев подвыборки среднего, выполняющих обработку изображения. Последний слой последовательности выравнивается; то есть каждый нейрон в результирующей серии сверток двумерных массивов копируется в одну линию нейронов. На этом этапе два полносвязных слоя и классификатор softmax завершают сеть и обеспечивают вывод с точки зрения вероятности. Сеть LeNet5 действительно является основой всех последующих CNN. Воссоздание архитектуры с использованием Keras объяснит ее слой за слоем и продемонстрирует, как строить свои собственные сверточные сети.

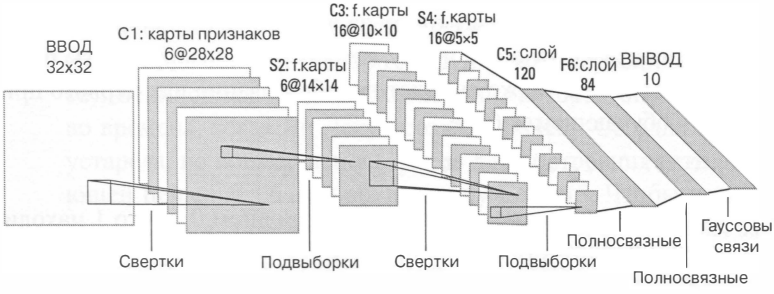


Рис. 10.6. Архитектура LeNet5, нейронная сеть для распознавания рукописных цифр

Построение собственной сети LeNet5

Эта сеть будет обучена работе с соответствующим объемом данных (предоставленный Keras набор данных цифр содержит более 60 000 примеров). Таким образом, вы можете получить преимущество, если запустите его на Colab, как описано в главе 3, или на локальном компьютере, если у вас есть графический процессор. Открыв новый блокнот, вы начнете с импорта необходимых пакетов и функций из Keras, используя следующий код.

```
import keras
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
```

```
from keras.layers import Conv2D, AveragePooling2D
from keras.layers import Dense, Flatten
from keras.losses import categorical_crossentropy
```

После импорта нужных инструментов вам необходимо собрать данные.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

При первом выполнении этого кода, команда `mnist` загрузит все данные из Интернета, что может занять некоторое время. Загруженные данные состоят из одноканальных 28 × 28-пиксельных изображений, представляющих рукописные числа от нуля до девяти. В качестве первого шага вам нужно преобразовать переменную ответа (`y_train` для фазы обучения и `y_test` для теста после завершения обучения модели) в нечто, что нейронная сеть может понять и обработать.

```
num_classes = len(np.unique(y_train))
print(y_train[0], end=' => ')
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
print(y_train[0])
```

Этот фрагмент кода переводит ответ из чисел в векторы чисел, где значение в позиции, соответствующей числу, которое прогнозирует сеть, равно 1, а остальные равны 0. Код также выведет преобразование для первого примера изображения в обучающем наборе.

```
5 => [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

СОВЕТ

Обратите внимание, что вывод, в основном 0, и что 1 находится в позиции, соответствующей числу 5. Это используется потому, что нейронной сети необходим слой ответов (response layer), представляющий собой набор нейронов (отсюда вектор), который должен активироваться, если предоставленный ответ правильный. В данном случае вы видите десять нейронов, а на этапе обучения код активирует правильный ответ (в правильной позиции установлено значение 1) и отключает другие (их значения равны 0). На этапе тестирования нейронная сеть использует свою базу данных примеров, чтобы включить правильный нейрон или, по крайней мере, включить наиболее правильный. Следующий код готовит учебные и тестовые данные.

```
X_train = X_train.astype(np.float32) / 255
X_test = X_test.astype(np.float32) / 255
img_rows, img_cols = X_train.shape[1:]
X_train = X_train.reshape(len(X_train),
                          img_rows, img_cols, 1
```

```
X_test = X_test.reshape(len(X_test),
                        img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)
```

Значения пикселей в диапазоне от 0 до 255 преобразуются в десятичное значение в диапазоне от 0 до 1. Первые две строки кода оптимизируют сеть для правильной работы с большими числами, которые могут вызвать проблемы. Следующие строки изменяют изображение, чтобы оно имело нужную высоту, ширину и каналы.

Следующая строка кода определяет архитектуру LeNet5. Вы начинаете с вызова функции `sequential`, предоставляющей пустую модель.

```
lenet = Sequential()
```

Первый добавленный слой — это сверточный слой по имени "C1".

```
lenet.add(Conv2D(6, kernel_size=(5, 5), activation='tanh',
    input_shape=input_shape, padding='same', name='C1'))
```

Свертка работает с фильтром размером 6 (это означает, что будет шесть новых каналов, созданных свертками) и размером ядра 5 × 5 пикселей.

СОВЕТ

Функция активации всех слоев сети, кроме последнего, — это *tanh* (Hyperbolic Tangent function — функция гиперболического тангенса), нелинейная функция активации, которая была современной во времена, когда Ян Лекун создал LetNet5. Сегодня эта функция устарела, но пример использует ее для построения сети, напоминающей первоначальную архитектуру LetNet5. Чтобы использовать такую сеть для своих собственных проектов, вы должны заменить функцию tanh на современную функцию ReLU (см. подробнее на `https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning`). В примере добавлен слой подвыборки S2, использующий ядро размером 2 × 2 пикселя.

```
lenet.add(AveragePooling2D(pool_size=(2, 2), strides=(1, 1),
          padding='valid'))
```

На этом этапе код переходит к последовательности, всегда выполняемой со слоями свертки и подвыборки, но на этот раз с использованием большего количества фильтров.

```
lenet.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1),
                 activation='tanh', padding='valid'))
lenet.add(AveragePooling2D(pool_size=(2, 2), strides=(1, 1),
          padding='valid'))
```

LeNet5 завершает работу поэтапно, используя свертку с 120 фильтрами. Эта свертка не имеет слоя подвыборки, а скорее выравнивающий слой, проецирующий нейроны в последний сверточный слой как *плотный слой* (dense layer).

```
lenet.add(Conv2D(120, kernel_size=(5, 5), activation='tanh',
          name='C5'))
lenet.add(Flatten())
```

Закрытие сети представляет собой последовательность из двух плотных слоев, обрабатывающих вывод свертки с использованием активации `tanh` и `softmax`. Эти два слоя поддерживают конечные выходные слои, где нейроны активируют вывод, чтобы сигнализировать о прогнозируемом ответе. На самом деле выходным является слой softmax, как указано в `name='OUTPUT'`.

```
lenet.add(Dense(84, activation='tanh', name='FC6'))
lenet.add(Dense(10, activation='softmax', name='OUTPUT'))
```

ЗАПОМНИ!

Когда сеть будет готова, для ее компиляции вам понадобится Keras. (Позади всего кода Python есть некоторый код на языке C.) Keras компилирует его на основе оптимизатора SGD.

```
lenet.compile(loss=categorical_crossentropy, optimizer='SGD',
              metrics=['accuracy'])
lenet.summary()
```

На этом этапе вы можете запустить сеть и подождать, пока она обработает изображения.

```
batch_size = 64
epochs = 50
history = lenet.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(X_test,
                                     y_test))
```

Завершение цикла занимает 50 эпох, каждая из которых обрабатывает партии по 64 изображения за один раз. (*Эпоха* (epoch) — это прохождение всего набора данных через нейронную сеть один раз, в то время как *пакет* (batch) является частью набора данных, что означает, в этом случае, разделение набора данных на 64 блока.) На каждой эпохе (продолжительностью около 8 секунд, если вы используйте Colab), вы можете следить за индикатором, показывающим время, необходимое для завершения этой эпохи. Вы также можете прочитать показатели точности для обучающих (оптимистическую оценку качества своей модели, подробные сведения о том, что именно означает качество, см. на `https://towardsdatascience.com/`

measuring-model-goodness-part-1-a24ed4d62f71) и тестовых наборов (более реалистичный взгляд). В последнюю эпоху вы должны прочитать, что сеть LeNet5, построенная за несколько этапов, достигает точности 0,989, а это означает, что из каждых 100 рукописных чисел, которые сеть пытается распознать, она должна правильно угадать около 99.

Обнаружение краев и фигур в изображениях

Свертки обрабатывают изображения автоматически и работают лучше, чем плотный слой, поскольку они изучают шаблоны в изображениях на локальном уровне и могут отслеживать их в любой другой части изображения (такая характеристика как *трансляционная инвариантность* (translation invariance)). С другой стороны, традиционные плотные нейронные слои способны определять общие характеристики изображения без преимуществ трансляционной инвариантности. Это как различие между изучением книги с запоминанием наиболее значащих фрагментов текста и запоминанием ее слово в слово. Ученики (свертки), изучившие книгу фрагментами способны лучше абстрагировать ее содержание и готовы применить эти знания в подобных случаях. Ученик (плотный слой), выучивший книгу слово в слово, изо всех сил будет пытаться извлечь из нее что-то полезное.

CNN не волшебство и не черный ящик. Вы можете понять их в ходе обработки изображений и использовать их функциональность для расширения своих возможностей при решении новых задач. Эта функция позволяет решить ряд проблем с компьютерным зрением, которые аналитики данных считали слишком трудными для решения с использованием прежних стратегий.

Визуализация сверток

Для выполнения определенных задач иерархическим способом, CNN использует различные слои. Ян Лекун (см. раздел "Начнем обзор CNN с распознавания символов" в начале этой главы) заметил, что LeNet сначала обрабатывает края и контуры, затем мотивы, затем категории и, наконец, объекты. Недавние исследования раскрывают, как на самом деле работают свертки.

- **Начальные слои**. Обнаруживают края изображения.
- **Средние слои**. Обнаруживают сложные формы (созданные по краям).
- **Финальные слои**. Обнаруживают отличительные особенности изображения, характерные для того типа изображения, которое сеть должна классифицировать (например, нос собаки или уши кошки)

Эта иерархия шаблонов, обнаруживаемая свертками, также объясняет, почему глубокие сверточные сети работают лучше, чем мелкие: чем больше сложенных сверток, тем лучше сеть может изучать все больше сложных и полезных шаблонов для успешного распознавания изображений. Рис. 10.7 дает представление о том, как все это работает. Изображение собаки обрабатывается свертками, и первый слой выявляет шаблоны. Второй слой получает эти шаблоны и собирает их в собаку. Если шаблоны, обработанные первым слоем, кажутся слишком общими, чтобы их можно было использовать, шаблоны, представленные вторым слоем, воссоздают более характерные признаки собаки, обеспечивающие нейронную сеть преимуществом при распознавании собак.

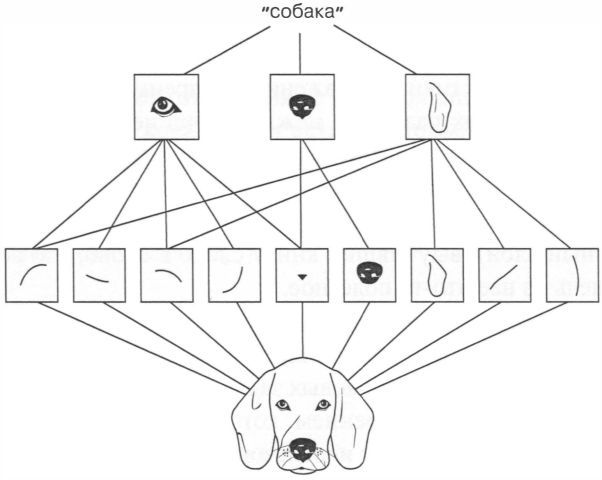


Рис. 10.7. Обработка изображения собаки с использованием сверток

Трудность в изучении работы свертки заключается в понимании того, как ядро (матрица чисел) создает свертки и как они работают с фрагментами изображений. Когда у вас много сверток, работающих одна за другой, определить результат с помощью прямого анализа сложно. Тем не менее, методика, разработанная для понимания таких сетей, создает изображения, которые активируют большинство сверток. Когда изображение активирует определенный слой, у вас появляется представление о том, что именно этот слой воспринимает больше всего.

СОВЕТ

Анализ свертки помогает понять, как все работает. Но чтобы избежать предвзятости в прогнозировании, следует разработать новые способы обработки изображений. Например, вы можете обнаружить, что ваша CNN отличает собак от кошек, используя фон изображения, поскольку применяемые для обучения изображения собак сняты на улице, а кошек — в помещении.

Этот процесс подробно описан в 2017 году в статье *Feature Visualization* Криса Олаха (Chris Olah), Александра Мордвинцева (Alexander Mordvintsev) и Людвига Шуберта (Ludwig Schubert) из команд Google Research и Google Brain Team (`https://distill.pub/2017/feature-visualization/`). Вы даже можете проверить изображения самостоятельно, щелкнув и указав на слои GoogleLeNet, CNN, модели созданной Google по адресу `https://distill.pub/2017/feature-visualization/appendix/`.

Изображения на Feature Visualization могут напоминать вам изображения приложения *deepdream*, если вы имели возможность увидеть их, когда они стали популярными в Интернете (оригинальную статью о Deepdream и некоторые изображения можно найти на странице `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`). Это та же техника, но вместо поиска изображений, которые активируют слой лучше всего, вы выбираете сверточный слой и позволяете ему преобразовывать изображение.

Используя аналогичную технику, основанную на использовании сверток для преобразования существующего изображения, вы также можете скопировать стиль работ великого художника прошлого, такого как Пикассо или Ван Гога, это процесс *переноса художественного стиля* (artistic style transfer). Полученная в результате картина современна, но стиль — нет. Вы можете видеть некоторые интересные примеры переноса художественного стиля в оригинальной статье Леона Гатиса (Leon Gatys), Александра Экера (Alexander Ecker) и Матиаса Бетга (Matthias Bethge) по адресу `https://arxiv.org/pdf/1508.06576.pdf`.

На рис. 10.8 оригинальное изображение преобразуется в стиле рисунка и цветовых характеристик японской гравюры на дереве в стиле укиё-э “Большая волна в Канагаве”, выполненной японским художником Кацусики Хокусая, жившего с 1760 по 1849 годы.

Успешные архитектуры

В последние годы аналитики данных достигли значительного прогресса благодаря более глубокому изучению работы CNN. Другие методы также внесли свою лепту в прогресс CNN. Конкурсы изображений сыграли важную роль,

побуждая исследователей улучшать свои сети, что сделало доступными большое количество изображений.

Рис. 10.8. Содержимое изображения преобразуется с переносом стиля

Процесс обновления архитектур начался прошлой зимой искусственного интеллекта. Фей-Фей Ли (Fei-Fei Li), профессор информатики Иллинойсского университета в Урбане-Шампейне (а ныне главный научный сотрудник Google Cloud и профессор в Стэнфорде), решила предоставить больше реальных наборов данных для улучшения алгоритмов тестирования нейронных сетей. Она начала накапливать невероятное количество изображений, представляющих большое количество классов объектов. Она и ее команда решили такую огромную задачу, используя Amazon Mechanical Turk, сервис, которым вы пользуетесь, чтобы попросить людей выполнить за вас микрозадачи (например, классифицировать изображение) за небольшую плату.

Полученный набор данных, завершенный в 2009 году, назывался ImageNet и первоначально содержал 3,2 млн помеченных изображений (теперь он содержит более 10 млн изображений), сгруппированных в 5247 иерархически организованных категорий. Если вам интересно, вы можете изучить набор данных по адресу `http://www.image-net.org/` или прочитать оригинальную статью по адресу `http://www.image-net.org/papers/imagenet_cvpr09.pdf`.

Набор ImageNet вскоре появился на конкурсе 2010 года в котором нейронные сети, использующие свертку (отсюда и возрождение и дальнейшее развитие технологии, разработанной Яном Лекуном в 1990-х годах), доказали свою способность правильно классифицировать изображения, сгруппированные в 1000 классов. За семь лет конкурсов (они были закрыты в 2017 году), алгоритмы-победители повысили точность распознавания изображений с 71,8% до 97,3%, что превосходит возможности человека (да, люди делают ошибки при классификации объектов). Вот некоторые известные архитектуры CNN, которые были разработаны для конкурса.

- **AlexNet (2012)**. Создана Алексом Крижевски (Alex Krizhevsky) из Университета Торонто. Он использовал CNN с фильтром 11 × 11 пикселей. Он победил в конкурсе и представил использование графических процессоров для обучения нейронных сетей наравне с активацией ReLU для контроля переобучения.
- **VGGNet (2014)**. Существует в двух версиях, 16 и 19. Она была создана Visual Geometry Group в Оксфордском университете и определила новый стандарт 3 × 3 в размере фильтра для CNN.
- **ResNet (2015)**. Создана корпорацией Microsoft. Эта CNN не только расширила идею различных версий сети (50, 101, 152), но и ввела *пропускаемые слои* (skip layer). Это способ соединения более глубоких слоев с более мелкими, для предотвращения проблемы исчезающего градиента (см. главы 8 и 9), что позволяет создавать гораздо более глубокие сети, способные куда лучше распознавать шаблоны в изображениях.

Вы можете воспользоваться всеми инновациями, представленными на конкурсе ImageNet, и даже использовать каждую из нейронных сетей. Эта доступность позволяет вам копировать производительные сети, отмеченные на конкурсе, и успешно распространять их на множество других задач.

Перенос обучения

Сети, различающие объекты и правильно классифицирующие их, требуют большого количества изображений, более длительного времени обработки и огромных вычислительных мощностей, чтобы узнать, что делать. Адаптация возможностей сети к новым типам изображений, которые не были частью начального обучения, означает перенос существующих знаний на новую задачу. Этот процесс адаптации возможностей сети называется *переносом обучения* (transfer learning), а сеть, которую вы адаптируете, зачастую называют *предварительно обученной* (pretrained). Вы не можете применить перенос обучения к другим алгоритмам машинного обучения; только глубокое обучение способно перенести то, что он узнал по одной задаче, на другую.

ЗАПОМНИ!

Перенос обучения является чем-то новым для большинства алгоритмов машинного обучения и открывает возможный рынок для передачи знаний из одного приложения в другое и из одной компании в другую. Компания Google уже делает это; она делится своим огромным хранилищем данных, обнародовав сети, которые она построила на TF Hub (`https://www.tensorflow.org/hub`).

Например, вы можете преобразовать сеть, способную различать собак и кошек, для выполнения работ, подразумевающих определение блюд из макарон и сыра. С технической точки зрения эта задача решается по-разному, в зависимости от того, насколько прежняя задача похожа на новую, и сколько новых изображений у вас есть для обучения. (Небольшой набор изображений составляет примерно несколько тысяч изображений, иногда даже меньше.)

Если ваша новая задача с изображением похожа на старую, в вашей сети уже могут быть все свертки (слои краев, фигур и высокоуровневых признаков), необходимые для распознавания и классификации похожих изображений. В этом случае вам не нужно вводить в обучение слишком много изображений, добавлять вычислительные мощности или слишком глубоко адаптировать свою предварительно обученную сеть. Этот тип передачи является наиболее распространенным типом применения передачи обучения, и вы обычно применяете его, используя сеть, обученную во время конкурса ImageNet (поскольку эти сети были обучены на стольких изображениях, что у вас, вероятно, уже есть все свертки, необходимые для переноса знаний на другие задачи).

СОВЕТ

Предположим, что задача, которую вы намериваетесь решить, подразумевает не только поиск собак на изображениях, но и определение их породы. Вы используете большинство слоев сети ImageNet, таких как VGG16, как есть, без дополнительной настройки. При передаче обучения вы замораживаете значения коэффициентов предварительно обученных сверток, чтобы они не подвергались никакому дальнейшему обучению, и сеть не подвергнется переобучению на ваших данных, если они слишком малы.

Затем вы обучаете выходные слои на новых изображениях, характерных для новой задачи (процесс *точной настройки* (fine-tuning)). В скором времени и всего с несколькими примерами сеть применит то, чему она научилась при различении собак и кошек, для пород собак. Она будет работать даже лучше, чем нейронная сеть, обученная распознавать только породы собак, поскольку при тонкой настройке она использует то, чему сеть научилась ранее на миллионах изображений.

ЗАПОМНИ!

Нейронная сеть будет идентифицировать только те объекты, которые была обучена идентифицировать. Следовательно, если вы обучите CNN распознавать основные породы собак, такие как лабрадор-ретривер или хаски, CNN не будет распознавать помеси этих двух пород, например, лабски. Вместо этого CNN выведет наиболее близкое совпадение на основе внутренних весов, которые она совершенствует во время обучения.

Если задача, которую следует перенести в существующую нейронную сеть, отличается от задачи, для которой она была обучена: выявление блюд из макарон и сыра, начав с сети, используемой для идентификации собак и кошек, у вас есть несколько вариантов.

- Если у вас мало данных, вы можете заморозить первый и средний слои предварительно обученной сети и отбросить конечные слои, поскольку они содержат высокоуровневые признаки, которые, вероятно, бесполезны для вашей новой задачи. Вместо окончательных сверток вы добавляете слой ответа, подходящий для вашей задачи. Точная настройка выработает наилучшие коэффициенты для слоя ответов, учитывая уже имеющиеся предварительно подготовленные сверточные слои.
- Если у вас много данных, вы добавляете подходящий слой ответа в предварительно обученную сеть, но не замораживаете сверточные слои. Вы используете предварительно обученные веса в качестве отправной точки и позволяете сети наилучшим образом приспособиться к новой задаче, поскольку вы можете обучаться на большом количестве данных.

Пакет Keras предлагает несколько предварительно обученных моделей, которые вы можете использовать для переноса обучения. Вы можете прочитать обо всех доступных моделях и их архитектурах по адресу `https://keras.io/applications/`. В описании моделей упоминается также о некоторых отмеченных наградами сетях, упомянутых ранее в этой главе: VGG16, VGG19 и ResNet50. Глава 12 демонстрирует, как использовать эти сети на практике и как перенести коэффициенты, полученные на конкурсе ImageNet, на другие задачи.

Глава 11

Введение в рекуррентные нейронные сети

В ЭТОЙ ГЛАВЕ...

- Важность изучения данных в последовательности
- Создание подписей к изображениям и перевод с помощью глубокого обучения
- Технология долгой краткосрочной памяти (LSTM)
- Возможные альтернативы LSTM

В этой главе рассматривается, как глубокое обучение может справляться с потоковой информацией. Реальность не просто изменчива, но изменчива прогрессивным способом, который может быть предсказуем из наблюдения прошлого. Если изображение представляет собой статический моментальный снимок, видео, состоящее из последовательности связанных изображений, является потоком информации, и фильм может рассказать вам гораздо больше, чем отдельная фотография или серия фотографий. Аналогично для коротких и длинных текстовых данных (от твитов до целых документов и книг) и для всех числовых рядов, представляющих нечто происходящее на временной шкале (например, серию данных о продажах продукта или качестве воздуха за каждый день в городе).

В этой главе описывается ряд новых слоев, рекуррентные сети и все их улучшения, такие как слои LSTM и GRU. Эти технологии лежат в основе самых удивительных приложений глубокого обучения, с которыми вы можете

экспериментировать сегодня. Вы часто видите их на своем мобильном телефоне или дома. Например, вы используете подобное приложение, когда общаетесь с такими собеседниками как Siri, Google Home или Alexa. Другое приложение, Google Translate, переводит ваш разговор на другой язык.

За каждой из этих технологий стоит особая нейронная архитектура, а для обучения используются специфические для данной области данные — некоторые из них общедоступны, а некоторые нет. Даже с учетом этих различий в источнике данных и методике, слои, делающие все это возможным, являются точно теми же слоями, которые вы импортируете из TensorFlow и Keras (об этих двух средах глубокого обучения рассказывается в главе 4) и которые используете в коде своих приложений.

Рекуррентные сети

Нейронные сети преобразуют ваш ввод в желаемый результат. Даже в глубоком обучении процесс один и тот же, хотя преобразование куда сложней. В отличие от более простой нейронной сети, состоящей из нескольких слоев, глубокое обучение опирается на бо́льшее количество слоев для выполнения сложных преобразований. Вывод источника данных подключается к входному слою нейронной сети, и входной слой начинает обработку данных. Скрытые слои ищут соответствие шаблонам и связывают их с определенным результатом, который может быть значением или вероятностью. Этот процесс отлично работает для любого вида ввода и особенно хорошо работает для изображений, как описано в главе 10.

После того, как каждый слой обработает свои данные, он выводит преобразованные данные на следующий слой. Этот следующий слой обрабатывает данные совершенно независимо от предыдущих слоев. Использование этой стратегии подразумевает, что если вы подаете в свою нейронную сеть видео, сеть будет обрабатывать каждое изображение отдельно, одно за другим, и результат не изменится вообще, даже если вы перетасовали порядок предоставленных изображений. При работе сети таким образом, с использованием архитектур, описанных в предыдущих главах этой книги, вы не получите никаких преимуществ от порядка обработки информации.

Однако опыт также учит, что для понимания процесса иногда приходится наблюдать за событиями в определенной последовательности. Используя опыт, полученный на предыдущем шаге, для изучения следующего шага, вы можете снизить кривую обучения и сократить количество времени и усилий, необходимых для понимания каждого шага.

Моделирование последовательностей с использованием памяти

Виды нейронных архитектур, рассмотренных до сих пор, не позволяют обрабатывать всю последовательность элементов одновременно, используя один вход. Например, если у вас есть серия ежемесячных продаж продукта, вы обрабатываете эти данные, используя двенадцать входов, по одному на каждый месяц, и позволяя нейронной сети анализировать их одновременно. Из этого следует, что когда у вас есть длинные последовательности, вам нужно разместить их, используя большее количество входов, и ваша сеть становится довольно большой, поскольку каждый вход должен соединяться с неким другим входом. В итоге вы получите сеть, характеризующуюся большим количеством соединений (что выражается во множестве весов).

Рекуррентные нейронные сети (Recurrent Neural Network — RNN) представляют собой альтернативу решениям из предыдущих глав, таким как перцептрон в главе 7 и CNN в главе 10. Они впервые появились в 1980-х годах, и множество исследователей работали над их улучшением, пока недавно они не приобрели популярность благодаря достижениям в области глубокого обучения и появлению достаточной вычислительной мощи. Идея, лежащая в основе RNN, проста: они проверяют каждый элемент последовательности по одному разу и сохраняют его в памяти, чтобы использовать при рассмотрении следующего элемента последовательности. Это похоже на то, как работает человеческий разум при чтении текста: человек читает текст буква за буквой, но понимает слова, помня каждую букву в слове. Аналогичным образом RNN может связать слово с результатом, запоминая последовательность букв, которые она получает. Расширение этого метода позволяет запросить RNN определить, имеет ли фраза положительный или отрицательный оттенок — это широко распространенный *анализ настроений* (sentiment analysis). Сеть связывает положительный или отрицательный ответ с определенными последовательностями слов, которые она встречала в учебных примерах.

Графически RNN представляют как нейронный *модуль* (unit) (или *ячейку* (cell)), соединяющий вход с выходом, но также соединенный с самим собой, как показано на рис. 11.1. Это подключение к самому себе представляет концепцию *рекурсии*, являющуюся функцией, применяемой к себе самой до тех пор, пока не будет достигнут определенный результат. Одним из наиболее часто используемых примеров рекурсии является вычисление факториала, как описано на `https://www.geeksforgeeks.org/recursion/`. На рисунке показан конкретный пример RNN, использующий последовательность букв для создания слова *jazz*. Справа на рисунке представлено поведение модуля RNN,

получающего слово *jazz* в качестве ввода, но как показано слева, фактически есть только один модуль.

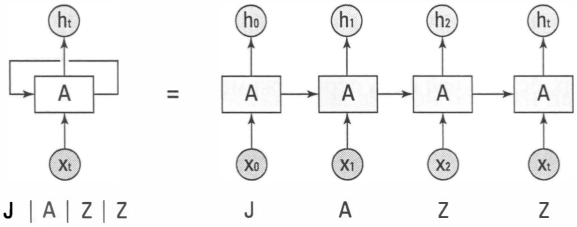


Рис. 11.1. Свернутая и развернутая ячейки RNN, обрабатывающие ввод последовательности

Рис. 11.1 демонстрирует слева рекурсивную ячейку и ее развернутый вид как серию модулей, получающую отдельные буквы слова *jazz* справа. Последовательность начинается с буквы *j*, сопровождаемой другими буквами. В ходе этого процесса, RNN осуществляет вывод и изменяет свои внутренние параметры. Изменяя свои внутренние параметры, устройство извлекает уроки из полученных данных и из запомненных предыдущих данных. Суммарный результат такого обучения является состоянием ячейки RNN.

ЗАПОМНИ!

При обсуждении нейронных сетей в предыдущих главах эта книга упоминает исключительно веса. При RNN вам также нужен термин *состояние* (state). Веса помогают преобразовать ввод в вывод RNN, но *состояние* содержит следы информации, которую RNN уже встречал ранее, поэтому состояние влияет на функционирование RNN. Состояние — это вид краткосрочной памяти, которая сбрасывается после завершения последовательности. Когда ячейка RNN получает фрагменты последовательности, она выполняет следующие действия.

1. Обрабатывает их, меняя состояние с каждым вводом.
2. Передает вывод.
3. После просмотра последнего вывода, RNN узнает наилучшие весовые коэффициенты для соотнесения ввода с правильным выводом при использовании обратного распространения.

Распознавание и перевод речи

Способность распознавать и переводить речь на разных языках становится все более важной с каждым днем благодаря глобализации экономики во всем мире. Языковой перевод — это область, в которой искусственный

интеллект имеет определенное преимущество перед людьми, причем настолько, что авторы таких статей как https://www.digitalistmag.com/digital-economy/2018/07/06/artificial-intelligence-is-changing-translation-industry-but-will-it-work-06178661 и https://www.forbes.com/sites/bernardmarr/2018/08/24/will-machine-learning-ai-make-human-translators-an-endangered-species/#535ec9703902 начинают сомневаться, как долго еще продержится человечек-переводчик.

Конечно, столь качественный процесс перевода стал возможен благодаря использованию глубокого обучения. С точки зрения нейронной архитектуры у вас есть несколько вариантов.

- Сохранять все выходные данные, предоставленные ячейкой RNN.
- Сохранять только последний вывод ячейки RNN.

Последний вывод — это вывод всей RNN, поскольку он создается после завершения обработки последовательности. Тем не менее, вы можете использовать предыдущий вывод, если вам нужно прогнозировать другую последовательность или вы намерены сложить больше ячеек RNN после текущей, например, при работе со сверточными нейронными сетями (CNN). Вертикальное стекирование RNN позволяет сети изучать сложные шаблоны последовательностей, а также быть более эффективными в прогнозировании.

RNN можно также разместить горизонтально, на том же уровне. Разрешение изучать последовательность нескольким RNN может помочь получить из данных больше. Использование нескольких RNN аналогично CNN, где каждый отдельный слой использует глубину сверток для изучения деталей и шаблонов в изображении. В случае нескольких RNN слой может уловить различные нюансы последовательности, которую они исследуют.

Проектирование структуры RNN как по горизонтали, так и по вертикали, улучшает прогнозирующие характеристики. Однако решение о том, как использовать выходные данные, определяет, чего может достичь архитектура глубокого обучения, основанная на RNN. Ключом является количество элементов, используемых в качестве вводов, и ожидаемая длина вывода. Поскольку сеть глубокого обучения синхронизирует выводы RNN, вы получаете желаемый результат.

При использовании нескольких RNN, у вас есть несколько возможностей, как показано на рис. 11.2.

- **Один к одному**. Когда у вас есть один вход и ожидается один выход. Примеры в этой книге до сих пор используют именно этот подход. Они берут один случай, состоящий из определенного количества информативных переменных, и дают оценку, такую как число или вероятность.

- **Один ко многим**. Здесь есть один вход, и вы ожидаете последовательность выходов в результате. Автоматический подход к нейронным сетям таков: вы вводите одно изображение и создаете фразу, описывающую содержание изображения.
- **Многие к одному**. Классический пример RNN. Например, вы вводите текстовую последовательность и ожидаете одного результата в качестве вывода. Этот подход используется для получения оценки при анализе настроений или другой классификации текста.
- **Многие ко многим**. В качестве ввода вы предоставляете последовательность и ожидаете результирующую последовательность в качестве вывода. Это базовая архитектура для многих наиболее впечатляющих приложений искусственного интеллекта с глубоким обучением. Этот подход используется для машинного перевода (когда сеть, способна автоматически переводить фразу с английского на немецкий), чат-ботов (нейронная сеть, способная отвечать на ваши вопросы и беседовать с вами), и маркировки последовательности (классификация каждого изображения в видео).

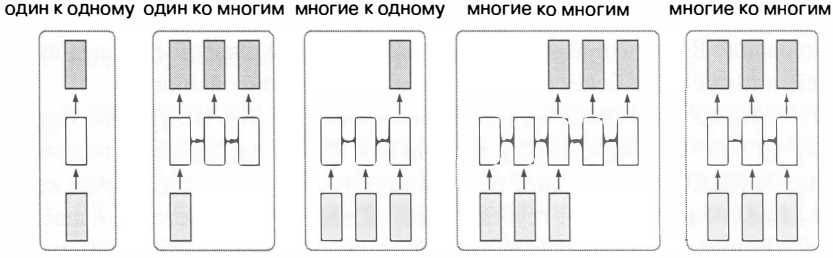


Рис. 11.2. Различные конфигурации вводов и выводов RNN

Машинный перевод (machine translation) — это способность машины правильно и осмысленно переводить с одного человеческого языка на другой. Это именно то, к чему ученые стремились на протяжении долгого времени, особенно в военных целях. Вы можете прочитать увлекательный рассказ обо всех попытках машинного перевода американских и российских ученых в статье Василия Зубарева по адресу `http://vas3k.com/blog/machine_translation/`. Настоящий прорыв произошел только после запуска Google Neural Machine Translation (GNMT), о котором вы можете прочитать на блоге Google AI: `https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html`. Для чтения последовательности слов на языке, который вы хотите перевести (*слой кодера* (encoder layer)), и возвращения результатов на другой уровень RNN (*слой декодера* (decoder layer)), превращающий его в переведенный вывод, GNMT опирается на ряд RNN, использующих парадигму "многие ко многим".

Нейронный машинный перевод требует двух слоев, поскольку грамматика и синтаксис одного языка могут сильно отличаться от другого. Одна RNN не может поддерживать две языковые системы одновременно, поэтому для работы с двумя языками необходима пара кодер-декодер. Система не идеальна, но это невероятный шаг вперед по сравнению с предыдущими решениями, описанными в статье Василия Зубарева, значительно уменьшающая количество ошибок в порядке слов, лексических ошибок (выбранное слово в переводе) и грамматике (как используются слова).

Кроме того, производительность зависит от обучающего комплекта, различий между языками и их специфических характеристик. Например, из-за структурного построения предложений на японском языке, правительство Японии сейчас инвестирует средства в разработку голосового переводчика реального времени, чтобы помочь во время Олимпийских игр в Токио в 2020 году и стимулировать туризм. О разработке усовершенствованного решения для нейронной сети см. на `https://www.japantimes.co.jp/news/2015/03/31/reference/translation-tech-gets-olympic-push/`.

ЗАПОМНИ!

RNN — это причина, по которой ваш голосовой помощник может отвечать вам, или ваш автоматический переводчик может дать вам перевод на иностранный язык. Поскольку RNN — это просто повторяющаяся операция умножения и суммирования, сети глубокого обучения не могут по-настоящему понять какой-либо смысл; они просто обрабатывают слова и фразы на основе того, что они узнали во время обучения.

Размещение правильной подписи на картинках

Другим возможным применением сетей RNN, использующим подход "многие ко многим", является *создание подписей* (caption generation), подразумевающее передачу изображения в нейронную сеть и получение текстового описания, объясняющего, что происходит на изображении. В отличие от чат-ботов и машинных переводчиков, вывод которых используется людьми, создание подписей связано с робототехникой. Это больше, чем просто создание описания изображения или видео. Создание подписей может помочь людям с ослабленным зрением воспринимать окружающую среду с помощью таких носимых устройств как Horus, или создать мост между изображениями и базами знаний (на текстовой основе) для роботов, что позволит им лучше понять свое окружение. Вы начинаете со специально разработанных наборов данных, таких как Pascal Sentence Dataset (см. `http://vision.cs.uiuc.edu/pascal-sentences/`); Flickr 30K (`http://shannon.cs.illinois.edu/DenotationGraph/`), состоящий из изображений Flickr, аннотированных

краудсорсингом; или набор данных MS Coco (`http://cocodataset.org`). Каждое изображение во всех этих наборах данных содержат одну или несколько фраз, поясняющих содержимое изображения. Например, в примере № 5947 набора данных MS Coco (`http://cocodataset.org/#explore?id=5947`) вы видите четыре летящих самолета, которые можно правильно обозначить как.

- Четыре самолета в небе над головой в пасмурный день.
- Четыре одномоторных самолета в облачный день.
- Группа из четырех самолетов, летящих в строю.
- Группа самолетов, летящих по небу.
- Флот самолетов, летящих по небу.

Хорошо обученная нейронная сеть должна быть способна производить аналогичные фразы, если ей будет представлена подобная фотография. Компания Google впервые опубликовала статью о решении этой проблемы *Show and Tell или Neural Image Caption* (NIC), в 2014 году, а затем обновила ее год спустя (см. статью на `https://arxiv.org/pdf/1411.4555.pdf`).

С тех пор Google открыла исходный код NIC и предложила его как часть инфраструктуры TensorFlow. Как нейронная сеть, она состоит из предварительно обученной CNN (такой как Google LeNet, победитель конкурса ImageNet 2014 года; см. подробности в главе 10), обрабатывающей изображения аналогично переносу обучения. Изображение превращается в последовательность значений, представляющих особенности изображения высокого уровня, обнаруженные CNN. Во время обучения изображение передается на слои RNN, которые запоминают характеристики изображения в своем внутреннем состоянии. CNN сравнивает результаты, полученные RNN, со всеми возможными описаниями, предоставленными учебным изображением, и вычисляется ошибка. Затем ошибка обратно распространяется в часть сети RNN, чтобы отрегулировать веса и помочь RNN научиться правильно подписывать изображения. После многократного повторения этого процесса с использованием разных изображений сеть готова увидеть новые изображения и предоставить описания этих новых изображений.

Долгая краткосрочная память

Может показаться, что использование краткосрочной памяти в RNN способно решить любую возможную проблему глубокого обучения. Тем не менее, нет RNN совершенно без изъянов. Проблемы с RNN возникают из-за их ключевой характеристики — рекурсии той же информации с течением времени. Одна и

та же информация, проходящая много раз через одни и те же ячейки, может постепенно затухнуть, а затем и исчезать, если вес ячейки слишком мал. Это так называемая проблема *исчезающего градиента* (vanishing gradient), когда сигнал обратного распространения ошибки исчезает при прохождении через нейронную сеть. Из-за этой проблемы вы не можете добавить в RNN слишком много слоев, поскольку их обновление становится затруднительным.


ЗАПОМНИ!

У RNN есть проблемы и посложнее. При обратном распространении, градиент (коррекция) имеет дело с исправлением ошибок, которое сети допускают при прогнозировании. Слои перед прогнозирующими слоями распределяют градиент по входным слоям, и они обеспечивают правильное обновление весов. Слои, достигаемые обновлением градиента, но небольшим, фактически останавливают обучение.

Фактически, внутренние сигналы обратного распространения RNN имеют тенденцию исчезать после нескольких рекурсий, поэтому нейронная сеть лучше обновляет и изучает самые последние последовательности. Сеть забывает прежние сигналы и не может связать их с более свежим вводом. Таким образом, RNN может стать совсем недальновидной, и вы не сможете успешно применить ее к задачам, требующим больше памяти.


ЗАПОМНИ!

Обратное распространение на уровне RNN работает как через слой в направлении других слоев, так и внутри каждой ячейки RNN, регулируя ее память. К сожалению, независимо от того, насколько силен сигнал, через некоторое время градиент исчезает.

Краткосрочная память и исчезающий градиент мешают RNN изучать достаточно длинные последовательности. Такие приложения, как создающие подписи к изображениям или машинный перевод, нуждаются в хорошей памяти во всех частях последовательности. Следовательно, для большинства приложений требуется альтернатива, и простые RNN были заменены различными рекуррентными ячейками.

Определение различий в памяти

Два ученых изучали проблему исчезающего градиента в RNN, и в 1997 году опубликовали важный документ, в котором предлагалось решение. Зепп Хохрайтер (Sepp Hochreiter), ученый в области информатики, внесший большой вклад в области машинного обучения, глубокого обучения и биоинформатики, а также Юрген Шмидхубер (Jürgen Schmidhuber), пионер в области искусственного интеллекта, опубликовали в журнале *Neural Computation* издательства MIT Press статью *Long Short-Term Memory* (`http://www.bioinf.at/`

publications/older/2604.pdf). В статье была представлена новая концепция рекуррентных ячеек, служащая теперь основой для всех невероятных приложений глубокого обучения, использующих последовательности. Первоначально от идеи отказались из-за ее слишком большой новизны (она опередила свое время). Новая концепция ячейки, предложенная в статье, называлась LSTM (сокращение от Long Short-Term Memory — *долгая краткосрочная память*), и сегодня используется в более 4 миллиардах нейронных операций в день. Личная домашняя страница Шмидхубера расположена по адресу `http://people.idsia.ch/~juergen/`. LTSM считается стандартом для машинного перевода и чат-ботов.

ЗАПОМНИ!

Компании Google, Apple, Facebook, Microsoft и Amazon разработали свои продукты на основе технологии LSTM, предложенной Хохрайтером и Шмидхубером. Если бы LSTM не был изобретен, такие продукты, как интеллектуальные голосовые помощники и машинные переводчики, работали бы иначе.

Основная идея LSTM заключается в том, что RNN различает краткосрочное и долгосрочное состояния. Состояние памяти ячейки и LSTM разделяются на такие каналы.

- **Краткосрочный**. Входные данные непосредственно смешиваются с данными, поступающими из последовательности.
- **Долгосрочный**. Из краткосрочной памяти выбираются только те элементы, которые необходимо хранить на протяжении длительного времени.

Более того, канал долгосрочной памяти имеет меньше параметров для настройки. Долгосрочная память использует некие сложения и умножения с элементами, поступающими из краткосрочной памяти, и ничего более, что делает ее почти прямой информационной магистралью. (Исчезающий градиент не может остановить поток информации.)

Архитектура LSTM

В основе LSTM лежат *вентили* (gate), являющиеся внутренними механизмами, использующими суммирование, умножение и функцию активации для регулирования потока информации внутри ячейки LSTM. Регулируя поток, вентиль способен содержать, улучшать или отбрасывать поступившую из последовательности информацию, как в краткосрочной, так и в долгосрочной памяти. Этот поток напоминает электрическую цепь. На рис. 11.3 показана внутренняя структура LSTM.

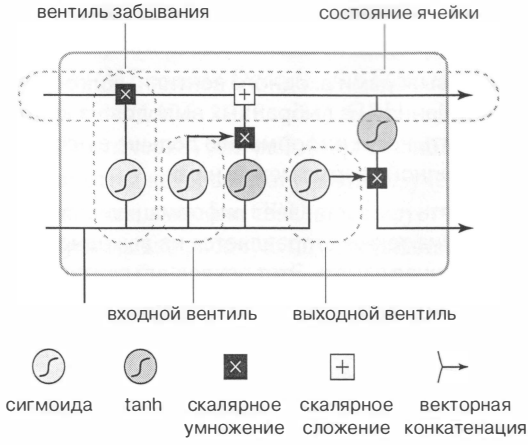


Рис. 11.3. Внутренняя структура LSTM, с двумя потоками памяти и вентилями

Поначалу разные маршруты и вентили могут показаться немного сложными, но следующая последовательность шагов поможет вам понять их.

1. Краткосрочная память, происходящая из предыдущего состояния (или случайных значений), встречает вновь введенную часть последовательности, и они смешиваются вместе, создавая первое ветвление.
2. Сигнал краткосрочной памяти, несущий как выходной сигнал, так и вновь введенный сигнал, пытается достичь долгосрочной памяти, проходя через *вентиль забывания* (forget gate), используемый для забывания определенных данных. (Технически, вы видите ветвление, где сигнал дублируется.)
3. Вентиль забывания решает, какую краткосрочную информацию следует отбросить перед передачей в долгосрочную память. Сигмоидная функция активация отменяет бесполезные сигналы и улучшает то, что кажется важным сохранить и запомнить.
4. Информация, проходящая через вентиль забывания, поступает в канал долгосрочной памяти, переносящий информацию из предыдущих состояний.
5. Значения долгосрочной памяти и вывод вентиля забывания перемножаются.
6. Краткосрочная память, не прошедшая через вентиль забывания, снова дублируется и занимает другую ветвь; то есть одна часть переходит к выходному вентилю, а другая — к входному вентилю.
7. На входном вентиле данные краткосрочной памяти проходят отдельно через сигмоидную функцию и функцию tanh. Затем выводы этих двух функций сначала перемножаются, а затем добавляются в долгосрочную память. Влияние на

долгосрочную память зависит от сигмоидной функции, которая забывает или запоминает сигнал в зависимости от его важности.

8. После сложения с выводами входного вентиля, долгосрочная память не меняется. Будучи созданной из выбранных выводов из краткосрочной памяти, долгосрочная память несет информацию дольше в последовательности и не реагирует на временной разрыв между ними.

9. Долгосрочная память предоставляет информацию непосредственно в следующее состояние. Она также отправляется на выходной вентиль, куда также сходится краткосрочная память. Этот последний вентиль нормализует данные из долгосрочной памяти, используя активацию tanh, и фильтрует краткосрочную память, используя сигмоидную функцию. Два результата умножаются друг на друга, а затем отправляются в следующее состояние.


СОВЕТ

Для своих вентилей LSTM используют как сигмоидную функцию активации, так и tanh. Следует помнить, что функция tanh нормализует входной сигнал от –1 до 1, а сигмоидная функция уменьшает его от 0 до 1. Следовательно, в то время как функция активации tanh сохраняет ввод внутри работоспособного диапазона значений, сигмоида может отключать ввод, поскольку она подталкивает более слабые сигналы к нулю и гасит их. Другими словами, сигмоидная функция позволяет запоминать (усиливать) или забывать (ослаблять) сигнал.

Открывая интересные варианты

Есть несколько вариантов LSTM, каждый из которых маркируется дополнительными цифрами или буквами в имени, например LSTM4, LSTM4a, LSTM5, LSTM5a и LSMT6, чтобы показать, что они имеют измененную архитектуру, хотя основные концепции решения прежние. Наиболее популярной и актуальной модификацией, среди этих вариантов, является использование *смотровых глазков* (peephole connection), являющихся просто конвейерами данных, позволяющих всем или некоторым слоям вентилей просматривать долгосрочную память (в терминах RNN, состояния ячеек). Позволяя заглянуть в долгосрочную память, RNN может основывать краткосрочные решения на шаблонах замеченных ранее, которые были консолидированы в процессе работы. В Keras вы можете найти обычную реализацию LSTM с помощью команды `keras.layers.LSTM` (`keras.layers.CuDNNLST` — версия для графического процессора), чего достаточно для большинства приложений. Если вам нужно протестировать варианты глазков, вы можете изучить реализацию TensorFlow, которая предлагает дополнительные параметры на уровне архитектуры ячейки LSTM.

Другой вариант радикальней. *Управляемый рекуррентный блок* (Gated Recurrent Unit — GRU) впервые появился в документе *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation* (`https://arxiv.org/pdf/1406.1078.pdf`). GRU действуют как упрощение архитектуры LSTM. Фактически, они работают с использованием информационных вентилей, параметры которых могут быть изучены так же, как у LSTM. В целом, поток информации в ячейке GRU использует линейный маршрут, поскольку GRU использует только рабочую память (эквивалентную долгосрочной памяти в терминах LSTM). Эта рабочая память обновляется *вентилем обновления* (update gate) с использованием текущей информации, предоставленной сети. Затем обновленная информация снова суммируется с исходной рабочей памятью в вентиле, объединяющем эти два элемента. Он называется *вентилем сброса* (reset gate), поскольку он выбирает информацию рабочей памяти, чтобы эффективно сохранять память данных, освобождаемую на следующем шаге последовательности. Вы можете увидеть простую схему потока на рис. 11.4.

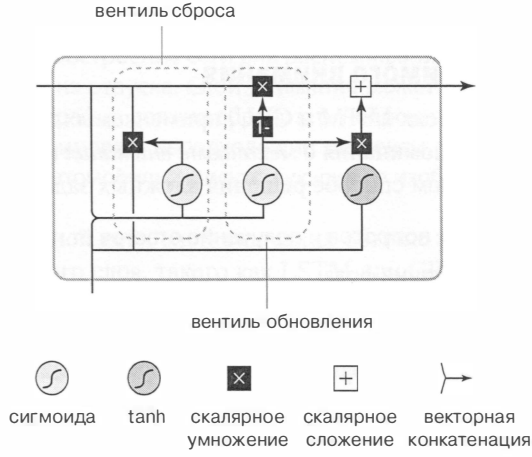


Рис. 11.4. Внутренняя структура GRU с одним потоком памяти и двумя вентилями

В отличие от LSTM, GRU используют вентили сброса, которые останавливают информацию, подлежащую забыванию. GRU используют также вентиль обновления, поддерживающий полезные сигналы. GRU имеют уникальную память, без различия между долгосрочной и краткосрочной.

В своих сетях вы можете использовать как слои GRU, так и LSTM, не слишком меняя код. Импортируйте слой, используя код `keras.layers.GRU` (или `keras.layers.CuDNNGRU` версия для графического процессора с использованием библиотеки NVIDIA CuDNN; более подробную информацию см. на

https://developer.nvidia.com/cudnn), и взаимодействуйте с ним как со слоем LSTM. Вы задаете параметр units, определяя количество блоков GRU, необходимых в одном слое. Переход с LSTM на GRU обеспечивает следующие преимущества, а также некоторые компромиссы.

- GRU обрабатывают сигналы как LSTM и потенциально избегают проблемы исчезающего градиента, но они не различают длинную и короткую память, поскольку полагаются на одну рабочую память — состояние ячейки, многократно обрабатываемое ячейками GRU.
- GRU проще, чем LSTM, но они также менее способны запоминать прошлые сигналы, поэтому LSTM имеют преимущество при работе с более длинными последовательностями.
- GRU обучается быстрее, чем LSTM (у них меньше параметров для настройки).
- GRU работают лучше, чем LSTM, когда у вас меньше учебных данных, поскольку они с меньшей вероятностью переобучатся на полученной информации.

Получение необходимого внимания

Когда вы читаете о слоях LSTM и GRU, применяемых к языковым задачам, вы нередко встречаете упоминания о *механизме внимания* (attention mechanism) как наиболее эффективном способе решения сложных задач, таких как

- Задавание вопросов и получение ответов от нейронной сети.
- Классификация фраз.
- Перевод текста с одного языка на другой.

Механизм внимания рассматривается как современное решение для этих сложных задач, и, несмотря на то, что он отсутствует среди доступных в настоящее время слоев в пакетах TensorFlow и Keras, можно найти работоспособную реализацию с открытым исходным кодом или даже ее не сложно реализовать самостоятельно.

СОВЕТ

Вы можете начать создавать собственный механизм внимания, рассмотрев реализацию с открытым исходным кодом, разработанную инженером-исследователем Филиппом Реми (Philippe Rémy), см. https://github.com/philipperemy/keras-attention-mechanism.

Слои внимания впервые описаны в статье *Neural machine translation by jointly learning to align and translate*, написанной Дмитрием Богдановым

(Dzmitry Bahdanau), Кунг Хун Чо (Kyunghyun Cho) и Йошуа Бенджио (Yoshua Bengio) в 2014 году (`https://arxiv.org/abs/1409.0473v7`). *Слои внимания* (attention layer), реализующие механизм внимания, — это векторы весов, выражающие важность элемента в наборе, обрабатываемом глубокой нейронной сетью. Набор элементов зачастую включает в себя последовательность, обрабатываемую RNN, но это может также быть изображение. Фактически, слой внимания может решать два вида задач.

- При обработке длинных последовательностей слов, связанные слова могут встречаться в последовательности далеко друг от друга. Например, местоимения, как правило, трудно обработать в RNN, поскольку они не могут связать местоимение с элементами, переданными в последовательности ранее. Слой внимания может выделить ключевые элементы во фразе до того, как RNN начнет обработку последовательности.
- При обработке больших изображений, множество встречающихся на нем объектов могут отвлечь нейронную сеть от обучения правильной классификации целевых объектов. Примером может служить построение сети для распознавания ориентиров на фотографиях из отпуска. Слой внимания может определить, какую часть фотографии должна обработать нейронная сеть, и предложить RNN игнорировать неподходящие элементы, такие как человек, собака или автомобиль, присутствующие на изображении.

ЗАПОМНИ!

В нейронной сети слой внимания обычно располагается после рекуррентного слоя, такого как LSTM или GRU. В 2017 году исследователи из Google создали автономный механизм внимания, способный работать, не полагаясь на предыдущие рекуррентные слои, и работающий намного лучше, чем предыдущие решения. Они назвали такую архитектуру *Transformer*.

3 Взаимодействие с глубоким обучением

В ЭТОЙ ЧАСТИ...

- Классификация изображений
- Работа с CNN
- Основы обработки текстов на естественном языке
- Создание произведений изобразительного искусства и музыки
- Глубокое обучение с подкреплением

Глава 12

Классификация изображений

В ЭТОЙ ГЛАВЕ...

- Признание ключевого вклада задач распознавания образов
- Важность приращения изображения
- Использование набора данных немецких дорожных знаков
- Создание собственной сети CNN, способной классифицировать дорожные знаки

Понимание работы сверточных слоев (см. главу 10) является лишь отправной точкой. Теория может только объяснить, как все работает, но она не может адекватно описать успех решений глубоких нейронных сетей в области распознавания образов. Большая часть успеха этой технологии, особенно в приложениях искусственного интеллекта, обусловлена наличием подходящих данных для обучения и тестирования сетей изображений, их применения к различным задачам, благодаря обучению с помощью переноса, и дальнейшему совершенствованию технологии, позволяющему ей отвечать на сложные вопросы о содержании изображения.

В этой главе мы углубимся в тему классификации объектов и задач обнаружения, чтобы раскрыть их вклад в основы современного ренессанса глубокого обучения. Конкурсы, например, основанные на наборе данных ImageNet, не только предоставляют подходящие данные для обучения многоразовых сетей, применимых для различных целей (благодаря переносу обучения, как обсуждалось в главе 10), но и ведут исследователей к поиску более новых разумных решений для увеличения возможностей нейронной сети по пониманию

изображения. Слои локальной нормализации и пересечения являются технологическими решениями, слишком сложными для обсуждения в этой книге, но вы должны знать, что они революционные. Все они были представлены нейронными сетями, победившими в конкурсе ImageNet: AlexNet (в 2012 году), GoogleLeNet (в 2014 году) и ResNet (в 2015 году).

Благодаря набору данных о немецких дорожных знаках (German Traffic Sign Benchmark), предоставленному Рурским университетом в Бохуме (Германия), глава завершается примером использования набора данных изображений. Используя набор данных, вы создадите свою собственную CNN для распознавания дорожных знаков, применяя приращение изображений и взвешивание для балансировки частоты различных классов в примерах.

ЗАПОМНИ!

Вам не нужно вводить исходный код примеров из этой главы вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файле `DL4D_12_German_Traffic_Sign_Benchmark.ipynb`.

Конкурсы по классификации изображений

Слои CNN для распознавания изображений были впервые задуманы Яном Лекуном и командой исследователей. Компания AT&T фактически внедрила LeNet5 (нейронную сеть для рукописных цифр, описанную в главе 10) в считыватели чеков ATM. Однако изобретение не помешало еще одной зиме искусственного интеллекта, которая началась в 1990-х годах, когда многие исследователи и инвесторы снова потеряли веру в то, что компьютеры смогут достичь какого-либо прогресса в содержательном разговоре с людьми, переводе с разных языков, понимании изображений и рассуждениях в манере людей.

На самом деле, общественное доверие подорвали уже экспертные системы. *Экспертные системы* (expert system) — это набор автоматических правил, устанавливаемых людьми, чтобы компьютеры могли выполнять определенные операции. Тем не менее, новая зима искусственного интеллекта помешала развитию нейронных сетей в пользу различных алгоритмов машинного обучения. В то время компьютерам не хватало вычислительной мощности, были также определенные ограничения, такие как проблема исчезающего градиента. (Исчезающий градиент и другие ограничения, мешавшие глубоким нейронным архитектурам, обсуждает глава 9.) В то время данным не хватало также сложности, а потому сложная и революционная CNN, такая как LeNet5, уже работавшая с технологией и ограничениями своего времени, имела мало шансов показать свою истинную силу.

Лишь немногие исследователи, такие как Джеффри Хинтон, Ян Лекун, Юрген Шмидхубер и Йошуа Бенджио, продолжали разрабатывать технологии нейронных сетей, стремясь добиться прорыва, который закончил бы зиму искусственного интеллекта. Между тем, в 2006 году Фей-Фей Ли, профессор информатики в Университете Иллинойс в Эрбана-Шампейн (в настоящее время доцент в Стэнфорде, а также директор Стэнфордской лаборатории искусственного интеллекта и Стэнфордской лаборатории зрения), попыталась сделать это, предоставив больше реальных наборов данных для лучшей проверки алгоритмов. Она начала накапливать невероятное количество изображений, представляющих большое количество классов объектов. Вы можете прочитать об этих усилиях в главе 10. Предлагаемые классы охватывают различные типы объектов, как естественных (например, 120 пород собак), так и рукотворных (например, средства передвижения). Вы можете изучить их все по адресу `http://image-net.org/challenges/LSVRC/2014/browse-synsets`. Используя этот огромный набор данных изображений для обучения, исследователи заметили, что их алгоритмы стали работать лучше (в то время не существовало ничего подобного ImageNet), а затем они начали проверять новые идеи и улучшать архитектуры нейронных сетей.

ImageNet и MS COCO

Важность и влияние конкурса ImageNet (известного также как ImageNet Large Scale Visual Recognition Challenge или ILSVRC; `http://image-net.org/challenges/LSVRC/`) на разработку решений глубокого обучения распознаванию изображений можно резюмировать в трех ключевых моментах.

- **Помощь в возрождении глубоких нейронных сетей**. Архитектура AlexNet CNN (разработанная Алексом Крижевски, Ильей Суцкевером (Ilya Sutskever) и Джеффри Хинтоном) выиграла конкурс ILSVRC 2012 года с большим отрывом от других решений.
- **Стимулирование различных команд исследователей для разработки более сложных решений**. ILSVRC улучшил производительность CNN. VGG16, VGG19, ResNet50, Inception V3, Xception и NASNet — это нейронные сети, проверенные на изображениях ImageNet, которые вы можете найти в пакете Keras (`https://keras.io/applications/`). Каждая архитектура представляет собой улучшение по сравнению с предыдущими архитектурами и вводит ключевые инновации глубокого обучения.
- **Возможность переноса обучения**. Конкурс ImageNet помог выработать набор весов, которые сделали это возможным. 1,2 миллиона учебных изображений ImageNet, распределенных по 1000 классов, помогли создать сверточные сети, верхние слои которых на самом деле способны обобщать задачи, отличные от ImageNet.

Недавно несколько исследователей начали подозревать, что новейшие нейронные архитектуры превосходят набор данных ImageNet. В конце концов, для выбора лучших сетей на протяжении многих лет использовался один и тот же набор тестов, как отмечают исследователи Бенджамин Рехт (Benjamin Recht), Ребекка Рулофс (Rebecca Roelofs), Людвиг Шмидт (Ludwig Schmidt) и Вайшаал Шанкар (Vaishaal Shankar), см. `https://arxiv.org/pdf/1806.00451.pdf`.


ЗАПОМНИ!

Другие исследователи из команды Google Brain (Саймон Корнблит (Simon Kornblith), Джонатон Шленс (Jonathon Shlens) и Куок В.Ле (Quoc V.Le)) обнаружили корреляцию между точностью, полученной в ImageNet, и производительностью, полученной при переносе обучения той же сети на другие наборы данных. Они опубликовали свои выводы в статье *Do Better ImageNet Models Transfer Better?* (`https://arxiv.org/pdf/1805.08974.pdf`). Интересно, что они также указали, что, если сеть переобучена на ImageNet, с обобщением могут возникнуть проблемы. Поэтому рекомендуется проверять перенос обучения на основе самой последней и наиболее эффективной сети ImageNet, но не останавливаться на достигнутом. Вы можете обнаружить, что некоторые менее эффективные сети на самом деле лучше для вашей задачи.

Другие возражения по поводу использования ImageNet заключаются в том, что обычные изображения в повседневных сценах содержат больше объектов и что эти объекты могут быть нечетко видны, если они частично закрыты другими объектами или потому, что они смешиваются с фоном. Если вы хотите использовать предварительно обученную сеть ImageNet в повседневной жизни, например, при создании приложения или робота, производительность может вас разочаровать. Следовательно, с тех пор, как прекратился конкурс ImageNet (утверждалось, что повышение производительности в результате продолжения работы с набором данных оказалось невозможным), исследователи все больше внимания уделяют использованию альтернативных общедоступных наборов данных для совершенствования своих собственных CNN и улучшения современного состояния в распознавании изображений. Альтернативы на данный момент таковы.

- **PASCAL VOC (Visual Object Classes — классы визуальных объектов)** `http://host.robots.ox.ac.uk/pascal/VOC/`. Этот набор данных, разработанный Оксфордским университетом, устанавливает стандарт обучения нейронной сети для маркировки нескольких объектов на одном изображении, стандарт PASCAL VOC XML. Конкурс, связанный с этим набором данных, был прекращен в 2012 году.

- **SUN** `https://groups.csail.mit.edu/vision/SUN/`. Созданный Массачусетским технологическим институтом (MIT), этот набор данных предоставляет эталонные тесты, позволяющие определять производительность CNN. Никаких конкурсов с этим не связано.
- **MS COCO** `http://cocodataset.org/`. Этот набор данных подготовлен корпорацией Microsoft для целой серии активных конкурсов.

В частности, Microsoft Common Objects в наборе данных Context (отсюда и название MS COCO) предлагают меньше учебных изображений, чем ImageNet, но каждое изображение содержит несколько объектов. Кроме того, все объекты отображаются в реалистичных положениях (не в постановке) и условиях (зачастую на открытом воздухе и в общественных местах, таких как дороги и улицы). Для различения объектов в наборе данных предусмотрены как контуры в пиксельных координатах, так и надписи в стандарте PASCAL VOC XML, причем каждый объект определяется не только классом, но и его координатами на изображениях (прямоугольник, указывающий его положение на изображении). Этот прямоугольник, *рамка* (bounding box), определяется простым способом с использованием четырех пикселей, в отличие от множества пикселей, необходимых для определения объекта по его контурам.


ЗАПОМНИ!

Набор данных ImageNet недавно начал предлагать, по меньшей мере, один миллион изображений, множество объектов для распознавания и ограничивающие их рамки.

Магия приращения данных

Даже если для вашей модели глубокого обучения есть доступ к большим объемам данных, таким как наборы данных ImageNet и MS COCO, этого может быть недостаточно из-за множества параметров, встречающихся в большинстве сложных нейронных архитектур. На самом деле, даже если вы используете такие методы, как отсев (как описан в главе 9), переобучение все же возможно. *Переобучение* (overfitting) происходит, когда сеть запоминает входные данные, а не изучает полезные шаблоны в них. Помимо отсева, от переобучения могут помочь другие методы, такие как LASSO, Ridge и ElasticNet. Тем не менее, ничто не является настолько эффективным для повышения прогностических возможностей вашей нейронной сети, как добавление большего количества примеров для обучения.

Приращение изображений решает проблемы отсутствия примеров для передачи в нейронную сеть за счет искусственного создания новых изображений из существующих. *Приращение изображения* (image augmentation) подразумевает

различные операций обработки изображения, осуществляемые отдельно или совместно для получения изображения, отличного от исходного. Результат помогает нейронной сети лучше понять задачу распознавания.

ЗАПОМНИ!

Первоначально LASSO, Ridge и ElasticNet были способами ограничения весов модели линейной регрессии, являющейся статистическим алгоритмом для вычисления регрессионных оценок. В нейронной сети они работают аналогичным образом, снижая до минимума общую сумму весов в сети без ущерба для правильности прогнозов. LASSO стремится свести множество весов к нулю, выбирая, таким образом, только лучшие веса. Ridge напротив, имеет тенденцию ослаблять все веса, избегая более крупных весов, способных привести к переобучению. Наконец, ElasticNet представляет собой смесь подходов LASSO и Ridge, что обеспечивает компромисс между стратегиями выбора и демпфирования.

Например, если у вас есть слишком яркие или слишком размытые учебные изображения, обработка преобразует их в более темные и более резкие версии. Эти новые версии иллюстрируют характеристики, на которых должна сосредоточиться нейронная сеть, а не предоставляют примеры, ориентированные на качество изображения. Кроме того, могут помочь, поворот, обрезка или отражение изображения, как показано на рис. 12.1, поскольку они заставляют сеть снова изучать полезные признаки изображения, независимо от того, как выглядит объект.

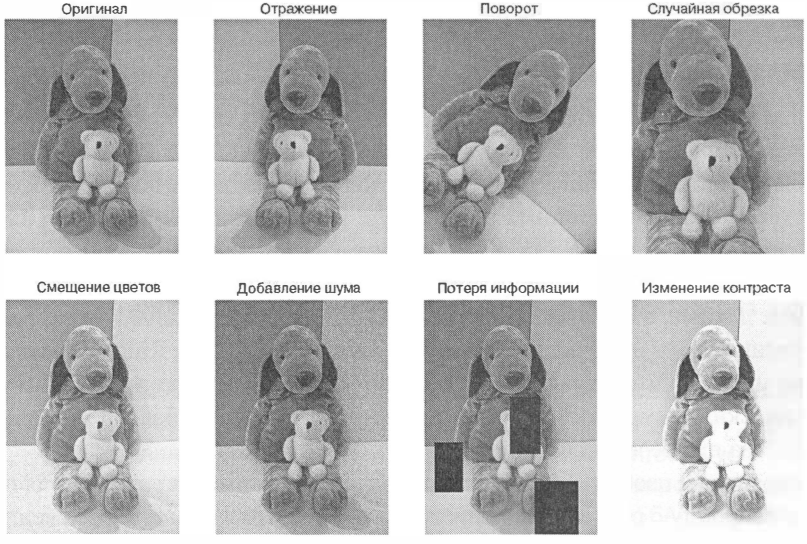


Рис. 12.1. Некоторые общие способы приращения изображений

Наиболее распространенными процедурами приращения изображений, как показано на рис. 12.1, являются следующие.

- **Отражение**. Отражение изображения по оси проверяет способность алгоритма находить его независимо от точки зрения. Общий смысл вашего изображения должен сохраняться даже при зеркальном отражении. Некоторые алгоритмы не могут найти объекты, отраженные по вертикали или горизонтали, особенно если оригинал содержит слова или другие специфические признаки.
- **Поворот**. Поворот изображения позволяет проверить алгоритм на распознавание под определенными углами; моделирование разных точек зрения или неточно откалиброванной визуализации.
- **Случайная обрезка**. Обрезка изображения заставляет алгоритм сосредоточиться на компоненте изображения. Вырезание области и расширение ее до того же размера, что и у стандартного изображения, позволяет проверить распознавание частично скрытых признаков изображения.
- **Смещение цветов**. Изменение нюансов цветов изображения обобщает ваш пример, поскольку в реальном мире цвета могут меняться или записываться иначе.
- **Добавление шума**. Добавление случайных шумов проверяет способность алгоритма обнаруживать объект, даже если качество изображения ниже идеального.
- **Потеря информации**. Случайное удаление частей изображения имитирует визуальную обструкцию. Это также помогает нейронной сети научиться полагаться на общие признаки изображения, а не на его детали (которые могут быть удалены случайным образом).
- **Изменение контраста**. Изменение яркости делает нейронную сеть менее чувствительной к условиям освещения (например, к дневному или искусственному свету).

СОВЕТ

Вам не нужно специализироваться на обработке изображений, чтобы использовать эту мощную технику. Keras предлагает способ легко включить приращение в любое обучение, используя функцию `ImageDataGenerator` (`https://faroit.github.io/keras-docs/1.2.2/preprocessing/image/`).

Основная цель функции `ImageDataGenerator` — создавать пакеты входных данных для передачи в нейронную сеть. Это значит, что вы можете получить данные в виде фрагментов из массива NumPy, используя метод `.flow`. Кроме

того, вам не нужно хранить все учебные данные в памяти, поскольку метод `.flow_from_directory` может получить их непосредственно с диска. Когда `ImageDataGenerator` извлекает пакеты изображений, он может преобразовывать их, используя масштабирование (изображения состоят из целых чисел в диапазоне от 0 до 255, но нейронные сети лучше всего работают с числами с плавающей запятой в диапазоне от нуля до единицы) или применяя такие преобразования как.

- **Стандартизация**. Приведение всех ваших данных к единой шкале в результате установки среднего значения в ноль и стандартного отклонения в единицу (в качестве статистической стандартизации) на основе среднего значения и стандартного отклонения всего набора данных (*по признакам*) или отдельно для каждого изображения (*по примерам*).
- **Отбеливание ZCA**. Удаление из изображения любой избыточной информации при сохранении сходства с исходным изображением.
- **Случайные поворот, сдвиг и отражение**. Меняют внешний вид объекта на отличный от оригинала.
- **Изменение размеров**. Сопоставление размеров данных между изображениями. Например, преобразование изображений BGR (формат цветного изображения, ранее популярный среди производителей камер) в стандартный RGB.

Когда вы используете `ImageDataGenerator` для обработки пакетов изображений, вы ограничены не размером памяти компьютера в вашей системе, а размером вашего хранилища (например, размером жесткого диска) и скоростью передачи из него. Вы даже можете получать необходимые данные из Интернета, если ваше соединение достаточно быстрое.


СОВЕТ

Вы можете получить еще более мощные приращения изображений, используя такой пакет как albumentations (`https://github.com/albu/albumentations`). Его создали Александр Буслаев (Alexander Buslaev), Алекс Паринов (Alex Parinov), Владимир И. Игловиков (Vladimir I. Iglovikov) и Евгений Хведченя (Evegene Khvedchenya), основываясь на своем опыте в решении многих задач распознавания изображений. Пакет предлагает невероятный набор возможных инструментов для обработки изображений в зависимости от выполняемой задачи и типа используемой нейронной сети.

Распознавание дорожных знаков

После обсуждения теоретических оснований и характеристик CNN, вы можете попробовать их построить. TensorFlow и Keras могут создать классификатор изображений с разделителями для конкретной задачи. Для успешного выполнения конкретных задач не требуется изучения большого разнообразия признаков изображения. Следовательно, вы можете легко решать их, используя простые архитектуры, такие как LeNet5 (CNN, революционизировавшая распознавание изображений, см. главу 10) или нечто подобное. В этом примере выполняется интересная, реалистичная задача с использованием набора данных о немецких дорожных знаках (German Traffic Sign Recognition Benchmark — GTSRB), который можно найти на странице института нейроинформатики Рурского университета в Бохуме (Institute für NeuroInformatik at Ruhr-Universität Bochum) по адресу: `http://benchmark.ini.rub.de/?section=gtsrb`.

ЗАПОМНИ!

Чтение дорожных знаков является сложной задачей из-за различий во внешнем виде в реальных условиях. GTSRB является контрольной задачей для оценки различных алгоритмов машинного обучения, применяемых к этой задаче. Вы можете прочитать о создании этой базы данных в статье *Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition* Дж. Сталлкампа (J. Stallkampand) и других авторов по адресу `https://www.ini.rub.de/upload/file/1470692859_c57fac98ca9d02ac701c/stallkampetal_gtsrb_nn_si2012.pdf`.

Набор данных GTSRB предлагает более 50 000 изображений, сгруппированных в 42 класса (дорожные знаки), что позволяет использовать его для задач мультиклассовой классификации. В задаче мультиклассовой классификации вы указываете вероятность того, что изображение принадлежит к классу, и принимаете наибольшую вероятность в качестве правильного ответа. Например, знак "Дорожные работы" приведет к тому, что алгоритм классификации будет вырабатывать высокие вероятности для всех предупреждающих знаков. (Максимальная вероятность должна соответствовать его классу.) Размытость изображения, плохое разрешение, плохое освещение и тока зрения делают задачу довольно сложной для компьютера (а иногда и для человека), как вы можете убедиться на некоторых примерах, извлеченных из набора данных (рис. 12.2).

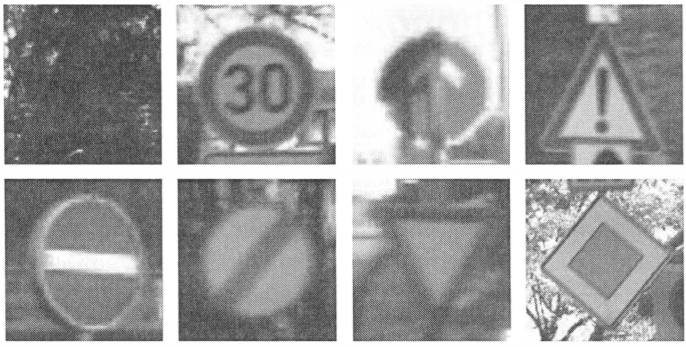


Рис. 12.2. Несколько примеров из набора немецких дорожных знаков

Подготовка данных изображения

Пример начинается с настройки модели, настройки оптимизатора, предварительной обработки изображений и создания сверток, слоев подвыборки и плотных слоев, как показано в следующем коде. (О работе с TensorFlow и Keras см. главу 4.)

```
import numpy as np
import zipfile
import pprint
from skimage.transform import resize
from skimage.io import imread
import matplotlib.pyplot as plt
% matplotlib inline

import warnings
warnings.filterwarnings("ignore")

from keras.models import Sequential
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import (Flatten, Dense, Dropout)
```

СОВЕТ

Набор данных содержит более 50 000 изображений, и подходящая нейронная сеть может достичь почти человеческого уровня точности при распознавании дорожных знаков. Такое приложение потребует большого количества компьютерных вычислений, и выполнение этого кода локально может занять много времени на вашем компьютере, в зависимости от его типа. Аналогично, его выполнение на Colab может занять еще больше времени в зависимости от ресурсов,

которые Google предоставляет вам, в том числе от того, есть ли у вас доступ к графическому процессору, как указано в главе 4. Замер времени для этого первоначального приложения поможет вам узнать, какая среда быстрее для запуска больших наборов данных, ваша локальная машина или Colab. Однако лучшая среда — это та, которая дает наиболее последовательные и надежные результаты. У вас может не быть надежного соединения с Интернетом, что делает Colab плохим выбором.

В этот момент пример получает набор данных GTSRB из его расположения в Интернете (указанном ранее веб-сайте Рурского университета в Бохуме). Следующий фрагмент кода загружает его в тот же каталог, что и код Python. Обратите внимание, что процесс загрузки может занять некоторое время, поэтому сейчас самое время наполнить чашку.

```
import urllib.request
url = "http://benchmark.ini.rub.de/Dataset/\
GTSRB_Final_Training_Images.zip"
filename = "./GTSRB_Final_Training_Images.zip"
urllib.request.urlretrieve(url, filename)
```

После получения набора данных в виде файла `.zip` из Интернета, код устанавливает размер изображения. (Все изображения меняются на квадратные, поэтому размер представляет стороны в пикселях.) Код выделяет также часть данных для тестирования, что означает исключение определенных изображений из обучения, чтобы получить более надежную меру работоспособности нейронной сети.

Циклический перебор файлов, хранящихся в загруженном файле `.zip`, извлекает отдельные изображения, изменяет их размер, сохраняет метки классов и добавляет изображения в два отдельных списка: один для обучения и один для тестирования. При сортировке используется хеш-функция, переводящая название изображения в число и на основе этого числа решающая, куда добавить изображение.

```
IMG_SIZE = 32
TEST_SIZE = 0.2
X, Xt, y, yt = list(), list(), list(), list()

archive = zipfile.ZipFile(
                './GTSRB_Final_Training_Images.zip', 'r')
file_paths = [file for file in archive.namelist()
              if '.ppm' in file]

for filename in file_paths:
    img = imread(archive.open(filename))
```

```
    img = resize(img,
                 output_shape=(IMG_SIZE, IMG_SIZE),
                 mode='reflect')
    img_class = int(filename.split('/')[-2])

    if (hash(filename) % 1000) / 1000 > TEST_SIZE:
        X.append(img)
        y.append(img_class)
    else:
        Xt.append(img)
        yt.append(img_class)

archive.close()
```

После выполнения задания код сообщает о последовательностях обучающих и тестовых примеров.

```
test_ratio = len(Xt) / len(file_paths)
print("Train size:{} test size:{} ({:0.3f})".format(len(X),
                len(Xt),
                test_ratio))
```

Размер обучающего набора составляет более 30 000 изображений, а тестового — почти 8 000 (20% от общего количества).

```
Train size:31344 test size:7865 (0.201)
```

Ваши результаты могут немного отличаться от приведенных. Например, следующий запуск примера показал размер обучающего набора 31 415 и тестового 7 794. Нейронные сети могут лучше решать задачи множественной классификации, когда классы схожи по численности, или они имеют тенденцию концентрировать свое внимание на изучении лишь более многочисленных классов. Следующий код проверяет распределение классов.

```
classes, dist = np.unique(y+yt, return_counts=True)
NUM_CLASSES = len(classes)
print ("No classes:{}".format(NUM_CLASSES))

plt.bar(classes, dist, align='center', alpha=0.5)
plt.show()
```

На рис. 12.3 показано, что классы не сбалансированы. Некоторые дорожные знаки встречаются чаще других (например, знак "Движение без остановки запрещено" встречаются чаще, чем знак "Дикие животные").

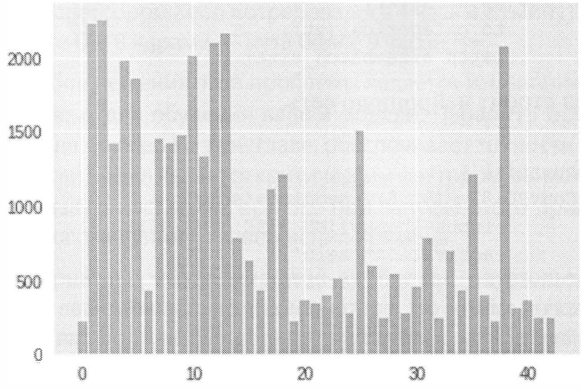


Рис. 12.3. Распределение классов

В качестве решения, код вычисляет *вес* (weight), являющийся отношением, основанным на частотах классов, которые нейронная сеть использует для усиления сигнала, получаемого из более редких примеров, и для ослабления более частых.

```
class_weight = {c:dist[c]/np.sum(dist) for c in classes}
```

Выполнение задачи классификации

После установки весов код определяет генератор изображений, часть кода, извлекающая изображения в пакетном режиме (примеры предварительно определенного размера) для обучения и проверки, нормализует их значения и для борьбы с переобучением применяет приращение в результате небольшого смещения и поворота изображений. Обратите внимание, что следующий код применяет приращение только к генератору обучающих изображений, а не к генератору тестовых изображений, поскольку тестировать необходимо только оригинальные изображения.

```
batch_size = 256
tgen=ImageDataGenerator(rescale=1./255,
                        rotation_range=5,
                        width_shift_range=0.10,
                        height_shift_range=0.10)

train_gen = tgen.flow(np.array(X),
                      to_categorical(y),
                      batch_size=batch_size)

vgen=ImageDataGenerator(rescale=1./255)
```

```
val_gen = vgen.flow(np.array(Xt),
                    to_categorical(yt),
                    batch_size=batch_size)
```

Наконец, код строит нейронную сеть.

```
def small_cnn():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), padding='same',
                     input_shape=(IMG_SIZE, IMG_SIZE, 3),
                     activation='relu'))
    model.add(Conv2D(64, (5, 5), activation='relu'))
    model.add(Flatten())
    model.add(Dense(768, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(NUM_CLASSES, activation='softmax'))
    return model

model = small_cnn()
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

Нейронная сеть состоит из двух сверток, одна с 32 каналами, другая с 64, обе работают с ядром размером (5,5). За свертками следует плотный слой из 768 узлов. Исключение (отбрасывание 40% узлов) упорядочивает этот последний слой, а softmax активирует его (таким образом, сумма выходных вероятностей всех классов составит 100%).

ЦЕНА РЕАЛИСТИЧНОГО ВЫВОДА

Как уже несколько раз упоминалось в этой книге, глубокое обучение может занять значительное время. Всякий раз, когда вы видите в коде функцию подбора, такую как `model.fit_generator`, вы, вероятно, просите систему осуществить обучение. Пример кода всегда будет стремиться предоставить вам реалистичный вывод — то, что ученый в реальном мире посчитал бы приемлемым.

К сожалению, реалистичный вывод может стоить вам слишком много времени. Не каждый имеет доступ к новейшей высокотехнологичной системе, и не все получат GPU на Colab. В некоторых примерах этой главы обучение занимает много времени. Например, при тестировании кода на Colab для его завершения потребовалось чуть более 16 часов, когда Colab не предоставил GPU. Тот же пример может сработать всего за час, если Colab предоставит GPU. (Больше о проблеме с графическим процессором сказано в главе 4.) Аналогично, для завершения обучения при использовании системы с только с 16-ядерным

графическим процессором Xeon потребовалось 4 часа и 23 минуты, а для процессора Intel Core i7 с 8 ядрами — чуть более 9 часов.

Одним из способов решения этой проблемы является изменение количества эпох, используемых для обучения вашей модели. Параметр `epochs = 100`, используемый для примера в этой главе, обеспечивает точность вывода чуть более 99%. Но если время является критическим фактором, вы можете использовать более низкую настройку `epochs` при запуске этого примера, чтобы уменьшить время ожидания до завершения примера.

Еще одно альтернативное решение этой проблемы — использование поддержки GPU на вашем локальном компьютере. Но чтобы использовать эту альтернативу, у вас должен быть адаптер дисплея с микросхемой правильного типа. Поскольку настройка сложна, и у вас может не быть подходящего графического процессора, в этой книге выбран путь только для CPU. Однако, используя главу 4 в качестве отправной точки, и добавив затем поддержку CUDA, вы, безусловно, можете установить правильную поддержку. Дополнительные сведения содержатся в статье на `https://towardsdatascience.com/tensorflow-gpu-installationmade-easy-use-conda-instead-of-pip-52e5249374bc`.

Что касается оптимизации, потери при минимизации являются категориальной *кросс-энтропией* (cross-entropy). Код измеряет успех как *точность* (accuracy) — процент правильных ответов, предоставленных алгоритмом. (Ответом является класс дорожных знаков с наибольшей прогнозируемой вероятностью.)

```
history = model.fit_generator(train_gen,
                  steps_per_epoch=len(X) // batch_size,
                  validation_data=val_gen,
                  validation_steps=len(Xt) // batch_size,
                  class_weight=class_weight,
                  epochs=100,
```

Использование `fit_generator` в модели, приводит к произвольному извлечению изображений, нормализации и приращению для фазы обучения. После извлечения всех обучающих изображений код видит *эпоху* (epoch) (итерацию обучения с использованием полного прохода набора данных) и вычисляет *оценку валидации* (validation score) на валидационных изображениях. По прохождении 100 эпох, обучение и модель завершены.

СОВЕТ

Если вы не используете никакого приращения, вы можете обучить вашу модель всего за 30 эпох и достичь такой производительности модели, которая почти сопоставима с навыком водителя распознавать различные дорожные знаки (точность составляет около 98,8%).

Чем агрессивнее используемое приращение, тем больше эпох требуется для достижения моделью своего максимального потенциала, хотя показатели точности также будут выше. На этом этапе код отображает график, изображающий состояние точности обучения и проверки во время обучения.

```
print("Best validation accuracy: {:0.3f}"
      .format(np.max(history.history['val_acc'])))

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.ylabel('accuracy'); plt.xlabel('epochs')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```

Код сообщит о наилучшей зарегистрированной точности проверки и строит кривые точности, достигнутой на учебных и тестовых данных в течение периодов роста обучения, как показано на рис. 12.4. Обратите внимание, что в конце обучения точности при обучении и проверке достоверности почти одинаковы, хотя проверка всегда лучше, чем обучение. Это легко объяснимо, поскольку проверочные изображения на самом деле "легче" распознать, чем обучающие, ведь к ним не применяется никакого приращения.

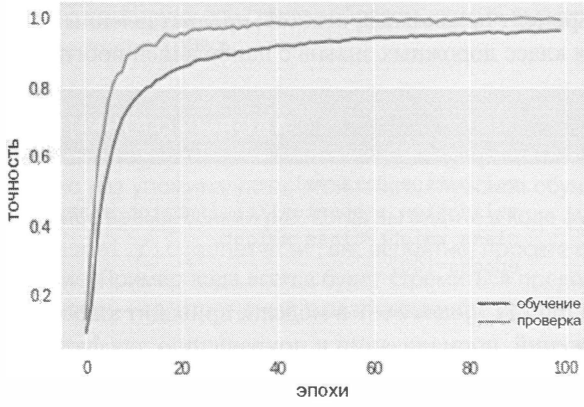


Рис. 12.4. Сравнение ошибок обучения и проверки

С учетом того, что код может инициализировать нейронную сеть различными способами, в конце оптимизации обучения вы можете увидеть разные лучшие результаты. Однако к концу 100 эпох, установленных в коде, точность проверки должна превысить 99% (в Colab точность достигла 99,5%).

ЗАПОМНИ!

Существует разница между производительностью, которую вы получаете для учебных данных (которая зачастую меньше) и для проверки, поскольку учебные данные являются более сложными и изменчивыми, чем проверочные данные, учитывая приращения изображений, устанавливаемые кодом.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Вы должны считать этот результат превосходным, основываясь на современных тестах, о которых можно прочитать в статье Мринала Халой (Mrinal Haloi) *Traffic Sign Classification Using Deep Inception Based Convolutional Networks* (`https://arxiv.org/pdf/1511.02992.pdf`). В статье делается намек на то, что может быть легко достигнуто с точки зрения распознавания изображений при ограниченных чистых данных и доступных инструментах, таких как TensorFlow и Keras.

Глава 13

Передовые CNN

В ЭТОЙ ГЛАВЕ...

- Важность обнаружения объекта
- Различие между обнаружением, локализацией и сегментацией
- Проверка обнаружения объектов RetinaNet из реализации GitHub
- Понятие слабых мест CNN

Решения глубокого обучения по распознаванию изображений стали настолько впечатляющими в своей производительности на человеческом уровне, что их можно встретить как в разрабатываемых, так и уже реализованных приложениях, таких как беспилотные автомобили и устройства видеонаблюдения. Устройства видеонаблюдения уже выполняют такие задачи, как автоматический мониторинг спутниковых изображений, распознавание лиц, а также локализация и подсчет людей. Тем не менее, вы не можете представить себе сложное приложение, когда ваша сеть помечает изображение только одним прогнозом. Даже простой детектор собак или кошек может оказаться бесполезным, если анализируемые фотографии содержат несколько собак и кошек. Реальный мир грязный и сложный. Вы не можете ожидать, за исключением весьма ограниченных и контролируемых случаев, изображений лабораторного стиля, состоящих из отдельных, отчетливо видимых объектов.

Необходимость обработки сложных изображений проложила путь к появлению вариантов сверточных нейронных сетей (Convolutional Neural Network — CNN). Подобные варианты предлагают решения таких задач как обнаружение и локализация нескольких объектов, которые все еще разрабатываются и совершенствуются. Обнаружение нескольких объектов подразумевает выявление

многих разных объектов одновременно. Локализация может указать вам, где они находятся на рисунке, а сегментация поможет найти их точные контуры. Эти новые возможности требуют сложных нейронных архитектур обработки изображений, более совершенных, чем простые CNN, обсуждаемые в предыдущих главах. Эта глава иллюстрирует основы работы таких решений, рассматривает ключевые подходы и архитектуры и, наконец, проверяет одну из самых эффективных реализаций обнаружения объектов.

Глава завершается обсуждением вероятных слабостей невероятных технологий. Используя соответствующие методы манипуляции изображениями, некто может преднамеренно обманывать CNN, чтобы она неправильно обнаруживала или игнорировала видимые объекты. Это удивительное открытие создает новый фронт исследований, показывающий, что при глубоком обучении необходимо учитывать безопасность, как при частном, так и при публичном использовании.

Различные задачи классификации

CNN являются составными блоками распознавания изображений, основанного на глубоком обучении, но они отвечают только основной потребности классификации: они могут определить по изображению, может ли его содержимое быть связано с определенным классом изображений, изученным на предыдущих примерах. Поэтому, когда вы обучаете глубокую нейронную сеть распознаванию собак и кошек, вы можете передать ей фотографию и получить вывод, сообщающий, содержит ли фотография собаку или кошку. Если последним слоем сети является слой softmax, сеть выводит вероятности того, что фотография содержит собаку или кошку (два класса, которые вы научили ее распознавать), и сумма выводов составит 100%. Когда последним слоем является активированный сигмоидный слой, вы получаете оценки, которые можете интерпретировать как вероятности принадлежность содержимого каждому классу независимо. Сумма баллов не обязательно составит 100%. В обоих случаях классификация может потерпеть неудачу, когда происходит следующее.

- Основной объект — это не то, чему вы научили распознавать сеть, например, фотографии енота. В этом случае сеть выдаст неверный ответ о собаке или кошке.
- Основной объект частично перекрыт. Например, на представленной сети фотографии ваша кошка играет в прятки, и сеть не может ее обнаружить.

» Фотография содержит много различных объектов для обнаружения, возможно, включая животных, отличных от кошек и собак. В этом случае вывод сети будет предлагать один класс, а не включать все объекты.

На рис. 13.1 показано изображение 47780 (`http://cocodataset.org/#explore?id=47780`), из набора данных MS Coco (выпущенного в рамках лицензии Creative Commons Attribution 4.0 с открытым исходным кодом). Серия из трех выводов показывает, как CNN обнаружила, локализовала и сегментировала объекты на изображении (котенок и собака, стоящие на поле травы). Обычная CNN не может воспроизвести примеры на рис. 13.1, поскольку его архитектура будет выводить только изображение определенного класса. Чтобы преодолеть это ограничение, исследователи расширяют базовые возможности CNN, чтобы сделать их способными к следующему.

Рис. 13.1. Пример обнаружения, локализации и сегментации изображения из набора данных Coco

» **Обнаружение**. Определение наличия объекта на изображении. Обнаружение отличается от классификации, поскольку включает в себя только часть изображения, подразумевая, что сеть может обнаруживать несколько объектов одного и того же или разных типов. Способность выявлять объекты на частях изображения является *определением экземпляра* (instance spotting).

» **Локализация**. Определение точного местоположения обнаруженного объекта на изображении. Возможны разные типы локализаций. В зависимости от степени детализации они различают часть изображения, содержащую обнаруженный объект.

» **Сегментация**. Классификация объектов на уровне пикселей. Сегментация доводит локализацию до крайности. Этот тип нейронной модели назначает каждый пиксель изображения классу или даже объекту. Например, сеть помечает все пиксели на изображении относительно собак и различает каждый из них, используя отдельную метку (*сегментация экземпляра* (instance segmentation)).

Локализация

Локализация, пожалуй, самое простое расширение, которое можно получить на обычной сети CNN. Для этого требуется обучение регрессионной модели наряду с моделью классификации глубокого обучения. *Регрессор* (regressor) — это модель, прогнозирующая числа. Местоположение объекта на изображении можно определить с помощью координат пикселей по углам, а значит, вы можете обучить нейронную сеть выводить ключевые метрики, позволяющие с помощью ограничительной рамки легко определить, где классифицированный объект находится на изображении. Обычно ограничивающий прямоугольник использует координаты x и y левого нижнего угла, а также ширину и высоту области, которая окружает объект.

Классификация нескольких объектов

CNN может обнаруживать (предсказывать класс) и локализовать (предоставляя координаты) только один объект на изображении. Если на изображении есть несколько объектов, вы все равно можете использовать CNN и с помощью двух старых решений для обработки изображений найти каждый присутствующий на нем объект.

- **Скользящее окно**. За один раз анализируется только часть изображения (*изучаемая область* (region of interest)). Когда изучаемая область достаточно мала, она, вероятно, содержит только один объект. Небольшая изучаемая область позволяет CNN правильно классифицировать объект. Это метод *скользящего окна* (sliding window), поскольку программное обеспечение использует окно для ограничения видимости конкретной области изображения (как это делает окно в доме) и медленно перемещает это окно по изображению. Техника эффективна, но может обнаруживать одно и то же изображение несколько раз, либо некоторые объекты могут остаться незамеченными, в зависимости от размера окна, используемого для анализа изображений.
- **Пирамида изображений**. Решает проблему использования окна фиксированного размера, поскольку оно создает изображения все меньшего разрешения. Поэтому вы можете применить небольшое скользящее окно. Так преобразуются объекты на изображении, и одно из сокращений может точно вписаться в используемое скользящее окно.

Эти методы требуют большого объема вычислений. Чтобы применить их, вы должны изменить размер изображения несколько раз, а затем разделить его на фрагменты. Затем вы обрабатываете каждый фрагмент используя

классификацию CNN. Количество операций при этих действиях настолько велико, что отображение вывода в реальном времени невозможно.

Скользящее окно и пирамида изображений вдохновили исследователей, занимающихся глубоким обучением, открыть пару концептуально похожих подходов, требующих меньших вычислительных затрат. Первый подход — *одноэтапное обнаружение* (one-stage detection). Он работает, разделяя изображения сеткой, и нейронная сеть делает прогноз для каждой ячейки сетки, прогнозируя класс объекта внутри. Прогноз довольно грубый, в зависимости от разрешения сетки (чем выше разрешение, тем сложнее и медленнее сеть глубокого обучения). Одноэтапное обнаружение очень быстрое, имеет почти такую же скорость классификации, как простая сеть CNN. Чтобы собрать ячейки, представляющие один и тот же объект, результаты должны быть обработаны, и это может привести к дальнейшим неточностям. Нейронными архитектурами, основанными на этом подходе, являются Single-Shot Detector (SSD), You Only Look Once (YOLO) и RetinaNet. Одноэтапные детекторы очень быстры, но не очень точны.

Второй подход — это *двухэтапное обнаружение* (two-stage detection). Этот подход использует вторую нейронную сеть для уточнения прогноза первой. Первый этап — это сеть предложений, выводящая свои прогнозы на сетку. На втором этапе выполняется точная настройка этих предложений и выводится окончательное обнаружение и локализация объектов. R-CNN, Fast R-CNN и Faster R-CNN — все имеют двухэтапные модели обнаружения, которые намного медленнее своих одноэтапных эквивалентов, но более точны в своих прогнозах.

Аннотирование нескольких объектов на изображениях

Чтобы научить модели глубокого обучения обнаружению нескольких объектов, вам нужно предоставить больше информации, чем при простой классификации. Для каждого объекта следует предоставить как класс, так и координаты на изображении, используя процесс аннотации, отличный от маркировки, используемой при простой классификации изображения.

Маркировка изображений в наборе данных является сложной задачей даже при простой классификации. Учитывая эту картину, сеть должна обеспечить правильную классификацию для этапов обучения и тестирования. При маркировке сеть выбирает правильную метку для каждого изображения, и не все будут воспринимать полученное изображение одинаково. Люди, создавшие набор данных ImageNet, использовали классификацию, предоставленную несколькими пользователями краудсорсинговой инфраструктуры Amazon Mechanical Turk (создатели ImageNet использовали сервис Amazon настолько

интенсивно, что в 2012 году они стали самым важным академическим клиентом Amazon).

Аналогичным образом, при аннотировании изображения с помощью ограничительных рамок вы полагаетесь на работу нескольких людей. Аннотация требует, чтобы вы не только пометили каждый объект на картинке, но и определили наилучшую рамку, в которую будет заключен каждый объект. Эти две задачи делают аннотацию куда сложней маркировки, и более склонной к получению ошибочных результатов. Для правильного выполнения аннотации требуется больше людей, способных прийти к единому мнению относительно точности аннотации.


СОВЕТ

В аннотации изображений могут помочь некоторые программы с открытым исходным кодом (а также для сегментации изображений, обсуждаемой в следующем разделе). Два инструмента особенно эффективны.

- LabelImg, созданный TzuTa Lin (`https://github.com/tzutalin/labelImg`) с учебником по адресу `https://www.youtube.com/watch?v=p0nR2YsCY_U`.
- LabelMe (`https://github.com/wkentaro/labelme`) — мощный инструмент для сегментации изображений, предоставляющий сетевой сервис.
- FastAnnotationTool основан на библиотеке компьютерного зрения OpenCV (`https://github.com/christopher5106/FastAnnotationTool`). Пакет проработан недостаточно, но вполне работоспособен.

Сегментирование изображений

Семантическая сегментация прогнозирует класс для каждого пикселя в изображении, что отличается от маркировки или аннотации. Некоторые называют такую задачу *плотным прогнозированием* (dense prediction), поскольку она дает прогноз для каждого пикселя в изображении.

В задаче между разными объектами в прогнозе не делается особого различия. Например, семантическая сегментация может показать все пиксели класса `cat`, но она не предоставит никакой информации о том, что кошка (или кошки) делают на рисунке. Вы можете легко получить все объекты в сегментированном изображении с помощью *постобработки* (post-processing), поскольку после выполнения прогнозирования вы можете получить пиксельные области объекта и различать их разные экземпляры, если в рамках одного и того же класса прогнозирования существует несколько отдельных областей.

Различные архитектуры глубокого обучения способны обеспечить сегментацию изображения. Одними из самых эффективных архитектур являются *полностью сверточные сети* (Fully Convolutional Network — FCN) и U-NET. FCN для первой части (*кодер*), аналогична сети CNN. После начальной серии сверточных слоев FCN завершается другой серией CNN, работающих в порядке обратном для кодера (*декодер*). Декодер сконструирован так, чтобы воссоздать исходный размер входного изображения и вывести в виде пикселей классификацию каждого пикселя в изображении. Таким образом, FCN достигает семантической сегментации изображения. FCN слишком дорог в вычислительном отношении для большинства приложений реального времени. Кроме того, им требуются большие учебные наборы, чтобы хорошо освоить свои задачи; в противном случае результаты их сегментации зачастую бывают грубыми.

ЗАПОМНИ!

Распространенным явлением является применение кодирующей части FCN, предварительно обученной на наборе ImageNet, что ускоряет обучение и повышает его эффективность.

U-NET — это развитие FCN, разработанное Олафом Роннебергером (Olaf Ronneberger), Филиппом Фишером (Philipp Fischer) и Томасом Броксом (Thomas Brox) в 2015 году для медицинских целей (см. `https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/`). U-NET имеет преимущества по сравнению с FCN. Части кодирования (или *сокращения* (contraction)) и декодирования (или *расширения* (expansion)) являются совершенно симметричными. Кроме того, в сетях U-NET используются быстрые соединения между слоями кодера и декодера. Эти сокращения позволяют легко передавать детали объектов из части кодирования в части декодирования U-NET, и в результате сегментация получается точной и детальной.

СОВЕТ

Создание модели сегментации с нуля может быть сложной задачей, но вам и не нужно этого делать. Вы можете использовать некоторые предварительно обученные архитектуры U-NET и сразу же начать использовать нейронную сеть этого типа, используя сегментацию *зоопарк моделей* (model zoo) (термин, применяемый для описания коллекции предварительно обученных моделей, предлагаемых многими инфраструктурами), предоставляемую моделями сегментации, из пакета, предлагаемого Павлом Якубовским (Pavel Yakubovskiy). Вы можете найти инструкции по установке, исходный код и множество примеров использования по адресу `https://github.com/qubvel/segmentation_models`. Команды из пакета легко интегрируются с Keras.

Восприятие объектов в их окружении

Интеграция зрительных возможностей в систему восприятия беспилотного автомобиля может повысить уверенность его вождения и безопасность. Алгоритм сегментации может помочь автомобилю отличать полосы от тротуаров, а также от других препятствий, которые автомобиль должен замечать. Автомобиль может даже иметь полную сквозную систему, такую как NVIDIA, контролирующую рулевое управление, газ и тормоза реактивным образом на основе визуальных вводов. (NVIDIA является крупным игроком в области глубокого обучения, и книга упоминает об этом также в главах 4, 9 и 11. Вы можете узнать больше об усилиях NVIDIA по беспилотным автомобилям на `https://www.nvidia.com/en-us/self-driving-cars/`.) Визуальная система способна различать некоторые объекты на дороге, имеющие отношение к вождению, такие как дорожные знаки и светофоры. Она может также визуально отслеживать траектории других автомобилей. Во всех случаях, сеть глубокого обучения способна обеспечить решение.

В разделе "Различные задачи классификации" обсуждается улучшение распознавания при классификации одиночных объектов в CNN. В этом разделе рассматриваются архитектуры и современные модели двух основных подходов: одноэтапного и двухэтапного обнаружения. В данном разделе рассматривается работа одноэтапной системы обнаружения, а также принципы работы автономного транспортного средства.

Программирование такой системы распознавания с нуля было бы сложной задачей, требующей отдельной книги. К счастью, вы можете использовать проекты с открытым исходным кодом на GitHub, такие как Keras-RetinaNet (`https://github.com/fizyr/keras-retinanet`). Keras-RetinaNet — это реализация Keras модели RetinaNet, предложенной Линь Цзун-ух (Tsung-Yi Lin), Прайей Гойал (Priya Goyal), Россом Гиршик (Ross Girshick), Каймином Хе (Kaiming He) и Пиотром Доллар (Piotr Dollár) в статье *Focal Loss for Dense Object Detection*, опубликованной в августе 2017 года на `https://arxiv.org/abs/1708.02002`.

СОВЕТ

Исаак Ньютон сказал: "Если я видел дальше других, только потому, что стоял на плечах гигантов". Точно так же вы можете достичь большего в области глубокого обучения, когда используете существующие нейронные архитектуры и предварительно обученные сети. Например, вы можете найти много моделей на GitHub (`www.github.com`), например, зоопарк моделей TensorFlow (`https://github.com/tensorflow/models`).

Как работает RetinaNet

RetinaNet — это сложная и интересная модель обнаружения объектов, стремящаяся быть такой же быстрой, как и другие одноэтапные модели обнаружения, и в то же время обеспечивающей точность прогнозирования ограничивающей рамки двухэтапных систем обнаружения, таких как Faster R-CNN (модель с наилучшими характеристиками). Благодаря своей архитектуре, RetinaNet достигает своих целей, используя методы, аналогичные архитектуре U-NET, упоминавшейся при обсуждении семантической сегментации. RetinaNet является частью группы моделей Feature Pyramid Networks (FPN).

Своим успехом модель RetinaNet обязана своим авторам, Линь Цзун-ух, Прайе Гойал, Россу Гиршик, Каймину Хе и Пиотру Доллару, отметившим, что одноэтапные модели обнаружения не всегда обнаруживают объекты именно потому, что на них подавляюще воздействует присутствие отвлекающих элементов на изображениях, используемых для обучения. Их статья *Focal Loss for Dense Object Detection* (`https://arxiv.org/pdf/1708.02002.pdf`) содержит подробную информацию о методах, использованных RetinaNet. Проблема в том, что на изображениях представлено мало интересных объектов для обнаружения. Фактически, одноэтапные сети обнаружения обучаются выявлять класс каждой ячейки в изображении, разделенном на фиксированную сетку, где большинство ячеек не содержит интересующих объектов.

ЗАПОМНИ!

Целью классификации при семантической сегментации являются отдельные пиксели. При одноэтапном обнаружении цели представляют собой наборы смежных пикселей, выполняющие задачу, аналогичную семантической сегментации, но с другим уровнем детализации.

Вот что происходит, когда вы имеете такое преобладание пустых примеров в изображениях и используете обучающий подход, рассматривающий в качестве примеров все доступные ячейки. Сеть с большей вероятностью может заявить, что в ячейке обработанного изображения ничего нет, чем обеспечить правильную классификацию. Нейронные сети всегда выбирают наиболее эффективный путь обучения, и в этом случае заявить о фоне легче, чем сделать что-либо еще. В этой ситуации, при *несбалансированном обучении* (unbalanced learning), многие объекты не обнаруживаются нейронной сетью с использованием подхода обнаружения одиночного объекта.

В машинном обучении, когда вы хотите предсказать два численно различных класса (один является классом большинства, а другой — классом меньшинства), возникает проблема несбалансированной классификации. Большинство алгоритмов не работает должным образом, когда классы не

сбалансированы, поскольку они предпочитают класс большинства. Для этой проблемы доступно несколько решений.

- **Выборка**. Выбор одних примеров и отбрасывание других.
- **Уменьшение частоты дискретизации**. Уменьшение влияния класса большинства в ходе отбора для использования только его части, что уравновешивает прогнозы большинства и меньшинства. Во многих случаях это самый простой подход.
- **Увеличение частоты дискретизации**. Увеличение влияния класса меньшинства в ходе многократной репликации его примеров, пока для класса меньшинства не будет иметься такого же количества примеров, что и у класса большинства.

Создатели RetinaNet пошли другим путем, как они отмечают в своей статье *Focal Loss for Dense Object Detection*, упомянутой ранее в этом разделе. Они игнорируют примеры классов большинства, которые легче классифицировать, и концентрируются на ячейках, которые трудно классифицировать. В результате функция стоимости сети более фокусируется на адаптации своих весов для распознавания фоновых объектов. Это решает *функция фокальных потерь* (focal loss) и представляет собой хитрый способ заставить одноэтапное обнаружение работать корректно, но быстро, что является требованием приложений в реальном времени, например, при обнаружении препятствий или объектов беспилотным автомобилем, или обработки больших количеств изображений при видеонаблюдении.

Использование кода Keras-RetinaNet

Keras-RetinaNet, вышедший в рамках Apache License 2.0 с открытым исходным кодом, является проектом, спонсируемым голландской робототехнической компанией Fitz, и реализованной многими участниками (главные участники — Ханс Гайзер (Hans Gaiser), Маартен де-Фриз (Maarten de Vries)). Это реализация нейронной сети RetinaNet, написанная на языке Python с использованием Keras (`https://github.com/fizyr/keras-retinanet/`). Вы найдете, что Keras-RetinaNet успешно используется во многих проектах, наиболее заметным и впечатляющим из которых является победившая модель конкурса NATO Innovation Challenge, задачей которого было обнаружение автомобилей на аэрофотоснимках. (Вы можете прочитать рассказ команды победителей в сообщении блога `https://medium.com/data-from-the-trenches/object-detection-with-deep-learning-on-aerial-imagery-2465078db8a9`.)

Сетевой код обнаружения объектов слишком сложен, чтобы объяснить его на нескольких страницах, кроме того, вы можете использовать существующую сеть для настройки решений глубокого обучения, поэтому в данном разделе

показано, как загрузить и использовать Keras-RetinaNet на вашем компьютере. Прежде чем опробовать этот процесс, убедитесь, что вы настроили свой компьютер так, как описано в главе 4, и рассмотрите компромиссы, связанные с использованием различных вариантов запуска, описанных в главе 12.

В качестве первого шага вы загружаете необходимые пакеты и начинаете скачивать архивированную версию хранилища GitHub. В этом примере используется Keras-RetinaNet версии 0.5.0, которая была самой последней доступной на момент написания статьи.

```
import os
import zipfile
import urllib.request
import warnings
warnings.filterwarnings("ignore")
url = "https://github.com/fizyr/\
keras-retinanet/archive/0.5.0.zip"
urllib.request.urlretrieve(url, './'+url.split('/')[-1])
```

После загрузки архива кода пример кода автоматически извлекает его с помощью следующих команд.

```
zip_ref = zipfile.ZipFile('./0.5.0.zip', 'r')
for name in zip_ref.namelist():
    zip_ref.extract(name, './')
zip_ref.close()
```

При выполнении создается новый каталог по имени `keras-retinanet-0.5.0`, содержащий код для настройки нейронной сети. Затем код выполняет компиляцию и установку пакета с помощью команды `pip`.

```
os.chdir('./keras-retinanet-0.5.0')
!python setup.py build_ext --inplace
!pip install .
```

Все эти команды только получили код, создающий архитектуру сети. Теперь в примере требуются предварительно обученные веса, и они основаны на весах, обученных на наборе данных MS Coco с использованием ResNet50 CNN, нейронной сети, которую компания Microsoft использовала для победы в конкурсе ImageNet 2015 года.

```
os.chdir('../')
url = "https://github.com/fizyr/\
        keras-retinanet/releases/download/0.5.0/\
        resnet50_coco_best_v2.1.0.h5"
urllib.request.urlretrieve(url, './'+url.split('/')[-1])
```

Загрузка всех весов занимает некоторое время, поэтому сейчас самое время пополнить свою чашку кофе. После завершения этого этапа пример

готов импортировать все необходимые команды и инициализировать модель RetinaNet с использованием предварительно обученных весов, полученных из Интернета. Этот этап устанавливает также словарь для преобразования числовых сетевых результатов в понятные классы. Выбор классов полезен для детектора на беспилотном автомобиле или для любого другого решения, которое должно понимать изображения, полученные с дороги или перекрестка.

```
import os
import numpy as np
from collections import defaultdict
import keras
from keras_retinanet import models
from keras_retinanet.utils.image import (read_image_bgr,
        preprocess_image, resize_image)
from keras_retinanet.utils.visualization import (draw_box,
        draw_caption)
from keras_retinanet.utils.colors import label_color
import matplotlib.pyplot as plt
%matplotlib inline

model_path = os.path.join('.',
          'resnet50_coco_best_v2.1.0.h5')

model = models.load_model(model_path,
          backbone_name='resnet50')

labels_to_names = defaultdict(lambda: 'object',
          {0: 'person', 1: 'bicycle', 2: 'car',
           3: 'motorcycle', 4: 'airplane', 5: 'bus',
           6: 'train', 7: 'truck', 8: 'boat',
           9: 'traffic light', 10: 'fire hydrant',
          11: 'stop sign', 12: 'parking meter',
          25: 'umbrella'})
```

Чтобы сделать пример полезным, вам понадобится образец изображения для тестирования модели RetinaNet. Пример основан на бесплатном изображении из Викимедиа, содержащем перекресток с людьми, ожидающими перехода, несколькими автомобилями, светофорами и дорожными знаками.

```
url = "https://upload.wikimedia.org/wikipedia/commons/\
thumb/f/f8/Woman_with_blue_parasol_at_intersection.png/\
640px-Woman_with_blue_parasol_at_intersection.png"
urllib.request.urlretrieve(url, './'+url.split('/')[-1])
```

После завершения загрузки изображения, пришло время проверить нейронную сеть. Следующий фрагмент кода читает изображение с диска, а затем меняет каналы синего и красного цветов изображения (поскольку изображение загружается в формате BGR, а RetinaNet работает с изображениями RGB).

Наконец, код обрабатывает и изменяет размер изображения. Все эти шаги выполняются с использованием предоставленных функций и не требуют специальных настроек.

Модель выведет обнаруженные ограничивающие рамки, уровень достоверности (показатель вероятности того, что сеть действительно что-то обнаружила) и кодовую метку, преобразуемую в текст с использованием определенного ранее словаря меток. Цикл фильтрует поля, обозначенные на примере изображения. Код использует порог достоверности 0,5, что означает, что в этом примере будет сохраняться любое обнаружение, достоверность которого составляет не менее 50%. Использование более низкого доверительного порога приводит к большему количеству обнаружений, особенно тех объектов, которые на изображении кажутся небольшими, но также увеличивает количество ошибочных обнаружений (например, некоторые тени могут начать обнаруживаться как объекты).

СОВЕТ

В зависимости от ваших целей использования RetinaNet, вы можете решить, что использование более низкого доверительного порога — это нормально. Вы заметите, что при снижении достоверности доля получаемых точных догадок (с почти 100% достоверностью) будет уменьшаться. Такая пропорция называется *точностью* обнаружения, и, выбирая допустимую точность, вы можете устанавливать наилучшую достоверность для своих целей.

```
image = read_image_bgr('640px-Woman_with_blue_parasol_at_
    intersection.png')
draw = image.copy()
draw[:,:,0], draw[:,:,2] = image[:,:,2], image[:,:,0]

image = preprocess_image(image)
image, scale = resize_image(image)

boxes, scores, labels = model.predict_on_batch(np.expand_
    dims(image, axis=0))
boxes /= scale

for box, score, label in zip(boxes[0], scores[0], labels[0]):
    if score > 0.5:
      color = label_color(label)
      b = box.astype(int)
      draw_box(draw, b, color=color)
      caption = "{} {:.3f}".format(labels_to_names[label],
    score)
      draw_caption(draw, b, caption.upper())
```

```
plt.figure(figsize=(12, 6))
plt.axis('off')
plt.imshow(draw)
plt.show()
```

При первом запуске кода может потребоваться некоторое время, но после некоторых вычислений вы должны получить выходные данные, показанные на рис. 13.2.

Рис. 13.2. Результаты обнаружения объектов в Keras-RetinaNet

Сеть может успешно обнаруживать различные объекты, некоторые очень маленькие (например, человек на заднем плане), некоторые отображаются частично (например, перед автомобиля справа). Каждый обнаруженный объект очерчен своей ограничительной рамкой, что создает широкий спектр возможных применений.

Например, вы можете использовать сеть для определения, использует ли человек зонтик или какой-то объект. При обработке результатов вы можете связать тот факт, что два ограничивающих прямоугольника накладываются друг на друга, причем один из них является зонтиком, а другой — человеком, и что первое поле расположено поверх второго, чтобы сделать вывод о том, что человек держит зонтик. Это *визуальное обнаружение отношений* (visual relationship detection). Таким же образом, с помощью общей настройки обнаруженных объектов и их относительного положения, вы можете обучить вторую сеть глубокого обучения так, чтобы она выводила общее описание сцены.

Предотвращение преднамеренных атак на приложения глубокого обучения

Поскольку глубокое обучение интенсивно используются в беспилотных автомобилях, например, для обнаружения и интерпретации дорожных знаков и цветов светофоров; обнаружения дороги и ее полос; обнаружения пешеходных переходов и других транспортных средств; управления автомобилем с помощью руля и тормозов при сквозном подходе к автоматическому вождению; и так далее, могут возникнуть вопросы о безопасности беспилотного автомобиля. Вождение — не единственная область деятельность, переживающая революцию из-за приложений глубокого обучения. Недавно общественности стали доступны приложения распознавание лиц для защищенного доступа. (Вы можете прочитать об их использовании в банкоматах в Китае по адресу `https://www.telegraph.co.uk/news/worldnews/asia/china/11643314/China-unveils-worlds-first-facial-recognition-ATM.html`). Другое приложение глубокого обучения — распознавание речи, используется для *систем с голосовым управлением* (Voice Controllable System — VCS), разрабатываемых множеством компаний, таких как Apple, Amazon, Microsoft и Google для самых разных приложений, включая Siri, Alexa, и Google Home.

Некоторые из этих приложений глубокого обучения способны нанести экономический ущерб или даже угрожать жизни, если они не смогут дать правильный ответ. Поэтому нет ничего удивительного в том, что хакеры могут преднамеренно обманывать глубокие нейронные сети и приводить их к ошибочным прогнозам, используя специальные методы и состязательные примеры.

Состязательный пример (adversarial example) — это созданный вручную фрагмент данных, рассматриваемый нейронной сетью в качестве входных данных при обучении или тестировании. Хакер изменяет данные так, чтобы алгоритм не справился со своей задачей. Каждый состязательный пример имеет модификации, преднамеренно сделанные действительно незначительными и незаметными для человека. Модификации, хоть и неэффективны для людей, для нейронной сети довольно эффективны в снижении ее производительности и полезности. Зачастую такие злонамеренные примеры нацелены на то, чтобы заставить нейронную сеть предсказуемо ошибиться, чтобы создать некое незаконное преимущество для хакера. Вот лишь несколько примеров злонамеренного использования состязательных примеров (список далеко не исчерпывающий).

- Ввести в заблуждение беспилотный автомобиль.
- Получение денег в результате страхового мошенничества при предоставлении поддельных фотографий, которые автоматические системы считают истинными.
- Обман системы распознавания лиц, чтобы получить доступ к деньгам на банковском счете или личным данным на мобильном устройстве.


ЗАПОМНИ!

Генеративно-состязательные сети (generative adversarial network — GAN), обсуждаемые в главе 16, и *состязательное обучение* (adversarial training), имеют совершенно другое назначение, чем *состязательные примеры*. Эти методы — способ обучать глубокую нейронную сеть генерации новых примеров любого рода.

Обманные пиксели

Впервые обнародованные в статье *Intriguing Properties of Neural Networks* (см. `https://arxiv.org/pdf/1312.6199.pdf`), состязательные примеры привлекли большое внимание в последние годы, и в этой области было проведено много успешных (и шокирующих) открытий, исследователи разработали более быстрые и эффективные способы создания таких примеров, чем указывалось в оригинальной статье.

ОКАЗЫВАЕТСЯ, ЧТО МАФФИН — ЭТО НЕ ЧИХУАХУА

Иногда классификация образов глубоким обучением не дает правильного ответа, поскольку целевое изображение неоднозначно по своей сути или представляют загадочные наблюдения. Например, некоторые изображения настолько вводят в заблуждение, что способны на некоторое время мистифицировать даже человека, например, "Чихуахуа или маффин" (см. `https://imgur.com/QWQiBYU`) и "Лабрадудль или жареная курица" (см. `https://imgur.com/5EnWOJU`). Нейронная сеть может неправильно понимать запутанные изображения, если ее архитектура не соответствует задаче, и ее обучение не было исчерпывающим с точки зрения видимых примеров. Обозреватель технологий искусственного интеллекта Мария Яо (Mariya Yao) сравнила различные интерфейсы API компьютерного зрения (по адресу `https://www.freecodecamp.org/news/chihuahua-or-muffin-my-search-for-the-best-computer-vision-api-cbda4d6b425d/`) и обнаружила, что даже полностью оперившиеся продукты компьютерного зрения могут быть обмануты неоднозначными образами.

Недавно другие исследования бросили вызов глубоким нейронным сетям, предложив неожиданные точки зрения на известные объекты. В статье *Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects* на сайте `https://arxiv.org/pdf/1811.11553.pdf`. Исследователи обнаружили, что простая двусмысленность может обмануть классификаторы изображений и детекторы объектов, обученные на крупномасштабных наборах данных изображений. Зачастую объекты изучаются нейронными сетями по изображениям, сделанным в *канонических позах* (т.е. в наиболее популярных обычных ситуациях). Столкнувшись с объектом в необычной позе или вне его обычной среды, некоторые нейронные сети не могут классифицировать полученный объект. Например, вы ожидаете, что школьный автобус будет ездить по дороге, но если вы повернете его и поднимите в воздух, а затем положите его поперек дороги, нейронная сеть может легко увидеть в нем мусоровоз, боксерскую грушу или даже снегоуборщик. Вы можете утверждать, что неправильная классификация происходит из-за предвзятого отношения к обучению (обучение нейронной сети с использованием только изображений в канонических позах). Тем не менее, это означает, что в настоящее время вы не должны полагаться на такую технологию при любых обстоятельствах, особенно, как указывали авторы статьи, в приложениях для беспилотного автомобиля, поскольку объекты на дороге могут внезапно оказаться в новых позах или обстоятельствах.

ЗАПОМНИ!

Состязательные примеры все еще ограничиваются исследовательскими лабораториями глубокого обучения. По этой причине вы найдете много научных работ, цитируемых в этих параграфах при ссылке на различные виды примеров. Тем не менее, вы никогда не должны сбрасывать со счетов состязательные примеры как некое академическое отклонение, поскольку их потенциал для ущерба высок.

В основе всех этих подходов лежит идея о том, что смешивание изображения с некой числовой информацией, *вариацией* (perturbation), может привести к тому, что нейронная сеть будет вести себя не так, как ожидалось, хотя и контролируемым образом. Когда вы создаете состязательный пример, вы добавляете некий специально разработанный шум (выглядящий как нечто, кажущееся случайным числом) к существующему изображению, и этого достаточно, чтобы обмануть большинство CNN (поскольку один и тот же прием зачастую работает с разными архитектурами, обученными на одних и тех же данных). Как правило, такие вариации можно обнаружить, имея доступ к модели (ее архитектуре и весам). Затем вы используете ее алгоритм обратного распространения, чтобы систематически находить лучший набор числовой информации,

добавляемой к изображению, чтобы вы могли преобразовать один обнаруживаемый класс в другой.

СОВЕТ

Вы можете создать эффект вариации, изменив один пиксель в изображении. Исследователи получили прекрасно работающие состязательные примеры, используя этот подход, открытый учеными из Университета Кюсю и описанный в их статье *One Pixel Attack for Fooling Deep Neural Networks* (`https://arxiv.org/pdf/1710.08864.pdf`).

Взлом с помощью наклеек и других артефактов

Большинство состязательных примеров являются результатами лабораторных экспериментов по улучшению компьютерного зрения, и эти примеры могут продемонстрировать все свои возможности, поскольку они создаются в результате непосредственного изменения входных данных и проверочных изображений на этапе обучения. Однако многие приложения, основанные на глубоком обучении, работают в реальном мире, и использование лабораторных методов не предотвращает злонамеренных атак. Чтобы быть эффективными, такие атаки не нуждаются в доступе к базовой нейронной модели. Некоторые примеры могут иметь форму наклейки или неслышимого звука, с которым нейронная сеть не знает, как справиться.

Документ *Adversarial Examples in the Physical World* (доступный по адресу `https://arxiv.org/pdf/1607.02533.pdf`) демонстрирует, что различные атаки возможны также и в не лабораторных условиях. Все, что вам нужно, это распечатать состязательные примеры и показать их камере, питающей нейронную сеть (например, с помощью камеры на мобильном телефоне). Этот подход демонстрирует, что эффективность состязательного примера не строго связана с числовым вводом, подаваемым в нейронную сеть. Это ансамбль форм, цветов и контрастности, присутствующих в изображении, и вам не нужен прямой доступ к нейронной модели, чтобы узнать, какой ансамбль работает лучше всего. Вы можете увидеть, что сеть способна ошибочно принять изображение стиральной машины за сейф или громкоговоритель непосредственно из этого видео, созданного авторами, которые обманули демонстрационную камеру TensorFlow, приложения для мобильных устройств, осуществляющую классификацию изображений на лету: `https://www.youtube.com/watch?v=zQ_uMenoBCk`.

Другие исследователи из Университета Карнеги Меллона нашли способ обмануть систему распознавания лиц, заставив ее поверить в то, что человек является знаменитостью, создавая оправы для очков, способные повлиять на то, как глубокая нейронная сеть распознает объекты. По мере широкого

распространения автоматизированных систем безопасности, возможность обмануть их с помощью простых трюков, таких как очки, может превратиться в серьезную угрозу безопасности. Документ *Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition* (`https://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf`) описывает, как аксессуары могут позволить уклоняться от распознавания личности.

Наконец, еще одно вызывающее тревогу реальное использование состязательного примера упоминается в статье *Robust Physical-World Attacks on Deep Learning Visual Classification* (`https://arxiv.org/pdf/1707.08945.pdf`). Простые черно-белые наклейки, размещенные на знаке "Движение без остановки запрещено", могут повлиять на то, как беспилотный автомобиль понимает сигнал, неправильно понимая его за другой дорожный знак. Когда вы используете более красочные (но также и более заметные) наклейки, такие как описанные в статье *Adversarial Patch* (`https://arxiv.org/pdf/1712.09665.pdf`), вы можете направлять результаты нейронной сети в определенном направлении, чтобы она игнорировала все, кроме наклейки и вводящей в заблуждение информации. Как объясняется в статье, нейронная сеть может показать, что банан будет чем-то еще, если просто поместить на изображении рядом соответствующую лживую наклейку.

Теперь вы можете задаться вопросом, возможна ли какая-либо защита от состязательных примеров, или же они рано или поздно подорвут доверие общественности к приложениям глубокого обучения, особенно в области беспилотного вождения автомобиля. Интенсивно изучая, как ввести в заблуждение нейронную сеть, исследователи также находят и способы ее защиты от любого неправильного использования. Во-первых, нейронные сети могут аппроксимировать любую функцию. Если нейронные сети достаточно сложны, они также могут сами определять, как исключать состязательные примеры, когда они получены. Во-вторых, новые методы, такие как ограничение значений в нейронной сети или уменьшение размера нейронной сети после ее обучения (метод *дистилляции* (distillation), использовавшийся ранее для обеспечения работоспособности сети на устройствах с небольшим объемом памяти), были успешно проверены на различных типах состязательных атак.

Глава 14

Обработка текстов на естественном языке

В ЭТОЙ ГЛАВЕ...

- Обработка текстов на естественном языке
- Как при глубоком обучении превратить слова в числа
- Соотнесение слов и их значений в векторном представлении слов
- Создание системы анализа настроений с использованием RNN и глубокого обучения

Компьютер не может понять язык; он обрабатывает только конкретные приложения. Кроме того, компьютер не может обрабатывать язык, если он не очень формален и точен, как например язык программирования. Жесткие синтаксические и грамматические правила позволяют компьютеру превратить программу, написанную разработчиком на таком компьютерном языке как Python, в машинный код, определяющий, какие задачи будет выполнять компьютер. Человеческий язык совсем не похож на язык компьютера. Человеческому языку зачастую не хватает четкой структуры, он полон ошибок, противоречий и двусмысленностей, но все же он хорошо подходит для людей, и с некоторыми усилиями со стороны слушателя отлично служит человеческому обществу и прогрессу в области знаний.

Таким образом, программирование компьютера для обработки человеческого языка является сложной задачей, решение которой стало возможно только недавно, с появлением *обработки текстов на естественном языке* (Natural Language Processing — NLP), *рекуррентных нейронных сетей* (Recurrent Neural Network — RNN) глубокого обучающихся и векторного представления слов.

Векторное представление слов (word embedding) — это название технологии моделирования языка и изучения признаков в NLP, соотносящей словарь с векторами действительных чисел, с использованием таких продуктов, как Word2vec, GloVe и fastText. Вы также видите, что она используется в предварительно обученных сетях с открытым исходным кодом, таких как BERT от Google. В этой главе вы начнете с основ, необходимых для понимания NLP, и узнаете, как она может помочь вам в построении лучших моделей глубокого обучения для решения языковых задач. Затем в главе рассказывается о векторном представлении слов, о том, как предварительно обученные сети революционизируют глубокое обучение, и как компьютеры могут общаться через чат-боты. Глава завершается примером модели глубокого обучения применительно к анализу настроений, выявляющей мнения в тексте.

ЗАПОМНИ!

Вам не нужно вводить исходный код примеров из этой главы вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файлах `DL4D_14_Processing_Language.ipynb` и `DL4D_14_Movie_Sentiment.ipynb`.

Обработка языка

В упрощенном виде, язык можно считать последовательностью слов, состоящих из букв (а также знаков препинания, символов, смайликов и т.д.). Глубокое обучение лучше всего обрабатывает язык с использованием слоев RNN, таких как LSTM или GRU (см. главу 11). Однако знание того, как использовать RNN, ничего не говорит вам о том, как использовать последовательности в качестве ввода; вам нужно определить вид последовательностей. На самом деле, сети глубокого обучения принимают в качестве ввода только числовые значения. Компьютеры кодируют буквенные последовательности, понятные человеку, в числа в соответствии с протоколом, таким как Unicode Transformation Format-8 bit (UTF-8). UTF-8 — это наиболее широко используемая кодировка. (Справочник по кодировкам доступен на `https://www.alexreisner.com/code/character-encoding`.)

ЗАПОМНИ!

Глубокое обучение способно также обрабатывать текстовые данные с использованием сверточных нейронных сетей (CNN) вместо RNN, в ходе представления последовательностей в виде матриц (аналогично обработке изображений). Keras поддерживает слои CNN, такие как `Conv1D` (`https://keras.io/layers/convolutional/`), способные воздействовать на упорядоченные признаки во времени, то

есть последовательности слов или других сигналов. За выводом одномерной свертки обычно следует слой `MaxPooling1D`, суммирующий выходные данные. CNN, применяемые к последовательностям, находят предел в своей нечувствительности к глобальному порядку последовательности. (Как правило, они обнаруживают локальные шаблоны.) По этой причине их лучше всего использовать при обработке последовательности в сочетании с RNN, а не в качестве их замены.

Обработка текстов на естественном языке (Natural Language Processing — NLP) состоит из набора процедур, улучшающих обработку слов и фраз для статистического анализа, алгоритмов машинного обучения и глубокого обучения. NLP обязана своими корнями компьютерной лингвистике. Она использует основанные на правилах искусственного интеллекта системы, такие как экспертные системы, принимающие решения на основе компьютерного перевода человеческих знаний, опыта и образа мышления. NLP превращала неструктурированную текстовую информацию в более структурированные данные, чтобы экспертные системы могли легко манипулировать и оценивать ее. В наши дни взяло верх глубокое обучение, и экспертные системы ограничены конкретными приложениями, в которых первостепенное значение имеют интерпретация и контроль процессов принятия решений (например, в медицинских приложениях и системах принятия решений о поведении при вождении на некоторых беспилотных автомобилях). Тем не менее, конвейер NLP все еще весьма актуален для многих приложений глубокого обучения.

Определение понимания как лексического анализа

Первым шагом в конвейере NLP является получение необработанного текста. Обычно вы храните его в памяти или обращаетесь к нему на диске. Когда данные слишком велики для размещения в памяти, вы сохраняете на диске указатель на них (например, имя каталога и имя файла). В следующем примере вы используете три документа (представленные строковыми переменными), хранящиеся в списке (контейнер документов — это *корпус* компьютерной лингвистики).

```
import numpy as np

texts = ["My dog gets along with cats",
         "That cat is vicious",
         "My dog is happy when it is lunch"]
```

После получения текста вы обрабатываете его. При обработке каждой фразы, вы извлекаете из текста соответствующие признаки (создаете матрицу

набора слов (bag of words)) и передаете все в обучаемую модель, такую как алгоритм глубокого обучения. Для манипулирования текстом во время его обработки вы можете использовать различные преобразования (единственным обязательным преобразованием является лексический анализ).

- **Нормализация**. Убирает заглавные буквы.
- **Очистка**. Удаляет нетекстовые элементы, такие как знаки препинания и цифры.
- **Лексический анализ**. Разделение предложения на отдельные слова.
- **Удаление стоп-слов**. Удаление распространенных неинформативных слов, которые не привносят смысла в предложение, такие как *the* и *a*. Удаление отрицаний, таких как *not*, может оказаться вредным, если необходимо выяснить настроение.
- **Морфологический поиск**. Сокращение слова до его основы (формы слова перед добавлением флективных аффиксов, как можно прочитать здесь: `https://www.thoughtco.com/stem-word-forms-1692141`). Такой алгоритм как stemmer может выполнить это на основе ряда правил.
- **Лемматизация**. Преобразуйте слово в словарную форму (лемму). Это альтернатива морфологическому поиску, но она сложнее, поскольку вы используете не алгоритм. Вместо этого, для преобразования каждого слова в его словарную форму, используется словарь.
- **Частеречная разметка**. Помечает каждое слово во фразе его грамматической ролью в предложении (например, помечает слово как глагол или как существительное).
- **n-грамма**. Связывает каждое слово с определенным количеством (*n* в n-грамме) следующих слов и рассматривает их как уникальный набор. Лучше всего в целях анализа обычно работают *биграммы* (серия из двух смежных элементов или токенов) и *триграммы* (серия из трех смежных элементов или токенов).

Для выполнения этих преобразований вам может понадобиться специализированный пакет Python, такой как NLTK (`http://www.nltk.org/api/nltk.html`) или Scikit-learn (см. руководство по адресу `https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html`). При работе с глубоким обучением при большом количестве примеров вам нужны только базовые преобразования: нормализация, очистка и лексический анализ. Слои глубокого обучения могут определять, какую информацию извлекать и обрабатывать. При работе с несколькими примерами вам необходимо обеспечить как можно больше обработки NLP, чтобы помочь сети глубокого

обучения определить, что делать, несмотря на небольшое количество данных, имеющихся в нескольких примерах.


СОВЕТ

Keras предлагает функцию `keras.preprocessing.text.Tokenizer`, способную нормализовать (параметр `lower` в значении `True`) очищать (параметр `filters` содержит строку удаляемых символов, обычно это: `'!"#$%&()*+,-./:;<=>?@ [\]^_`{|}~ '`) и лексически анализировать.

Помещение всех документов в набор

После обработки текста вы должны извлечь соответствующие признаки, что означает преобразование оставшегося текста в числовую информацию для обработки нейронной сетью. Обычно это осуществляется с использованием *набора слов* (bag-of-words), получаемого в ходе частотного или двоичного кодирования текста. Этот процесс эквивалентен преобразованию каждого слова в столбец матрицы шириной, равной количеству слов, которое вам нужно представить. В следующем примере показано, как организовать этот процесс и что он подразумевает. В примере используется список текстов, созданный ранее в этой главе. В качестве первого шага вы подготавливаете базовую нормализацию и лексический анализ, используя несколько команд Python, чтобы определить размер словаря для обработки.

```
unique_words = set(word.lower() for phrase in texts for
                   word in phrase.split(" "))
print(f"There are {len(unique_words)} unique words")
```

Код сообщает о 14 словах. Теперь вы приступаете к загрузке функции `Tokenizer` из Keras и настраиваете ее на обработку текста, предоставляя ожидаемый размер словаря.

```
from keras.preprocessing.text import Tokenizer
vocabulary_size = len(unique_words) + 1
tokenizer = Tokenizer(num_words=vocabulary_size)
```


СОВЕТ

Использование слишком маленького значения `vocabulary_size` может привести к исключению важных слов из процесса обучения. Слишком большой словарь может бесполезно занять память компьютера. Вам необходимо предоставить функции `Tokenizer` правильную оценку количества отдельных слов, содержащихся в списке текстов. Вы также всегда добавляете 1 к значению `vocabulary_size`, чтобы обеспечить дополнительное слово для начала фразы. На этом этапе `Tokenizer` соотносит слова, присутствующие в текстах, с индексами — числовыми значениями, представляющие слова в тексте.

```
tokenizer.fit_on_texts(texts)
print(tokenizer.index_word)
```

В результате индексы выглядят следующим образом.

```
{1: 'is', 2: 'my', 3: 'dog', 4: 'gets', 5: 'along',
 6: 'with', 7: 'cats', 8: 'that', 9: 'cat', 10: 'vicious',
 11: 'happy', 12: 'when', 13: 'it', 14: 'lunch'}
```

Индекс представляет номер столбца, в котором находится информация о слове.

```
print(tokenizer.texts_to_matrix(texts))
```

Вот итоговая матрица.

```
[[0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.]]
```

Матрица состоит из 15 столбцов (14 слов плюс указатель начала фразы) и трех строк, представляющих три обработанных текста. Это текстовая матрица для обработки с использованием неглубокой нейронной сети (RNN требуют другого формата, как будет описано ниже), размер которой всегда определяется как `vocabulary_size` на количество текстов.

Числа внутри матрицы представляют количество раз, когда слово встречается во фразе. Но это не единственное возможное представление. Вот другие.

- **Частотное кодирование**. Сколько раз слово встречается во фразе.
- **Унитарное кодирование**. Отмечает наличие слова во фразе, независимо от того, сколько раз оно встретилось.
- **Частота термина — инверсная частота в документе** (Term Frequency–Inverse Document Frequency — TF-IDF). Кодирует меру количества вхождений слова в документе относительно общего количества слов в матрице. (Слова с высокими оценками более характерны; слова с низкими оценками менее информативны.)

Вы можете напрямую использовать трансформацию TF-IDF из Keras. `Tokenizer` предлагает метод `text_to_matrix`, который стандартно кодирует текст и преобразует его в матрицу, где столбцы — это слова, а строки — это тексты, а значения — это частоты слов в тексте. Если вы примените преобразование, указав `mode='tfidf'`, для заполнения значений матрицы преобразование использует TF-IDF вместо частоты слов.

```
print(np.round(tokenizer.texts_to_matrix(texts, mode='tfidf'), 1))
```

Обратите внимание, что, используя матричное представление, независимо от того, используете ли вы двоичный, частотный или более сложный подсчет

TF-IDF, вы теряете смысл словосочетания, существующий во фразе. Во время обработки слова разбросаны по разным столбцам, и нейронная сеть не может угадать порядок слов во фразе. Именно из-за отсутствия порядка такой подход называется *набором слов* (bag-of-words). Подход набора слов используется во многих алгоритмах машинного обучения, с результатами, зачастую варьирующимися от хорошего до удовлетворительного, и вы можете применить его к нейронной сети, используя слои с плотной архитектурой. Преобразования слов, закодированных в `n_grams` (обсуждаются в предыдущем абзаце как преобразование обработки NLP), предоставляют некоторую дополнительную информацию, но, опять же, связать слова вы не можете.

RNN отслеживают последовательности, поэтому они по-прежнему используют унитарное кодирование, но кодируют не всю фразу, а индивидуально каждый *токен* (token) (который может быть словом, символом или даже группой символов). По этому они ожидают последовательности индексов, представляющих фразу.

```
print(tokenizer.texts_to_sequences(texts))
```

Когда каждая фраза передается на вход нейронной сети в виде последовательности порядковых чисел, число превращается в унитарно кодированный вектор. Затем векторы с унитарным кодированием подаются в слои RNN по одному, что облегчает их изучение. Вот, например, преобразование первой фразы в матрице.

```
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

В этом представлении вы получаете отдельную матрицу для каждого фрагмента текста. Каждая матрица представляет отдельные тексты в виде отдельных слов с использованием столбцов, но теперь строки представляют порядок следования слов. (Первая строка — это первое слово, вторая строка — это второе слово и т.д.)

Запоминание имеющих значение последовательностей

Работа с TF-IDF и n-граммами (буквами или словами) позволяет создавать языковые модели на нескольких примерах. Кодирование фраз в виде

последовательностей однословных унитарных кодировок помогает эффективно использовать RNN. Однако лучший способ обрабатывать текстовые данные с большей скоростью (способ создания мощных моделей глубокого обучения) — это использовать *векторное представление* (embedding).

Векторные представления имеют долгую историю. Понятие векторного представления появилось в статистическом многомерном анализе под названием *многомерного анализа соответствия* (multivariate correspondence analysis). С 1970-х годов Жан-Поль Бенцери (Jean-Paul Benzécri), французский статистик и лингвист, а также многие другие французские исследователи из Французской школы анализа данных, нашли способ отображать ограниченный набор слов в низкоразмерные пространства (обычно это двумерные представления, такие как топографическая карта). Создание процесса превращения слова в зн*а*чимые числа и проекции привело ко многим открытиям в лингвистике и социальных науках и проложило путь для последних достижений в обработке языков с использованием глубокого обучения.

Понимание семантики по векторным представлениям слов

Нейронные сети невероятно быстры в обработке данных и поиске правильных весов для достижения наилучших прогнозов, как и все слои глубокого обучения, которые обсуждались до сих пор: от CNN до RNN. Эти нейронные сети имеют пределы эффективности, обусловленные данными, которые они должны обрабатывать, такие как нормализация данных, позволяющая нейронной сети работать должным образом, или фиксация диапазона входных значений от 0 до +1 или от –1 до +1, чтобы уменьшить проблемы при обновлении весов сети.

ЗАПОМНИ!

Нормализация выполняется внутри сети с помощью функций активации, таких как tanh. Она сжимает значения для отображения в диапазоне от –1 до +1 (`https://tex.stackexchange.com/questions/176101/plotting-the-graph-of-hyperbolic-tangent`), или с помощью специализированных слоев, таких как BatchNormalization (`https://keras.io/layers/normalization/`), применяющих статистическое преобразование для значений, передаваемых из одного слоя в другой.

Другой тип проблемных данных, которые нейронной сети трудно обрабатывать, — это разреженные данные. Ваши данные разреженны, когда они в основном состоят из нулевых значений, что и происходит, когда вы обрабатываете текстовые данные с использованием частотного или двоичного кодирования, даже если вы не используете TF-IDF. При работе с разреженными данными не

только у нейронной сети будут проблемы с поиском хорошего решения (как объяснено в ответах Quora: `https://www.quora.com/Why-are-deep-neural-networks-so-bad-with-sparse-data`), но вам также понадобится огромное количество весов для входного слоя, поскольку разреженные матрицы обычно довольно широки (у них много столбцов).

Проблемы с разреженными данными мотивировали использование *векторного представления слов* (word embedding), позволяющего преобразовать разреженную матрицу в плотную. Векторное представление слов способно уменьшить количество столбцов в матрице с сотен тысяч до нескольких сотен. Кроме того, они не допускают нулевых значений внутри матрицы. Процесс векторного представления слов не происходит случайным образом, он разработан так, что слова получают одинаковые значения, если они имеют одинаковый смысл или находятся в одинаковых темах. Другими словами, это сложное сопоставление; каждый столбец — это специальная карта (или шкала, если хотите), и похожие или связанные слова оказываются рядом друг с другом.

СОВЕТ

Векторное представление слов — это не единственная передавая техника, которую можно использовать в решении глубокого обучения для обработки неструктурированного текста. Недавно появилась серия предварительно обученных сетей, которые еще больше упрощают моделирование языковых задач. Например, одним из наиболее перспективных методов является Google Bidirectional Encoder Representations from Transformers (BERT). Вот ссылка на сообщение в блоге Google AI, описывающее эту технику: `https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html`.

В качестве другого примера, у вас может быть векторное представление, преобразующее названия разных продуктов в столбцы числовых значений, представляющих собой матрицу векторных представлений слов. В этой матрице слова, обозначающие фрукты, могут иметь одинаковую оценку в определенном столбце. В одной колонке овощи могут иметь разные значения, но не слишком отличающиеся от фруктов. Наконец, названия мясных блюд могут сильно отличаться от фруктов и овощей. Векторное представление выполняет эту работу за счет преобразования слов в значения в матрице. Значения похожи, когда слова являются синонимами или относятся к похожему понятию. (Это *семантическое сходство* (semantic similarity), при этом термин *семантическое* относится к значению слов.)

ЗАПОМНИ!

Поскольку одно и то же семантическое значение может встречаться в разных языках, вы можете использовать тщательно выстроенные векторные представления для помощи в переводе с одного языка на другой: слово на одном языке будет иметь те же оценки векторного представления, что и то же слово на другом языке. Исследователи из лаборатории Facebook AI Research (FAIR) нашли способ синхронизировать различные векторные представления и использовать их для многоязычных приложений, основанных на глубоком обучении (см. подробности на `https://code.fb.com/mlapplications/under-the-hood-multilingual-embeddings/`).

При работе с векторными представлениями слов важно помнить, что они являются продуктом данных и, таким образом, отражают содержание данных, использованных для их создания. Поскольку векторные представления слов требуют большого количества примеров текста для правильной генерации, содержание текстов, вводимых в векторные представления во время обучения, зачастую автоматически извлекается из Интернета и не подвергается тщательному анализу. Использование непроверенного ввода может привести к ошибкам векторного представления слов. Например, вы можете быть удивлены, обнаружив, что векторные представления слов создают неправильные ассоциации между словами. Вы должны быть осведомлены об этом риске и тщательно проверять свое приложение, поскольку в результате вы можете внести некорректности в создаваемые приложения глубокого обучения.

На данный момент наиболее популярными векторными представлениями слов, зачастую используемыми для приложений глубокого обучения, являются.

» **Word2vec**. Создано группой исследователей во главе с Томашем Миколовым (Tomáš Mikolov) из Google (оригинальную статью об этом запатентованном методе можно прочитать здесь: `https://arxiv.org/pdf/1301.3781.pdf`). Он опирается на два мелких слоя нейронной сети, которые пытаются научиться прогнозировать слово, зная слова, предшествующие ему и следующие за ним. Word2vec поставляется в двух версиях: одна основана на чем-то, похожем на модель набора слов (*непрерывный набор слов* (Continuous Bag-Of-Words — CBOW)), которая менее чувствительна к порядку слов; и другая, основана на n-граммах (continuous skip-gram), и более чувствительна к порядку. Word2vec учится прогнозировать слово с учетом его контекста, используя *дистрибутивную семантику* (distributional hypothesis), а значит, что похожие слова появляются в сходных контекстах слов. Изучая, какие слова должны появляться в разных контекстах, Word2vec усваивает контексты. Обе версии

подходят для большинства приложений, но версия skip-gram на самом деле лучше представляет редкие слова.

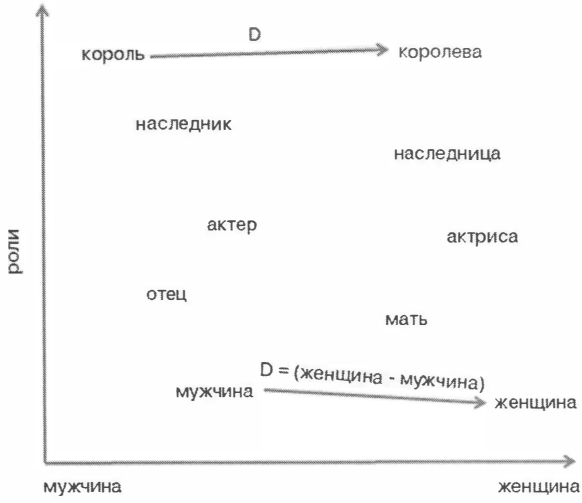
- **GloVe (Global Vectors)**. Разработанный как проект с открытым исходным кодом в Стэнфордском университете (`https://nlp.stanford.edu/projects/glove/`), подход GloVe похож на методы статистической лингвистики. Он получает статистику совпадений слово-слово из корпуса и сводит полученную разреженную матрицу к плотной с помощью *матричной факторизации* (matrix factorization), являющейся алгебраическим методом, широко используемым в многомерной статистике.
- **FastText**. Создано лабораторией Facebook's AI Research (FAIR) (`https://fasttext.cc/`), и представляет собой векторные представления слов, доступное на нескольких языках и работающее с подпоследовательностями слов, а не с отдельными словами. Оно разбивает слово на множество фрагментов из букв и создает их векторное представление. Этот метод имеет интересные последствия, поскольку fastText предлагает лучшее представление редких слов (которые часто состоят из подпоследовательностей, что не редкость) и определяет, как проецировать слова с ошибками. Возможность справляться с орфографическими ошибками позволяет эффективно использовать векторное представление с текстом, поступающим из социальных сетей, электронной почты и других источников, где люди обычно не используют проверку орфографии.

ПОЧЕМУ (КОРОЛЬ – МУЖЧИНА) + ЖЕНЩИНА = КОРОЛЕВА

Векторные представления слова превращает слово в серию чисел, представляющих его положение в самом векторном представлении. Эта серия чисел — *вектор слов* (word vector). Обычно он состоит из порядка 300 векторов (количество векторов, которые Google использовал в своей модели, обученной на наборе новостей Google), и нейронные сети используют его для более качественной и эффективной обработки текстовой информации. Фактически, слова со схожим значением, или используемые в сходных контекстах, имеют похожие векторы слов; следовательно, нейронные сети могут легко обнаружить слова с аналогичным значением. Кроме того, манипулируя векторами нейронные сети способны работать с аналогиями, а значит, вы можете получить удивительные результаты, такие как

- король – мужчина + женщина = королева
- Париж – Франция + Польша = Варшава

Это может показаться магией, но на самом деле — простая математика. Вы можете увидеть, как все работает, посмотрев на следующий рисунок, представляющий два вектора Word2vec.



Каждый вектор в Word2vec представляет различную семантику; это может быть вид пищи, качество человека, национальность или пол. Существует множество типов семантики, и они не определены заранее; обучение векторному представлению создало их автоматически на основе представленных примеров. На рисунке показаны два вектора из Word2vec: один представляет качество человека; другой — пол человека. Первый вектор определяет роли, начиная с короля и королевы, обладающих наиболее высокими оценками, проходит через актера и актрису и, наконец, завершается мужчиной и женщиной с более низкими оценками. Если добавить этот вектор к вектору пола, вы увидите, что мужской и женский варианты различаются оценками в этом векторе. Теперь, когда вы вычитаете из короля мужчину и добавляете женщину, вы просто переходите от координат короля и перемещаетесь вдоль вектора пола, пока не достигнете координат королевы. Этот простой прием координат Word2vec не подразумевает какого-либо понимания слов, и возможен потому, что все векторы векторного представления слов синхронизированы, представляя значения слов, и вы можете осмысленно переходить от одной координаты к другой, когда понятия в рассуждениях меняются.

Использование искусственного интеллекта для анализа настроений

Анализ настроений в вычислительном отношении происходит из анализа письменного текста и отношения автора (позитивное, негативное или нейтральное) к теме текста. Этот вид анализа полезен для специалистов в сфере маркетинга и коммуникаций, поскольку он помогает им понять, что клиенты и потребители думают о продукте или услуге, и, таким образом, действовать соответствующим образом (пытаясь, например, успокоить недовольных клиентов или применив другую стратегию продаж). Всем нужен анализ настроений. Например, при чтении текста, каждый, естественно, пытается определить чувства, которые затронули написавшего его человека. Но когда количество текстов, которые нужно прочитать и понять, слишком велико и текст постоянно добавляется, как в социальных сетях и клиентских электронных письмах, автоматизация задачи важна.

Следующий пример — это пробный запуск RNN с использованием Keras и TensorFlow, создающий алгоритм анализа настроений, способный классифицировать отношения, выраженные в обзоре фильма. Данные представляют собой выборку из набора данных IMDb, содержащую 50 000 обзоров фильмов (разделенных пополам на обучающие и тестовые). Обзоры сопровождаются меткой, выражающей их мнение (0 = отрицательное, 1 = положительное). Набор IMDb (`https://www.imdb.com/`) — это большая сетевая база данных, содержащая информацию о фильмах, сериалах и видеоиграх. Первоначально поддерживаемый любителями, теперь он контролируется дочерней компанией Amazon. На IMDb люди находят нужную им информацию о своем любимом шоу, а также публикуют свои комментарии или пишут обзоры для чтения другими посетителями.

Keras предлагает загружаемую оболочку для данных IMDb. Вы готовите данные, перетасовываете и разделяете их на обучающий и тестовый набор. Этот набор данных содержится среди других полезных наборов данных на `https://keras.io/datasets/`. В частности, текстовые данные IMDb, предлагаемые Keras, очищены от пунктуации, нормализованы в нижний регистр и преобразованы в числовые значения. Каждое слово закодировано в число, представляющее его ранг по частоте. Самые частые слова имеют низкие числовые значения, а редкие слова — высокие.

Для начала код импортирует из Keras функцию `imdb` и использует ее для извлечения данных из Интернета (загрузка около 17,5 МБ). Параметры, используемые в этом примере, охватывают только первые 10000 слов, и Keras должен перетасовать данные с использованием определенного случайного начального

числа. (Знание начального числа, при необходимости, позволяет воспроизводить результат, в противном случае результат случайный.) Функция возвращает два набора обучающих и тестовых данных, оба из текстовых последовательностей и результирующих настроений.

```
from keras.datasets import imdb

top_words = 10000
((x_train, y_train),
 (x_test, y_test)) = imdb.load_data(num_words=top_words, seed=21)
```

После завершения выполнения предыдущего кода вы можете проверить количество примеров, используя следующий код.

```
print("Training examples: %i" % len(x_train))
print("Test examples: %i" % len(x_test))
```

После запроса количества вариантов, доступных для использования на этапе обучения и тестирования нейронной сети, код выводит ответ из 25 000 примеров для каждой фазы. (Этот набор данных является относительно небольшим для языковой задачи; очевидно, набор данных предназначен главным образом для демонстрационных целей.) Кроме того, код определяет, является ли набор данных сбалансированным, что означает, что он имеет почти равное количество примеров положительных и отрицательных настроений.

```
import numpy as np
print(np.unique(y_train, return_counts=True))
```

Результат, `array([12500, 12500])`, подтверждает, что набор данных равномерно распределен между положительными и отрицательными результатами. Такой баланс между классами ответов обусловлен исключительно демонстративным характером набора данных. В реальном мире вы редко встретите сбалансированные наборы данных. Следующий этап создает несколько словарей Python, способных преобразовывать код, используемый в наборе данных, в настоящие слова. Фактически, набор данных, используемый в этом примере, предварительно обработан и предоставляет последовательности чисел, представляющих слова, а не сами слова. (Алгоритмы LSTM и GRU, имеющиеся в Keras, ожидают последовательности в виде чисел.)

```
word_to_id = {w:i+3 for w,i in imdb.get_word_index().items()}
id_to_word = {0:'<PAD>', 1:'<START>', 2:'<UNK>'}
id_to_word.update({i+3:w for w,i in imdb.get_word_index().
  items()})

def convert_to_text(sequence):
    return ' '.join([id_to_word[s] for s in sequence if s>=3])

print(convert_to_text(x_train[8]))
```

Предыдущий фрагмент кода определяет два словаря преобразований (из слов в числовые коды и наоборот) и функцию, переводящую примеры набора данных в читаемый текст. В качестве примера, код выводит девятый случай: "this movie was like a bad train wreck as horrible as it was . . ." (этот фильм был похож на ужасное крушение поезда, столь же ужасное, как и раньше). Как легко догадаться из этой выдержки, отношение к данному фильму позитивным не является. Такие слова, как *bad, wreck* и *horrible* (плохо, крушение и ужас), передают сильное негативное чувство, и это позволяет легко выявить правильное мнение.


СОВЕТ

В этом примере вы получаете числовые последовательности и превращаете их обратно в слова, но и обратное преобразование встречается нередко. Обычно вы получаете фразы, состоящие из слов, и превращаете их в последовательности целых чисел для передачи в слой RNN. Keras предлагает специальную функцию `Tokenizer` (см. `https://keras.io/preprocessing/text/#tokenizer`), способную сделать это за вас. Она использует метод `fit_on_text`, чтобы научиться соотносить слова с целыми числами из обучающих данных, и `text_to_matrix`, чтобы преобразовать текст в последовательность.

Тем не менее, в других фразах вы можете не найти таких откровенных слов. Отношение выражается более тонким или косвенным образом, и понимание настроения в начале текста может быть невозможным, поскольку ключевые фразы и слова могут появиться намного позже в обсуждении. Поэтому вам необходимо также решить, какую часть фразы вы хотите анализировать. Традиционно берут начальную часть текста и используют ее в качестве образца всего обзора. Иногда, чтобы выяснить смысл, вам просто нужно несколько начальных слов, например, первые 50, но иногда нужно больше. Особенно длинные тексты не показывают своего отношения сразу. Поэтому вам необходимо понять тип текста, с которым вы работаете, и решить, сколько слов анализировать, используя глубокое обучение. В этом примере рассматриваются только первые 200 слов, которых должно быть вполне достаточно.


СОВЕТ

Возможно вы заметили, что коды слов выдаются начиная с цифры 3, пропуская коды от 0 до 2. Значения ниже используются для специальных тегов, таких как сигнал начала фразы, заполнения пустых мест для фиксации определенной длины последовательности и отметки исключаемых слов, поскольку они встречаются не достаточно часто. Этот пример использует только 10000 самых частых слов. Использование тегов для указания начала, конца и специальных ситуаций — это трюк, хорошо работающий с RNN, особенно при машинном переводе.

```
from keras.preprocessing.sequence import pad_sequences

max_pad = 200
x_train = pad_sequences(x_train,
                        maxlen=max_pad)

x_test = pad_sequences(x_test,
                       maxlen=max_pad)

print(x_train[0])
```

Используя функцию `pad_sequence` из Keras со значением `max_pad` равным 200, код рассматривает первые двести слов каждого отзыва. В случае если обзор содержит менее двухсот слов, перед последовательностью должно быть столько нулевых значений, сколько необходимо для достижения заданного количества элементов последовательности. Обрезка последовательностей до определенной длины и заполнение пустот нулевыми значениями — это *дополнение ввода* (input padding), важная операция обработки при использовании RNN, и алгоритмов глубокого обучения. Теперь код разрабатывает архитектуру.

```
from keras.models import Sequential
from keras.layers import Bidirectional, Dense, Dropout
from keras.layers import GlobalMaxPool1D, LSTM
from keras.layers.embeddings import Embedding

embedding_vector_length = 32
model = Sequential()
model.add(Embedding(top_words,
                    embedding_vector_length,
                    input_length=max_pad))

model.add(Bidirectional(LSTM(64, return_sequences=True)))
model.add(GlobalMaxPool1D())
model.add(Dense(16, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print(model.summary())
```

Предыдущий фрагмент кода определяет форму модели глубокого обучения, где используется несколько специализированных слоев для обработки текстов на естественном языке из Keras. В этом примере требуется также сводка модели (команда `model.summary()`), чтобы определить, что происходит с архитектурой, использующей различные нейронные слои.

У вас есть слой Embedding, преобразующий числовые последовательности в плотные векторные представления слов. Этот тип векторного представления слов больше подходит для изучения слоем RNN, как обсуждалось ранее в этой главе. Keras предоставляет слой Embedded, который, помимо необходимости быть первым слоем сети, может выполнять две задачи.

- Применение векторного представления слова (например, Word2vec или GloVe) для ввода последовательности. Вам просто нужно передать матрицу, содержащую векторное представление, его параметру weights.
- Создание векторного представления слова с нуля на основе получаемых входных данных.

Во втором случае Embedding просто нужно знать.

- input_dim. размер словаря, ожидаемого для данных.
- output_dim. размер создаваемого векторного представления (размерность).
- input_length. ожидаемый размер последовательности.

После определения параметров, в ходе обучения, слой Embedding найдет лучшие веса для преобразования последовательностей в плотную матрицу. Размер плотной матрицы определяется длиной последовательностей и размерностью векторного представления.

ЗАПОМНИ!

Если вы используете слой Embedding предоставленный Keras, вы должны помнить, что функция предоставляет только весовую матрицу размером словаря на размер желаемого векторного представления. Он соотносит слова со столбцами матрицы, а затем настраивает веса матрицы для предоставленных примеров. Это решение, хотя и практичное для нестандартных языковых задач, не аналогично обсуждавшемуся ранее векторному представлению слов.

В примере используется пакет Bidirectional — слой LSTM из 64 ячеек. Bidirectional —это модификация обычного слоя LSTM за счет его удвоения: на первой стороне применяется обычная последовательность вводимых данных; на второй проходит обратная последовательность. Этот подход используется потому, что иногда вы используете слова в другом порядке, и построение двунаправленного слоя уловит любой случай применения слова, независимо от порядка. Реализация Keras действительно проста: вы просто применяете ее как функцию к слою, который хотите сделать двунаправленным.

Двунаправленный LSTM настроен на возврат последовательностей (`return_sequences=True`); то есть для каждой ячейки он возвращает результат, полученный после просмотра каждого элемента последовательности. Результатами для каждой последовательности является выходная матрица 200 × 128, где 200 — это количество элементов последовательности, а 128 — количество ячеек LSTM, используемых слоем. Этот метод не позволяет RNN получать последний результат каждой ячейки LSTM. Фактически, подсказки о настроении текста могут появиться где угодно в последовательности векторных представлений слов.

Короче говоря, важно брать не последний результат каждой ячейки, а лучший результат. Поэтому для проверки каждой последовательности результатов, предоставляемой каждой ячейкой LSTM, и сохранения только максимального результата, код опирается на следующий слой, `GlobalMaxPool1D`. Это должно гарантировать, что пример выбирает самый сильный сигнал из каждой ячейки LSTM, которая, как мы надеемся, специализируется на его способности учиться выбирать некие значимые сигналы.

После фильтрации нейронных сигналов, пример содержит слой из 128 выходов, по одному для каждой ячейки LSTM. Код уменьшает и смешивает сигналы, используя последовательный плотный слой из 16 нейронов с активацией ReLU (пропускающий, таким образом, только положительные сигналы; подробности см. в главе 8). Архитектура завершается конечным узлом, использующим сигмоидную активацию, сжимающую результаты в диапазон 0–1 и делающую их похожими на вероятности. Определив архитектуру, вы можете обучать сеть. Трех эпох (передача данных по сети трижды, чтобы она изучала шаблоны) будет достаточно. Каждый раз код использует пакеты из 256 обзоров, что позволяет сети увидеть каждый раз достаточно разнообразных слов и настроений, прежде чем обновлять свои веса с помощью обратного распространения. Наконец, код фокусируется на результатах, предоставленных проверочными данными (которые не являются частью учебных данных). Получение хорошего результата на проверочных данных означает, что нейронная сеть обрабатывает ввод правильно. Код сообщает о проверочных данных сразу после окончания каждой эпохи.

```
history = model.fit(x_train, y_train,
                    validation_data=(x_test, y_test),
                    epochs=3, batch_size=256)
```

Получение результатов требует времени, но если вы используете графический процессор, его вполне хватит, чтобы выпить чашку кофе. На этом этапе вы можете оценить результаты, снова используя проверочные данные. (В

результатах не должно быть каких-либо неожиданностей или отличий от того, что код сообщал во время обучения.)

```
loss, metric = model.evaluate(x_test, y_test, verbose=0)
print("Test accuracy: %0.3f" % metric)
```

Окончательная точность, представляющая собой процент правильных ответов глубокой нейронной сети, будет составлять около 85-86%. При каждом запуске эксперимента результат будет немного меняться из-за рандомизации при построении нейронной сети. Это совершенно нормально, учитывая небольшой размер данных, с которыми вы работаете. Если вы начнете с правильных весов, обучение пройдет легче и быстрее.

В конце концов, ваша сеть — это анализатор настроений, способный выявить настроения, выраженные в обзорах фильмов, примерно в 85% случаев. Используя еще больше обучающих данных и более сложные нейронные архитектуры, вы можете получить еще более впечатляющие результаты. В маркетинге аналогичный инструмент используется для автоматизации многих процессов, требующих чтения текста и принятия мер. Опять же, вы можете связать подобную сеть с такой нейронной сетью, которая слышит голос и превращает его в текст. (Это еще одно применение RNN, которое теперь используется для Alexa, Siri, Google Voice и многих других личных помощников.) Переход позволяет приложению понимать настроения даже в голосовых выражениях, таких как телефонный звонок от клиента.

Глава 15

Создание произведений изобразительного искусства и музыки

В ЭТОЙ ГЛАВЕ...

- Как имитировать творческий процесс
- Чего не может создать глубокое обучение
- Развитие искусства на основе установленных стилей
- Сочинение музыки на основе установленных стилей

В Интернете идут серьезные дискуссии о том, способны ли компьютеры на творчество при использовании глубокого обучения. Дискуссия раскрывает саму суть того, что значит творчество. Философы и другие мыслители обсуждали эту тему бесконечно, на протяжении всей истории человечества, так и не придя к выводу о том, что именно означает творчество. Следовательно, одна глава книги, написанной всего за несколько месяцев, не решит эту проблему.

Но чтобы обеспечить основу для обсуждений в этой книге, определим *творчество* как способность вырабатывать новые идеи, модели, отношения и так далее. Акцент делается на слове "*новые*", это оригинальность, прогрессивность и воображение, присущие людям. Это не подразумевает копирования чужого стиля, а выработку собственного. Конечно, это определение почти

наверняка вызовет гнев одних и согласие других, но чтобы обсуждение вообще сработало, нам нужно определение. Имейте в виду, это определение не исключает творчества не людьми. Например, некоторые могут привести аргументы в пользу творчества у обезьян.

Эта глава поможет вам понять, как творчество и компьютеры могут объединиться в увлекательном сотрудничестве. В первую очередь следует учесть, что компьютеры делают все на математике, и искусство и музыка не являются исключением. Компьютер может передавать существующие художественные или музыкальные шаблоны в нейронную сеть и использовать результат для создания чего-то, что выглядит новым, но на самом деле основывается на существующем шаблоне. Тем не менее, наряду с этим откровением, второе соображение заключается в том, что алгоритм, используемый для статистического анализа модели и последующего вывода нового произведения искусства, разработал человек. Другими словами, компьютер выполняет эту задачу не сам по себе; он полагался на человека, чтобы обеспечить средства выполнения этой задачи. Более того, именно человек решит, какой стиль имитировать, и определит, какой результат может оказаться эстетически приятным. Короче говоря, компьютер становится инструментом исключительно умного человека для автоматизации процесса создания того, что можно считать новым, но на самом деле это не так.

В рамках процесса определения того, как некоторые воспринимают творчество компьютеров, в главе также рассматривается имитация компьютером установленного стиля. Вы можете убедиться, что глубокое обучение зависит исключительно от математики, даже при выполнении задач, вообще не связанных с математикой. Художник или музыкант не полагается на вычисления, чтобы создать что-то новое, но другие выполняя свою задачу вполне могут на них полагаться. Когда художник или музыкант использует математику для изучения стиля другого, этот процесс называется обучением, а не творчеством. Конечно, вся эта книга о том, как глубокое обучение выполняет задачи обучения, и даже этот процесс сильно отличается от того, как учатся люди.

Учимся подражать искусству и жизни

Вы, вероятно, видели интересные визуализации об искусстве искусственного интеллекта, такие как упомянуты в статье на `https://news.artnet.com/art-world/ai-art-comes-to-market-is-itworth-the-hype-1352011`. Произведения искусства, несомненно, имеют эстетическую привлекательность. Фактически, в статье упоминается, что Кристи, один из самых известных аукционных домов в мире, первоначально предполагал продавать

такие произведения искусства за 7 000–10 000 долларов США, но на самом деле его продали за 432 000 долларов, по данным *Guardian* (https://www.theguardian.com/artanddesign/shortcuts/2018/oct/26/call-that-art-can-a-computer-be-a-painter) и *New York Times* (https://www.nytimes.com/2018/10/25/arts/design/ai-art-sold-christies.html). Так что у произведения привлекателен не только вид, оно может принести и много денег. Тем не менее, в каждой непредвзятой статье, которую вы читаете, остается вопрос о том, является ли искусство искусственного интеллекта вообще искусством. Следующие разделы помогут вам понять, что компьютерное поколение не связано с творчеством — оно превращается в удивительные алгоритмы, использующие новейшие статистические данные.

Передача художественного стиля

Одним из отличительных черт искусства является художественный стиль. Даже когда некто берет фотографию и представляет ее как произведение искусства, метод получения фотографии, ее обработки и, при необходимости, ретуширования, определяет особый стиль. Во многих случаях, в зависимости от мастерства художника, вы даже не можете сказать, что смотрите на фотографию из-за ее художественных элементов (https://www.pinterest.com/lorimcneeartist/artistic-photography/?lp=true).

Некоторые художники становятся настолько известными благодаря своему особому стилю, что другие тратят время на его глубокое изучение и совершенствование своей техники. Например, очень часто имитируют уникальный стиль Винсента Ван Гога (https://www.artble.com/artists/vincent_van_gogh/more_information/style_and_technique). Стиль Ван Гога — это его использование цветов, методов, средств, предмета и множество других соображений — требует интенсивного изучения, чтобы люди могли его копировать. Люди импровизируют, поэтому в качестве описания стиля человека нередко фигурирует суффикс *esque* (в манере). Критик может сказать, что некий художник работает в манере Ван Гога (Goghesque).

ЗАПОМНИ!

При создании произведений искусства компьютер использует определенный художественный стиль для изменения внешнего вида исходного изображения. В отличие от человека, компьютер может идеально воспроизвести конкретный стиль при достаточном количестве последовательных примеров. Конечно, вы можете создать своего рода смешанный стиль, используя примеры из разных периодов жизни художника. Дело в том, что компьютер не создает новый стиль и не импровизирует. Исходное изображение тоже не новое. При работе с компьютером вы видите идеально скопированный

стиль и идеально скопированное исходное изображение, вы просто переносите стиль в исходное изображение, чтобы создать нечто, похожее на оба.

Процесс, используемый для передачи стиля исходному изображению и создания вывода, является сложным и вызывает много дискуссий. Например, важно учитывать, где заканчивается исходный код, и начинаются такие элементы, как обучение. В статье на `https://www.theverge.com/2018/10/23/18013190/ai-art-portrait-auction-christies-belamy-obvious-robbie-barrat-gans` обсуждается одна из таких ситуаций, когда использовался существующий код, но отличающийся от оригинальной реализации, которая заставляет людей задуматься над такими вопросами, как атрибуция, когда произведение искусства создается компьютером. Имейте в виду, что все обсуждения сосредоточены на людях, которые создают код и выполняют обучение компьютера; сам компьютер не участвует в обсуждении, поскольку компьютер просто перебирает цифры.

ДРУГИЕ ВИДЫ ПРОИЗВЕДЕНИЙ ИСКУССТВА

Имейте в виду, что в этой книге обсуждается особый вид компьютерного искусства — произведения сети глубокого обучения. Вы можете найти все другие виды компьютерных произведений искусства, которые не обязательно полагается на глубокое обучение. Одним из наиболее ранних примеров произведений искусства является фрактал (`http://www.arthistory.net/fractal-art/`), созданный с использованием уравнения. Первым из этих фракталов является множество Мандельброта (`http://mathworld.wolfram.com/MandelbrotSet.html`), открытое в 1980 году Бенуа Мандельбротом, польским математиком. Некоторые фракталы сегодня довольно красивы (`https://www.creativebloq.com/computer-arts/5-eye-popping-examples-fractal-art-71412376`) и даже содержат некоторые элементы реального мира. Несмотря на это, творчество принадлежит не компьютеру, который просто вычисляет числа, а математику или художнику, который разрабатывает алгоритм, используемый для генерации фрактала.

Следующий шаг в создании произведений искусства — это *компьютерные изображения* (Computer Generated Imagery — CGI). Скорее всего, вы видели несколько удивительных примеров изображений CGI в фильмах, сегодня они встречается практически везде (`https://www.vice.com/en_us/topic/cgi-art`). Некоторые люди ограничивают CGI трехмерным искусством, а некоторые ограничивают его трехмерным динамическим искусством, используемым для игр и фильмов. Независимо от того, какие ограничения вы накладываете на произведения искусства CGI, процесс по сути один и тот же.

Художник выбирает серию преобразований для создания эффектов на экране компьютера, таких как вода, которая выглядит влажной, и туман, который выглядит туманным (`https://www.widewalls.ch/cgiartworks/`). CGI находят также применение при построении моделей на основе проектов, таких как архитектурные чертежи (`https://archicgi.com/3d-modeling-things-youve-got-know/` и `https://oceancgi.com/`). Эти модели помогают вам визуально представить, как будет выглядеть готовый продукт задолго до того, как будет вынута первая лопата земли. Однако, в конце концов, вы видите творчество художника, архитектора, математика или другого человека, указывающего компьютеру, чтобы он выполнял различные виды вычислений, чтобы превратить дизайн в нечто, что выглядит реальным. Но сам компьютер во всем этом ничего не понимает.

Сведение проблемы к статистике

На самом деле, компьютеры не видят ничего. Некто берет цифровое изображение объекта реального мира или создает причудливый рисунок, подобный рисунку на рис. 15.1, и каждый пиксель в этом изображении отображается в виде наборов чисел, представляющих значения красного, синего и зеленого для каждого пикселя, как показано на рис. 15.2. Эти числа и являются тем, с чем взаимодействует компьютер, используя алгоритм. Компьютер не понимает, что числа образуют кортеж — это человеческое соглашение. Он знает только то, что алгоритм определяет операции, которые он должен выполнять над сериями чисел. Короче говоря, искусство становится вопросом манипулирования числами с использованием различных методов, включая статистику.

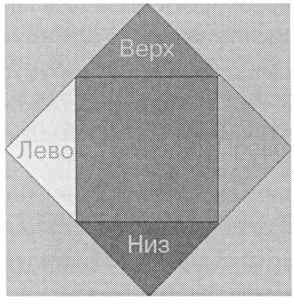


Рис. 15.1. Человек может увидеть причудливый рисунок


ЗАПОМНИ!

Чтобы отразить конкретный стиль, который вы хотите использовать, глубокое обучение опирается на ряд алгоритмов манипулирования пикселями в исходном рисунке различными способами. На самом

деле, вы можете найти головокружительный набор таких алгоритмов, поскольку у каждого, похоже, есть свое представление о том, как заставить компьютер создавать особые виды произведений искусства. Дело в том, что для решения подобных задач все эти методы основаны на алгоритмах, обрабатывающих наборы чисел; чтобы создать что-то новое, компьютер никогда не берет в руки кисть. Тем не менее, для этого существуют два метода.

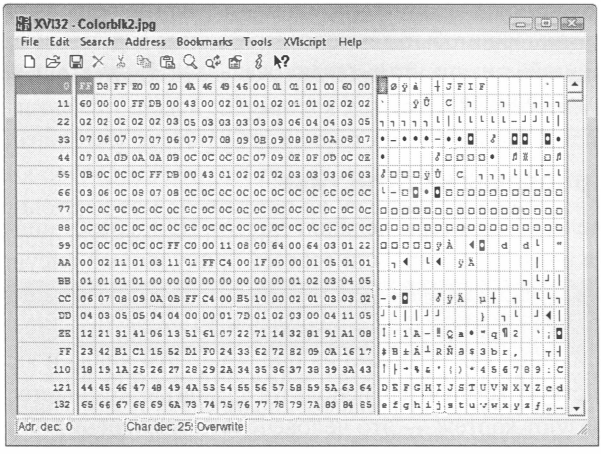


Рис. 15.2. Компьютер видит серию цифр

- **Сверточные нейронные сети (Convolutional Neural Network — CNN)**. См. подробности в главе 10, а также раздел "Определение нового произведения на основе одного примера" далее в этой главе.
- **Генеративно-состязательные сети (Generative Adversarial Network — GAN)**. См. подробности в главе 16, а также раздел "Визуализация мечты нейронных сетей" далее в этой главе.

Что глубокое обучение создать не может

Для произведений искусства, созданных с использованием глубокого обучения, изображения заимствуются, компьютер их вообще не понимает, и только использует алгоритмы для выполнения задач модификации изображений. Глубокое обучение даже не выбирает метод обработки изображений — это делает человек. Короче говоря, глубокое обучение — это интересный метод манипулирования изображениями, созданными кем-то другим, с использованием стиля, также созданного другим человеком.

ЗАПОМНИ!

Может ли глубокое обучение что-то создать, — это неправильный вопрос. Вопрос в том, могут ли люди оценить результат глубокого обучения. Несмотря на неспособность понимать или создавать, глубокое обучение способно дать некие удивительные результаты. Следовательно, творчество лучше оставить людям, но глубокое обучение может дать каждому выразительный инструмент, даже не творческим людям. Например, вы можете использовать глубокое обучение для создания портрета любимого человека в стиле Ван Гога, который будет висеть на вашей стене. Тот факт, что вы участвовали в этом процессе и что у вас есть нечто нарисованное профессионально, — это то, что и нужно учитывать, а не то, способен ли компьютер на творчество.

Подражание художнику

Глубокое обучение позволяет подражать конкретному художнику. Вы можете имитировать любого художника, которого хотите, поскольку компьютер ничего не понимает ни в стиле, ни в рисовании. Алгоритм глубокого обучения будет точно воспроизводить стиль, основанный на вводимых вами данных. Следовательно, имитация является гибким способом получения определенного результата, как описано в следующих разделах.

Определение нового произведения на основе одного примера

В приложениях глубокого обучения используются *сверточные нейронные сети* (CNN). Например, они используются для беспилотных автомобилей и систем распознавания лиц. В главе 10 приведены некоторые дополнительные примеры того, как CNN выполняют свою работу, но суть в том, что CNN могут хорошо выполнять задачи распознавания только при достаточном обучении.

Интересно, что CNN особенно хорошо распознают художественный стиль. Таким образом, вы можете объединить два произведения искусства в одно. Тем не менее, эти две части предоставляют два разных вида ввода для CNN.

» **Содержание**. Изображение, определяющее желаемый результат. Например, если вы предоставите изображение кошки, вывод будет выглядеть как кошка. Это не будет тот кот, с которого вы начали, но содержание определяет желаемый результат в отношении того, что увидит человек.

- **Стиль**. Изображение, определяющее желаемую модификацию. Например, если вы предоставите пример картины Ван Гога, результат будет отражать этот стиль.


СОВЕТ

В общем, вы видите CNN, полагающиеся на одно изображение содержимого и одно изображение стиля. Использование только двух таких изображений позволяет увидеть, как содержимое и стиль работают вместе при получении определенного результата. Пример метода объединения двух изображений таким способом приведен по адресу `https://medium.com/mlreview/making-ai-art-with-style-transfer-using-keras-8bb5fa44b216`.

Конечно, вам нужно решить, как комбинировать изображения. Фактически, именно здесь в игру вступает статистика глубокого обучения. Для выполнения этой задачи вы используете *нейронный перенос стиля* (neural style transfer), как обрисовано в общих чертах в статье Леона А. Гатиса (Leon A. Gatys), Александра С. Эккера (Alexander S. Ecker) и Матиаса Бетге (Matthias Bethge) *A Neural Algorithm of Artistic Style* (`https://arxiv.org/pdf/1508.06576.pdf` или `https://www.robots.ox.ac.uk/~vgg/rg/papers/1508.06576v2.pdf`).

Алгоритм работает с такими видами изображений: *изображение содержимого*, на котором изображен представляемый объект; *изображение стиля*, предоставляющее изобразительный стиль, который вы хотите имитировать; а также *входное изображение*, являющееся изображением для преобразования. Входное изображение обычно является случайным изображением или тем же изображением, что и изображение содержимого. Передача стиля подразумевает сохранение содержимого (т.е., если вы начнете с фотографии собаки, результат все равно будет содержать собаку). Однако преобразованное входное изображение ближе к изображению стиля в представлении. Используемый алгоритм определяет две меры потерь.

- **Потеря содержимого**. Определяет количество исходного изображения, используемого CNN для обеспечения вывода. Большая потеря означает, что результат будет лучше отражать предоставляемый стиль. Тем не менее, вы можете достичь точки, после которой потеря будет настолько велика, что вы больше не сможете видеть содержимое.
- **Потеря стиля**. Определяет способ применения стиля к содержимому. Более высокий уровень потерь означает, что содержимое сохраняет свой первоначальный стиль. Потеря стиля должна быть достаточно низкой, чтобы вы могли получить новое произведение искусства, отражающее желаемый стиль.

Наличие только двух изображений не позволяет проводить обширное обучение, поэтому вы используете предварительно обученную сеть глубокого обучения, такую как VGG-19 (победитель конкурса ImageNet, созданную Visual Geometry Group, VGG, в Оксфордском университете в 2014 году). Предварительно обученная сеть глубокого обучения уже знает, как преобразовать изображение в элементы изображения различной сложности. Алгоритм нейронной передачи стиля включает CNN VGG-19, исключая только последние полносвязные слои. Таким образом, у вас есть сеть, действующая как фильтр обработки изображений. Когда вы отправляете изображение, сеть VGG-19 преобразует его в представление нейронной сети, которое может полностью отличаться от оригинала. Однако когда в качестве фильтров изображения вы используете только верхние слои сети, сеть преобразует полученное изображение, но не меняет его полностью.

Используя преимущества преобразующих свойств нейронной сети, нейронный перенос стиля не использует все свертки VGG-19. Вместо этого она отслеживает их, используя две меры потерь, для гарантии того, что, несмотря на преобразования, примененные к изображению, сеть обеспечит вывод содержимого и применение стиля. Таким образом, когда вы пропускаете входное изображение через сеть VGG-19 несколько раз, ее веса корректируются для выполнения двойной задачи: сохранения содержимого и изучения стиля. После нескольких итераций, которые фактически требуют большого количества вычислений и обновлений весов, сеть преобразует ваше входное изображение в ожидаемое изображение и художественный стиль.

СОВЕТ

Вывод CNN зачастую называют *стилизацией* (pastiche). Это причудливое слово, которое обычно означает художественное произведение, состоящее из элементов, заимствованных из мотивов или техник других художников. Учитывая характер глубокого обучения искусству, термин уместен.

Объединение стилей для создания нового произведения искусства

Если вы действительно хотите быть модным, вы можете создать стилизацию на основе нескольких стилевых изображений. Например, вы могли бы обучить CNN, используя несколько работ Моне так, что стилизация больше походила на работу Моне. Конечно, вы можете так же легко объединить стили нескольких художников-импрессионистов, чтобы создать уникальное произведение искусства, отражающее стиль импрессионизма в целом. Фактические методы выполнения этой задачи различается, но в статье на странице

https://ai.googleblog.com/2016/10/supercharging-style-transfer.html предлагаются идеи для решения этой задачи.

Визуализация мечты нейронных сетей

Использование CNN, по сути, является ручным процессом в отношении выбора функций потерь. Успех или неудача CNN зависит от человека, устанавливающего различные значения. GAN использует другой подход. Для автоматической настройки значений и обеспечения лучшего вывода она опирается на две интерактивные глубокие сети. Эти две глубокие сети обычно называют как.

- **Генератор**. Создает изображение на основе введенных вами данных. Изображение должно сохранять оригинальное содержимое, но с соответствующим уровнем стилизации, который трудно отличить от оригинала.
- **Дискриминатор**. Определяет, является ли вывод генератора достаточно реальным для передачи в качестве оригинала. Если нет, дискриминатор обеспечивает обратную связь, сообщая генератору, что не так со стилизацией.

Чтобы эта настройка работала, вы фактически обучаете две модели: одну для генератора и другую для дискриминатора. Они действуют согласованно: генератор создает новые примеры, а дискриминатор сообщает генератору, что не так с каждым примером. Процесс продолжается назад и вперед между генератором и дискриминатором, пока стилизация не достигнет определенного уровня совершенства. В главе 16 вы можете найти более подробное объяснение того, как работают GAN.

СОВЕТ

Этот подход выгоден потому, что он обеспечивает более высокий уровень автоматизации и более высокую вероятность хороших результатов, чем использование CNN. Недостатком является то, что этот подход также требует значительного времени для реализации, и требования к обработке намного выше. Следовательно, использование подхода CNN зачастую лучше для достижения достаточно хорошего результата. Вы можете увидеть пример подхода GAN по адресу https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0.

Использование сети для сочинения музыки

В этой главе основное внимание уделяется изобразительному искусству, поскольку вы можете легко судить о происходящих с ним тонких изменениях. Тем не менее, те же методы работают и с музыкой. Вы можете использовать

сети CNN и GAN для создания музыки на основе определенного стиля. Компьютеры не могут видеть произведений визуального искусства, музыку они также не могут слышать. Музыкальные тона становятся числами, которыми компьютер манипулирует так же, как он манипулирует числами, связанными с пикселями. Компьютер не видит никакой разницы вообще.

Тем не менее, глубокое обучение обнаруживает разницу. Да, для музыки вы используете те же алгоритмы, что и для визуального искусства, но используемые настройки отличаются, и обучение также уникально. Кроме того, некоторые источники говорят, что обучение музыке намного сложнее, чем изобразительному искусству (см. подробности на `https://motherboard.vice.com/en_us/article/qvq54v/why-is-ai-generated-music-still-so-bad`). Конечно, часть трудностей связана с различиями людей, слушающих музыку. Как сообществу, людям, казалось бы, трудно определить эстетически приятную музыку, и даже людям, которые любят определенный стиль или определенных композиторов, редко нравится все, что они создают.

В некоторых отношениях инструменты, используемые для сочинения музыки с использованием искусственного интеллекта, более формализованы и зрелы, чем инструменты, используемые для изобразительного искусства. Это не означает, что инструменты музыкальной композиции всегда дают отличные результаты, но это значит, что вы можете легко купить пакет для выполнения задач музыкальной композиции. Вот два самых популярных предложения на сегодня.

- **Amper**. `https://www.ampermusic.com/`
- **Jukedeck**. `https://www.jukedeck.com/`

ЗАПОМНИ!

Музыкальная композиция искусственного интеллекта отличается от создания произведений изобразительного искусства, потому что музыкальные инструменты существуют уже долгое время, согласно статье на `https://www.theverge.com/2018/8/31/17777008/artificial-intelligence-taryn-southern-amper-music`. Покойный автор песен и исполнитель Дэвид Боуи (David Bowie) использовал приложение Verbasizer (`https://motherboard.vice.com/en_us/article/xygxpn/the-verbasizer-was-david-bowies-1995-lyric-writing-mac-app`) еще в 1995 году, для помощи в своей работе. Ключевая идея здесь заключается в том, что этот инструмент помогал, а не делал всю работу. Человек — это творческий талант; искусственный интеллект служит лишь творческим инструментом для создания лучшей музыки. Следовательно, в музыке искусственный интеллект, скорее помощник, а не центральная фигура.

Глава 16

Построение генеративно-состязательных сетей

В ЭТОЙ ГЛАВЕ...

- Как нейронные сети могут создавать достоверные данные
- Сети GAN, способные создавать рукописные числа
- Графические и музыкальные приложения GAN

Глубокое обучение превратилось в ведущую технологию, и новые исследования все время дают все более впечатляющие открытия. Открытия всегда появляются чаще во время Конференций по машинному обучению и нейровычислениям (Conference on Neural Information Processing Systems — (NeurIPS)) (`https://neurips.cc/`), служащих сценой для всего, что связано с глубоким обучением. Конференция проводится ежегодно в разных местах по всему миру (совсем недавно, до публикации этой книги, в Монреале, Канада).

Конференция делает доступными для людей все новые технологии, но некоторым областям уделяется больше внимания. На недавней конференции представлено впечатляющее разнообразие приложений и новых технологий, связанных с глубоким обучением. Вот те, на которые стоит обратить наибольшее внимание: обработка текстов на естественном языке (особенно для таких предварительно обученных систем векторного представления как BERT, обсуждавшихся в главе 14); обучение с подкреплением (тема следующей главы); и генеративно-состязательная сеть (Generative Adversarial

Network — GAN). GAN — это идея без границ. Ян Лекун, ныне директор Facebook AI, определяет ее как "самую интересную идею в машинном обучении за последние десять лет ".

В этой главе описывается, что такое GAN, и демонстрируется, как они способны генерировать новые данные, особенно изображения, из уже существующих. Глава завершает обзор GAN созданием такой сети с использованием Keras и TensorFlow.

После того, как вы увидите GAN в действии, мы обсудим в этой главе наиболее интересные разработки и достижения GAN.


ЗАПОМНИ!

Вам не нужно вводить исходный код примеров из этой главы вручную. Намного проще использовать загружаемый исходный код (подробности о загрузке исходного кода см. во введении). Исходный код примеров из этой главы представлен в файле `DL4D_16_MNIST_GAN.ipynb`.

Создание состязающихся сетей

В 2014 году на факультете информатики Монреальского университета Ян Гудфеллоу (Ian Goodfellow) и другие исследователи (включая одного из самых известных ученых Йошуа Бенджио (Yoshua Bengio), занимающегося искусственными нейронными сетями и глубоким обучением) опубликовали первую статью о GAN. Вы можете прочитать эту работу по адресу `https://arxiv.org/pdf/1406.2661v1.pdf` или `https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`. В последующие месяцы статья привлекла внимание и была признана инновационной благодаря предложенному сочетанию глубокого обучения и теории игр. Идея получила широкое распространение благодаря своей доступности с точки зрения архитектуры нейронной сети: вы можете получить вполне работоспособную сеть GAN, используя стандартный компьютер. (Техника работает лучше, если вы можете позволить себе больше вычислительной мощи.)

В отличие от других нейронных сетей глубокого обучения, классифицирующих изображения или их последовательности, специализация GAN в их способности генерировать новые данные, вдохновляясь учебными данными. Эта способность становится особенно впечатляющей при работе с данными изображений, поскольку хорошо обученные сети GAN способны создавать новые произведения искусства, которые люди продают на аукционах (например, произведения искусства, продаваемые на аукционе Кристи почти за полмиллиона долларов, упомянутые в главе 15: `https://www.dezeen.com/2018/10/29/christies-ai-artwork-obvious-portrait-edmond-de-belamy-design/`).

Этот подвиг невероятен, поскольку предыдущие результаты, полученные с использованием других математических и статистических методов, были далеко не заслуживающими доверия или пригодными для использования.

Поиск ключа в состязании

В названии сети GAN содержится слово "состязательная", поскольку ключевая идея, лежащая в основе GAN, — это конкуренция между двумя сетями, играющими роль соперников друг для друга. Ян Гудфеллоу, главный автор оригинальной статьи о GAN, использовал простую метафору, чтобы описать, как все работает. Гудфеллоу описал процесс как бесконечное состязание между фальсификатором и детективом: фальсификатор создает поддельные произведения искусства, копируя некий настоящий шедевр. Затем детектив исследует фальшивую картину и решает, создал ли фальсификатор настоящее произведение искусства или просто фальшивку. Если детектив видит подделку, фальсификатор получает уведомление о том, что с работой что-то не так (но не конкретное указание ошибки). Когда фальсификатор демонстрирует настоящее произведение искусства, благодаря отрицательной обратной связи с детективом, детектив получает уведомление о своей ошибке и меняет технику обнаружения, чтобы избежать ошибок во время следующей попытки. Поскольку фальсификатор продолжает попытки обмануть детектива, и он, и детектив становятся все более компетентными в своих обязанностях. Со временем качество создаваемого фальсификатором произведения искусства становится чрезвычайно высоким, и оно уже почти неотличимо от реальных произведений для тех, кто не обладает опытным взглядом.

На рис. 16.1 показана история GAN в виде простой схемы, где вводы и нейронные архитектуры взаимодействуют друг с другом в замкнутом цикле обратной связи. Сеть *генератора* (generator) играет роль фальсификатора, а сеть *дискриминатора* (discriminator) — роль детектива. GAN используют термин *дискриминатор* из-за сходства по назначению с электронными схемами, которые принимают или отклоняют сигналы на основе их характеристик. Дискриминатор в GAN принимает (ошибочно) или отбрасывает (правильно) работу, созданную генератором. Интересным аспектом этой архитектуры является то, что генератор никогда не видит ни одного обучающего примера. Только дискриминатор получает доступ к таким данным при обучении. Для обеспечения случайной начальной точки, генератор получает каждый раз случайные входные сигналы (шум), что вынуждает его выдавать другой результат.

Может показаться, что генератор забирает себе всю славу (ведь он генерирует данные произведения). Тем не менее, настоящая мощь архитектуры — это дискриминатор. Дискриминатор вычисляет ошибки, которые

распространяются обратно в его собственную сеть, чтобы научиться лучше отличить реальные данные от поддельных. Ошибки распространяются также на генератор, который оптимизирует себя, чтобы обмануть дискриминатор на следующем раунде.

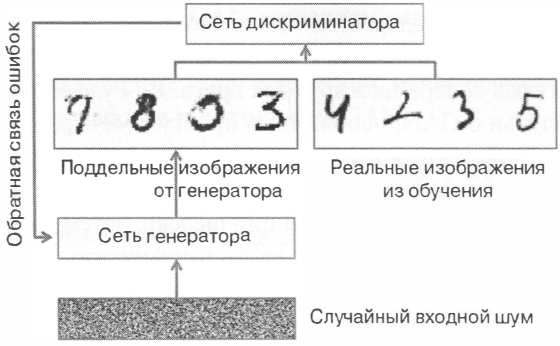


Рис. 16.1. Как работает GAN

ПРОБЛЕМА С ЛОЖНЫМИ ДАННЫМИ

Подобно тому, как GAN способна генерировать впечатляющие произведения искусства, она способна создавать искусственные изображения людей. Посмотрите на `https://www.thispersondoesnotexist.com/`, — это изображение несуществующего человека. Если не знать, где искать, фотография действительно вполне убедительна. Тем не менее, мелкие детали ее выдают.

- Фоны выглядит мутными или лишенными реалистичности.
- Тем, кто смотрел фильм *Матрица*, знакомы эпизодические глюки, появляющиеся на некоторых изображениях.
- Текстура пикселей на переднем плане может быть не совсем правильной. Например, вы можете увидеть муар (`https://photographylife.com/what-is-moire`) там, где его не ожидается.

Но для распознавания такого рода проблем необходим человек. Кроме того, по мере улучшения GAN различные проблемы исчезают. GAN способны подделывать не только картинки. Вы можете создать полностью фальшивую человеческую личность за невероятно короткое время без особых усилий. У GAN могут быть все нужные записи во всех нужных местах. Сегодня существует технология создания поддельной человеческой личности, способной появиться в таких местах, где было бы крайне трудно ее выявить. Это не роботы-убийцы, а реальная проблема, о которой вы должны знать.

ЗАПОМНИ!

Сети GAN могут показаться творческими. Однако на самом деле они являются генеративными: они учатся на примерах, как меняются данные, и они могут генерировать новые выборки, как если бы они были взяты из одних и тех же данных. GAN учится имитировать ранее существующее распределение данных; но не может создать нечто новое. Как упоминалось в других главах, глубокое обучение не является творческим.

Достижение более реалистичных результатов

Даже если концепция GAN ясна, ее архитектура на первый взгляд может показаться сложной. Создание простого примера GAN стало вполне доступным с помощью Keras и TensorFlow, а обучение на практике — хороший способ объяснить детали технологии, которая в противном случае оставалась бы сугубо теоретической. У процесса есть несколько сложных элементов, но, в конце концов, все именно так, как описано в предыдущих параграфах, согласно метафоре Яна Гудфеллоу.

На следующих страницах вы создадите простую сеть GAN, которая научится воссоздавать рукописные числа от нуля до девяти после изучения набора данных MNIST. Набор данных MNIST представляет собой набор оцифрованных, нормализованных, примеров рукописных цифр размером 28×28 пикселей (написанных как учащимися средней школы, так и сотрудниками Американского бюро переписей), которые часто используются для обучения систем распознавания изображений. Вы можете найти этот набор данных на веб-сайте Яна Лекуна по адресу `http://yann.lecun.com/exdb/mnist/`.

Пример начинается с импорта необходимых функций и классов. Для этой задачи вам не нужно ничего особенного, и вы уже имели дело или даже уже опробовали все, что импортирует этот код.

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Dropout
from keras.layers import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
from keras.optimizers import Adam
import matplotlib.pyplot as plt
%matplotlib inline
```

Обратите внимание, что код загружает набор данных MNIST с помощью функции Keras `mnist`. Отдельные массивы изображений размером 28×28 пикселей содержат значения пикселей от 0 до 255. Чтобы сделать их приме-

нимыми для сети глубокого обучения, код обрабатывает их, используя следующие шаги.

1. Изменяет форму данных, сделав их вектором, то есть списком значений.
2. Преобразует значения в тип с плавающей запятой, используя 32-разрядную точность, подходящую для графических процессоров, поскольку 64-разрядная версия применима только для обработки CPU.
3. Преобразует значения в диапазон 0–1.


СОВЕТ

Нормализация — это процесс преобразования данных изображения перед обработкой системой глубокого обучения. Можно использовать различные виды нормализации, такие как масштабирование до диапазона от 0 до 1, от –1 до 1 или применение статистической нормализации за счет вычитания среднего значения и деления на стандартное отклонение. Обычно хорошим рабочим решением является масштабирование всех значений в диапазон от 0 до 1.

```
def normalize(X):
    X = X.reshape(len(X), 784)
    X = X.astype('float32')/255
    return X

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train = normalize(X_train)
```

Подготовив набор данных для обучения нейронной сети, вы можете приступить к подготовке архитектуры GAN. Вы начинаете с определения нескольких параметров, таких как тип вводимых данных, предоставляемых GAN для генерации изображений. Хорошим выбором для этого проекта является использование списка случайных чисел. Считайте эти случайные числа инструкциями для GAN, чтобы она решила, что представлять. Вы мало контролируете то, что сеть GAN делает с числами, но в других моделях вы вполне можете использовать ввод, чтобы получать желаемый результат.

Вы также задаете оптимизатор (в данном случае оптимизатор Adam) и определяете первую часть архитектуры — генератор. Генератор получает случайный ввод и пропускает его через серию из четырех плотных слоев. Единственным заметным аспектом этого процесса является то, что функциями активации LeakyReLU снабжены все слои, за исключением последнего, что демпфирует отрицательные входные данные. BatchNormalization контролирует распределение выводов, применяя к ним статистическую нормализацию. Использование этого подхода позволяет избежать ситуации, когда во время обучения появляется экстремальное число.

Примечательно, что последний слой отличается; для генерации выходных данных в диапазоне от нуля до единицы он использует сигмоидную активацию. Этот последний слой выдает созданное GAN изображение, что делает его генераторной частью архитектуры. Поскольку он генерирует 784 вывода, значения которых находятся в диапазоне от 0 до 1, выходные данные можно легко преобразовать в массивы размером 28×28 пикселей со значениями в диапазоне от 0 до 255 (то есть изображение MNIST).

```
input_dim = 100
np.random.seed(42)
optimizer = Adam(lr=0.0002, beta_1=0.5)

gen = Sequential()
gen.add(Dense(256, input_dim=input_dim))
gen.add(LeakyReLU(alpha=0.2))
gen.add(BatchNormalization())
gen.add(Dense(512))
gen.add(LeakyReLU(alpha=0.2))
gen.add(BatchNormalization())
gen.add(Dense(1024))
gen.add(LeakyReLU(alpha=0.2))
gen.add(BatchNormalization())
gen.add(Dense(784, activation='sigmoid'))
gen.compile(loss='binary_crossentropy',
            optimizer=optimizer)
```

Вторая часть архитектуры, дискриминатор, похожа по конструкции на генератор. Он также имеет четыре плотных слоя, и все, кроме последнего, снабжены функциями активации LeakyReLU. Дискриминатор не использует BatchNormalization, но у него есть Dropout, чтобы избежать переобучения, поскольку эта часть выполняет контролируемую задачу классификации. Фактически, вывод является единственным узлом, выводящим значения вероятности от 0 до 1. Задача этой части нейронной сети заключается в дифференциации поддельных изображений, частично полученных генератором из реальных изображений.

```
dsc = Sequential()
dsc.add(Dense(1024, input_dim=784))
dsc.add(LeakyReLU(alpha=0.2))
dsc.add(Dropout(0.3))
dsc.add(Dense(512))
dsc.add(LeakyReLU(alpha=0.2))
dsc.add(Dropout(0.3))
dsc.add(Dense(256))
dsc.add(LeakyReLU(alpha=0.2))
dsc.add(Dropout(0.3))
```

```
dsc.add(Dense(1, activation='sigmoid'))
dsc.compile(loss='binary_crossentropy',
            optimizer=optimizer)
```

На этом этапе есть сложная часть, вы соединили первую и вторую половину сети и убедились, что они работают вместе. Используя функции API Keras (см. подробнее на `https://keras.io/getting-started/functional-api-guide/`), вы устанавливаете архитектуры, являющиеся куда более сложными, чем последовательные архитектуры, использовавшиеся ранее в этом разделе. В целом, часть генератора обрабатывает входные данные и выводит результат в часть дискриминатора. Дискриминатор действует как математическая функция, применяемая к другим функциям, то есть это функция дискриминатора (функция генератора — это [input]). Таким образом, вы управляете также оптимизацией сети, поскольку можете заморозить ее часть, используя такую функцию `make_trainable` как `def make_trainable(dnn, flag)`. (На самом деле вы обучаете генератор и дискриминатор для достижения максимальных целей).

```
    dnn.trainable = flag
    for l in dnn.layers:
        l.trainable = flag

make_trainable(dsc, False)
inputs = Input(shape=(input_dim, ))
hidden = gen(inputs)
output = dsc(hidden)
gan = Model(inputs, output)
gan.compile(loss='binary_crossentropy',
            optimizer=optimizer)
```

Теперь можно проверить настройки. Вы готовите две полезные, удобные функции для генерации входного шума и отображения результатов генератора.

```
def create_noise(n, z):
    return np.random.normal(0, 1, size=(n, z))

def plot_sample(n, z):
    samples = gen.predict(create_noise(n, z))
    plt.figure(figsize=(15,3))
    for i in range(n):
    plt.subplot(1, n, (i+1))
    plt.imshow(samples[i].reshape(28, 28),
               cmap='gray_r')
    plt.axis('off')
plt.show()
```

Фактический тест начинается с настройки кода для 100 эпох обучения и установки учебной партии на 128 изображений. Код начинает итерации по количеству эпох и пакетов, необходимых для передачи всех обучающих образов в GAN. Как и с другими примерами в книге, для запуска требуется некоторое время. Если вы можете получить доступ к графическому процессору, вам лучше запустить его на Google Colab или на компьютере с видеокартой. Когда вы сможете получить доступ к графическому процессору, запланируйте примерно полчаса ожидания на запуск в Google Colab. (Ваш локальный графический процессор может быть лучше.) На системе только с CPU выполнение примера может занять несколько часов.

```
epochs = 100
batch_size = 128
batch_no = int(len(X_train) / batch_size)
gen_errors, dsc_errors = (list(), list())

for i in range(0, epochs):
    for j in range(batch_no):

    # Выбор случайного примера из обучающего набора
    rand_sample = np.random.randint(0, len(X_train),
        size=batch_size)
    image_batch = X_train[rand_sample]

    # Создание шумовых вводов для генератора
    input_noise = create_noise(batch_size, input_dim)

    # Генерация поддельных изображений из шумового ввода
    generated_images = gen.predict(input_noise)
    X = np.concatenate((image_batch,
        generated_images))

    # Создание неких шумовых меток
    y = np.concatenate([[0.9]*batch_size,
                        [0.0]*batch_size])

    # Обучение дискриминатора отличать подделки от
    # настоящих изображений
    make_trainable(dsc, True)
    dsc_loss = dsc.train_on_batch(X, y)
    make_trainable(dsc, False)

    # Обучение созданию подделок
    input_noise = create_noise(batch_size, input_dim)
    fakes = np.ones(batch_size)
    for _ in range(4):
        gen_loss = gan.train_on_batch(input_noise,
            fakes)
```

```
# Регистрация потерь
gen_errors.append(gen_loss)
dsc_errors.append(dsc_loss)

# Отображение промежуточных результатов
if i % 10 == 0:
    print("Epoch %i" % i)
    plot_sample(10, input_dim)
```

Когда код завершает выполнение множества вычислений, он может пересмотреть необходимые шаги.

1. Создать несколько поддельных изображений, вызвав только функцию генератора. Поскольку это простой прогноз, не требующий обучения, выходные изображения генератора вначале будут казаться совершенно случайными.
2. Объединить поддельные изображения с партией реальных изображений.
3. Передать изображения на дискриминатор, чтобы определить, способен ли дискриминатор отделить поддельные изображения от настоящих. Это обучение, и дискриминатор учится отделять текущие изображения генератора от истинных исходных изображений.
4. Заморозить дискриминатор после того, как он закончит обучение, чтобы код мог запустить его вместе с генератором, но на этот раз учиться будет только генератор. На этом этапе код передает несколько случайных вводов через генератор для преобразования в изображения, а затем передает поддельные данные дискриминатору, чтобы определить, можно ли обмануть дискриминатор и выдать их за реальные изображения.

Когда дискриминатор способен определить, что эти изображения являются продуктом генератора, код будет использовать оценку дискриминатора в качестве ошибки для генератора, чтобы учиться у него (успех дискриминатора является неудачей для генератора).

СОВЕТ

Код содержит несколько трюков, гарантирующих, что GAN всегда даст хорошие результаты.

- При обучении дискриминатора вы придаете меткам истинности некоторую неопределенность, делая тем самым дискриминатор менее строгим.
- Каждый раз, когда вы обучаете дискриминатор, вы также четыре раза обучаете генератор. Дело в том, что обучение созданию изображений на самом деле является более длительным процессом, и использование этого подхода ускоряет данный процесс.

Это два наиболее эффективных трюка, но вы можете прочитать о них еще на этой странице, которую поддерживает Сумит Чинтала (Soumith Chintala), по адресу `https://github.com/soumith/ganhacks`. Ясно, что глубокое обучение все еще больше искусство, чем наука. Вывод некоторых результатов, как показано в следующем коде, демонстрирует, что сеть GAN научилась генерировать почти достоверные рукописные цифры, хотя они и не идеальны. Взглянув на рис. 16.2, вы можете увидеть, чего может достичь GAN за такое короткое время обучения.

```
# Вывод окончательного результата
plot_sample(10, input_dim)
```

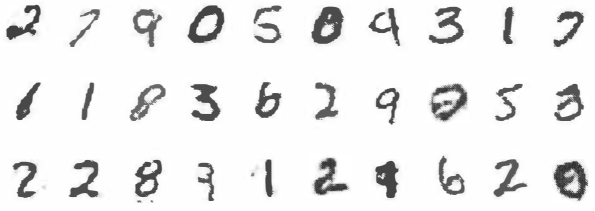


Рис. 16.2. Некоторые результаты сети GAN, обучавшейся 100 эпох

Вы также можете наблюдать ошибки, которые две составляющие GAN сети получали во время обучения. Используйте следующий код (выходные данные демонстрирует рис. 16.3).

```
# Вывод ошибок
plt.figure(figsize=(15, 5))
plt.plot(dsc_errors, label='discriminitive loss')
plt.plot(gen_errors, label='generative loss')
plt.legend()
plt.show()
```

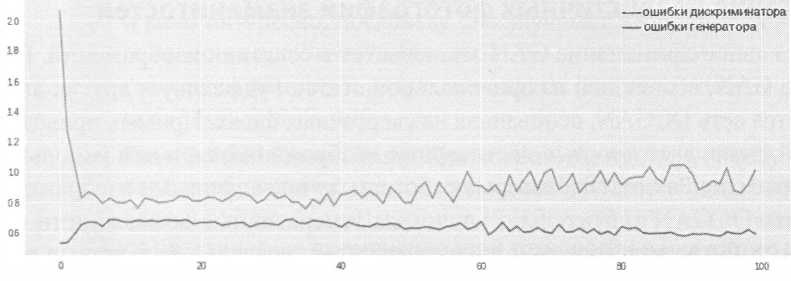


Рис. 16.3. Ошибки генератора и дискриминатора при обучении сети GAN

На рис. 16.3 показано, что ошибки имеют разный масштаб, поскольку ошибка дискриминатора всегда ниже, чем ошибка генератора. Кроме того, ошибка дискриминатора имеет тенденцию уменьшаться, поскольку просмотр бо́льшего количества примеров помогает ему отличать поддельные изображения от реальных, даже если генератор улучшает свои возможности. Что касается генератора, ошибки сначала быстро уменьшаются, но затем обычно стабилизируются, поскольку дискриминатор приобретает опыт в распознавании. Если вы зададите больше эпох, то увидите, что ошибка генератора приобретает синусоидальную форму, т.е. частота ошибок регулярно увеличивается (поскольку генератор становится все более умелым), а затем снова уменьшается (после обнаружения некого нового трюка, чтобы обмануть дискриминатор). Это бесконечная борьба между двумя частями сети GAN — борьба, которая всегда дает все более реалистичные изображения по мере продолжения обучения.

Рост поля деятельности

Начав с простой реализации, аналогичной только что завершенной, исследователи превратили идею GAN в большое количество вариантов, решающих задачи куда более сложные, чем просто создание новых изображений. Список сетей GAN и их приложений растет с каждым днем, и поддерживать его сложно. Авинаш Хиндупур (Avinash Hindupur) в своей статье *GAN Zoo* (Зоопарк сетей GAN), описывает все варианты, и с каждым днем эта задача становится все сложнее. (Вы можете увидеть самые последние обновления на сайте `https://github.com/hindupuravinash/the-gan-zoo`.) Чжэн Лю (Zheng Liu) напротив, предпочитает исторический подход, и вы можете увидеть временну́ю шкалу GAN, которую он поддерживает, на `https://github.com/dongb5/GAN-timeline`. Независимо от того, как вы подходите к GAN, весьма полезно увидеть, как каждая новая идея проистекает из предыдущих.

Создание реалистичных фотографий знаменитостей

Основное применение GAN заключается в создании изображений. Первой сетью GAN, возникшей из оригинальной статьи Гудфеллоу и других авторов, является сеть DCGAN, основанная на сверточных слоях. Пример, приведенный в этой главе, дает простые достоверные изображения, но в нем используются плотные слои, а не CNN, которые работают лучше с данными изображений.

Сеть DCGAN значительно улучшила генеративные возможности оригинальных GAN, и вскоре они впечатлили всех, создавая фальшивые изображения лиц, получая примеры из фотографий знаменитостей. Конечно, не все лица, созданные DCGAN, были достаточно реалистичными, но усилия были

лишь отправной точкой в стремлении создать более реалистичные изображения. Сети EBGAN-PT, BEGAN и Progressive GAN были куда лучше, они достигли более высокой степени реализма. Чтобы получить более точное представление о качестве, достигаемом такими современными методами, можно прочитать документ NVIDIA, подготовленный для Progressive GAN: `https://research.nvidia.com/publication/2017-10_Progressive-Growing-of`.

Следующим большим усовершенствованием GAN стали сети Conditional GAN (CGAN). Хотя создание сетью реалистичных изображений всех видов интересно, это бесполезно, когда вы не можете каким-либо образом контролировать тип получаемых результатов. Сети CGAN манипулируют вводом и сетью GAN, чтобы указать то, что она должна произвести. Теперь, например, у вас могут быть сети производящие изображения лиц несуществующих людей, основываясь уже на ваших предпочтениях в том, как выглядят волосы, глаза и другие детали. Это показано в демонстрационном видео от NVIDIA по адресу `https://www.youtube.com/watch?v=kSLJriaOumA`.

Улучшение деталей и преобразование изображений

Создание изображений более высокого качества и, возможно, управление полученным результатом открыло путь для большего количества приложений. В этой главе нет места для их обсуждения, но в следующем списке представлен обзор того, что вы можете найти.

- **Cycle GAN**. Применяется к нейронному переносу стиля (как описано в главе 10). Например, вы можете превратить лошадь в зебру или картину Моне в картину Ван Гога. Изучив проект по адресу `https://github.com/junyanz/CycleGAN`, вы можете увидеть, как он работает, и рассмотреть, какие преобразования можно применить к изображениям.
- **Super Resolution GAN (SRGAN)**. Преобразует изображения, делая размытые изображения с низким разрешением более четкими и с высоким разрешением. Применение этой техники в фотографии и кино интересно, поскольку она улучшает изображения низкого качества практически без затрат. Вы можете найти статью, описывающую технику и результаты, здесь: `https://arxiv.org/pdf/1609.04802.pdf`.
- **Pose Guided Person Image Generation**. Контролирует позу человека, изображенного на созданном изображении. В документе по адресу `https://arxiv.org/pdf/1705.09368.pdf` описывается практическое применение этой технологии в индустрии моды. Она используется для создания большего количества поз модели, и как ни удивительно, при таком же подходе можно создавать

видеоролики одного человека, танцующего точно так же, как другой: https://www.youtube.com/watch?v=PCBTZh41Ris.

- **Pix2Pix**. Переводит эскизы и карты в реальные изображения и наоборот. Вы можете использовать это приложение для преобразования архитектурных эскизов в изображение реального здания или для преобразования спутниковой фотографии в рисованную карту. В документе https://arxiv.org/pdf/1611.07004.pdf обсуждаются дополнительные возможности, предлагаемые сетью Pix2Pix.
- **Image repairing**. Восстанавливает или изменяет существующее изображение, добавляя то, что отсутствует, удалено или скрыто: https://github.com/pathak22/context-encoder.
- **Face Aging**. Определяет, как лицо будет стареть. Вы можете прочитать об этом на https://arxiv.org/pdf/1702.01983.pdf.
- **Midi Net**. Создает музыку в вашем любимом стиле, как описано на https://arxiv.org/pdf/1703.10847.pdf.

Глава 17

Глубокое обучение с подкреплением

В ЭТОЙ ГЛАВЕ...

- Обучение с подкреплением
- Использование OpenAI Gym для экспериментов
- Как работает Deep Q-Network (DQN)
- Работа с AlphaGo, AlphaGo Zero и Alpha Zero

После рассмотрения примера GAN у вас может возникнуть искушение отождествить глубокое обучение с обучением прогнозированию с учителем. Тем не менее, глубокое обучение вы используете также и для обучения без учителя, и для *обучения с подкреплением* (Reinforcement Learning — RL). Обучение без учителя поддерживает ряд устоявшихся методик, таких как авто-кодеры и самоорганизующиеся карты (Self-Organizing Map — SOM), которые в этой книге не рассматриваются. Методы обучения без учителя могут помочь разделить ваши данные на однородные группы или обнаружить аномалии в ваших переменных.

Среди практиков, методы обучения с подкреплением даже более популярны, чем методы обучения без учителя. Недавно, в результате интенсивных исследований, объект RL позволил находить куда более разумные решения для таких задач, как парковка автомобиля, обучение вождению всего за двадцать минут (как показано в документе `https://arxiv.org/abs/1807.00412`), управление промышленными роботами и многое другое. (Полный список приложений приведен в статье Юйси Ли (Yuxi Li) `https://medium.com/@yuxili/rl-applications-73ef685c07eb`.) В этой главе рассказывается о некоторых

из этих методов, в том числе об AlphaGo, ставшем первым алгоритмом, который в честной игре превзошел профессионального игрока в го (древняя китайская настольная игра).

Вы также получите практический опыт работы с некоторыми примерами, знакомящими вас с OpenAI Gym (`https://gym.openai.com/`), полным набором инструментов для экспериментов с глубоким обучением, и с keras-rl (`https://github.com/keras-rl/keras-rl`) — готовой к использованию реализацией самых современных алгоритмов RL, таких как Google Deep Q-Network (DQN). DQN — это алгоритм, используемый для игры в классические игры Atari 2600 на уровне эксперта. (DQN — это только одно из возможных применений этой техники, запатентованное Google DeepMind). После того, как вы узнаете на примере, как создать работающую сеть с глубоким обучением, способную успешно играть в простую игру, вы рассмотрите, как работает AlphaGo и почему его победа является такой вехой для глубокого обучения в частности и для AI в целом.

ЗАПОМНИ!

Вам не нужно вводить исходный код примеров из этой главы вручную. Намного проще использовать загружаемый исходный код примеров из (подробности о загрузке исходного кода см. во введении). Исходный код этой главы представлен в файле `DL4D_17_Reinforcement_Learning.ipynb`.

Играя в игру с нейронными сетями

Будучи малышом, вы, возможно, с удовольствием открывали для себя окружающий мир и рисковали, чтобы проверить свои способности под бдительным взором родителей. Только впоследствии вы начали заменять знания, полученные на непосредственном опыте, знаниями, полученными от других. Подобно тому, как алгоритм машинного обучения с учителем напоминает учащегося, изучающего мир, исходя из чьего-либо прошлого опыта, описанного в книгах (в этой метафоре опыт — это данные), алгоритм RL больше похож на малыша, накапливающего знания методом проб и ошибок, получая за них одобрение или порицание.

RL не только обеспечивает компактный способ обучения без сбора больших массивов данных, но и включает в себя также сложное взаимодействие с внешним миром. Поскольку RL начинается без каких-либо данных, взаимодействие с внешним миром и получение обратной связи определяет метод, используемый для получения требуемых данных. Вы можете использовать этот подход для робота, перемещающегося в физическом мире, или для бота, блуждающего в цифровом мире. В частности, RL кажется заманчивым для задач, которые

нелегко решить, используя только статические (предоставленные) данные. Примерами таких задач являются обучение компьютера самостоятельной игре или выработка наилучшего возможного результата в неопределенных ситуациях, таких как оптимизация рекламы в Интернете. Реклама является одним из лучших примеров, поскольку приложение должно предоставлять правильные рекламные кампании нужной аудитории, но предыдущий опыт отсутствует (для статических или существующих данных), поскольку все кампании новые.

Знакомство с обучением с подкреплением

В RL есть *агент* (agent) (который может быть роботом в реальном мире или ботом в цифровом), взаимодействующий со средой, способной включать в себя реальный или виртуальный мир с его собственными правилами. Агент может получать информацию из среды (*состояние* (state)) и может воздействовать на нее, иногда изменяя ее. Однако важней всего то, что агент может получать информацию из среды, положительную или отрицательную, в зависимости от последовательности действий или бездействия. Вход является *вознаграждением* (reward), даже если оно отрицательно. Цель RL — заставить агента научиться вести себя так, чтобы максимизировать общую сумму вознаграждений, полученных в ходе приобретения опыта в среде.

Вы можете определить отношения между агентом и средой на рис. 17.1. Обратите внимание на пометки времени. Если рассматривать текущий момент времени как t, предыдущий момент будет равен $t - 1$. В момент времени $t - 1$ агент действует, а затем получает от среды и состояние, и вознаграждение. На основе наборов значений, относящихся к действию в момент времени t, состояния в момент времени $t - 1$ и вознаграждения в момент времени t, алгоритм RL может выучить действие, необходимое для получения определенного состояния среды.

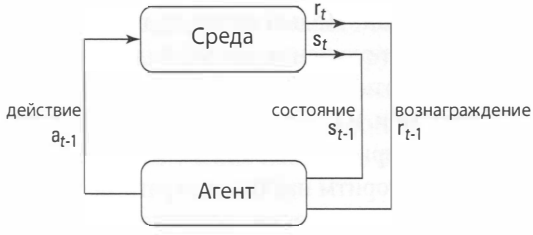


Рис. 17.1. Схема связи агента и среды в RL

Ян Гудфеллоу, исследователь искусственного интеллекта, стоявший за созданием GAN, считает, что лучшая интеграция между RL и глубоким обучением является одним из главных приоритетов для дальнейших достижений в

области глубокого обучения. Лучшая интеграция ведет к более умным роботам (см. подробнее на https://www.forbes.com/sites/quora/2017/07/21/whats-next-for-deep-learning/#36131b871002). Интеграция сейчас является горячей темой, но до недавнего времени RL, как правило, имели более прочные связи со статистикой и алгоритмами, чем нейронные сети. Некоторые люди пытались заставить их работать вместе и раньше. В начале 1990-х годов Джеральд Тезауро (Gerald Tesauro) из исследовательского центра IBM разработал способ, позволяющий компьютеру научиться играть в нарды (одну из старейших известных настольных игр: http://www.bkgm.com/rules.html) и победить чемпиона мира (человека). Он успешно использовал нейронную сеть для снабжения алгоритма RL, создав компьютерную программу TD-Gammon. Программа вызвала широкий интерес к применению нейронных сетей для решения проблем RL, поэтому многие люди после Тезауро пытались показать какое-то другое возможное использование этой техники, но все они потерпели неудачу, и идея умерла.

Позже некоторые исследователи заметили, что нарды — это игра, частично основанная на случайности. Другие игры (такие как шахматы или го) и задачи реального мира, которые плохо реагируют на комбинацию глубокого обучения и RL, не зависят от удачи. Отсутствие компонента случайности лишь частично объясняет проблему с хорошей работой глубокого обучения с некоторыми играми (например, покер — это азартная игра, но она некоторое время был вне досягаемости RL и глубокого обучения). Несмотря на это понимание (то есть глубокое обучение лучше работает с неопределенностью), ученые все еще не могли найти решение, которое позволило бы нейронным сетям поддерживать RL для новых задач, пока через несколько лет команда исследователей глубокого обучения Google не доказала обратное.

В Google DeepMind взяли хорошо известную технику RL, называемую Q-learning, и заставили ее работать с глубоким обучением, а не с классическим алгоритмом вычислений. Новый вариант, получивший название Deep Q-Learning, использует для получения входных данных и их обработки как свертки, так и обычные плотные слои. Это решение не только снова объединило глубокое обучение и RL, но и дало сверхчеловеческие возможности для игры в некоторые игры Atari 2600 (см. https://www.youtube.com/watch?v=V1eYniJ0Rnk). Алгоритм научился играть за относительно короткое время и нашел выигрышные стратегии, которые используют только самые опытные игроки.

СОВЕТ

Команда DeepMind опубликовала также документ *Human-level control through deep reinforcement learning* (https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf).

Несмотря на свою техническую тему, статья вполне читабельна. Она показывает, почему Deep Q-Learning хорошо работает с одними играми и плохо с другими. Проблема возникает, когда нейронной сети необходимо выработать сложные и долгосрочные стратегии.

Имитация игровой среды

Даже если вы не используете предварительно созданные наборы данных при работе с RL (это означает, что вам не нужно собирать и маркировать данные), вы должны учитывать взаимодействие между алгоритмом и внешним миром, что является другой проблемой. Например, если вы хотите создать алгоритм RL, способный победить вас в шахматах, вам сначала нужно создать шахматную компьютерную игру, включающую все правила. Алгоритм будет взаимодействовать с этим набором правил как с частью его ввода.

Чтобы позволить большему количеству исследователей и практиков опробовать эту концепцию, OpenAI (`https://openai.com/`), некоммерческая исследовательская компания по искусственному интеллекту, разработала пакет Gym с открытым исходным кодом. (Вы можете найти этот код по адресу `https://github.com/openai/gym`, а также документ, описывающий решение, по адресу `https://arxiv.org/pdf/1606.01540.pdf`.) Gym — это полный набор инструментов, который поможет всем разработчикам применять алгоритмы RL как к простым, так и к сложным задачам, предлагая готовые к использованию среды.


ЗАПОМНИ!

Пакет Gym от OpenAI позволяет проверить, являются ли ваши алгоритмы общими, поскольку все среды используют один и тот же командный интерфейс. Чтобы проверить свое решение RL в другой ситуации, вы просто меняете имя среды.

В пакете есть также веб-сайт, где вы можете опубликовать свои результаты, сравнивая, как ваш алгоритм RL сочетается с другими решениями. Используя следующие команды, вы легко установите на локальном компьютере пакет `gym` и его необходимые компоненты (пакет `h5py`) из оболочки Anaconda (`pip` подключится к Интернету для получения пакетов и их локальной установки).

```
pip install h5py
pip install gym
conda install -c menpo ffmpeg
```

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

В отличие от других примеров книги, примеры в этой главе не могут быть запущены в Google Colab по техническим причинам — процедуры слишком сложны. Этот код нужно запускать на локальном компьютере.

При использовании Gym, вам больше не нужно беспокоиться о среде. Доступны различные среды, некоторые представляют алгоритмические задачи (такие как обучение копированию последовательности), некоторые текстовые, некоторые связанные с роботами (например, управление рукой робота), и большее количество основанных на старых играх Atari, таких как Space Invaders или Breakout. Все доступные среды можно увидеть на `https://gym.openai.com/envs/#classic_control`. Вы начинаете с классической среды, как описано в научной литературе по RL, но вы также можете изучить и другие возможности, предлагаемые пакетом.

ЗАПОМНИ!

Умение находить решения для игр с использованием RL также поможет вам находить лучшие решения для реальных задач. Инженеры сетевой транспортной компании Uber, изучая алгоритмы RL, размышляют о том, как RL работает, и анализируют, как RL принимает решения для развития доверия и уверенности в искусственном интеллекте, как можно прочитать в блоге Uber по адресу `https://eng.uber.com/atari-zoo-deep-reinforcement-learning/`.

Пакет Gym создан на основе базовых принципов RL, поэтому вы найдете функции и методы для описания агента и среды. Вы также можете запросить агент выполнить действие или бездействовать в среде. Среда ответит, предоставив обратную связь в двух формах: новое состояние, которое вы можете использовать для обобщения новой ситуации в среде; и вознаграждение, которое является показателем успеха или неудачи. Единственная часть, которую вам нужно программировать, это RL, и вы можете запустить простой пример, используя всего несколько строк Python.

Средой для эксперимента RL является задача CartPole (см. подробнее на `https://gym.openai.com/envs/CartPole-v1/`). Столб свободно закреплен на тележке, которая движется по дорожке (трение не учитывается). Маятник запускается в вертикальном положении в неустойчивом равновесии, и цель среды заключается в том, чтобы предотвратить его падение (для которого требуется отклонение от вертикали на угол больше 15 градусов). В качестве действий вы определяете, увеличивать или уменьшать скорость тележки в том или ином направлении.

На рис. 17.2 показано представление среды, предоставляемой пакетом OpenAI Gym. Вы также можете увидеть пример того, как сбалансировать

CartPole в реальном эксперименте инженерного факультета Критского института технологического образования `https://www.youtube.com/watch?v=XWhGjxdug0o`.

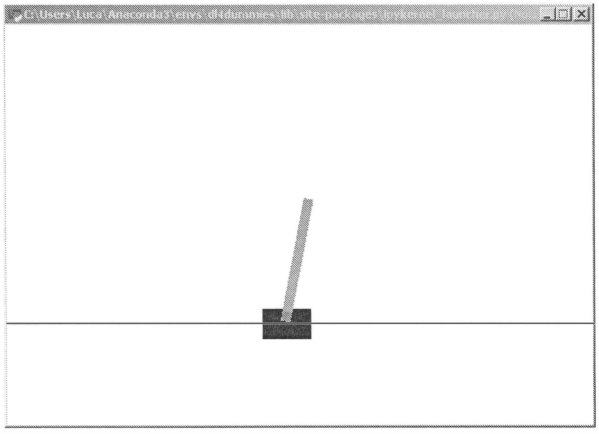


Рис. 17.2. Среда CartPole в OpenAI Gym

Среда CartPole работает, сообщая о наблюдениях следующих состояний.

- Позиция тележки.
- Скорость тележки.
- Угол столба.
- Скорость столба на конце.

Вы можете управлять средой на основе следующих состояний.

- Толкнуть тележку влево.
- Толкнуть тележку вправо.

Следующий код создает среду и проверяет с ней несколько случайных команд.

```
import numpy as np
import gym
env = gym.make('CartPole-v0')
np.random.seed(42), env.seed(42)
nb_actions = env.action_space.n
input_shape = (1, env.observation_space.shape[0])
```

Вы создаете среду с помощью одной команды `make`, которая возвращает класс Python, используемый для получения общей информации о

среде (например, о действиях, которые вы можете выполнить с помощью `env.action_space`), управления потоком времени или выполнения каких-то определенных действий внутри среды.

Следующие несколько строк сбрасывают только что созданную среду. Все перезапускается с начальной позиции (некоторые аспекты среды выбираются случайным образом). В коде используется цикл из 200 итераций для выполнения различных случайных действий, выбранных из диапазона возможных доступных действий (сила, применяемая к тележке в диапазоне от –1 до +1). Когда итерации завершены, игра заканчивается неудачей (когда столб отклоняется больше 15 градусов от вертикали), или тележка перемещается более, чем на 2,4 единицы от центра, переменная `done` приобретает значение `true`, и эксперимент завершается (количество шагов будет меняться, поскольку это процесс случайного выбора).

```
observation = env.reset()
for t in range(200):
    env.render()
    act = env.action_space.sample()
    obs, rwrd, done, info = env.step(act)
    if done:
        print("Episode concluded after %i timesteps" % (t+1))
        break
env.close()
```

Q-обучение

Построение решения RL, основанного на глубоком обучении, требует значительных усилий по написанию кода, но вы можете использовать существующий пакет keras-rl (`https://github.com/keras-rl/keras-rl`), содержащий самые современные алгоритмы RL. Этот пакет, разработанный Матиасом Плаппертом (Matthias Plappert), научным сотрудником OpenAI, может легко интегрироваться с нейронными сетями, созданными с помощью Keras, и сред OpenAI. Вы устанавливаете пакет, выполнив в оболочке следующую команду.

```
pip install keras-rl
```

После установки keras-rl вы импортируете из Keras необходимые функции (т.е. используете нейронную сеть для решения RL) и специализированные функции keras-rl для создания агента RL. (Подробности о том, как они работают, приведены ниже в этой главе.)

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Flatten, Dropout
from keras.optimizers import Adam
from rl.agents.dqn import DQNAgent
```

```
from rl.policy import EpsGreedyQPolicy
from rl.memory import SequentialMemory
```

Первым шагом является создание сети, способной вычислить результат с точки зрения вознаграждения от определенного состояния среды. Это подход *обучения на базе оценок* (value-based learning), и в этом идея Deep Q-Network и Deep Q-Learning: приблизительное определение вознаграждения после выполнения определенного действия с учетом текущего состояния. Этот метод не учитывает непосредственно предыдущие действия и связанное с ними состояние или полную последовательность действий, которые должен выполнить агент, однако он эффективно работает при решении многих задач, выявляя среди альтернатив единственное лучшее действие, которое следует предпринять.

```
model = Sequential()
model.add(Flatten(input_shape=input_shape))
model.add(Dense(12))
model.add(Activation('relu'))
model.add(Dense(nb_actions))
model.add(Activation('linear'))

print(model.summary())
```

Нейронная сеть, которую создает код, проста, она состоит из трех слоев уменьшающегося количества нейронов. Все слои активируются функцией ReLU, но последний слой активируется линейно, чтобы получить выходное значение, используемое как действие выполняемое ботом.

ЗАПОМНИ!

Алгоритм DQN не понимает, как работает среда. В человеческом смысле алгоритм просто связывает состояние и действия с ожидаемыми вознаграждениями, что делается с использованием математической функции. Поэтому алгоритм не может понять, играет ли он в какую-то конкретную игру; его понимание среды ограничено знанием сообщаемого состояния, полученного в результате предпринятых действий.

Эта нейронная сеть питает алгоритм DQN, наравне с политикой (*политика* (policy) — это функция, выбирающая последовательность действий), а также памятью о предыдущих действиях и состояниях. Память необходима, чтобы позволить коду обучить нейронную сеть. Она записывает предыдущий опыт работы агента со средой, и код может осуществлять выборку из нее, чтобы получить серию действий с заданным состоянием. Нейронная сеть использует память, чтобы научиться оценивать вероятное вознаграждение за действие, совершенное в данном состоянии.

Политика Eps Greedy Q выполняет одно из следующих действий.

- Предпринимает случайное действие с вероятностью эпсилон (epsilon).
- Предпринимает лучшее текущее действие с вероятностью (1 – эпсилон).

Две политики показывают *компромисс между исследованием и использованием* (exploration/exploitation trade-off). Когда функция политики Eps Greedy Q выбирает действие случайным образом, алгоритм осуществляет исследование, поскольку он должен принять решение о неожиданном действии, и это действие может привести к интересному результату. Например, в игре Atari Breakthrough пробивание дыры в стене и ускорение мяча, разрушающего стены сверху — это явно стратегия, возникшая случайным образом в результате исследования, которую алгоритм RL зарегистрировал и усвоил как чрезвычайно полезную.

```
policy = EpsGreedyQPolicy(eps=0.3)
memory = SequentialMemory(limit=50000,
                          window_length=1)

dqn = DQNAgent(model=model,
               nb_actions=nb_actions,
               memory=memory,
               nb_steps_warmup=50,
               target_model_update=0.01,
               policy=policy)

dqn.compile(Adam(lr=0.001))

training = dqn.fit(env, nb_steps=30000,
                   visualize=False, verbose=1)
```

Система обучается, используя тот же подход, что и другие сети глубокого обучения. После изучения 30 000 примеров, она готова к проверке.

```
env = gym.make('CartPole-v0')
mon = gym.wrappers.Monitor(env,
                           "./gym-results",
                           force=True)
mon.reset()
dqn.test(mon, nb_episodes=1, visualize=True)
mon.close()
env.close()
```

Проверка должна показать высокое вознаграждение (ожидаемый результат — около 200, но цифра может и отличаться, поскольку тест имеет случайный обучающий элемент). Вы можете просмотреть поведение тележки, используя директивы DQN, находящиеся в видео, записанном во время теста.

```
import io
import base64
from IPython.display import HTML

template =
    './gym-results/openaigym.video.%s.video000001.mp4'
video = io.open(template % mon.file_infix, 'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''
<video width="520" height="auto" alt="test" controls>
<source src="data:video/mp4;base64,{0}"
    type="video/mp4"/>
</video>'''.format(encoded.decode('ascii')))
```

Объяснение Alpha-Go

Шахматы и Го являются популярными настольными играми, имеющие общие характеристики, например, в них играют два игрока, которые ходят по очереди, и нет элемента случайности (как в игре в кости или нардах). Кроме того, у них разные правила игры и сложность. В шахматах у каждого игрока есть 16 фигур для перемещения на доске в соответствии с их типом, и игра завершается, когда фигура короля оказывается в тупике и не может двигаться дальше. Эксперты подсчитали, что возможно около 10^{123} различных вариантов игры, что очень много, если учесть, что ученые оценивают количество атомов в известной вселенной примерно в 10^{80}. Тем не менее, компьютеры способны освоить игру в шахматы, определив будущие возможные ходы достаточно далеко вперед, чтобы иметь преимущество перед любым человеческим соперником. В 1997 году суперкомпьютер Deep Blue от IBM, предназначенный для игры в шахматы, победил Гарри Каспарова, чемпиона мира по шахматам.

ЗАПОМНИ!

Компьютер не может заранее просчитать всю игру в шахматы, используя грубую силу (вычисляя каждый возможный ход от начала до конца игры). Он использует некую эвристику и свою способность анализировать определенное количество следующих ходов. Deep Blue был компьютером с высокой вычислительной производительностью, который мог прогнозировать больше будущих ходов в игре, чем любой предыдущий компьютер.

В игре Го есть сетка линий размером 19×19, содержащая 361 позицию, на которые каждый игрок кладет камень (обычно черного или белого цвета), когда игрок делает ход. Цель игры в том, чтобы захватить камнями большую часть

доски, чем у противника. Учитывая, что в среднем каждый игрок имеет около 250 возможностей на каждом ходу, и что игра состоит из порядка 150 ходов, компьютеру потребуется объем памяти достаточный, чтобы вместить 150^{250} игр, что составляет порядка 10^{360} досок. С точки зрения ресурсов, игра Го значительно сложней шахмат, и эксперты полагали, что ни одно компьютерное программное обеспечение не сможет победить мастера в течение следующего десятилетия, используя тот же подход, что и Deep Blue. Тем не менее, AlphaGo сделал это, используя методы RL.

Исследовательский центр DeepMind в Лондоне, принадлежащий компании Google, разработал компьютерную систему под названием AlphaGo в 2016 году, где использовались способности игры в Го, никогда не достигнутые ранее ни одним аппаратным или программным решением. После настройки системы, DeepMind проверил AlphaGo на самом сильном чемпионе Го в Европе, Фан Ги (Fan Gui), трижды чемпионе Европы. DeepMind провел закрытый матч, и AlphaGo выиграла все игры, оставив Фан Ги пораженным стилем игры, продемонстрированным компьютером.

После того, как Фан Ги помог улучшить навыки AlphaGo, команда Deep Mind, возглавляемая их генеральным директором Демисом Хассабисом (Demis Hassabis) и главным научным сотрудником Дэвидом Сильвером (David Silver), бросила вызов Ли Седолю (Lee Sedol), южнокорейскому профессиональному игроку в Го, имеющему девятый дан — наивысший уровень, которого может достичь мастер. AlphaGo выиграла серию из четырех игр против Ли Седоля и проиграл только одну. Помимо матча, проигранного из-за неожиданного хода чемпиона, она фактически обыграла чемпиона в других играх, проведя неожиданные, эффектные ходы. Фактически, оба игрока, Фан Ги и Ли Седоль, чувствовали, что игра против AlphaGo напоминала игру против соперника, пришедшего из другой реальности: ходы AlphaGo не напоминали ничего, что они видели раньше.


СОВЕТ

История AlphaGo настолько увлекательна, что кто-то сделал из нее фильм. Его стоит посмотреть: `https://www.imdb.com/title/tt6700846/`.

Если вы собираетесь выиграть

В шахматах вы можете просчитывать будущие ходы и далеко опередить обычный компьютер. Количество фигур, их ограниченные ходы и состояние доски — все это облегчает определение того, что может произойти дальше. Кроме того, вы можете получить оценку того, насколько хорошо развивается игра или как может развиваться ее ход из-за характера самой игры (например,

шахматные фигуры имеют смысл). В Го эти определения сделать нельзя, поскольку количество возможных ходов возрастает взрывообразно всего за несколько ходов. Кроме того, вы не можете определить значение хода, поскольку вы должны предвидеть, как завершится игра, прежде чем понять, как каждый ход способствовал итогу игры.

Поскольку основополагающая стратегия Го отличается от шахмат, компьютерные программы, играющие в Го, используют другой подход для определения того, какие ходы делать. Этот подход — *поиск по дереву Монте-Карло* (Monte Carlo Tree Search — MCTS). В MCTS компьютер моделирует множество завершенных игр из существующего состояния доски, используя сначала случайные ходы, а затем самые удачные ходы, которые он находит во время случайной игры. Это не слишком отличается от подхода исследования и использования в RL. Моделируя достаточно игр, чтобы получить надежный ответ, компьютер может определить, хорош ли данный ход в го или нет.

AlphaGo использует MCTS, но поддерживает работу алгоритма с использованием нейронных сетей. Система состоит из двух компонентов.

- **Взгляд на систему будущих ходов**. Метод прогнозирования, аналогичный используемому в Deep Blue. Это древовидная поисковая система, поскольку она перебирает возможные варианты игры и полагается на MCTS.
- **Несколько CNN**. Предоставляет руководство для древовидной системы поиска.

Сети глубокого обучения бывают двух видов: *сети политик* (policy network) и *сети оценок* (value network). Обе сети обрабатывают изображение доски в поисках локальных и общих закономерностей, подобных используемым для обработки изображений при различении собак и кошек. Роли двух сетей политик (одна медленнее, но точнее, другая быстрее, но грубее) в выборе действий. Эти сети политик выводят вероятности для каждого возможного хода, поэтому MCTS может моделировать реалистичные игры на основе их предложений, а не случайным образом. Сеть оценок обеспечивает вероятность выигрыша с учетом состояния на доске.


СОВЕТ

Используя как оценочные сети, обеспечивающие интуитивное восприятие игровой ситуации, так и сети политик, помогающие компьютеру заранее представить будущие ходы, AlphaGo способна выбрать наилучшую стратегию и ходы во время игры.

Учитывая, что такая архитектура на самом деле не является сквозной, поскольку в ней задействовано так много различных систем, для запуска нейронных сетей инженеры Deep Mind сначала научили AlphaGo играть в те игры, в

которые играют сильные любители. (Они использовали 160 000 любительских игр, собранных сетевым сообществом Го.) Наконец, они позволили AlphaGo играть против себя самой, чтобы узнать, как улучшить и усовершенствовать игровые навыки. Здесь ключевую роль сыграла техника RL: они научили компьютеры играть в нарды, шахматы, покер, скрэббл и, наконец, Го, заставляя AlphaGo соперничать с собой миллионы раз, работая в настолько быстрой для накопления опыта среде, которую только смогли создать люди.


ЗАПОМНИ!

Дэвид Сильвер, главный исследователь проекта AlphaGo, заявил, что самообучение столь эффективно при создании интеллектуальных систем потому, что соперник, с которым соревнуются эти системы, всегда на правильном уровне мастерства, он никогда не бывает слишком низким или слишком высоким. Позволить системе учиться, играя сама с собой, — это то, что мы видели в TD-Gammon в 1992 году, а также у компьютера WOPR в фильме *Военные игры* (WarGames) 1983 года. (В этом смысле компьютер WOPR является таким же символом искусственного интеллекта, что и HAL9000 в фильме *Космическая одиссея 2001 года*.)

Применение самообучения в масштабе

Команда DeepMind, создавшая AlphaGo, не остановилась после успеха своего решения и приступила к созданию еще более невероятных систем. Вначале команда создала AlphaGo Zero, которая обучалась на AlphaGo, играя против себя. Затем они создали Alpha Zero, общую программу, способную научиться играть сама с собой в шахматы и сёги (shogi), японские шахматы.

Если AlphaGo продемонстрировала, как решить проблему, которая считалась невозможной для компьютеров, AlphaGo Zero продемонстрировала, что компьютеры могут достичь сверхспособностей с помощью самообучения (что, по сути, является RL). В конце результаты получились даже лучше, чем при начале с человеческого опыта: система AlphaGo Zero бросила вызов AlphaGo и выиграла 100 матчей, не проиграв ни одного.

В статье, опубликованной в журнале *Nature* (и доступной на веб-сайте DeepMind по адресу `https://deepmind.com/research/publications/mastering-game-go-without-human-knowledge/`), объясняется, что AlphaGo Zero начала обучение, делая случайные ходы. Это действие похоже на то, как алгоритм с подкреплением DQN научился балансировать тележку. Приблизительно в 29 миллионах игр, AlphaGo Zero, играя сама с собой, достигла уровня, превышающего предыдущую систему AlphaGo. Более того, AlphaGo Zero является менее сложной с точки зрения моделей глубокого обучения. Для этого

требуется один компьютер и четыре специальных чипа TPU от Google, тогда как для оригинальной системы AlphaGo требовалось несколько машин и 48 чипов TPU.

ВАЖНОСТЬ ALPHA ZERO

Подвиг Alpha Zero куда важнее достижения AlphaGo. В этой книге часто упоминается роль данных в поиске путей развития глубокого обучения. Простая модель с большим количеством данных способна превзойти умный алгоритм, использующий меньше данных. Тем не менее, Alpha Zero удалось достичь вершины производительности, начав с нуля данных. Эта возможность выходит за рамки идеи о том, что данные способны помочь искусственному интеллекту достичь любых целей (как заявили Алон Галеви (Alon Halevy), Питер Норвиг (Peter Norvig) и Фернандо Перейра (Fernando Pereira) всего несколько лет назад в официальном документе на `https://static.googleusercontent.com/media/research.google.com/it//pubs/archive/35179.pdf`). Система Alpha Zero стала возможна потому, что нам известны генеративные процессы, используемые игроками Го, и исследователи DeepMind смогли воссоздать идеальную среду Го.

С точки зрения законов, можно определить гораздо больше ситуаций, чем в Го. Например, ученым известны основные законы того, как устроен физический мир, поскольку люди потратили столетия, исследуя их, и самые яркие умы пытались их понять — от Исаака Ньютона до Альберта Эйнштейна и Стивена Хокинга. Эти знания открывают возможности для создания генеративных моделей, способных копировать и моделировать мыслительные процессы, используемые для создания данных, необходимых для глубокого обучения и моделей искусственного интеллекта. Если этот процесс кажется действительно передовым, обратите внимание: он уже есть. Как можно прочитать в статье `https://www.inverse.com/article/26307-grand-theft-auto-open-ai`, люди уже обсуждают, как видеоигра может помочь в создании более совершенных автомобилей.

Системы AlphaGo, AlphaGo Zero и Alpha Zero представляют новый горизонт RL, а также надежду на будущие приложения. На самом деле, кроме игр в Го, шахматы и сёги, эти системы не способны ни на что другое. Как и Deep Blue, эти системы концентрируются на одной задаче, которую они могут решить на качественно сверхчеловеческом уровне. Исследователи из DeepMind предполагают дальнейшие возможные области применения, которые в настоящее время трудны для человека, например, изучение процессов сворачивания белка, оптимизация потребления энергии в сети или открытие новых материалов в химии.

Великолепные десятки

В ЭТОЙ ЧАСТИ...

- Реальные приложения, использующие глубокое обучение
- Некоторые из лучших инструментов для задач глубокого обучения
- Поиск профессии, опирающейся на глубокое обучение

Глава 18

Десять приложений, требующих глубокого обучения

В ЭТОЙ ГЛАВЕ...

- Прямое взаимодействие с людьми
- Определение эффективности зеленых технологий
- Использование вероятности для предсказания будущего
- Подражание творческим процессам

Эта глава слишком коротка. Она даже не пытается описать, как глубокое обучение повлияет на вас в будущем. Считайте эту главу закуской, способной разжечь ваш аппетит для дальнейшего изучения мира глубокого обучения. Некоторые из описанных в этой главе приложений уже популярны. Сегодня вы, наверное, использовали хотя бы одно из них, и, скорее всего, больше, чем одно. Прочитав эту главу, вы, возможно, захотите уделить время рассмотрению всех способов, которыми глубокое обучение в настоящее время влияет на вашу жизнь. Хотя технология начала широко использоваться, это только начало. Мы находимся только в начале чего-то, и искусственный интеллект на данный момент еще довольно незрелый.

В этой главе не обсуждаются роботы-убийцы, антиутопическое будущее, искусственный интеллект и любые другие сенсационные сценарии, которые

вы можете увидеть в фильмах. В этой главе рассказывается о реальной жизни, и обсуждаются существующие приложения искусственного интеллекта, с которыми вы можете взаимодействовать уже сегодня.

Восстановление цвета на черно-белых видео и изображениях

Возможно, у вас есть черно-белые видео или фотографии членов семьи или неких событий, которые вы хотели бы увидеть в цвете. Цвет состоит из трех элементов: оттенок (фактически цвет), яркость (темный или яркий цвет) и насыщенность (интенсивность цвета). Больше об этих элементах можно прочитать на `http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/`. Как ни странно, многие художники являются дальтониками и активно используют значение цвета в своих творениях (читай `https://www.nytimes.com/2017/12/23/books/a-colorblind-artist-illustrator-childrens-books.html` как один из многих примеров). Таким образом, отсутствие цветового оттенка (элемент, которого нет в черно-белом изображении) — это не конец света. Наоборот, некоторые художники рассматривают это как преимущество (см. подробнее на `https://www.artsy.net/article/artsy-editorial-the-advantages-of-being-a-colorblind-artist`).

При просмотре чего-либо в черно-белом режиме вы видите яркость и насыщенность, но не оттенок цвета. Раскрашивание — это процесс добавления оттенка. Художники обычно осуществляют этот процесс используя тщательный подбор отдельных цветов, как описано на `https://fstoppers.com/video/how-amazing-colorization-black-and-white-photos-are-done-5384` и `https://www.diyphotography.net/know-colors-add-colorizing-black-white-photos/`. Однако искусственный интеллект автоматизировал этот процесс с использованием сверточных нейронных сетей (CNN), как описано на `https://emerj.com/ai-future-outlook/ai-is-colorizing-and-beautifying-the-world/`.

ЗАПОМНИ!

Самый простой способ использовать CNN для раскрашивания — это найти библиотеку, которая поможет вам. Сайт Algorithmia по адресу `https://demos.algorithmia.com/colorize-photos/` предлагает такую библиотеку и показывает пример кода. Вы также можете опробовать приложение, вставив адрес URL в указанное поле. Статья на `https://petapixel.com/2016/07/14/app-magically-turns-bw-photos-color-ones/` описывает, насколько хорошо работает это приложение. Это абсолютно потрясающе!

Аппроксимация позы человека в реальном времени

Позы человека говорят не о том, кто именно находится в видеопотоке, а о том, какие элементы человека находятся в нем. Например, использование позы человека может сказать вам, есть ли локоть человека на видео и где он находится. В статье на `https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5` рассказывается больше о том, как работает вся эта техника визуализации. Фактически, вы можете увидеть работу этой системы на коротком видео одного человека в первом случае и трех человек во втором.

Позы человека могут использоваться для самых разных целей. Например, вы можете использовать позу человека, чтобы помочь людям улучшить свою форму в различных видах спорта — от гольфа до боулинга. Позы человека могут также сделать возможными новые виды видеоигр. Представьте себе возможность отслеживать положение человека для игры без обычного ассортимента громоздкого снаряжения. Теоретически, вы можете использовать позы человека для проведения анализа места преступления или для определения возможности совершения человеком преступления.

Другое интересное применение для определения позы — в медицинских и реабилитационных целях. Программное обеспечение, основанное на глубоком обучении, способно указать вам, правильно ли вы выполняете упражнения, и отслеживать ваши улучшения. Приложение такого рода может улучшить работу профессионального реабилитолога, заботясь о вас, когда вы не находитесь в медицинском учреждении (мероприятие телереабилитации; см. подробнее на `https://matrc.org/telerehabilitation-telepractice`).

ЗАПОМНИ!

К счастью, сегодня вы можете, по крайней мере, начать работу с позами человека, используя библиотеку tfjs-models (PoseNet) по адресу `https://github.com/tensorflow/tfjs-models/tree/master/posenet`. Загрузка примера занимает некоторое время, поэтому вам нужно набраться терпения.

Анализ поведения в реальном времени

Анализ поведения выходит за рамки анализа, описанного в предыдущем разделе. Когда вы проводите анализ поведения, вопрос остается тем же, не кто, а как. Это конкретное приложение искусственного интеллекта влияет на

то, как производители разрабатывают продукты и веб-сайты. Такие статьи, как https://amplitude.com/blog/2016/06/14/10-steps-behavioral-analytics идут на все, чтобы определить и охарактеризовать использование анализа поведения. В большинстве случаев анализ поведения помогает понять, что процесс, которого ожидал разработчик, не соответствует процессу, который вы фактически используете.

Анализ поведения играет важную роль и в других сферах жизни. Например, это может помочь людям медицинских профессий выявить потенциальные проблемы с людьми, имеющими определенные медицинские предпосылки, такие как аутизм, и помочь пациенту преодолеть эти проблемы (см. подробности на https://www.autismspeaks.org/applied-behavior-analysis-aba-0). Анализ поведения способен также помочь учителям физкультуры показать ученикам, как отточить свои навыки. Вы также можете увидеть его применение в юридических профессиях, чтобы помочь выяснить мотив. (Вина очевидна, но то, почему человек что-то делает, важно для справедливого исправления нежелательного поведения.)


СОВЕТ

К счастью, вы уже можете начать анализ поведения с помощью Python. Например, на сайте https://rrighart.github.io/GA/ обсуждается методика (а также предоставлен исходный код) с точки зрения веб-аналитики.

Перевод

Интернет создал среду, способную помешать вам узнать, с кем вы действительно общаетесь, где находится этот человек, а иногда даже когда этот человек общался с вами. Однако одно не изменилось: необходимость переводить с одного языка на другой, когда две стороны не говорят на общем языке. В некоторых случаях неправильный перевод может быть смешон, если у обеих сторон есть чувство юмора. Однако неправильный перевод приводит также к всевозможным серьезным последствиям, включая войну (см. https://unbabel.com/blog/translation-errors-war-iraq-hiroshima-vietnam/). Следовательно, даже, несмотря на то, что программное обеспечение для перевода чрезвычайно доступно в Интернете, важен тщательный выбор используемого продукта. Одним из самых популярных приложений этого типа является Google Translate (https://translate.google.com/), но доступно и много других приложений, таких как DeepL (https://www.deepl.com/en/translator). По мнению Forbes, машинный перевод — это область, в которой искусственный интеллект превосходит другие средства (см. https://www.

forbes.com/sites/bernardmarr/2018/08/24/will-machine-learning-ai-make-human-translators-an-endangered-species/#114274573902).

ЗАПОМНИ!

Приложения для перевода обычно полагаются на двунаправленные рекуррентные нейронные сети (BRNN), как описано на https://blog.statsbot.co/machine-learning-translation-96f0ed8f19e4. Вам не нужно создавать свою собственную сеть BRNN, поскольку вы можете выбирать из множества существующих API. Например, вы можете получить доступ в Python к API Google Translate с помощью библиотеки, расположенной по адресу https://pypi.org/project/googletrans/. Дело в том, что перевод, возможно, является одной из наиболее популярных областей применения глубокого обучения, и его используют многие люди, даже не задумываясь об этом.

Оценка потенциала солнечной энергии

Попытка определить, будет ли солнечная энергия действительно работать в вашем регионе, затруднительна, если ее еще не используют многие другие люди. Еще сложнее узнать, какой уровень экономии вы можете получить. Конечно, вы не хотите устанавливать солнечную батарею, если она не удовлетворит вашей цели по ее использованию, что может фактически не подразумевать долгосрочную экономию средств (хотя, как правило, это так). Некоторые проекты по глубокому обучению способны помочь вам отгадывать догадки о солнечной энергии, включая проект Sunroof, доступный по адресу https://www.google.com/get/sunroof. К счастью, вы также можете получить поддержку для такого прогноза в своем приложении Python по адресу https://github.com/ColasGael/Machine-Learning-for-Solar-Energy-Prediction.

Победа над людьми в компьютерных играх

Конкурс AI-versus-people (искусственный интеллект против людей) продолжает вызывать интерес. От победы в шахматах до победы в Го, искусственный интеллект, похоже, стал непобедим — по крайней мере, непобедим в одной игре. В отличие от людей, искусственный интеллект специализируется, и тот искусственный интеллект, который может выиграть в Го, вряд ли преуспеет в шахматах. Несмотря на это, 2017 год часто называют началом конца людей в играх против искусственного интеллекта, как описано на https://newatlas.

com/ai-2017-beating-humans-games/52741/. Конечно, соревнования продолжались на протяжении довольно длительного времени, и вы, вероятно, можете найти соревнования, в которых искусственный интеллект выигрывал гораздо раньше, чем в 2017 году. Действительно, некоторые источники (https://en.wikipedia.org/wiki/AlphaGo) указывают дату победы в го уже в октябре 2015 года. В статье на https://interestingengineering.com/11-times-ai-beat-humans-at-games-art-law-and-everything-in-between описывается 11 других случаев побед искусственного интеллекта.

Проблема в том, чтобы создать искусственный интеллект, способный выигрывать в конкретную игру, и понять, что, специализируясь на этой игре, искусственный интеллект может не справиться с другими играми. Процесс создания искусственного интеллекта только для одной игры может выглядеть сложным.

СОВЕТ

Тем не менее, высказанное немного ранее утверждение о том, что люди вышли из игры, на самом деле немного преждевременно. В будущем люди могут соревноваться с искусственным интеллектом в нескольких играх. Примеров такого рода соревнований уже предостаточно, где люди выступают в триатлоне игр, состоящем из трех соревнований, а не одного. Теперь это соревнование на гибкость: искусственный интеллект не может просто сесть и выучить еще одну игру, поэтому у человека есть преимущество в гибкости. Этот вид использования искусственного интеллекта демонстрирует, что людям и искусственному интеллекту, возможно, придется сотрудничать в будущем, при этом искусственный интеллект будет специализироваться на конкретных задачах, а человек обеспечит гибкость, необходимую для выполнения всех необходимых задач.

Генерация голоса

Ваша машина уже может говорить с вами; сейчас многие автомобили регулярно общаются с людьми. Как ни странно, созданный голос зачастую настолько хорош, что его трудно отличить от реального. В таких статьях как https://qz.com/1165775/googles-voice-generating-ai-is-now-indistinguishable-from-humans/ рассказывается о том, что поиск компьютерных голосов, звучащих вполне реалистично становится все более распространенной задачей. Сейчас, когда многие справочные службы говорят вам, что вы разговариваете с компьютером, а не с человеком, эта проблема привлекает достаточно много внимания.

СОВЕТ

Хотя ответы на звонки основаны на сценариях ответов, что позволяет генерировать ответы с чрезвычайно высоким уровнем достоверности, выполнить распознавание голоса немного сложнее (но сейчас это значительно легче). Чтобы успешно работать с распознаванием голоса, вам зачастую нужно ограничивать ввод определенными ключевыми словами. Используя ключевые слова, понятные при распознавании голоса, вы избегаете необходимости повторения запроса пользователем. Эта потребность в определенных условиях говорит о том, что вы разговариваете с компьютером — просто спросите что-нибудь неожиданное, и компьютер не будет знать, что с этим делать.

Самый простой способ реализовать собственную голосовую систему — это использовать существующий API, такой как Cloud Speech to Text (`https://cloud.google.com/speech-to-text/`). Конечно, вам может понадобиться что-то, что вы можете настроить. В этом случае использование API окажется полезным. В статье на `https://medium.com/@sundarstyles89/create-your-own-google-assistant-voice-based-assistant-using-python-94b577d724f9` рассказывается, как создать собственное голосовое приложение с использованием Python.

Прогнозирование демографии

Демография — это жизненно важная или социальная статистика, группирующая людей по определенным характеристикам, она всегда была частично искусством и частично наукой. Вы можете найти любое количество статей о том, как заставить ваш компьютер генерировать демографические данные для клиентов (или потенциальных клиентов). Демографические данные широко используются, но вы видите, что они используются для предсказания того, какой продукт будет покупать конкретная группа (по сравнению с конкурентами). Демография является важным средством категоризации людей и последующего прогнозирования некоторых действий с их стороны на основе их групповых ассоциаций. Вот наиболее популярные методы, предлагаемые для искусственного интеллекта при сборе демографических данных.

» **Историческая справка**. Основываясь на предыдущих действиях, искусственный интеллект обобщает, какие действия вы можете выполнить в будущем.

- **Текущая деятельность**. На основании действий, которые вы выполняете сейчас, и, возможно, других характеристик, таких как пол, компьютер прогнозирует ваше следующее действие.
- **Характеристики**. Основываясь на определяющих вас свойствах, таких как пол, возраст и регион проживания, компьютер прогнозирует выбор, который вы, скорее всего, сделаете.

ВНИМАНИЕ!

Вы можете найти статьи о прогнозирующих возможностях искусственного интеллекта, которые кажутся слишком хорошими, чтобы быть правдой. Например, в статье на `https://medium.com/@demografy/artificial-intelligence-can-now-predict-demographic-characteristics-knowing-only-your-name-6749436a6bd3` говорится, что искусственный интеллект теперь может прогнозировать вашу демографию исключительно на основе вашего имени. Компания Demografy в этой статье (`https://demografy.com/`) утверждает, что выявляет пол, возраст и культурную близость, исключительно на основании имени. Несмотря на то, что сайт утверждает, что он на 100% точен, эта статистика крайне маловероятна, поскольку некоторые имена являются неоднозначными по признаку пола, например, Renee, а другим присваивается один пол в одних странах и другой пол в других. Да, демографический прогноз может сработать, но будьте осторожны, прежде чем верить всему, что вам говорят эти сайты.

Если вы хотите поэкспериментировать с демографическим прогнозированием, вы можете найти в Интернете ряд API. Например, интерфейс API DeepAI по адресу `https://deepai.org/machine-learning-model/demographic-recognition` обещает помочь предсказать возраст, пол и культурное происхождение на основе присутствия человека на видео. Каждый сетевой API действительно специализирован, поэтому вам нужно выбрать API, ориентируясь на тип вводимых данных, которые вы можете предоставить.

Создание произведений искусства из реальных фотографий

Глава 15 дает вам некоторое представление о том, как можно использовать глубокое обучение для комбинирования некой реальной картины и существующего стиля мастера. Фактически, некоторые произведения искусства, созданные с использованием этого подхода, были проданы на аукционе довольно

дорого. Вы можете найти множество статей об этом особом поколении искусства, например, статью *Wired* по адресу https://www.wired.com/story/we-made-artificial-intelligence-art-so-can-you/.

Но, несмотря на то, что такие картины хорошо смотрятся на стенах, вы можете захотеть создать и другие виды произведений искусства. Например, вы можете создать трехмерную версию изображения, используя такие продукты, как Smoothie 3-D. В статьях по адресу https://styly.cc/tips/smoothie-3d/ и https://3dprint.com/38467/smoothie-3d-software/ описывается, как работает это программное обеспечение. Это не то же самое, что создание скульптуры; скорее, вы используете 3-D принтер для создания 3-D версии вашего изображения. В статье на https://thenextweb.com/artificial-intelligence/2018/03/08/try-this-ai-experiment-that-converts-2d-images-to-3d/ предлагается эксперимент, в ходе которого вы можете увидеть, как работает этот процесс.

ЗАПОМНИ!

Вывод искусственного интеллекта не обязательно должен состоять из чего-то визуального. Например, глубокое обучение позволяет создавать музыку на основе содержимого изображения, как описано на странице https://www.cnet.com/news/baidu-ai-creates-original-music-by-looking-at-pictures-china-google/. Эта форма искусства делает используемый искусственным интеллектом метод более понятным. Искусственный интеллект преобразует содержимое, которого он не понимает, из одной формы в другую. Как люди, мы видим и понимаем трансформацию, но все, что видит компьютер, — это числа для обработки с использованием умных алгоритмов, созданных другими людьми.

Прогнозирование природных катастроф

Люди пытаются предсказывать стихийные бедствия с тех пор, пока существуют люди и стихийные бедствия. Никто не хочет стать жертвой землетрясения, торнадо, извержения вулкана или любого другого стихийного бедствия. Способность вовремя уйти — это главное соображение в данном случае, учитывая, что люди еще не могут достаточно хорошо контролировать свою среду обитания, чтобы предотвратить любое стихийное бедствие.

Глубокое обучение предоставляет средства для поиска чрезвычайно тонких моделей, поражающих умы людей. Эти шаблоны могут помочь предсказать природную катастрофу, согласно статье о решении Google на http://www.digitaljournal.com/tech-and-science/technology/

google-to-use-ai-to-predict-natural-disasters/article/533026. Тот факт, что программное обеспечение способно предсказать любую катастрофу, просто удивителен. Тем не менее, статья на http://theconversation.com/ai-could-help-us-manage-natural-disasters-but-only-to-an-extent-90777 предупреждает, что полагаться исключительно на такое программное обеспечение было бы ошибкой. Чрезмерное доверие к технологиям является постоянной темой всей этой книги, поэтому не удивляйтесь, что глубокое обучение также не идеально подходит и для предсказания природных катастроф.

Глава 19

Десять инструментов глубокого обучения

В ЭТОЙ ГЛАВЕ...

- Дополнение вашей среды разработки
- Создание специализированных сред
- Решение бизнес-задач
- Доступ к специализированному оборудованию

Глубокое обучение — это сложная задача, и если вы попытаетесь написать каждый нужный вам кусочек кода, у вас уже не будет никакого времени для выполнения любого анализа, который сам по себе занимает значительное время. Следовательно, вам нужны инструменты, которые помогут вам выполнить работу с меньшими усилиями. На протяжении всей книги вы видели множество описанных и используемых инструментов. Однако, за исключением TensorFlow и Keras, описанные ранее инструменты, как правило, являются хорошей отправной точкой или тем, что нужно учитывать для облегчения обучения. Инструменты в этой главе особенные. Они помогают вам решать самые разные задачи с профессиональными результатами.

Компиляция математических выражений с использованием Theano

Theano (`http://deeplearning.net/software/theano/`) — это библиотека Python, позволяющая быстро работать с различными математическими

выражениями. При желании, вы можете заменить копию TensorFlow, установленную вами ранее в главе 4, на Theano. Выбор того, что использовать, может быть довольно сложным, как показано в обсуждениях на https://www.analyticsindiamag.com/tensorflow-vs-theano-researchers-prefer-artificial-intelligence-framework/ и https://www.reddit.com/r/MachineLearning/comments/4ekywt/tensorflow_vs_theano_which_to_learn/. Тем не менее, высокая скорость Theano, кажется, не под вопросом.

Изучив несколько моделей в этой книге, вы уже знаете, что скорость важна, даже очень важна. Например, код главы 12 действительно может нуждаться в ускорении, и библиотеки, подобные данной могут помочь вам в этом (см. обсуждение вопросов скорости в главе 12). Вот основные признаки, делающие библиотеку Theano невероятно быстрой.

- Прозрачное использование GPU.
- Динамическая генерация C-кода.
- Специализированные оптимизации.

ВНИМАНИЕ!

Theano в настоящее время является четвертой наиболее часто используемой инфраструктурой (как описано на https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a), именно поэтому она обсуждается в этой главе. Однако, как указано на https://groups.google.com/forum/m/#!msg/theano-users/7Poq8BZutbY/ rNCIfvAEAwAJ, разработчики Theano больше ничего с этим не делают. Вы можете увидеть последние заметки об обновлении на http://www.deeplearning.net/software/theano/NEWS.html и прочитать реакцию разработчиков на недостатки на https://www.quora.com/Is-Theano-deep-learning-library-dying. Многие разработчики все еще приводят веские аргументы в пользу ее использования, как обсуждалось на https://www.reddit.com/r/MachineLearning/comments/47qh90/is_there_a_case_for_still_using_torch_theano/.

Дополнение TensorFlow с помощью Keras

Глава 4 и некоторые другие главы этой книги описывают использование Keras (https://keras.io/) совместно с TensorFlow. В главе 4 рассказывается, как получить экземпляры этих продуктов и установить их. Многие примеры из книги не обойдутся без Keras , так что вы, возможно, уже видели несколько примеров того, что Keras может сделать для вас.

К счастью, если вы решите пойти по пути Theano, а не TensorFlow, у вас все равно есть возможность использовать Keras вместе с ним. С TensorFlow вы также можете использовать встроенную версию Keras (см. подробнее на `https://www.tensorflow.org/api_docs/python/tf/keras`). Связь между Keras и TensorFlow станет сильнее только после окончательного выпуска TensorFlow 2.0 (см. подробнее на `https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a`).

СОВЕТ

Как ни странно, вы также можете использовать Keras с Microsoft Cognitive Toolkit (CNTK). Keras поддерживает все через бэкэнд, как описано на `https://keras.io/backend/`. Вам просто нужно использовать другой базовый инструментарий, чтобы внести изменения в файл конфигурации. В результате вы можете поэкспериментировать, чтобы убедиться, какой инструментарий лучше всего соответствует вашим потребностям, и ваш код Keras останется прежним. Однако, одно предостережение: вы должны написать свой код, используя абстрактный интерфейс API бэкенда Keras, чтобы он был совместим с несколькими базовыми наборами инструментов. В этой книге не показано, как использовать абстрактный API бэкэнда Keras, поэтому данный метод потребует от вас дополнительного времени на изучение.

Динамическое вычисление графиков с помощью Chainer

Когда-то вы могли использовать такую библиотеку как Pylearn2 (которая построена на TensorFlow; см. подробнее на `http://deeplearning.net/software/pylearn2/`), чтобы преодолеть разрыв между алгоритмами и глубоким обучением. Такие продукты как Chainer (`https://chainer.org/`), вышли на сцену по тем причинам, которые обсуждались на `https://www.quora.com/Which-is-better-for-deep-learning-TensorFlow-or-Chainer`. Упор делается на упрощении доступа к функциям, которые большинство систем могут предоставить сегодня, или доступным через сетевые хосты. Следовательно, вы можете обратиться к Chainer для предоставления следующих функций.

- Поддержка CUDA для доступа к GPU.
- Поддержка нескольких GPU без особых усилий.
- Поддержка отдельных пакетов.

- Поддержка различных сетей, включая сети прямой связи, CNN, рекуррентные сети и рекурсивные сети.
- Контроль операторов потока в прямом вычислении без обратного распространения ошибок.
- Значительная функциональность отладки для облегчения поиска ошибок.

Создание среды, похожей на MATLAB, с помощью Torch

Чтобы получить оптимальную производительность от решений глубокого обучения, вам нужна поддержка GPU, и именно здесь Torch (`http://torch.ch/`) вступает в игру. Он ставит GPU на первое место при разработке решений, что позволяет получить дополнительные ядра и оптимизированные функции обработки, способные предоставить графические процессоры. Чтобы обеспечить максимальную скорость, Torch использует компилятор LuaJIT (`http://luajit.org/`) для компиляции приложения, вместо его интерпретации. (Интерпретаторы могут замедлять работу приложений.) Он также имеет базовый язык C и реализацию CUDA (`https://www.geforce.com/hardware/technology/cuda`), превращающую ваш код высокого уровня в код низкого уровня, для ускорения работы.

Torch поставляется с некоторыми функциями, похожими на таковые у NumPy, но с упором на глубокое обучение. (Документацию к пакету можно найти по адресу `http://torch.ch/docs/package-docs.html`.) Например, можно найти следующее.

- N-мерные массивы.
- Особенности манипуляций с матрицей.
- Процедуры линейной алгебры.

В дополнение к этим вы найдете некоторые функциям, специально предназначенные для нужд искусственного интеллекта, включая глубокое обучение.

- Модели нейронных сетей.
- Энергетические модели.
- Процедуры числовой оптимизации.
- Быстрая и надежная поддержка GPU.

Динамическое выполнение задач с помощью PyTorch

PyTorch (https://pytorch.org/) является серьезным конкурентом для TensorFlow. Одним из пунктов на главной странице, который, вероятно, привлечет ваше внимание, является то, что вы можете щелкать на различных параметрах, чтобы отобразить необходимые инструкции по установке для вашей инфраструктуры, используя технику, которую вы действительно хотите использовать. На самом деле, из всех продуктов, которые вы найдете в Интернете, этот может быть самым простым в установке. Простота установки распространяется и на другие аспекты этого продукта, такие как отладка, как описано на https://medium.com/@NirantK/the-silent-rise-of-pytorch-ecosystem-693e74b33f1e. Обратите внимание, что в этой статье описаны также некоторые недостающие элементы и способы их исправления.

Вы используете PyTorch так же, как и TensorFlow и Keras, но есть различия, о которых вам нужно знать, как описано на https://hub.packtpub.com/what-is-pytorch-and-how-does-it-work/. Эти отличия неплохи, и вполне можно утверждать, что они способствуют быстрому росту PyTorch (см. подробнее на https://venturebeat.com/2018/10/16/github-facebooks-pytorch-and-microsofts-azure-have-the-fastest-growing-open-source-projects/). Многие разработчики связывают PyTorch с другими продуктами, такими как Fastai, который описан по адресу https://twimlai.com/twiml-talk-186-the-fastai-v1-deep-learning-framework-with-jeremy-howard/.

Ускорение исследований в области глубокого обучения с использованием CUDA

Вы можете найти CUDA (https://developer.nvidia.com/how-to-cuda-python) в различных формах для разных языков и различных потребностей. Например, версия для C/C ++ находится по адресу https://developer.nvidia.com/cuda-math-library. В этом разделе рассматривается предложение для Python, но существуют и другие версии, и их функции отличаются от версии, обсуждаемой здесь. Независимо от формы, CUDA предполагает использование графических процессоров, особенно графических процессоров на устройствах NVIDIA, таких как Titan V (https://www.nvidia.com/en-us/titan/titan-v/).

На самом деле, вам не обязательно нужен графический процессор в системе для использования CUDA. Вместо этого вы можете получить доступ к графическим процессорам на любом из множества сайтов, включая Amazon AWS, Microsoft Azure и IBM SoftLayer. Фактически экземпляр поставляется с поддерживаемым NVIDIA CUDA Amazon Machine Image (AMI) на AWS, поэтому вам даже не придется прилагать много усилий, чтобы получить доступ к этой поддержке.

ВНИМАНИЕ!

CUDA придает вам большую гибкость в использовании различных источников GPU. В большинстве случаев гибкость обеспечивается ценой более сложного обучения и дополнительного кода, поскольку всего не предусмотришь. Следовательно, прежде чем устанавливать этот пакет, обязательно прочитайте сообщение в блоге по адресу `https://devblogs.nvidia.com/numba-python-cuda-acceleration/`, где более подробно рассказывается о том, как использовать CUDA при выполнении реальных задач. Но после того, как вы пройдете обучение, вы обнаружите, что можете выполнять невероятный набор задач, которые иначе вы не сможете выполнить.

При работе с CUDA многие разработчики связывают его с библиотекой CUDA Deep Neural Network (cuDNN) (`https://developer.nvidia.com/cudnn`). Это специальная библиотека оптимизированных подпрограмм, которые поддерживают использование CUDA для нужд глубокого обучения.

ЭТИКА ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Об этике и искусственном интеллекте можно легко написать целую книгу, поскольку эта технология имеет невероятный потенциал для неправильного использования. Например, в недавней статье обсуждалось использование алгоритмов предотвращения самоубийств при мониторинге пользователей Facebook. Фактически, Facebook регулярно использует алгоритмы для мониторинга использующих сервис людей, и считает, что делать это без разрешения вполне приемлемо. Генеральный директор Марк Цукерберг (Mark Zuckerberg) считает, что конфиденциальность мертва, и каждый должен привыкнуть к этому (см. подробнее на `http://www.nbcnews.com/id/34825225/ns/technology_and_science-tech_and_gadgets/t/privacy-dead-facebook-get-over-it/#.XFh14FVKhpg`).

Всплеск продаж таких книг, как *1984* Джорджа Оруэлла наблюдался (см. `https://www.nytimes.com/2017/01/25/books/1984-george-orwell-donald-trump.html`) отчасти из-за чувства незащищенности личной информации у людей. Согласно статье на `https://www.recode.`

net/2018/2/12/16998750/facebooks-teen-users-decline-instagram-snap-emarketer, тенденция Facebook постоянно держать вашу информацию в открытом доступе — это одна из причин потери пользователей. Все эти статьи выражают понимание того, что людям известно, что компания широко использует подобные технологии, и не довольны этим.

Проблема возникает, когда люди не знают, что делает компания. Быть в курсе всех технологических изменений сегодня невозможно, поскольку технологии развиваются очень быстро. В этом случае сотрудники организации должны сообщать о своих проблемах. Так произошло, когда сотрудники Amazon обратились к Джеффу Безосу (Jeff Bezos) с просьбой о продаже Rekognition (https://aws.amazon.com/rekognition/) правоохранительным органам, которые будет использовать его для массового наблюдения и распознавания лиц (https://www.pcmag.com/commentary/366229/the-ai-industrys-year-of-ethical-reckoning).

После Второй мировой войны общество становилось все более сложным, но и более хрупким. Чтобы защитить людей от опасностей, возникающих внутри и за пределами государства, военные и правоохранительные органы иногда используют эти новые технологии для наблюдения, контроля и воздействия. Если ученые не будут жаловаться и регулировать использование новых технологий для таких целей, их широкое и неизбирательное использование может подорвать права людей и даже создать тоталитарное государство, подобное тому, которое было в книге *1984*. Только этичное поведение ученых, которые знают об использовании их технологий, поможет смягчить это явно неэтичное поведение.

Поддержка бизнес-потребностей с Deeplearning4j

Когда дело касается данных, бизнес выглядит скучным, поскольку он выполняет одни и те же повторяющиеся задачи с разными параметрами. Однако использовать нейронную сеть для удовлетворения потребностей бизнеса в данных довольно сложно, поскольку для одной модели нейронной сети задачи различаются слишком сильно, чтобы соответствовать всем ситуациям. Deeplearning4J (https://deeplearning4j.org/) позволяет комбинировать различные мелкие сети (слои) для создания глубокой нейронной сети. Такой подход значительно сокращает время, необходимое для обучения глубокой нейронной сети, а время, как правило, у компаний крайне ограничено.

ЗАПОМНИ!

Это конкретное решение написано на Java и будет работать с любым JVM-совместимым языком, включая Scala, Clojure или Kotlin. Базовые вычисления написаны на языках C и CUDA, поэтому вы также можете использовать это решение с данными языками, если все, что вы хотите сделать, — это получить доступ к базовым вычислениям. Чтобы использовать это решение с Python, вам нужно запустить его на Keras. В примере на `https://www.javacodegeeks.com/2018/11/deep-learning-apache-kafka-keras.html` показано, что необходимо для создания решения в этой среде. Прежде, чем сделать решающий шаг, удостоверьтесь, что уделили достаточно времени его изучению.

Получение данных с использованием Neural Designer

Многие из перечисленных в этой главе продуктов не совсем закончены; создается такое впечатление, что это эксперименты исследователей. Однако некоторым людям просто нужно решение, которое работает. Neural Designer (`https://www.neuraldesigner.com/`) — это именно такое решение, и оно выполняет множество задач, в том числе.

- Обнаружение сложных отношений.
- Распознавание неизвестных шаблонов.
- Прогнозирование тенденций.
- Распознавание ассоциаций по данным.

В отличие от многих других решений, которые вы найдете, Neural Designer уделяет также особое внимание конкретным отраслям. Вы можете найти конкретную информацию для следующего.

- Банковское дело и страхование (`https://www.neuraldesigner.com/solutions/solutions-banking-insurance`).
- Проектирование и производство (`https://www.neuraldesigner.com/solutions/solutions-engineering-manufacturing`).
- Розничная торговля и потребительский спрос (`https://www.neuraldesigner.com/solutions/solutions-retail`).
- Здравоохранение (`https://www.neuraldesigner.com/solutions/solutions-health`).

Алгоритмы обучения с использованием Microsoft Cognitive Toolkit (CNTK)

Microsoft Cognitive Toolkit (CNTK) (`https://www.microsoft.com/en-us/cognitive-toolkit/`) — это еще одна базовая среда, используемая для глубокого обучения, во многом похожая на TensorFlow и Theano. Вы можете запустить Keras на любой из трех.

Помимо сравнения скорости трех сред и других проблем с производительностью, вам также необходимо учесть их функции. Очевидно, что все три будут поддерживать Keras — обычно с некоторыми модификациями. Тем не менее, каждая из трех систем обладает также особыми функциональными возможностями. Например, если вы хотите использовать Azure, CNTK, вероятно, будет вашим лучшим решением, поскольку ученые Microsoft наиболее знакомы с текущими и будущими функциями Azure. Конечно, вы ожидаете такого рода возможностей от CNTK.

СОВЕТ

Одна из приятных особенностей CNTK — обширная галерея моделей на `https://www.microsoft.com/en-us/cognitive-toolkit/features/model-gallery/`. Вы найдете примеры на нескольких языках, причем некоторые примеры относятся только к одному языку, а другие поддерживают несколько языков. Посмотрите внимательно на эту страницу, и вы увидите, что она включает в себя модели для языков C++, C# и .NET, что вам может быть трудно найти у других систем.

Полное использование возможностей графического процессора с помощью MXNet

MXNet (`https://mxnet.apache.org/`) имеет некоторые интересные функции, с которыми можно поэкспериментировать на данном этапе, но продукт, вероятно, еще не готов к работе, поскольку сайт сообщает, что он все еще разрабатывается. Этот продукт предоставляет несколько удивительных моделей, которые значительно сократят время, необходимое для создания многих приложений глубокого обучения.

ЗАПОМНИ!

Для работы с MXNet вы полагаетесь на Gluon (теоретически вы также можете использовать API модуля, но на данный момент это выглядит немного сложно). Gluon — это обязательный интерфейс,

описанный по адресу https://beta.mxnet.io/guide/crash-course/index.html (еще раз обратите внимание, что это бета-сайт, а не завершенный). Первое, что вы замечаете, проходя ускоренный курс, это то, что работа с Gluon действительно выглядит легкой. Чтобы использовать Gluon с Python, вы должны прочитать о пакете Python по адресу https://mxnet.incubator.apache.org/api/python/gluon/gluon.html. Информация на https://beta.mxnet.io/ поможет вам получить достаточно хорошие навыки, хотя и с некоторым трудом.

К счастью, документация MXNet для Gluon великолепна (https://mxnet.apache.org/api/python/gluon/model_zoo.html), и вы можете найти дополнительные ресурсы на Medium (https://medium.com/apache-mxnet). Наиболее впечатляющим является огромное количество моделей, которые этот продукт уже поддерживает. Кроме того, вы можете найти значительное количество примеров, облегчающих ваше обучение (https://github.com/apache/incubator-mxnet/tree/master/example), а также учебные пособия. В целом, это продукт заслуживает внимания, поскольку он имеет значительный потенциал для облегчения вашей работы.

Глава 20

Десять профессий, использующих глубокое обучение

В ЭТОЙ ГЛАВЕ...

- Работа с людьми
- Разработка новых технологий
- Анализ данных
- Проведение исследований

Эта книга охватывает множество различных способов применения глубокого обучения — от голосовых функций вашего цифрового помощника до беспилотных автомобилей. Конечно, хорошо использовать глубокое обучение для улучшения своей повседневной жизни, но большинству людей нужны другие причины для использования этой технологии, например, поиска работы. К счастью, глубокое обучение влияет не только на вашу способность находить информацию быстрее, но также предлагает некоторые действительно интересные возможности для работы, которые способно обеспечить только глубокое обучение. В этой главе представлен обзор десяти интересных профессий, которые в некоторой степени полагаются сегодня на глубокое обучение. Этот материал представляет собой только верхушку айсберга; гораздо больше профессий, чем может уместиться в этой книге, уже используют глубокое обучение, и каждый день добавляются новые.

Управление людьми

Триллер *Сфера* (The Circle) (https://www.amazon.com/exec/obidos/ASIN/B071GB3P5N/datacservip0f-20/) заставит вас поверить, что современные технологии будут еще более агрессивными, чем "Большой брат" в книге *1984* Джорджа Оруэлла. Частью сюжета фильма является установка камер повсюду, даже в спальнях. Главный герой просыпается каждое утро, чтобы приветствовать всех, кто смотрит на него. Да, это может случиться, если вы позволите.

Тем не менее, по-настоящему глубокое обучение не сводится к мониторингу и оценке людей, по большей части. Это больше похоже на облако глобального управления персоналом от Oracle (Global Human Resources Cloud) (https://cloud.oracle.com/en_US/global-human-resources-cloud). Отнюдь не страшно, эта конкретная технология позволит вам выглядеть умнее и быть в курсе всех мероприятий дня, как показано в видео на https://www.youtube.com/watch?v=NMm_cIHeEZ0&list=PL2Gxt-CBX-Ep2n5ytNGkl3bRUnUKAMI1Z. Видео немного избыточное, но оно дает вам хорошее представление о том, насколько глубокое обучение в настоящее время способно облегчить вашу работу.


ЗАПОМНИ!

Идея этой технологии заключается в облегчении достижения успеха людьми. Если вы посмотрите видео Oracle и связанные с ним материалы, то обнаружите, что эта технология помогает руководству предлагать потенциальные пути к целям сотрудников в организации. В некоторых случаях сотрудникам нравится их текущая ситуация, но программное обеспечение все еще может предложить способы сделать их работу более интересной и увлекательной. Программное обеспечение предотвращает текучесть кадров в системе и помогает управлять персоналом на индивидуальном уровне, чтобы к каждому сотруднику был индивидуальный подход.

Улучшение медицинского обслуживания

Глубокое обучение серьезно влияет на медицинскую практику, как вы можете заметить, когда обращаетесь к врачу или проводите время в больнице. Глубокое обучение помогает диагностировать болезни (https://www.cio.com/article/3305951/health-care-industry/the-promise-of-artificial-intelligence-in-diagnosing-illness.html) и находить правильное лечение (https://emerj.com/ai-sector-overviews/

machine-learning-medical-diagnostics-4-current-applications/). Глубокое обучение используется для улучшения процесса диагностики даже трудно обнаруживаемых проблем, в том числе проблем со зрением (https://www.theverge.com/2018/8/13/17670156/deepmind-ai-eye-disease-doctor-moorfields). Тем не менее, одной из наиболее важных областей применения для глубокого обучения в медицине является исследование.

Казалось бы, простой акт подбора правильных пациентов для использования в исследованиях на самом деле не так прост. Пациенты должны соответствовать строгим критериям, иначе результаты тестирования могут оказаться недействительными. Теперь исследователи полагаются на глубокое обучение для выполнения таких задач, как поиск подходящего пациента (https://emerj.com/ai-sector-overviews/ai-machine-learning-clinical-trials-examining-x-current-applications/), разработка критериев тестирования и оптимизация результатов. Очевидно, что медицине понадобится много людей, которые обучены как медицине, так и использованию методов глубокого обучения в медицине, чтобы продолжать достигать успехов в их нынешнем темпе.

Разработка новых устройств

Инновации в некоторых областях компьютерных технологий, таких как базовые системы, которые сейчас является товаром, с годами замедлились. Однако инновации в областях появившихся только недавно, значительно возросли. Сегодня изобретатель имеет больше возможностей для новых устройств, чем когда-либо прежде. Одна из таких новых областей — это средства для выполнения задач глубокого обучения (https://www.oreilly.com/ideas/specialized-hardware-for-deep-learning-will-unleash-innovation). Чтобы создать потенциал для решения более сложных задач глубокого обучения, многие организации теперь используют специализированное аппаратное обеспечение, превосходящее возможности графических процессоров (сегодняшнюю технологию обеспечения глубокого обучения).

В этой книге много говорится о различных технологиях глубокого обучения, но технология находится в зачаточном состоянии, поэтому сообразительный изобретатель может придумать что-то интересное, даже не работая по-настоящему в поте лица. В статье на https://blog.adext.com/en/artificial-intelligence-technologies-2019 рассказывается о новых технологиях искусственного интеллекта, но даже эти технологии не начинают раскрывать глубины того, что может произойти.

СОВЕТ

Глубокое обучение привлекает внимание, как изобретателей, так и инвесторов из-за его способности обходить действующее патентное законодательство и способы, которыми люди создают новые вещи (https://marketbrief.edweek.org/marketplace-k-12/artificial-intelligence-attracting-investors-inventors-academic-researchers-worldwide/). Интересная часть большинства статей такого рода в том, что они прогнозируют значительное увеличение количества рабочих мест, связанных с различными видами глубокого обучения, большинство из которых связаны с созданием чего-то нового. По сути, если вы можете каким-то образом использовать глубокое обучение и связать его с текущей активной профессией, вы можете найти работу или развить собственный бизнес.

Обеспечение поддержки клиентов

Многие обсуждения в этой книге касаются чат-ботов (см. главы 1, 2, 11 и 14) и других форм поддержки клиентов, включая услуги переводчиков. Если интересно, вы можете пообщаться с чат-ботом по адресу https://pandorabots.com/mitsuku/. Однако использование чат-ботов и других технологий поддержки клиентов вызывает беспокойство.

Некоторые группы потребителей говорят, что служба поддержки пользователей обречена, как в статье на https://www.forbes.com/sites/christopherelliott/2018/08/27/chatbots-are-killing-customer-service-heres-why/. Тем не менее, если вам когда-либо приходилось иметь дело с чат-ботом при выполнении чего-либо сложного, вы знаете, что этот опыт менее чем привлекателен. Таким образом, новая парадигма — это комбинация человека и чат-бота, как описано на сайте https://chatbotsmagazine.com/bot-human-hybrid-the-new-era-of-customer-support-346e1633e910.

ЗАПОМНИ!

Многие современные технологии, которые якобы заменяют человека, в большинстве случаев этого не делают. В настоящее время ожидается возникновение множества ситуаций, когда люди и боты работают вместе как команда. Бот уменьшает нагрузку при выполнении физически сложных, скучных и повседневных задач. Человек будет выполнять более интересные вещи, и предоставлять творческие решения неожиданных ситуаций. Следовательно, люди должны пройти обучение, необходимое для работы в этих областях, и чувствовать уверенность в том, что они будут продолжать сохранять оплачиваемую работу.

Просмотр данных новыми способами

Взглянув на ряд веб-сайтов и других источников данных можно заметить одну вещь: все они представляют данные по-разному. Компьютер не понимает различий в представлениях и не зависит от того, как они выглядят. На самом деле он не понимает данные; он ищет шаблоны. Глубокое обучение позволяет приложениям самостоятельно собирать больше данных, гарантируя, что приложение сможет увидеть соответствующие шаблоны, даже если эти шаблоны отличаются от того, что приложение видело ранее (см. подробнее на `https://www.kdnuggets.com/2018/09/data-capture-deep-learning-way.html`). Несмотря на то, что глубокое обучение улучшит и ускорит сбор данных, интерпретировать их придется все равно человеку. Фактически, людям все еще нужно убедиться, что приложение собирает подходящие данные, поскольку в данных приложение действительно ничего не понимает.

Еще один способ увидеть данные по-новому — выполнить приращение данных (`https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced`). Опять же, основную работу выполняет приложение, но требуется человек, чтобы определить, какое приращение предоставить. Другими словами, человек выполняет интересную, творческую часть, а приложение просто трудится, гарантируя, что все сработает.

ЗАПОМНИ!

Эти две первые области применения глубокого обучения интересны, и они будут продолжать создавать рабочие места, но наиболее интересное применение глубокого обучения — это для действий, которых еще не существуют. Творческий человек может смотреть на то, как другие используют глубокое обучение, и придумывать что-то новое. В статье `https://www.wordstream.com/blog/ws/2017/07/28/machine-learning-applications` описываются некоторые интересные способы использования искусственного интеллекта, машинного обучения и глубокого обучения, которые только сейчас становятся практическими.

Ускорение выполнения анализа

Когда большинство людей говорит об анализе, они подразумевают исследователя, ученого или специалиста. Тем не менее, глубокое обучение укореняется в таких интересных местах, требующих участия человека, как

прогнозирование дорожно-транспортных происшествий: https://www.hindawi.com/journals/jat/2018/3869106/.

Представьте, что полицейский департамент распределяет ресурсы на основе схем движения транспорта, так что офицер уже ждет на месте ожидаемой аварии. Лейтенанту полиции необходимо знать, как использовать приложение такого рода. Конечно, конкретно это приложение еще не распространено, но такое вполне вероятно, поскольку оно уже возможно даже с использованием существующих технологий. Таким образом, выполнять анализ будут не только люди с приставкой "Доктор" перед именем, но и все остальные.

Сам по себе анализ не так уж и полезен. Полезным становится объединение анализа с конкретной потребностью в конкретной среде. То, что вы делаете с анализом, определяет влияние этого анализа на вас и окружающих. Человек может понять концепцию анализа с некоторой целью, а решение глубокого обучения способно только выполнить анализ и дать результат.

Создание лучшей рабочей среды

В этой книге рассказывается о том, как работает глубокое обучение, но на самом деле все это означает, что глубокое обучение сделает вашу жизнь лучше, а вашу работу приятней, если у вас появятся навыки, позволяющие вам успешно взаимодействовать с искусственным интеллектом. В статье на https://www.siliconrepublic.com/careers/future-ai-workplaceoffice рассказывается, как искусственный интеллект может изменить рабочее место в будущем. Важный элемент этого обсуждения — сделать работу более привлекательной.

В какой-то момент истории человечества работа была на самом деле приятной для большинства людей. Не то чтобы они все время бегали, пели и смеялись, но многие люди с нетерпением ждали начала каждого дня. Позже, во время промышленной революции, другие люди использовали этот труд для выполнения работ, делая единственным удовольствием каждый день без работы, чем наслаждались лишь некоторые люди. Проблема стала настолько серьезной, что вы можете найти популярные песни по этой теме о ней, такие как *Working for the Weekend*. Удаляя грязь с рабочего места, глубокое обучение может снова сделать работу приятной.


СОВЕТ

Глубокое обучение серьезно влияет на рабочую среду разными способами, а не только на фактическое выполнение работы. Например, технологии, основанные на глубоком обучении, способны улучшить ваше здоровье (https://www.entrepreneur.com/

article/317047), а, следовательно, вашу производительность. Это победа для всех, поскольку вы будете наслаждаться жизнью и работать больше, в то время как ваш босс получит больше отдачи от ваших усилий.

Одна из вещей, о которых вы не часто задумываетесь, — это влияние на падение рождаемости в развитых странах. Статья на https://www.mckinsey.com/featured-insights/future-of-work/ai-automation-and-the-future-of-work-ten-things-to-solve-for посвящена, в некоторой степени, этой проблеме и предоставляет диаграмму, показывающую потенциальное влияние глубокого обучения на различные отрасли. Если текущая тенденция сохранится, то из-за уменьшения количества доступных работников потребуется интенсификация на рабочем месте.

Тем не менее, вы можете задуматься о своем будущем, если беспокоитесь, что не сможете приспособиться к новой реальности. Проблема в том, что вы не знаете, действительно ли вы в безопасности. В книге *Искусственный интеллект для чайников* Джона Пола Мюллера и Луки Массарона можно встретить обсуждения профессий, недоступных для искусственного интеллекта, а также новых профессий, которые искусственный интеллект создаст. Вы даже можете обнаружить, что в какой-то момент вы сможете работать в космосе. К сожалению, не все пойдут на такие действия, как луддиты во время промышленной революции (см. подробнее на https://www.history.com/news/industrial-revolution-luddites-workers). Конечно, обещания искусственного интеллекта будут иметь последствия, даже бо́льшие, чем промышленная революция и будут еще более разрушительными. Некоторые политики, такие как Эндрю Вонг (Andrew Wang) (https://www.yang2020.com/policies/), уже ищут краткосрочные решения, такие как базовый универсальный доход. Эти решения, если они будут приняты, помогут уменьшить влияние искусственного интеллекта, но это не долгосрочные решения. В какой-то момент времени общество станет существенно отличаться от нынешнего, в результате искусственный интеллект изменит общество так же, как и промышленная революция.

Исследование неясной или подробной информации

Одну вещь компьютеры могут делать исключительно хорошо — это сопоставление с шаблоном (причем намного лучше, чем люди). Если у вас когда-либо было чувство, что вы плаваете в информации, и ничего из этого не связано с вашими текущими потребностями, вы не одиноки. Информационная

перегрузка была проблемой в течение многих лет и ухудшается с каждым годом. Вы можете найти множество советов по борьбе с информационной перегрузкой, например, на сайте https://www.interaction-design.org/literature/article/information-overload-why-it-matters-and-how-to-combat-it. Проблема в том, что вы все еще тонете в информации. Глубокое обучение позволяет вам найти иголку в стоге сена, причем за разумное количество времени. Вместо месяцев поиска, хорошее решение глубокого обучения может найти нужную информацию, как правило, за считанные часы.

Однако, знания того, что информация существует, обычно недостаточно. Вам нужна информация, достаточно подробная, чтобы полностью ответить на ваш вопрос, что зачастую означает поиск нескольких источников и консолидацию информации. Опять же, решение глубокого обучения поможет найти шаблоны и объединить данные за вас, чтобы вам не приходилось комбинировать входные данные из нескольких источников вручную.

ЗАПОМНИ!

После того, как искусственный интеллект найдет данные и объединит несколько источников в один сплоченный отчет (вы надеетесь), он сделает все возможное за вас. Человек должен понять информацию и определить способ ее успешного применения. Компьютер не устранит творческую часть задачи; он устранит тяжелую работу по поиску ресурсов, необходимых для выполнения творческой части задачи. Поскольку количество информации продолжает увеличиваться, ожидается и увеличение количества людей, специализирующихся на поиске подробной или малоизвестной информации.

Информационный брокер становится важной частью общества, что представляет интересный карьерный путь, о котором многие люди даже не слышали. Статья на https://www1.cfnc.org/Plan/For_A_Career/Career_Profile/Career_Profile.aspx?id=edMrqnSJebpXYIKXsDcurwXAP3DPAXXAP3DPAX предлагает хорошее резюме о том, что делают информационные брокеры.

Проектирование зданий

Большинство людей рассматривают архитектуру как творческую профессию. Представьте себе проект следующего Эмпайр-стейт-билдинг или другого здания, которое выдержит испытание временем. В прошлом проектирование такого здания занимало годы. Как ни странно, подрядчик построил Эмпайр-стейт-билдинг фактически всего чуть больше, чем за год (см. подробнее на http://www.designbookmag.com/empirestatebuilding.htm), но обычно это не так. Глубокое обучение и компьютерные технологии способны

помочь значительно сократить время на проектирование и строительство зданий, включая такие вещи, как виртуальные пошаговые руководства (https://pdf.wondershare.com/real-estate/virtual-tour-software-for-real-estate.html). Фактически, использование глубокого обучения значительно улучшает жизнь архитекторов, как указано на https://www.autodesk.com/redshift/machine-learning-in-architecture/.

Тем не менее, превращение дизайна в виртуальный тур не является даже самым впечатляющим подвигом глубокого обучения в этой области. Использование глубокого обучения позволяет дизайнерам находить потенциальные инженерные проблемы, проводить стресс-тестирование и обеспечивать безопасность другими способами, прежде чем проект покинет чертежную доску. Эти возможности сводят к минимуму количество проблем, возникающих после того, как здание начинает функционировать, и архитектор может наслаждаться лаврами успеха, а не тревожиться о потенциальной трагедии при обрушении.

Повышение безопасности

Аварии случаются! Однако глубокое обучение может помочь предотвратить несчастные случаи, по крайней мере, по большей части. Анализируя сложные модели в реальном времени, глубокое обучение может помочь людям, вовлеченным в различные аспекты обеспечения безопасности. Например, благодаря отслеживанию различных схем движения и заблаговременному прогнозированию потенциальной возможности аварии, решение глубокого обучения способно предоставить экспертам по безопасности советы по предотвращению аварий вообще. Человек не может выполнить такой анализ из-за слишком большого количества переменных. Тем не менее, решение глубокого обучения способно выполнить такой анализ, а затем предоставить результаты человеку для потенциальной реализации.


ЗАПОМНИ!

Как и в любом другом занятии, подразумевающем использование глубокого обучения, человек действует как его часть, понимающая решения. Разнообразие видов несчастных случаев превосходит способности любого решения глубокого обучения предоставлять каждый раз точные решения. Люди не предсказуемы, но другие люди могут уменьшить вероятность того, что случится нечто ужасное, если получить правильную информацию. Решение глубокого обучения предоставляет правильную информацию, но для ее интерпретации необходимо человеческое понимание и интуиция.

Глибоке навчання надає засоби для виявлення шаблонів в даних, що лежать в основі онлайнових операцій, медицині, дослідженнях, соціальних мережах і в багатьох елементах у повсякденному житті. У цій книзі надана вся інформація, необхідна для усунення загадковості цієї теми, а також описані всі основні технології, пов'язані з нею. Незабаром ви зможете розібратися в досить заплутаних алгоритмах і знайдете просте та безпечне середовище для експериментів з глибоким навчанням.

Науково-популярне видання

Мюллер, Джон Пол, **Массарон**, Лука

Глибоке навчання для чайників

(Рос. мовою)

Зав. редакцією *С. М. Тригуб*

Із загальних питань звертайтеся до видавництва "Діалектика" за адресою:
info@dialektika.com, http://www.dialektika.com

Підписано до друку 16.09.2020. Формат 60x90/16
Ум. друк. арк. 25,0. Обл.-вид. арк. 20,1
Зам. № 20-2402

Видавець ТОВ "Комп'ютерне видавництво "Діалектика"
03164, м. Київ, вул. Генерала Наумова, буд. 23-Б.
Свідоцтво суб'єкта видавничої справи ДК № 6758 від 16.05.2019.

Надруковано ТОВ "АЛЬФА ГРАФІК "
03067, м. Київ, вул. Машинобудівна, 42
Свідоцтво суб'єкта видавничої справи ДК № 6838 від 09.07.2019.

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ ДЛЯ ЧАЙНИКОВ

Джон Пол Мюллер
Лука Массарон

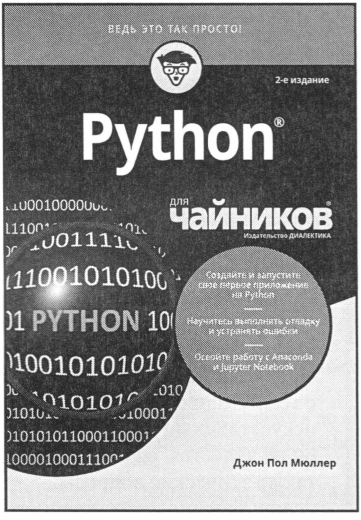
www.dialektika.com

Искусственный интеллект является захватывающим и немного жутковатым. Он вокруг нас. Искусственный интеллект помогает защитить мошенничества, контролировать расписание медицинских процедур, он способен работать в клиентской службе и даже помогает вам в выборе телешоу и приборке вашего дома. Хотите узнать больше? Эта книга восполняет пробелы, знакомя вас с тем, что представляет собой искусственный интеллект и чем он не является, рассматриваются также этические вопросы использования искусственного интеллекта, его современное применение и некоторые из удивительных вещей, на которые он, вероятно, будет способен завтра. Будь вы технофилом или просто любопытны, вы будете очарованы тем, что узнаете!

ISBN 978-5-907114-57-9 **в продаже**

PYTHON ДЛЯ ЧАЙНИКОВ
2-Е ИЗДАНИЕ

Джон Пол Мюллер



www.dialektika.com

Python — это мощный язык программирования, на котором можно создавать самые разные приложения, не зависящие от платформы. Он идеально подходит для новичков, особенно если нужно быстро научиться программировать и начать создавать реальные проекты. Благодаря пошаговым инструкциям, приведенным в книге, вы сможете в краткие сроки освоить основы языка. Работая в среде Jupyter Notebook, вы будете применять принципы грамотного программирования для создания смешанного представления кода, заметок, математических уравнений и графиков.

Основные темы книги:

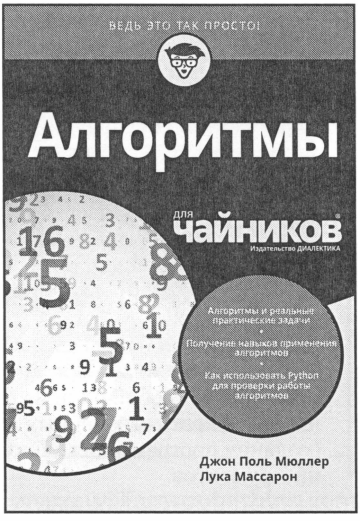
- загрузка и установка Python;
- использование командной строки;
- знакомство со средой Jupyter Notebook;
- основы программирования на Python;
- создание коллекций и списков;
- взаимодействие с пакетами;
- поиск и устранение ошибок.

ISBN: 978-5-907144-26-2

в продаже

АЛГОРИТМЫ ДЛЯ ЧАЙНИКОВ

Джон Поль Мюллер
Лука Массарон



www.dialektika.com

Эта книга — действительно книга для чайников, поскольку основная ее задача не научить программировать реализации тех или иных давно известных алгоритмов, а познакомить вас с тем, что же такое алгоритмы, как они влияют на нашу повседневную жизнь, и каково состояние дел в этой области человеческих знаний сегодня.
В книге рассматривается крайне широкий спектр вопросов, связанных с алгоритмами — это и стандартные сортировка и поиск, и работа с графами (но с уклоном не в стандартные базовые алгоритмы, а в приложении их к таким явлениям сегодняшнего дня, как, например, социальные сети), работа с большими данными и вопросы искусственного интеллекта.
При этом материал книги — не просто отвлеченный рассказ о том или ином аспекте современных алгоритмов, но и демонстрация реализаций алгоритмов с конкретными примерами на языке программирования Python.
Книга будет полезна всем, кто интересуется современным состоянием дел в области программирования и алгоритмов.

ISBN 978-5-9909446-2-6

в продаже

C#ДЛЯ ЧАЙНИКОВ

Джон Пол Мюллер

www.dialektika.com

Даже если вы никогда не имели дела с программированием, эта книга поможет вам освоить язык C# и научиться писать на нем программы любой сложности. Для читателей, которые уже знакомы с каким-либо языком программирования, процесс изучения C# только упростится, но иметь опыт программирования для чтения книги совершенно необязательно.

Из этой книги вы узнаете не только о типах, конструкциях и операторах языка C#, но и о ключевых концепциях объектно-ориентированного программирования, реализованных в этом языке, который в настоящее время представляет собой один из наиболее приспособленных для создания программ для Windows инструментов.

Если вы в начале большого пути в программирование, смелее покупайте эту книгу: она послужит вам отличным путеводителем, который облегчит ваши первые шаги на этом длинном, но очень увлекательном пути.

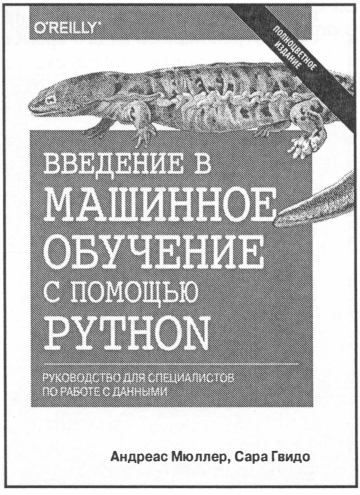
ISBN 978-5-907144-43-9

в продаже

ВВЕДЕНИЕ В МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ PYTHON

РУКОВОДСТВО ДЛЯ СПЕЦИАЛИСТОВ ПО РАБОТЕ С ДАННЫМИ

Андреас Мюллер
Сара Гвидо



www.williamspublishing.com

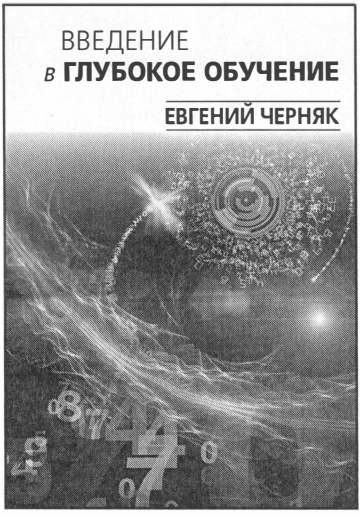
Эта полноцветная книга — отличный источник информации для каждого, кто собирается использовать машинное обучение на практике. Ныне машинное обучение стало неотъемлемой частью различных коммерческих и исследовательских проектов, и не следует думать, что эта область — прерогатива исключительно крупных компаний с мощными командами аналитиков. Эта книга научит вас практическим способам построения систем МО, даже если вы еще новичок в этой области. В ней подробно объясняются все этапы, необходимые для создания успешного проекта машинного обучения, с использованием языка Python и библиотек scikit-learn, NumPy и matplotlib. Авторы сосредоточили свое внимание исключительно на практических аспектах применения алгоритмов машинного обучения, оставив за рамками книги их математическое обоснование. Данная книга адресована специалистам, решающим реальные задачи, а поскольку область применения методов МО практически безгранична, прочитав эту книгу, вы сможете собственными силами построить действующую систему машинного обучения в любой научной или коммерческой сфере.

ISBN 978-5-9908910-8-1

в продаже

ВВЕДЕНИЕ В ГЛУБОКОЕ ОБУЧЕНИЕ

Евгений Черняк



www.dialektika.com

Автор книги — давний исследователь искусственного интеллекта, специализирующийся на обработке естественного языка, революцию в котором сделало глубокое обучение. К сожалению, ему потребовалось много времени, чтобы это понять. Можно сказать, что нейронные сети угрожают революцией уже третий раз, а отнюдь не первый. Тем не менее автор внезапно оказался далеко позади и изо всех сил пытался наверстать упущенное. Именно поэтому он сделал то, что сделал бы на его месте любой уважающий себя профессор: запланировал преподавание материала и начал ускоренный курс, просматривая веб-страницы. Этим объясняется несколько выдающихся особенностей этой книги. Во-первых, краткость. Во-вторых, она сильно зависит от проекта. Автор считает, что материал по информатике лучше изучать при написании программ, поэтому книга во многом отражает его привычки в преподавании. Эта книга, в первую очередь, задумана как учебник для курса по глубокому обучению. Курс, который автор преподает в Брауне, предназначен как для выпускников, так и для остальных студентов, и охватывает весь материал.

ISBN 978-5-907203-10-5

в продаже

BESTSELLING
BOOK
SERIES

Глубокое обучение для чайников®

СЕРИЯ
КОМПЬЮТЕРНЫХ
КНИГ ОТ
ДИАЛЕКТИКИ

Шпаргалка

Глубокое обучение влияет на все сферы вашей жизни, от использования смартфона до врачебной диагностики. Python — это невероятный язык программирования, который вы можете использовать для выполнения задач глубокого обучения с минимальными усилиями. Комбинируя огромное количество доступных библиотек Python с совместимыми с ним средами, вы можете избежать написания низкоуровневого кода, обычно необходимого для создания приложений глубокого обучения. Все, на чем вам нужно сосредоточиться — это выполнение работы. Эта шпаргалка содержит наиболее популярные советы, которые помогут вам легко и быстро освоить программирование.

Наиболее популярные инфраструктуры глубокого обучения

Для решения задач глубокого обучения с помощью языка Python без разработки большого количества кода, который уже был разработан кем-то другим, вам нужна инфраструктура глубокого обучения. Инфраструктура — это абстракция, которая создает среду, в которой может выполняться ваш код. В отличие от библиотеки, которую вы импортируете в свое приложение для предоставления сервисов, управляемых вашим приложением, инфраструктура содержит ваше приложение и предоставляет общую среду, которую вы используете для создания и развертывания приложений, работающих предсказуемым и надежным образом.

Шпаргалка

Следующий список (упорядоченный по популярности) расскажет вам о наиболее распространенных инфраструктурах глубокого обучения, совместимых с языком Python. В этой статье различные инфраструктуры ранжируются по ряду показателей, в том числе популярности.

- **TensorFlow.** В настоящее время — это самая популярная инфраструктура. Она поддерживает как язык C++, так и Python. Вы можете создавать разумные приложения, используя ее как продукт с открытым исходным кодом, но она предлагает также и платные возможности для разработчиков, которым нужно нечто большее. Два из самых больших преимуществ TensorFlow в том, что она проста в установке, и с ней относительно легко работать. Тем не менее, некоторые люди утверждают, что она может быть медленной в работе.
- **Keras.** Это не столько инфраструктура, сколько API. Вы также можете рассматривать ее как интегрированную среду разработки (IDE), но, как правило, она применяется совместно с другими инфраструктурами глубокого обучения. Чтобы использовать Keras, вы также должны иметь систему глубокого обучения, такую как TensorFlow, Theano, MXNet или CNTK. На самом деле Keras поставляется в комплекте с TensorFlow, что также делает ее простым решением для снижения сложности работы с TensorFlow. Связь между Keras и TensorFlow будет только укрепляться, когда, наконец, будет выпущена инфраструктура TensorFlow 2.0. К счастью, если вы решите выбрать Theano вместо TensorFlow, у вас все еще будет возможность использовать Keras вместе с ней. Keras поддерживает языки базовой инфраструктуры.
- **PyTorch.** Написанная на языке Lua, инфраструктура PyTorch — это ответвление инфраструктуры Chainer (показана далее в этом списке). Изначально инфраструктуру PyTorch разработала компания Facebook, но сегодня ее используют многие другие организации, включая Twitter, Salesforce и Оксфордский университет. PyTorch чрезвычайно удобна для пользователя, она эффективно использует память, относительно быстра, и обычно используется для исследований. Эта инфраструктура поддерживает только язык Python.
- **Theano.** Эта инфраструктура совершенствуется не очень активно, что потенциально является большой проблемой. Тем не менее, отсутствие обновлений не мешает разработчикам использовать Theano, поскольку она обеспечивает обширную поддержку для числовых задач. Кроме того, разработчики считают, что у Theano лучшая поддержка GPU, чем в среднем у других. Эта инфраструктура поддерживает только язык Python.

Шпаргалка

- **MXNet.** Основная причина использования MXNet — это скорость. Определить, что быстрее, MXNet или CNTK (обсуждается далее в этом списке), может быть довольно сложно, но оба продукта довольно быстрые и часто используются теми, кого не устраивает медлительность TensorFlow. Особенностями MXNet являются расширенная поддержка графических процессоров, способность работать на любом устройстве, обеспечение высокопроизводительных императивных API, простое обслуживание моделей и высокая масштабируемость. Эта инфраструктура поддерживает множество языков программирования, включая C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl и Wolfram Language.
- **Microsoft Computational Network TookKit (CNTK).** CNTK — это продукт с открытым исходным кодом, требующий от вас изучения нового языка, Network Description Language (NDL), поэтому он имеет некоторую сложность в изучении. Он поддерживает разработку на языках C++, C#, Java и Python, поэтому он обеспечивает большую гибкость, чем многие решения. Кроме того, считается, что он обеспечивает значительную расширяемость, поэтому вы можете изменить работу инфраструктуры с большей легкостью.
- **Caffe2.** Возможно, вы захотите взглянуть на этот продукт, если у вас есть острая потребность в общих задачах глубокого обучения при отсутствии хороших навыков разработчика. Одна из причин, по которой людям действительно нравится Caffe2, заключается в том, что вы можете обучать и развертывать модель, не создавая никакого кода. Вместо этого вы выбираете одну из заранее написанных моделей и добавляете ее в файл конфигурации (который удивительно похож на код JSON). На самом деле, большой выбор предварительно обученных моделей является частью Model Zoo, на который вы можете положиться во многих нуждах. Этот продукт поддерживает языки C++ и Python непосредственно. Теоретически вы можете расширить его, используя Protobuf, но, согласно обсуждению на GitHub, такое расширение рискованно.
- **Chainer.** Эта инфраструктура обеспечивает простоту доступа к функциям, которые сегодня способно предоставить большинство инфраструктур или сетевых хостов. Следовательно, вы можете обратиться к Chainer для доступа к таким функциям как: поддержка CUDA для доступа к графическому процессору; поддержка нескольких GPU без особых усилий; поддержка различных сетей; поддержка разных архитектур; контроль операторов потока в прямом вычислении без потери обратного распространения; а также передовые функции отладки, упрощающие поиск ошибок. Многие разработчики используют эту инфраструктуру для замены таких библиотек как Pylearn2, которые построены на базе TensorFlow, для преодоления разрыва между алгоритмами и глубоким обучением. Эта инфраструктура поддерживает только язык Python.

...FOR Dummies ™
BESTSELLING BOOK SERIES

Глубокое обучение для чайников®

...ДЛЯ чайников ™
СЕРИЯ КОМПЬЮТЕРНЫХ КНИГ ОТ ДИАЛЕКТИКИ

Шпаргалка

Типы слоев сверточных нейронных сетей

Идею сверточных нейронных сетей (CNN) выдвинул французский ученый Ян Лекун совместно с другими известными учеными в конце 1980-х годов, и полностью развили ее технологию в 1990-х годах. Тем не менее, только сейчас, примерно через 25 лет, такие сети начали давать удивительные результаты, достигая в отдельных задачах распознавания даже лучшей производительности, чем люди. Изменения произошли потому, что такие сети могут быть собраны в сложные архитектуры, способные улучшить их обучение на основе множества полезных данных.

Большая часть успеха этой технологии, особенно в приложениях искусственного интеллекта, обусловлена наличием подходящих данных для обучения и тестирования сетей изображений, их применения к различным задачам, благодаря обучению с помощью переноса, и дальнейшему совершенствованию технологии, позволяющему ей отвечать на сложные вопросы о содержании изображения. Технология опирается на следующие слои.

- **Сверточный слой.** Сверточный слой выполняет своего рода обработку сигналов для входных данных. Идея в том, чтобы искать определенные признаки изображения. Результатом этого слоя является карта признаков или карта активации. Сверточный слой имеет следующие гиперпараметры.
  - **Размер фильтра.** Размер окна, используемого для взаимодействия с изображением.
  - **Шаг.** Количество пикселей для смещения при скольжении окна во время обработки больших изображений.
- **Слой подвыборки.** Упрощает вывод, полученный от сверточных слоев, сокращая тем самым количество выполняемых последовательных операций и обеспечивая использование меньшего количества сверточных операций для выполнения фильтрации. Обычно этот слой находится после сверточных слоев. Работая так же, как свертки (используя размер окна для фильтра и шаг, для перемещения по нему), слои подвыборки работают с фрагментами получаемого входного сигнала и уменьшают фрагменты до единого числа. Это эффективно сокращает объем данных, проходящих через нейронную сеть. Слои подвыборки имеют следующие версии, в зависимости от размерности входных данных.
  - Одномерная подвыборка. Работает с векторами. Таким образом, одномерная подвыборка идеальна для данных последовательности, таких как временные данные (данные, представляющие события, следующие друг за другом во времени) или текст (представленные в виде последовательностей букв или слов). Возвращает максимальное или среднее значение смежных частей последовательности.

Шпаргалка

- **Двумерная подвыборка.** Подходит для пространственных данных, соответствующих матрице. Вы можете использовать двумерную подвыборку для изображения в градациях серого или для каждого канала изображения RBG по отдельности. Возвращает максимальное или среднее значение небольших пятен (квадратов) данных.
- **Трехмерная подвыборка.** Подходит для пространственных данных, представленных в виде пространственно-временных данных. Вы можете использовать трехмерную подвыборку для изображений, снятых во времени. Типичным примером является использование магнитно-резонансной томографии (МРТ) при медицинском обследовании. Этот тип подвыборки возвращает максимальное или среднее значение небольших кусков (кубов) данных.

■ **Полносвязный слой.** Работает с плоским входом, где все входы связаны с каждым из нейронов. Этот слой обычно находится в конце архитектуры, и вы используете его для оптимизации. Например, вы можете использовать его для получения оценок классов для ввода, чтобы определить информацию о классе изображения.

Типы рекуррентных нейронных сетей

Реальность не просто изменчива, но изменчива прогрессивным способом, который может быть предсказуем из наблюдения прошлого. Если изображение представляет собой статический моментальный снимок, видео, состоящее из последовательности связанных изображений, является потоком информации, и фильм может рассказать гораздо больше, чем отдельная фотография или серия фотографий. Аналогично для коротких и длинных текстовых данных (от твитов до целых документов и книг) и для всех числовых рядов, представляющих нечто происходящее на временной шкале (например, серию данных о продажах продукта или качестве воздуха за каждый день в городе).

Рекуррентные нейронные сети (RNN) позволяют интерпретировать эти изменяющиеся данные для выполнения таких задач как распознавание речевого ввода и его преобразование в команды. RNN стоят за самыми удивительными приложениями глубокого обучения, с которыми вы можете экспериментировать сегодня. Вы часто видите их на своем мобильном телефоне или дома. Например, вы используете подобное приложение, когда общаетесь с такими умными ораторами как Siri, Google Home или Alexa.

Шпаргалка

В следующей таблице перечислены типы приложений RNN, используемые сегодня:

Тип RNN	*Пример использования*	*Описание*
Один к одному	Традиционная нейронная сеть	Когда у вас есть один вход и ожидается один выход. Традиционная нейронная сеть использует именно этот подход. Она получает один случай (входные данные), состоящий из определенного количества информативных переменных, и дает оценку, такую как число (возможно, представляющее класс) или вероятность
Один ко многим	Создание музыки	Здесь есть один вход, и вы ожидаете последовательность выходов в результате. Автоматический подход к нейронным сетям таков: вы вводите одно изображение и создаете фразу, описывающую содержание изображения
Многие к одному	Классификация настроений	Классический пример RNN. Например, вы вводите текстовую последовательность (например, описание) и ожидаете один результат (изображение, соответствующее этому описанию). Этот подход используется для получения оценки при анализе настроений или другой классификации текста
Многие ко многим	Распознавание имен и машинный перевод	В качестве ввода вы предоставляете последовательность и ожидаете результирующую последовательность в качестве вывода. Это базовая архитектура для многих наиболее впечатляющих приложений искусственного интеллекта с глубоким обучением. Этот подход используется для машинного перевода (сеть, способная автоматически переводить фразу с английского на немецкий), чат-ботов (нейронная сеть, способная отвечать на ваши вопросы и беседовать с вами) и маркировки последовательности (классификация каждого изображения в видео)

Погрузитесь глубже в глубокое обучение!

Глубокое обучение предоставляет средства для выявления шаблонов в данных, лежащих в основе онлайновых операций, медицине, исследованиях, социальных сетях и во многих элементах повседневной жизни. В этой книге предоставлена вся информация, необходимая для устранения загадочности этой темы, а также описаны все основные технологии, связанные с ней. Вскоре вы сможете разобраться в весьма запутанных алгоритмах и найдете простую и безопасную среду для экспериментов с глубоким обучением.

В книге...

- Создание примеров с использованием TensorFlow и Keras
- Практические занятия, упрощающие обучение
- Правильные инструменты, позволяющие применять глубокое обучение эффективней
- Смысл сложных алгоритмов

Джон Пол Мюллер — автор более 100 книг, включая *Искусственный интеллект для чайников, Python и наука о данных для чайников, Machine Learning for Dummies* и *Алгоритмы для чайников.*

Лука Массарон — аналитик данных, интерпретирующий большие данные и превращающий их в интеллектуальные данные с помощью самых простых и эффективных методов интеллектуального анализа и машинного обучения. Является экспертом Google Developer Expert (GDE) в области машинного обучения.

Комп'ютерне видавництво "ДІАЛЕКТИКА"
www.dialektika.com

для **чайников**®

Глубокое обучение

ISBN 978-617-7874-07-1