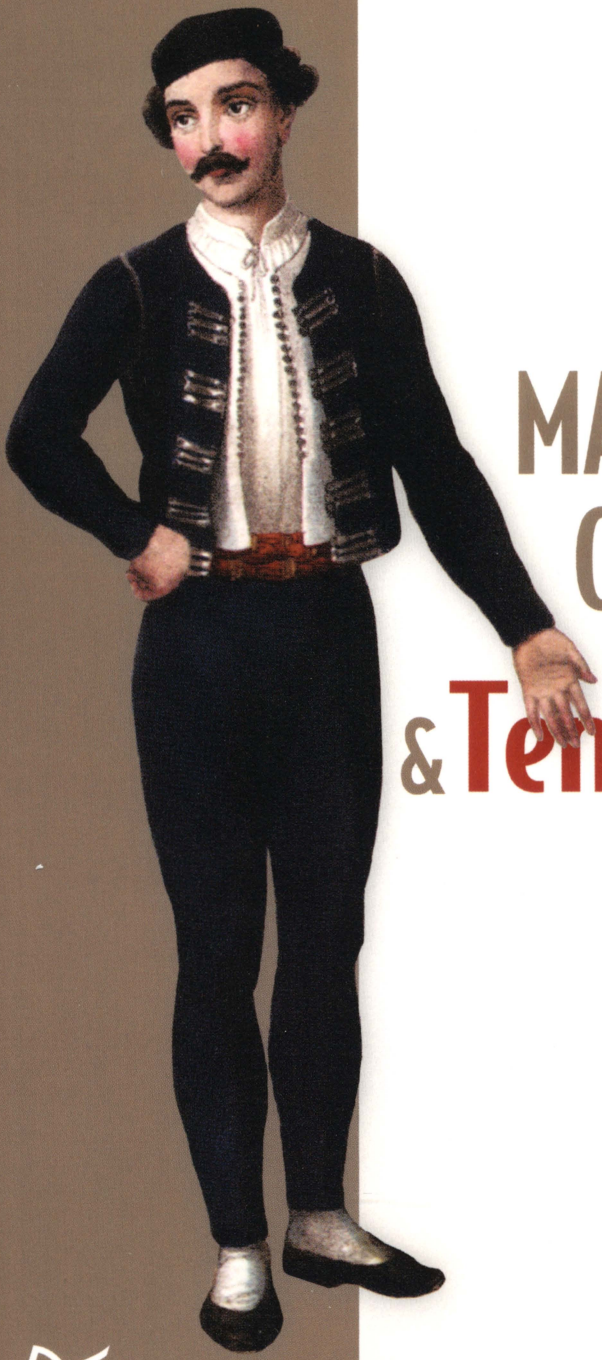


МАШИННОЕ ОБУЧЕНИЕ & TensorFlow

Нишант Шакла

MANNING

ПИТЕР®

Machine Learning with TensorFlow

NISHANT SHUKLA
WITH KENNETH FRICKLAS

MANNING
SHELTER ISLAND

Нишант Шакла
при участии Кена Фрикласа

МАШИННОЕ ОБУЧЕНИЕ & TensorFlow

ПИТЕР
Санкт-Петербург · Москва · Екатеринбург · Воронеж
Нижний Новгород · Ростов-на-Дону · Самара · Минск

2019

ББК 21.818
УДК 004.85
Ш17

Шакла Нишант

Ш17 Машинное обучение и TensorFlow. — СПб.: Питер, 2019. — 336 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-0826-8

Знакомство с машинным обучением и библиотекой TensorFlow похоже на первые уроки в автошколе, когда вы мучаетесь с параллельной парковкой, пытаетесь переключить передачу в нужный момент и не перепутать зеркала, лихорадочно вспоминая последовательность действий, в то время как ваша нога нервно подрагивает на педали газа. Это сложное, но необходимое упражнение. Так и в машинном обучении: прежде чем использовать современные системы распознавания лиц или алгоритмы прогнозирования на фондовом рынке, вам придется разобраться с соответствующим инструментарием и набором инструкций, чтобы затем без проблем создавать собственные системы.

Новички в машинном обучении оценят прикладную направленность этой книги, ведь ее цель — познакомить с основами, чтобы затем быстро приступить к решению реальных задач. От обзора концепций машинного обучения и принципов работы с TensorFlow вы перейдете к базовым алгоритмам, изучите нейронные сети и сможете самостоятельно решать задачи классификации, кластеризации, регрессии и прогнозирования.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 21.818
УДК 004.85

ISBN 978-1617293870 англ.
ISBN 978-5-4461-0826-8

Й

Оглавление

Й

Благодарности

Я хотел бы выразить глубокую признательность своей семье — Суман (маме), Умешу (папе) и Наташе (моей сестре) — за поддержку при написании этой книги. Их радость и чувство гордости всегда вдохновляли меня.

Моральную поддержку на протяжении долгих месяцев написания книги оказывали мне мои друзья по колледжу, мои Крутейшие Крутыши-Диджеи: Алекс Кац (Alex Katz), Аниш Симхал (Anish Simhal), Ясдев Сингх (Jasdev Singh), Джон Гиллен (John Gillen), Джонатан Блончек (Jonathon Blonchek), Кельвин Грин (Kelvin Green), Шив Синха (Shiv Sinha) и Винай Дандекар (Vinay Dandekar).

Выражаю свою благодарность Барбаре Блюменталь (Barbara Blumenthal), моему лучшему другу и даже больше чем другу, за связывание воедино галактик, туманностей и великих противоречий своими розовыми ленточками. Ты была моим спасением, исцеляя от творческого кризиса.

Я хотел бы выразить признательность за потрясающую обратную связь, которую получил от интернет-сообществ: внимание читателей к моим постам в Reddit (r/artificial, r/machinelearning, r/Python, r/Tensor-Flow, а также r/Programming) и Hacker News принесло невероятные плоды. Я благодарен тем, кто оставлял сообщения на официальном форуме книги и вносил свой вклад в репозиторий GitHub. Кроме того, благодарю удивительную группу рецензентов, возглавляемую Александром Драгосавлиевичем (Aleksandar Dragosavljević).

В нее также входят Нии Аттох-Окине (Nii Attoh-Okine), Томас Баллинджер (Thomas Ballinger), Джон Берриман (John Berryman), Гил Бирауд (Gil Biraud), Микаэль Дотрей (Mikaël Dautrey), Хамиш Диксон (Hamish Dickson), Мигель Эдуардо (Miguel Eduardo), Питер Хэмптон (Peter Hampton), Майкл Дженсен (Michael Jensen), Дэвид Криф (David Krief), Нат Луенгнаруэмитхай (Nat Luengnaruemitchai), Томас Пиклак (Thomas Peklak), Майк Штауфенберг (Mike Staufenberg,), Урсин Штосс (Ursin Stauss), Ричард Тобиас (Richard Tobias), Уильям Уилер (William Wheeler), Брад Видерхольт (Brad Wiederholt) и Артур Зубарев (Arthur Zubarev). Они исправляли технические ошибки, ошибки в терминологии и опечатки, а также вносили предложения по разделам книги. Содержание рукописи формировалось и дополнялось на каждом этапе редактирования, с каждым добавленным предложением, поступившим с форума.

Особую благодарность я хотел бы выразить Кену Фрикласу (Ken Fricklas), который сыграл роль старшего научного редактора, Джерри Гейнсу (Jerry Gaines), редактору-консультанту по техническим вопросам, и Дэвиду Фомбелла Помбалу (David Fombella Pombal), научному редактору книги. О лучших научных редакторах я не мог и мечтать.

И наконец, я хотел бы поблагодарить сотрудников издательства Manning Publications, которые сделали возможным появление этой книги: это издатель Марьян Бэйс (Marjan Bace) и все сотрудники издательского и производственного отделов, в том числе Джанет Вэйл (Janet Vail), Тиффани Тейлор (Tiffany Taylor), Шерон Уилки (Sharon Wilkey), Кэти Теннант (Katie Tennant), Деннис Далинник (Dennis Dalinnik), а также многие другие, чья работа осталась за кадром. Из всего множества других контактов, которыми я обзавелся в издательстве Manning, я выражаю мою глубочайшую признательность Тони Арритола (Toni Arritola), редактору, ответственному за работу с читателями книги. Ее непрестанное руководство и наставления на всем протяжении процесса подготовки издания сделали книгу доступной для значительно более широкой аудитории.

Об этой книге

Если вы новичок в машинном обучении или же новичок в использовании TensorFlow, в этой книге вы найдете полное руководство по этим темам. Для понимания некоторых примеров кода от вас потребуется знание объектно-ориентированного программирования на языке Python. В остальном же эта книга является введением в машинное обучение, начиная с его основ.

Содержание книги

Книга разделена на три части:

- Часть I начинается с изучения того, что такое машинное обучение, и в ней описывается ключевая роль TensorFlow. В главе 1 вводится терминология и дается теория машинного обучения. В главе 2 рассказывается о том, что вам нужно знать, чтобы начать использовать TensorFlow.
- Часть II содержит описание основных, проверенных временем алгоритмов. В главах 3–6 обсуждаются регрессия, классификация, кластеризация и скрытые марковские модели соответственно. В машинном обучении эти алгоритмы будут встречаться вам повсюду.
- Часть III раскрывает истинную мощь TensorFlow — нейронные сети. Главы 7–12 знакомят с автокодировщиками, обучением с подкреплением,

сверточными нейронными сетями, рекуррентными нейронными сетями, моделями *sequence-to-sequence* («последовательность-в-последовательность») и с практическим использованием сетей соответственно.

Если вы не продвинутый пользователь TensorFlow с большим опытом в машинном обучении, то я настоятельно рекомендую сначала прочитать главы 1 и 2. Во всех остальных случаях можете свободно просматривать эту книгу так, как вам будет удобно.

Исходный код

Идеи, лежащие в основе этой книги, неподвластны времени; и благодаря сообществу это также относится и к тем листингам программного кода, которые приведены в качестве примеров. Исходный код доступен на веб-сайте книги по адресу www.manning.com/books/machine-learning-with-tensorflow, а обновленная версия программного обеспечения хранится в официальном репозитории книги на GitHub, https://github.com/BinRoot/TensorFlow-Book. Вы также можете внести свой вклад в этот репозиторий через pull request (запрос на включение изменений) или загрузив на GitHub новые версии.

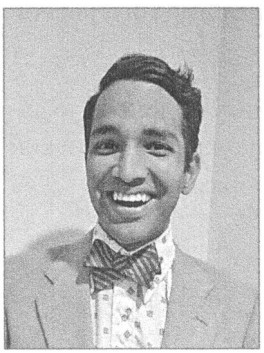
Форум книги

Покупка книги «Машинное обучение и TensorFlow» включает бесплатный доступ к частному веб-форуму (форум на английском языке), организованному издательством Manning Publications, где можно оставлять комментарии о книге, задавать технические вопросы, а также получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, перейдите на https://forums.manning.com/forums/machine-learning-with-tensorflow. Узнать больше о других форумах на сайте издательства Manning и познакомиться с правилами вы сможете на странице https://forums.manning.com/forums/about.

Издательство Manning обязуется предоставить своим читателям место встречи, на которой может состояться содержательный диалог между отдельными читателями и между читателями и автором. Однако со стороны автора отсутствуют какие-либо обязательства уделять форуму внимание в каком-то определенном

объеме — его присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать автору неожиданные вопросы, чтобы его интерес не угасал! Форум и архивы предыдущих обсуждений будут доступны на сайте издательства, пока книга находится в печати.

Об авторе



Нишант Шакла (http://shukla.io) является соискателем на степень доктора наук Калифорнийского университета Лос-Анджелеса, основные темы его исследований: методы машинного обучения и компьютерного зрения в робототехнике. Имеет степень бакалавра Computer Science и степень бакалавра математики Университета Вирджинии. В университете учредил лигу Hack.UVA (http://hackuva.io), а также читал один из самых посещаемых курсов лекций по Haskell (http://shuklan.com/haskell). Занимался разработкой для таких компаний, как Microsoft, Facebook и Foursquare, а также работал инженером по машинному обучению в компании SpaceX; является автором книги *Haskell Data Analysis Cookbook* (http://haskelldata.com). Кроме того, опубликовал ряд научных статей по широкому кругу тем — от аналитической химии до обработки естественного языка (http://mng.bz/e9sk). Свободное время проводит за настольными играми «Поселенцы Катана» и «Гвинт» (и периодически проигрывает).

Об обложке

Рисунок на обложке «*Машинное обучение и TensorFlow*» подписан как «Человек с острова Паг, Далмация, Хорватия». Эта иллюстрация взята с репродукции, опубликованной в 2006 году в книге коллекций костюмов и этнографических описаний XIX века под названием «*Далмация*» профессора Фрейна Каррара (1812–1854), археолога и историка, а также первого директора Музея античности города Сплита в Хорватии. Иллюстрации были любезно предоставлены библиотекарем Музея этнографии (в прошлом — Музея античности), расположенного в римском центре старого Сплита: руины дворца императора

Диоклетиана относятся к 304 году н. э. Книга содержит превосходные цветные рисунки людей из разных областей Далмации, сопровождающиеся описанием их костюмов и повседневной жизни.

С тех пор стиль одежды сильно изменился и исчезло разнообразие, свойственное различным областям и странам. Теперь трудно различить по одежде даже жителей разных континентов. Если взглянуть на это с оптимистичной точки зрения, мы пожертвовали культурным и внешним разнообразием в угоду более насыщенной личной жизни или в угоду более разнообразной и интересной интеллектуальной и технической деятельности.

В наше время, когда трудно отличить одну техническую книгу от другой, издательство Manning проявляет инициативу и деловую сметку, украшая обложки книг изображениями, которые показывают богатое разнообразие жизни.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

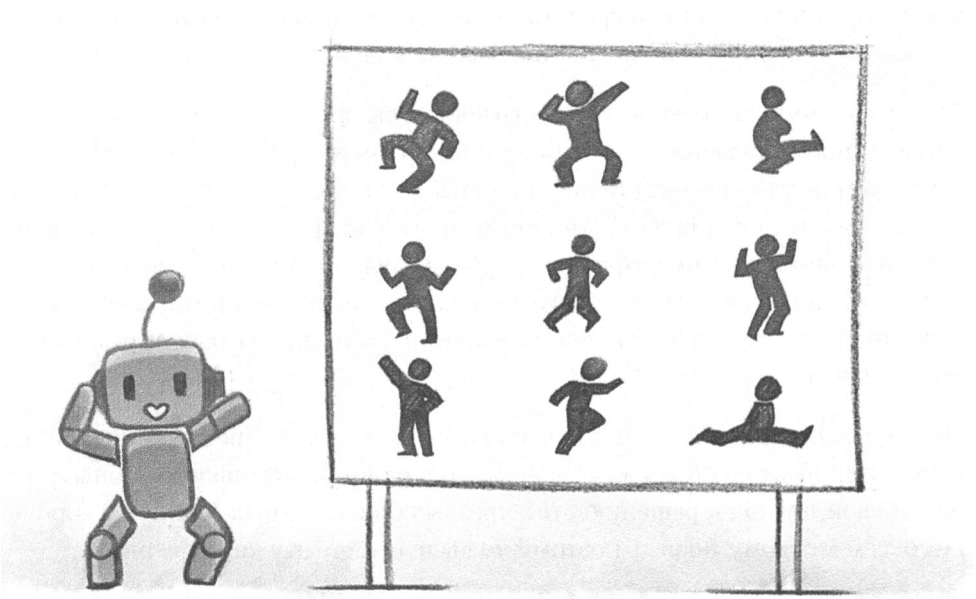
Часть I

Ваше снаряжение для машинного обучения

Обучение параллельной парковке поначалу представляется жутким испытанием. Первые несколько дней уходят на ознакомление с кнопками, вспомогательными камерами и чувствительностью двигателя. Знакомство с машинным обучением и библиотекой TensorFlow происходит аналогичным образом. Прежде чем использовать современные системы распознавания лиц или прогнозов фондовой биржи, необходимо в первую очередь освоиться с соответствующим инструментарием.

Чтобы подготовить надежную основу для машинного обучения, требуется учесть два аспекта. Первый аспект раскрывается в главе 1 и состоит в том, что необходимо освоить язык и теорию машинного обучения. Чтобы говорить на эту тему на одном языке, исследователи в своих публикациях дали точную терминологию и формулировки. Поэтому нам тоже лучше придерживаться этих правил, чтобы избежать путаницы. Второй аспект раскрывается в главе 2 и касается всего, что необходимо знать, чтобы начать использовать библиотеку TensorFlow. У самураев есть самурайский меч, у музыкантов — музыкальные инструменты, а у специалистов-практиков в машинном обучении есть TensorFlow.

Одиссея машинного обучения



Эта глава охватывает следующие темы:

- ✓ Основы машинного обучения
- ✓ Представление данных, признаки и нормы векторов
- ✓ Причины выбора TensorFlow

Вы когда-нибудь задумывались, есть ли предел того, что можно вычислить при помощи компьютерной программы? В наши дни компьютеры способны делать гораздо больше, чем просто решать математические уравнения. Во второй половине столетия программирование стало основным инструментом для автоматизации операций и средством экономии времени, но каков объем того, что мы можем автоматизировать и как вообще можно себе такое представить?

Может ли компьютер изучить фотографию и сказать: «Ага, вижу влюбленную парочку, прогуливающуюся по мосту под зонтом во время дождя»? Может ли программное обеспечение ставить настолько же точные медицинские диагнозы, как и опытный специалист? Могут ли прогнозы программного обеспечения относительно ситуации на фондовых рынках быть лучше, чем умозаключения человека? Достижения последнего десятилетия позволяют предположить, что ответом на все эти вопросы является многократное «да», а в основе реализации этих задач лежат общие методы.

Недавние достижения в теоретических исследованиях вкупе с новейшими технологиями дают возможность каждому при наличии компьютера попытаться найти свой подход к решению этих чрезвычайно сложных задач. Ну хорошо, не совсем каждому, но ведь поэтому-то вы и читаете эту книгу, верно?

Программисту больше не требуется знать запутанные подробности задачи, чтобы приступить к ее решению. Возьмем преобразование речи в текст: при традиционном подходе вам, вероятно, понадобилось бы разобраться с биологической структурой голосовых связок человека, чтобы иметь возможность декодировать выражения, используя для этого многочисленные, спроектированные вручную,

зависящие от предметной области и необобщаемые фрагменты программного кода. В наши дни можно написать программу, которая просмотрит множество примеров и выявит пути решения этой задачи при условии наличия достаточного количества времени и базы таких примеров.

Алгоритмы обучаются по данным аналогично тому, как люди учатся на собственном опыте. Люди учатся, читая книги, анализируя ситуации, обучаясь в школе, обмениваясь информацией во время разговоров и просматривая веб-сайты, и это помимо множества других методов. Как может машина развить способность обучаться? Окончательного ответа на этот вопрос нет, но исследователи мирового уровня разработали разумные программы для разных областей. В различных реализациях этих программ ученые заметили повторяющиеся образы в решении задач этого типа, что привело к возникновению специализированной области, которую сегодня называют *машинным обучением* (МО, machine learning).

По мере развития машинного обучения используемые инструменты становились все более стандартизованными, надежными, высокопроизводительными и масштабируемыми. Именно на этом этапе появилась TensorFlow. Эта библиотека программного обеспечения имеет интуитивный интерфейс, который позволяет программистам погрузиться в сложные идеи машинного обучения и сразу же применять их на практике. В следующей главе приводятся азы этой библиотеки, а все последующие главы содержат описание того, как использовать TensorFlow для каждой из различных областей применения машинного обучения.

НАДЕЖНЫЙ РЕЗУЛЬТАТ МАШИННОГО ОБУЧЕНИЯ

Распознавание образов теперь не является чертой, присущей исключительно человеческим существам. Взрывной рост тактовой частоты компьютера и объема используемой памяти привел нас к необычной ситуации: компьютеры теперь можно использовать для составления прогнозов, выявления аномалий, упорядочивания элементов и автоматической классификации изображений. Этот новый набор инструментов дает разумные ответы к задачам, которые не имеют четкого решения, но тут перед нами встает вопрос о доверии. Доверите ли вы компьютерному алгоритму выдачу жизненно важных рекомендаций относительно лечения: например, выполнять операцию на сердце или нет?

В таких вопросах нет места недостоверным методам машинного обучения. Доверие человека — слишком хрупкая субстанция, и наши алгоритмы, несомненно, должны быть надежными. Помните об этом и внимательно и с осторожностью изучайте информацию в этой главе.

1.1. Основы машинного обучения

Пытались ли вы когда-нибудь объяснить кому-либо, как правильно плавать? Объяснение ритма движений и форм обтекаемости ошеломляет своей сложностью. Аналогичным образом некоторые задачи программного обеспечения слишком сложны для нас, чтобы их можно было без труда охватить нашим сознанием. Именно для таких задач машинное обучение может оказаться самым подходящим инструментом.

Когда-то алгоритмы для выполнения этой работы собирались вручную и тщательно отлаживались, и это был единственный способ создания программного обеспечения. Проще говоря, традиционное программирование предполагает детерминированный выход для каждого набора входных данных. Машинное обучение, напротив, может решать класс задач, для которых соответствие входа и выхода недостаточно хорошо определено.

ПОЛНЫЙ ВПЕРЕД!

Машинное обучение является относительно молодым методом, а поэтому представьте, что вы — геометр в эпоху Евклида, прокладывающий путь в новой, только что открытой области. Или физик во времена Ньютона, обдумывающий нечто, подобное общей теории относительности, но для машинного обучения.

В машинном обучении используется программное обеспечение, которое обучается на основе ранее полученного опыта. Такая компьютерная программа улучшает свои результаты по мере того, как получает все новые и новые примеры. Тогда можно надеяться, что, если вы закинете в этот «механизм» достаточное количество данных, он научится распознавать образы и выдавать разумные результаты уже для новых входных данных.

Машинное обучение называют также *индуктивным обучением* (inductive learning), потому что код старается выявить структуру только лишь на основе данных. Это все равно что отправиться на каникулы за границу и читать местный журнал мод, пытаясь понять, как одеться, чтобы сойти «за своего». Изучая изображения людей в местной одежде, вы сможете сформировать в своем представлении некий образ локальной культуры. Такой способ обучения называют *индуктивным*.

Возможно, ранее вам никогда не доводилось применять такой подход к программированию, так как необходимость в индуктивном обучении есть не всегда. Предположим, вам нужно определить, четным или нечетным числом является сумма двух произвольных чисел . Уверен, вы можете представить себе тренировку алгоритма машинного обучения на миллионе обучающих примеров (рис. 1.1), но вы, безусловно, понимаете, что это крайность. Более прямой подход может без труда решить эту задачу.

Вход		Выход
$x_1 = (2, 2)$	→	y_1 = Четное
$x_2 = (3, 2)$	→	y_2 = Нечетное
$x_3 = (2, 3)$	→	y_3 = Нечетное
$x_4 = (3, 3)$	→	y_4 = Четное
...		...

Рис. 1.1. Каждая пара целых чисел при их суммировании дает четное или нечетное число. Перечисленные соответствия входа и выхода носят название контрольного набора данных (ground-truth dataset)

Например, сумма двух нечетных чисел всегда является четным числом. Убедитесь сами: возьмите два любых нечетных числа, сложите их между собой и проверьте, является ли их сумма четным числом. Вот как можно доказать этот факт:

- Для любого целого числа n выражение $2n + 1$ дает нечетное число. Более того, любое нечетное число можно записать как $2n + 1$ для некоторого целого числа n. Число 3 можно записать как $2(1) + 1$. А число 5 можно записать как $2(2) + 1$.

- Пусть у нас два нечетных числа, $2n + 1$ и $2m + 1$, где n и m — целые числа. Сложение двух нечетных чисел дает $(2n + 1) + (2m + 1) = 2n + 2m + 2 = 2(n + m + 1)$. Это — четное число, потому что умножение любого целого числа на 2 дает четное число.

Аналогичным образом мы видим, что сумма двух четных чисел тоже является четным числом: $2m + 2n = 2(m + n)$. И наконец, мы также приходим к выводу, что сумма четного и нечетного чисел является нечетным числом: $2m + (2n + 1) = 2(m + n) + 1$. На рис. 1.2 эта логика представлена более понятно.

m \ n	Четное	Нечетное
Четное	$2m + 2n =$ $2(m + n)$ Четное	$2m + (2n + 1) =$ $2m + 2n + 1$ Нечетное
Нечетное	$(2m + 1) + 2n =$ $2m + 2n + 1$ Нечетное	$(2m + 1) + (2n + 1) =$ $2(m + n + 1)$ Четное

Рис. 1.2. Таблица раскрывает внутреннюю логику соответствия выходных данных входным парам целых чисел

Вот и все! Безо всякого машинного обучения вы можете решить эту задачу для любой пары целых чисел, которую вам кто-нибудь подкинет. Эту задачу можно решить прямым применением математических правил. Однако в алгоритмах машинного обучения внутреннюю логику принято рассматривать как *черный ящик*; это означает, что логика происходящего внутри может быть не очевидна для интерпретации, как это показано на рис. 1.3.

Рис. 1.3. Подход к решению задач в машинном обучении можно представить как настройку параметров черного ящика до тех пор, пока он не начнет выдавать удовлетворительные результаты

1.1.1. Параметры

Иногда тот способ, который позволяет наилучшим образом реализовать алгоритм, преобразующий входной сигнал в соответствующий выходной, является слишком сложным. Например, если на вход подать серию чисел, кодирующих изображение в градациях серого, можно представить, насколько сложно написать алгоритм для маркировки каждого элемента этого изображения. Машинное обучение оказывается полезным, когда не слишком понятно, какая именно работа происходит внутри объекта. Оно предоставляет нам набор инструментов для написания программы без необходимости вдаваться в каждую деталь алгоритма. Программист может оставить некоторые значения неопределенными, тем самым давая возможность системе машинного обучения самой определить их наилучшие значения.

МАШИННОЕ ОБУЧЕНИЕ МОЖЕТ РЕШАТЬ ЗАДАЧИ, НЕ ПРОНИКАЯ В СУТЬ ПРЕДМЕТА

Искусство индуктивного решения задач — это палка о двух концах. Алгоритмы машинного обучения дают неплохой результат при решении определенных задач, однако, несмотря на это, попытки пошагово, с применением дедуктивных методов, проследить, как получился такой результат, могут не сразу увенчаться успехом. Тщательно продуманная система машинного обучения изучает тысячи параметров, но выяснение значения каждого параметра не всегда является ее основной задачей. Уверен, что, располагая этой информацией, вы откроете перед собой поистине волшебный мир.

Неопределенные значения называют *параметрами*, а их описание называют *моделью*. Ваша работа состоит в том, чтобы написать алгоритм, который проанализирует имеющиеся примеры и выявит, как наилучшим образом настроить параметры для получения оптимальной модели. Это невероятно важная мысль! Не волнуйтесь, эта концепция станет лейтмотивом книги, и вы столкнетесь с ней еще много раз.

УПРАЖНЕНИЕ 1.1

Предположим, вы три месяца собирали данные о ситуации на фондовом рынке. Вам хотелось бы прогнозировать дальнейшие тенденции этого рынка в целях переиграть систему и получать денежную выгоду. Как бы вы решили эту задачу, не используя методы машинного обучения? (Как вы узнаете в главе 8, эту задачу можно решить именно методами машинного обучения.)

ОТВЕТ

Верите вы или нет, но твердо установленные правила являются стандартным способом определить торговые стратегии фондового рынка. Например, часто используют простой алгоритм: «если цена падает на 5 %, акции можно покупать». Обратите внимание на то, что в этом случае машинное обучение не используется, а применяется только традиционная логика.

1.1.2. Обучение и вывод

Представьте, что вы пытаетесь приготовить десерт в духовке. Если вы новичок в готовке, то, чтобы получить что-то по-настоящему вкусное, вам потребуется несколько дней для выяснения правильного состава и точного соотношения ингредиентов. Если вы запишете этот рецепт, то впоследствии сможете быстро воспроизвести его и получить в точности то же самое вкусное блюдо.

Машинное обучение аналогичным образом использует идею с рецептами. Обычно мы проверяем алгоритм на двух стадиях: *стадии обучения* и *стадии логического вывода*. Цель этапа обучения — описание данных, которые называются *вектором признаков*, и сведение их в модели. Модель как раз и является

нашим рецептом. В сущности, модель — это программа с парой открытых интерпретаций, а данные позволяют устранять присущую ей двусмысленность.

> **ПРИМЕЧАНИЕ** *Вектор признаков объекта, признаковое описание* (feature vector), представляет собой упрощенное представление исходных данных. Вектор признаков можно рассматривать как сводку характеристик реальных объектов. Стадии обучения и вывода используют вектор признаков, а не сами исходные данные.

Аналогично тому как рецепты могут использоваться другими людьми, модель, полученная в результате обучения, может использоваться повторно в других программах. Больше всего времени занимает стадия обучения. Для получения полезной модели может потребоваться ждать выполнения алгоритма в течение нескольких часов, если не дней и недель. На рис. 1.4 показан процесс обучения.

Рис. 1.4. Метод обучения обычно следует структурированному рецепту. Сначала набор данных необходимо преобразовать в представление, чаще всего — в список признаков, который затем можно использовать в обучающем алгоритме. Обучающий алгоритм выбирает модель и подбирает ее параметры наилучшим образом

Стадия вывода использует полученную модель, чтобы сделать умозаключения в отношении данных, прежде ему неизвестных. Это похоже на использование рецепта, найденного в интернете. Процесс логического вывода обычно занимает на порядок меньше времени, чем обучение; вывод можно сделать достаточно быстро в режиме реального времени. К этапу вывода относят все, что касается тестирования модели на новых данных и анализа результатов в процессе испытания, как показано на рис. 1.5.

Рис. 1.5. Для логического вывода обычно используется модель, которая уже прошла стадию обучения или проходит ее. После преобразования данных в удобное представление, такое как вектор признаков, они используются моделью для получения выходных данных

1.2. Представление данных и признаки

Данные имеют первостепенное значение для машинного обучения. Компьютеры — это не более чем сложные калькуляторы, и поэтому данные, которые мы подаем в систему машинного обучения, должны быть вполне определенными математическими объектами, такими как векторы, матрицы или графы.

Главными элементами всех форм представления данных выступают *признаки*, которые являются наблюдаемыми свойствами объекта:

- *Векторы* имеют плоскую и простую структуру и обычно, в большинстве приложений для машинного обучения, представляют собой объединение данных. У них есть два атрибута: *размерность* вектора (это натуральное число) и *тип* его элементов (например, действительные числа, целые числа и т. п.). В качестве напоминания вот некоторые примеры двумерных целочисленных векторов: (1, 2) и (−6, 0) — некоторые примеры трехмерных векторов действительных чисел, такие как (1,1; 2,0; 3,9) и (π, $\pi/2$, $\pi/3$). Вы, вероятно, уже догадались: все это собрание чисел одного типа. В программе, использующей машинное обучение, вектор применяют для измерения свойств данных, таких как цвет, плотность, сила звука, близость к чему-либо, то есть всего того, что можно описать с помощью последовательности чисел — по одному числу для каждого оцениваемого свойства.
- А вектором векторов является *матрица*. Если каждый вектор описывает признаки одного объекта в наборе данных, то матрица описывает признаки всех объектов, при этом каждый элемент вектора является узлом, представляющим собой список признаков одного объекта.

- *Графы*, также используемые в машинном обучении, являются более выразительным представлением данных. Граф представляет собой набор объектов (*узлов*), которые могут быть соединены *ребрами* графа, образуя сеть. Графические структуры позволяют отражать связи между объектами, например круг друзей или навигационный маршрут системы метрополитена. Поэтому ими значительно труднее управлять в приложениях для машинного обучения. В этой книге входные данные будут редко включать графические структуры.

Вектор признаков является упрощенным представлением реальных данных, которые сами по себе могут быть слишком сложными, чтобы их можно было обрабатывать. Вместо того чтобы заниматься всеми мелкими деталями данных, используют упрощенный вектор признаков. Например, машина в действительности является чем-то большим, чем просто текст, используемый для ее описания. Продавец старается продать вам машину, а не высказанные или записанные слова, которые ее описывают. Эти слова — всего лишь отвлеченные понятия, так же как вектор признаков — это всего лишь краткая сводка информации об исходных данных.

Следующий сценарий объяснит это подробнее. Когда вы собираетесь купить новую машину, очень важно учитывать все мелкие детали различных марок и моделей. В конце концов, если вы решили потратить тысячи долларов, необходимо очень тщательно к этому подготовиться. Для этого потребуется составить список признаков каждой машины, а затем досконально сравнить их с характеристиками других машин. Этот упорядоченный список признаков и является признаковым описанием.

При поиске подходящей машины есть смысл сравнивать пробег, что намного целесообразнее, чем сравнивать менее значимые характеристики, такие как вес. Количество отслеживаемых признаков также должно быть выбрано правильно: их не должно быть слишком мало, иначе будет утеряна значимая часть информации; но и не должно быть слишком много, иначе их придется долго прослеживать. Процедура выбора количества показателей и определения того, какие именно показатели следует сравнивать, носит название конструирования признаков. В зависимости от проверяемых признаков эффективность системы может изменяться очень сильно. Выбор подходящих признаков для отслеживания может компенсировать низкую эффективность используемого алгоритма обучения.

Например, при обучении модели обнаруживать на изображении машины работоспособность и скорость системы значительно возрастут, если предварительно преобразовать изображение в режим градаций серого цвета. С помощью внесения собственных изменений в процесс предварительной обработки данных вы в конечном счете поможете алгоритму, так как программу не потребуется обучать тому, что цвета не имеют значения для обнаружения машины. Вместо этого алгоритм сможет сконцентрироваться на идентификации форм и текстур, что приведет к быстрому обучению, так как будут исключены попытки обрабатывать цвета.

Общее эмпирическое правило машинного обучения состоит в том, что больший объем данных приносит несколько лучшие результаты. Однако в отношении большего числа признаков это справедливо не всегда. Результат может оказаться даже прямо противоположным, то есть производительность системы может пострадать, если число отслеживаемых признаков окажется слишком большим. Заполнение пространства всех данных репрезентативными образцами требует экспоненциально больше данных, поскольку размерность вектора признаков становится больше. Поэтому конструирование признаков, как это показано на рис. 1.6, является одной из наиболее важных задач машинного обучения.

Рис. 1.6. Конструирование признаков является процессом выбора наиболее значимых из них для рассматриваемой задачи

ПРОКЛЯТИЕ РАЗМЕРНОСТИ

Для точного моделирования реальных данных нам, очевидно, потребуется больше двух опорных точек. Но точное число точек зависит от многих факторов, включая размерность вектора признаков. Добавление слишком большого числа признаков приводит к экспоненциальному росту числа опорных точек, которые требуются для описания пространства. Именно поэтому мы не можем спроектировать 1 000 000-мерный вектор признаков, чтобы использовать все возможные факторы, а затем ждать, пока алгоритм обучит модель. Это явление получило название *проклятия размерности*.

Возможно, вы не заметите этого сразу, но после того, как вы решите, за какими именно признаками лучше всего наблюдать, произойдет нечто важное. Философы много столетий обдумывали значение понятия *идентичность*; вы тоже не придете к этому сразу, но с определением признаков вы выявите определение *идентичности*.

Представьте, что вы пишете систему машинного обучения для обнаружения лиц на изображении. Пусть одним из необходимых признаков того, чтобы нечто можно было отнести к категории лиц, является наличие двух глаз. Теперь за лицо будет приниматься нечто с глазами. Можете представить себе, к какой неприятности это может привести? Если на фотографии человек моргает, детектор не обнаружит лицо, потому что не найдет двух глаз. То есть алгоритм не сможет определить лицо, когда человек моргает. Таким образом, мы сразу начали с некорректного описания того, что может относиться к категории лиц, и это стало понятным исходя из плохих результатов обнаружения.

Идентичность объекта можно разложить на признаки, из которых он состоит. Например, если признаки, которые вы отслеживаете для одной машины, в точности совпадают с соответствующими характеристиками другой машины, с вашей точки зрения они могут быть идентичными. Вам потребуется добавить в систему еще один признак, чтобы их различить, иначе их действительно придется считать идентичными. При выборе признаков нужно очень постараться, чтобы не впасть в философские рассуждения об идентичности.

УПРАЖНЕНИЕ 1.2

Представьте, что мы обучаем робота, как складывать одежду. Система восприятия видит футболку, лежащую на столе, как показано на рисунке ниже. Вы хотели бы представить футболку в виде признакового описания, чтобы ее можно было сравнивать с другими видами одежды. Определите, какие характеристики целесообразнее всего отслеживать (подсказка: какие слова используют продавцы при описании одежды в интернет-магазинах?).

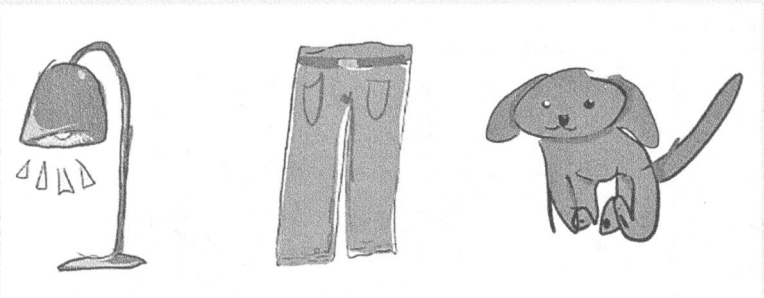
Робот пытается сложить футболку.
Какие признаки футболки целесообразно отслеживать?

ОТВЕТ

Ширина, высота, симметрия относительно оси *X*, симметрия относительно оси *Y* и плоскостность являются хорошими признаками для отслеживания при складывании одежды. Меньше всего имеют значение цвет, текстура ткани и материал.

УПРАЖНЕНИЕ 1.3

Теперь вместо складывания одежды вы амбициозно решили выявлять произвольные объекты; на рисунках ниже приведены некоторые примеры таких объектов. Какие из признаков являются наиболее характерными и позволят без труда отличать объекты друг от друга?



Здесь приведены изображения трех объектов: лампы, брюк и собаки. Какие именно признаки вы должны записать для сравнения и дифференцирования этих объектов?

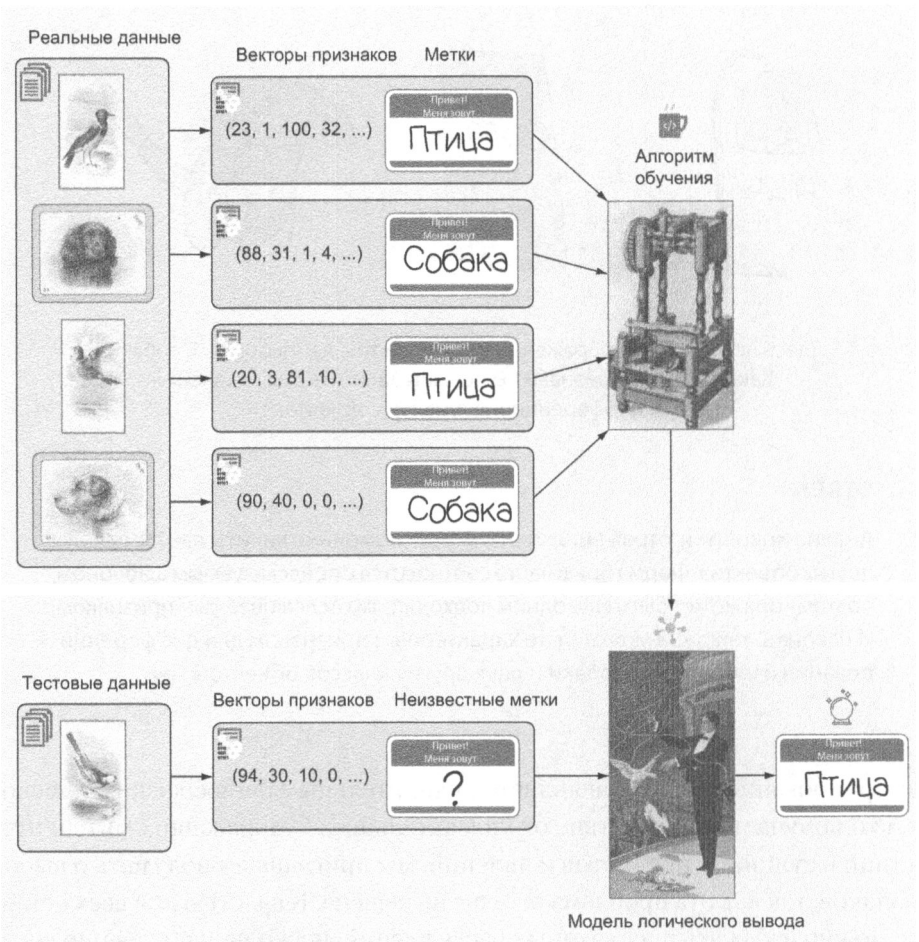
ОТВЕТ

Анализ яркости и отражения света может помочь отличить лампу от двух других объектов. Форма брюк часто соотносится с предсказуемым шаблоном, поэтому она может быть еще одним подходящим отслеживаемым признаком. И наконец, текстура может быть характерным признаком для дифференцирования изображения собаки и двух других классов объектов.

Конструирование признаков является удивительным философским поиском. Тех, кто получает удовольствие от увлекательных, будоражащих мысли путешествий в сущность предметов и явлений, мы приглашаем подумать о выборе признаков, так как эта проблема все еще не решена. К счастью для всех остальных, можно избежать пространных рассуждений, поскольку последние достижения в области машинного обучения позволяют автоматически определять, какие признаки объектов отслеживать. У вас будет возможность попробовать это сделать в главе 7.

ВЕКТОРЫ ПРИЗНАКОВ ИСПОЛЬЗУЮТСЯ КАК НА ЭТАПЕ ОБУЧЕНИЯ, ТАК И НА ЭТАПЕ ЛОГИЧЕСКОГО ВЫВОДА

Взаимодействие между обучением и логическим выводом дает полное представление о системе машинного обучения, как это показано на следующем рисунке. Сначала необходимо представить реальные данные в виде вектора



Векторы признаков являются формой представления реальных данных, которая используется как обучающими, так и формирующими логический вывод компонентами системы машинного обучения. Входными данными для алгоритма являются не реальные изображения, а векторы их признаков

признаков. Например, мы можем представить изображения вектором чисел, соответствующих интенсивности пикселов (как представлять изображения более детально, мы изучим в следующих главах). Мы можем показать нашему алгоритму метки контрольных данных (такие, как «птица» или «собака») вместе с соответствующими векторами признаков. При достаточном объеме данных алгоритм на основе машинного обучения создаст обученную модель. Мы сможем использовать эту модель для других реальных данных, чтобы выявить ранее неизвестные классы.

1.3. Метрики расстояния

Предположим, у вас есть список автомобилей, один из которых вы хотели бы купить, и их векторы признаков. Тогда, чтобы выяснить, какие два из них больше всего похожи, вы можете определить функцию расстояния для векторов признаков. Выявление сходства между объектами составляет значительную долю задач машинного обучения. Векторы признаков позволяют представлять объекты так, чтобы их можно было сравнивать различными способами. Стандартный метод заключается в использовании *евклидова расстояния*, которое является геометрическим расстоянием между двумя точками в пространстве и которое можно считать наиболее наглядным.

Предположим, что у нас два вектора признаков, $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$. Евклидово расстояние $\|x - y\|$ рассчитывается по формуле

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2}.$$

Например, евклидово расстояние между (0, 1) и (1, 0) будет

$$\|(0,1) - (1,0)\| =$$

$$= \|(-1,1)\| =$$

$$= \sqrt{(-1)^2 + 1^2} =$$

$$= \sqrt{2} = 1{,}414...$$

Ученые называют это нормой в пространстве L2 (*L2-нормой*). Однако это лишь одна из многих возможных метрик расстояния. Также существуют нормы L0,

L1 и L-бесконечность (бесконечная норма). Все эти нормы могут применяться для измерения расстояния. Рассмотрим их подробнее:

- *L0-норма* определяется как число ненулевых элементов вектора. Например, расстояние между началом отсчета (0, 0) и вектором (0, 5) равно 1, так как у вектора только один ненулевой элемент. Расстояние L0 между (1, 1) и (2, 2) равно 2, поскольку ни одна размерность не совпадает. Представим себе, что первая и вторая размерность представляют имя пользователя и пароль соответственно. Если расстояние по L0-норме между попыткой регистрации и действительными учетными данными равно 0, регистрация будет успешной. Если расстояние равно 1, тогда либо имя пользователя, либо пароль неверные, но не то и другое сразу. И наконец, если расстояние равно 2, значит, одновременно и имя пользователя, и пароль не были найдены в базе данных.
- *L1-норма*, представленная на рис. 1.7, определяется как $\sum|x_n|$. Расстояние между двумя векторами по норме L1 также называют *манхэттенским расстоянием*. Представим центральный район какого-нибудь города, например Манхэттен в Нью-Йорке, в котором улицы образуют сетку. Кратчайшее расстояние от одного пересечения улиц до другого пролегает вдоль кварталов. Аналогичным образом расстояние по норме L1 между двумя векторами пролегает перпендикулярно им.

  Расстояние между (0, 1) и (1, 0) по норме L1 равно 2. Расстояние по норме L1 между двумя векторами является суммой абсолютных разностей их координат и представляет собой полезную метрику сходства.

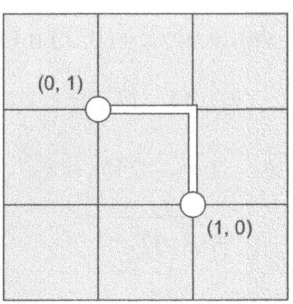


Рис. 1.7. Расстояние по норме L1 носит название *манхэттенского расстояния* (или метрики такси), поскольку напоминает маршрут машины по пересечениям улиц Манхэттена. Если машина перемещается из точки (0,1) в точку (1,0), то длина кратчайшего маршрута составит 2 единицы

- *L2-норма*, показанная на рис. 1.8, является евклидовой длиной вектора, $(\sum(x_n)^2)^{1/2}$. Это наиболее прямой маршрут, который можно выбрать на геометрической плоскости, чтобы добраться из одной точки в другую. Пояснение для тех, кому интересна математика: эта норма использует оценку по методу наименьших квадратов согласно теореме Гаусса—Маркова. Для всех остальных: это кратчайшее расстояние между двумя точками.

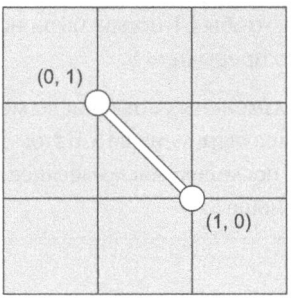


Рис. 1.8. L2-норма между точками (0,1) и (1,0) является длиной отрезка прямой между двумя точками

- *L-N-норма* является обобщением исходного правила и имеет вид $(\sum|x_n|^N)^{1/N}$. Конечные нормы выше L2 используются редко, здесь они приведены для полноты картины.
- *Бесконечная L-норма* имеет вид $(\sum|x_n|^\infty)^{1/\infty}$. Она определяется как наибольшая абсолютная величина из всех элементов вектора. Если мы имеем вектор (–1, –2, –3), то его бесконечная L-норма равна 3. Если вектор признаков содержит стоимости различных элементов, то минимизация бесконечной L-нормы вектора является попыткой снизить стоимость самого дорогостоящего элемента.

ГДЕ В РЕАЛЬНОМ МИРЕ ИСПОЛЬЗУЕТСЯ МЕТРИКА, ОТЛИЧНАЯ ОТ L2-НОРМЫ?

Допустим, вы работаете в стартапе и занимаетесь разработкой новой поисковой системы, пытаясь конкурировать с Google. Руководитель дал вам задание использовать машинное обучение, чтобы персонализировать результаты поиска для каждого пользователя.

Хорошим результатом с точки зрения пользователя можно считать не более пяти некорректных результатов поиска в месяц. Годовой оценкой данных пользователя является 12-мерный вектор (каждый месяц года является размерностью), отражающий количество некорректных результатов за каждый месяц. Вы стараетесь выполнить условие, чтобы бесконечная L-норма этого вектора была меньше 5.

Допустим, ваш руководитель изменил требования, заявив, что допустимо не более пяти ошибочных результатов поиска за весь год. В этом случае необходимо стремиться, чтобы L1-норма была ниже 5, так как суммарное число ошибок не должно превышать 5.

И вот требования были изменены снова: число месяцев с ошибочными результатами поисков должно быть меньше 5. В этом случае необходимо, чтобы L0-норма была меньше 5, поскольку число месяцев с ненулевым количеством ошибок должно быть меньше 5.

1.4. Типы обучения

Теперь вы умеете сравнивать векторы признаков, а значит, обладаете необходимым инструментарием и готовы использовать данные для практической реализации алгоритмов. Машинное обучение часто разделяется на три направления: обучение с учителем (контролируемое обучение), обучение без учителя (неконтролируемое обучение) и обучение с подкреплением. Давайте рассмотрим каждое из них.

1.4.1. Обучение с учителем

По определению *учитель* — это тот, кто находится выше в цепи командования. Как только у нас появляются сомнения, учитель диктует, что делать. То же справедливо и в отношении *обучения с учителем* — оно происходит на примерах, которые этот учитель дает (приблизительно как в школе).

Системе контролируемого машинного обучения необходимы разбитые по категориям данные, на основе которых она вырабатывает некие навыки распознавания, называемые *моделью*. Например, если такой системе предоставить множество фотографий людей с записями об их этнической принадлежности,

то мы можем обучить модель устанавливать этническую принадлежность ранее никогда не встречавшегося человека на произвольно выбранной фотографии. Проще говоря, модель является функцией, которая присваивает данным метку определенной категории. Она делает это с помощью собранных ранее примеров, именуемых *обучающей выборкой*, используя эти данные в качестве справочной информации.

Удобным способом описания моделей являются математические обозначения. Пусть x является элементом данных, таким, например, как вектор признаков. Соответствующей меткой класса, связанной с x, является $f(x)$, которую часто называют *контрольным значением* x. Обычно для простоты записи мы используем зависимую переменную $y = f(x)$. В примере с классификацией этносов человека по фотографиям x может быть 100-мерным вектором различных значимых признаков, а y является одной из пар значений для представления различных этносов. Поскольку y является дискретной величиной с несколькими значениями, модель называют *классификатором*. Если y может принимать много значений и эти значения упорядочены естественным образом, тогда модель называют *регрессионной*.

Обозначим модельный прогноз x через $g(x)$. Иногда можно настроить модель так, что качество ее прогноза может значительно измениться. Модели имеют параметры, которые настраиваются вручную или автоматически. Для представления этих параметров используем вектор θ. Если кратко, то $g(x|\theta)$ значительно полнее представляет модель, и это выражение следует читать как «g от x, заданного θ».

> **ПРИМЕЧАНИЕ** Модели могут также иметь *гиперпараметры*, которые являются ее дополнительными и специальными свойствами. Приставка *гипер-* в термине «*гиперпараметр*» сначала кажется несколько странной. Чтобы легче запомнить, можно представить более удачное название «*метапараметр*», потому что этот тип параметра является близким по смыслу к метаданным модели.

Успех прогноза модели $g(x|\theta)$ зависит от того, насколько хорошо она согласуется с наблюдаемым значением y. Для измерения расстояния между этими

двумя векторами необходимо выбрать способ измерений. Например, L2-норму можно использовать для измерения близости расположения этих векторов. Расхождение между наблюдаемыми значениями и прогнозом носит название *затрат* (cost).

Сущность алгоритма машинного обучения с учителем состоит в подборе таких параметров модели, которые приведут к минимальным *затратам*. Математически это означает, что мы ищем θ^*, которая минимизирует затраты среди всех точек данных $x \in X$. Один из способов формализовать эту задачу оптимизации заключается в следующем:

$$\theta^* = \arg\min_\theta \mathrm{Cost}(\theta \mid X),$$

$$\text{где } \mathrm{Cost}(\theta \mid X) = \sum_{x \in X} \| g(x \mid \theta) - f(x) \| .$$

Ясно, что вычисление методом полного перебора всех возможных комбинаций θ_s (известного также как *пространство параметров*) в конечном счете даст оптимальное решение, но на это уйдет слишком много времени. Обширная область исследований в машинном обучении посвящена написанию алгоритмов поиска эффективности в этом пространстве параметров. К первым посвященным этой области алгоритмам относятся *градиентный спуск*, *имитации отжига* и *генетические алгоритмы*. TensorFlow автоматически следит за деталями реализации средствами нижнего уровня этих алгоритмов, поэтому мы не будем вникать в их многочисленные детали.

После того как параметры были тем или иным образом получены в результате машинного обучения, модель можно наконец оценить, чтобы понять, насколько хорошо система выделяет образы из данных. Согласно эмпирическому правилу, не следует оценивать модель, используя те же данные, с помощью которых она была обучена, так как уже известно, как она работает на обучающих данных; необходимо проверить, работает ли она с данными, которые *не были* частью обучающего набора, то есть является ли она универсальной моделью и не *смещается* ли к данным, которые использовались при ее обучении. Используйте большую часть данных для обучения, а оставшуюся часть — для проверки. Например, если есть 100 помеченных точек данных, выберите случайным образом 70 из них для обучения модели, а остальные 30 для ее тестирования.

ДЛЯ ЧЕГО НУЖНО РАЗБИВАТЬ ДАННЫЕ?

Если разбиение данных в соотношении 70:30 кажется странным, представьте себе следующее. Допустим, что учитель физики устраивает пробный экзамен и говорит, что настоящий экзамен будет таким же. При этом можно запомнить ответы и получить превосходный балл, не понимая основ. Аналогично, если проверка модели выполняется на обучающих данных, вы не получите никакой пользы. При этом есть риск ложного чувства безопасности, поскольку модель может просто запомнить результаты. И какой в этом смысл?

Специалисты машинного обучения обычно делят наборы данных не в соотношении 70:30, а в соотношении 60:20:20. Обучение выполняется на 60 % от набора данных, проверка — на 20 %, а оставшиеся 20 % используются для *подтверждения*, суть которого объясняется в следующей главе.

1.4.2. Обучение без учителя

Обучение без учителя предполагает использование данных для моделирования, поступающих без соответствующих категорий и обработки. То, что можно вообще делать какие-либо заключения по необработанным данным, представляется магией. При достаточном объеме данных можно находить паттерны и структуры. Два наиболее мощных инструмента, которые используют специалисты для машинного обучения на основе только исходных данных, это кластеризация и уменьшение размерности.

Кластеризацией называют процесс разделения данных на отдельные сегменты сходных элементов. В этом отношении кластеризация похожа на классификацию данных без каких-либо соответствующих меток классов. Например, при расстановке книг на трех имеющихся полках вы будете ставить на одну полку книги, близкие по жанру, или располагать их по фамилиям авторов. Возможно, у вас будет одна секция для Стивена Кинга, другая для учебников, а третья для «чего-нибудь еще». Вам не важно, что все они разделены по одному признаку, а важно то, что в каждой из них есть нечто уникальное, и это позволяет разбить их примерно на равные, легко идентифицируемые группы. Один из наиболее популярных алгоритмов — это *метод k-средних*, который является частным случаем более мощного метода с названием *EM-алгоритм*.

Метод понижения размерности использует манипулирование данными для возможности рассматривать их в упрощенной проекции. Этот метод машинного обучения можно охарактеризовать фразой «Будь проще, дурачок». Например, избавившись от избыточных признаков, мы можем представить те же данные в пространстве меньшей размерности и увидеть, какие именно признаки важны. Это упрощение помогает также визуализировать данные или выполнить предварительную обработку для повышения эффективности модели. Одним из наиболее ранних алгоритмов этого направления является *метод главных компонент* (PCA), а одним из самых последних является *автокодировщик*, который описан в главе 7.

1.4.3. Обучение с подкреплением

Обучение с учителем или без предполагает наличие учителя на всех этапах или ни на каких соответственно. Но в одном хорошо изученном направлении машинного обучения окружающая среда играет роль учителя, предлагая советы, а не определенные ответы. Система обучения получает обратную связь от того, что она делает, без определенных обещаний того, что она действует в правильном направлении. Это может стать поиском в лабиринте или достижением явной цели.

ИССЛЕДОВАНИЕ ВМЕСТО ИСПОЛЬЗОВАНИЯ — ОСНОВА ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

Представим участие в видеоигре, которую вы никогда раньше не видели. Вы нажимаете те или иные кнопки и обнаруживаете, что определенное сочетание нажатий этих кнопок постепенно увеличивает заработанные очки. Прекрасно, теперь вы неоднократно используете эту находку, чтобы заработать много очков. Но подсознательно вы допускаете, что пропустили более результативную комбинацию. Будете ли вы и дальше использовать свою стратегию или рискнете найти новые варианты?

В отличие от обучения с учителем, в котором обучающие данные удобно помечены «учителем», *обучение с подкреплением* обучает на основе информации, собранной при наблюдении за тем, как окружающая среда реагирует на те или иные действия. Обучение с подкреплением относится к такому машинному

обучению, которое взаимодействует с окружающей средой, чтобы установить, какое сочетание действий дает наиболее благоприятный результат. Используя в алгоритмах такие понятия, как *окружающая среда* и *действие*, мы тем самым антропоморфизируем их, и ученые обычно называют такую систему автономным *агентом*. Этот тип машинного обучения естественным образом проявляет себя в области робототехники.

Для обсуждения агентов в окружающей среде вводятся два новых понятия: состояние и действие. Статус замершего на определенное время мира называют *состоянием*. Агент может выполнять одно из многих *действий*, чтобы изменить текущее состояние. Для того чтобы побудить агента выполнять действия, каждое состояние предоставляет соответствующую *награду*. Агент в конечном счете находит награду каждого состояния, которую называют *ценностью* состояния.

Как и в любой другой системе машинного обучения, эффективность этого метода повышается с увеличением объема данных. В этом случае данные являются архивом ранее приобретенного опыта. В случае обучения с подкреплением мы не знаем окончательной цены или награды серии действий, пока они выполняются. Эти состояния отображают неэффективность традиционного обучения с учителем, поскольку мы в точности не знаем, какое действие в выполненной последовательности действий несет ответственность за завершение процесса в состоянии с низкой ценностью. Единственное, что агент точно знает о ценности серии действий, — это то, что они уже были совершены, чего явно недостаточно. Целью агента является найти последовательность действий, которая приносит максимальную награду.

УПРАЖНЕНИЕ 1.4

Какое именно обучение, с учителем, без учителя или обучение с подкреплением, вы будете использовать, чтобы решить следующие задачи? (А) Разложить разные фрукты в три корзины без дополнительной информации. (Б) Прогноз погоды на основании данных датчика. (В) Обучение игре в шахматы после многочисленных проб и ошибок.

ОТВЕТ

(А) Обучение без учителя. (Б) Обучение с учителем. (В) Обучение с подкреплением.

1.5. Библиотека TensorFlow

Google открыл основу своей системы машинного обучения, библиотеку TensorFlow, в конце 2015 года с разрешения компании Apache 2.0. До этого библиотека использовалась Google как патентованное средство распознавания речи, поиска, обработки фотографий и для электронной почты Gmail вместе с остальными приложениями.

НЕМНОГО ИСТОРИИ

Использовавшаяся ранее масштабируемая распределенная система обучения DistBelief оказала первостепенное влияние на развитие используемой в настоящее время библиотеки TensorFlow. Вы когда-то написали запутанный код и хотели бы его переписать? В этом и состоит движение от DistBelief к TensorFlow.

Библиотека реализована на основе языка C++ и имеет удобный интерфейс прикладного программирования для Python, а также пользующийся меньшей популярностью интерфейс для C++. Благодаря простым зависимостям TensorFlow может быть быстро развернута в разных архитектурах.

Аналогично Theano (популярная вычислительная библиотека для языка Python, с которой вы уже, наверное, знакомы) вычисления описываются как блок-схемы, отделяя проектирование от реализации. Практически безо всякого труда этот процесс деления на части позволяет один и тот же проект использовать не только в крупномасштабных обучающих системах с тысячами процессоров, но и на мобильных устройствах. Одна такая система охватывает широкий диапазон платформ.

Одним из замечательных свойств TensorFlow является ее способность *автоматического дифференцирования*. Можно экспериментировать с новыми сетями без необходимости переопределения многих ключевых вычислений.

ПРИМЕЧАНИЕ Автоматическое дифференцирование упрощает выполнение обратного распространения ошибки обучения, которое приводит к сложным вычислениям при использовании метода *нейронных сетей*. TensorFlow скрывает мелкие детали обратного распространения, что дает возможность обратить внимание на более важные вопросы. В главе 7 приведено описание нейронных сетей с TensorFlow.

Можно абстрагироваться от всех математических аспектов, так как все они находятся внутри библиотеки. Это напоминает использование базы знаний *WolframAlpha* для поставленной задачи расчета.

Другой особенностью этой библиотеки является ее интерактивная среда визуализации, названная *TensorBoard*. Это средство показывает блок-схему преобразования данных, отображает итоговые журналы и отслеживает ход выполнения программы. На рис. 1.9 приведен пример того, как выглядит TensorBoard в работе. В следующей главе его использование будет описано более подробно.

Создание прототипа в TensorFlow значительно быстрее, чем в Theano (программа инициируется за секунды, а не за минуты), поскольку многие операции поступают предварительно скомпилированными. Благодаря выполнению подграфов становится легче отлаживать программу; весь сегмент программы можно использовать снова без повторных вычислений.

Поскольку TensorFlow используется не только в нейронных сетях, она также содержит готовый пакет матричных вычислений и инструменты манипулирования данными. Большинство библиотек, таких как Torch и Caffe, предназначено исключительно для глубоких нейронных сетей, но TensorFlow является более универсальной, а также обладает возможностью масштабирования.

Эта библиотека хорошо задокументирована и официально поддерживается Google. Машинное обучение является сложным предметом, поэтому только компания с исключительно высокой репутацией может поддерживать TensorFlow.

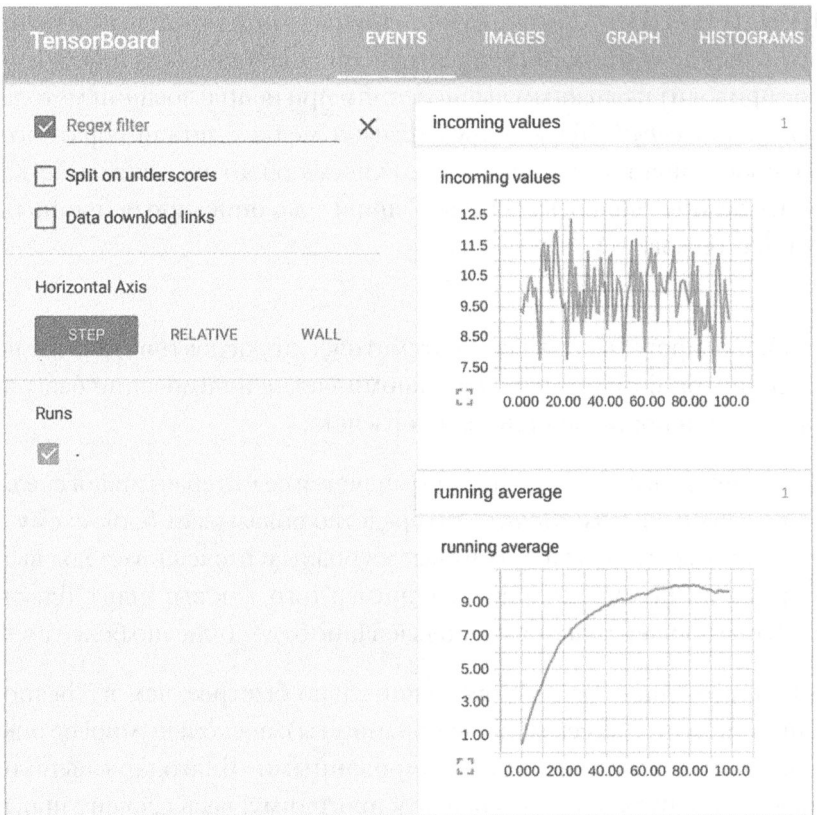


Рис. 1.9. Пример библиотеки TensorBoard в действии

1.6. Обзор предстоящих глав

В главе 2 показано, как использовать различные компоненты TensorFlow (рис. 1.10). В главах 3–6 показано, как использовать классические алгоритмы машинного обучения в TensorFlow, а в главах 7–12 приводится описание алгоритмов на основе нейронных сетей. Эти алгоритмы решают широкий круг задач, таких как прогноз, классификация, кластеризация, уменьшение размерности и планирование.

Различные алгоритмы могут решать одну и ту же задачу реального мира, и многие реальные задачи могут быть решены одним и тем же алгоритмом. В табл. 1.1 представлены алгоритмы, описанные в этой книге.

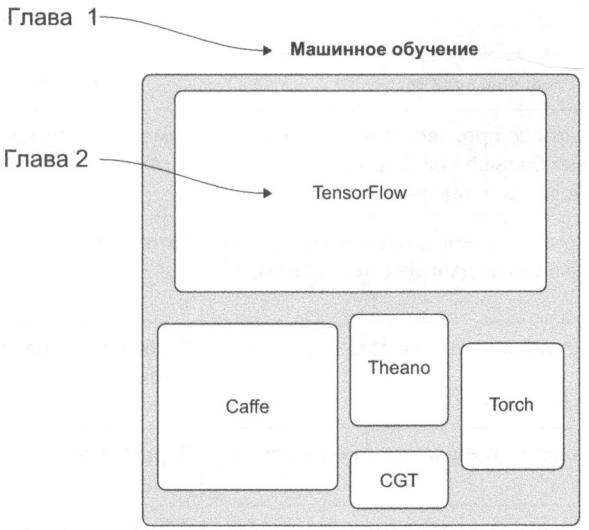


Рис. 1.10. В главе 1 приводятся фундаментальные принципы машинного обучения, а в следующей главе начинается знакомство с библиотекой TensorFlow. Существуют и другие инструменты применения алгоритмов машинного обучения (такие, как Caffe, Theano и Torch), но в главе 2 станет понятно, почему имеет смысл выбрать именно TensorFlow

Таблица 1.1. Многие реальные задачи могут быть решены с помощью алгоритмов, описание которых можно найти в соответствующих главах

Реальные задачи	Алгоритм	Глава
Прогнозирование тенденций, приближение данных с помощью кривой, описание связей между переменными	Линейная регрессия	3
Разделение данных на две категории, нахождение наилучшего способа разделения набора данных	Логистическая регрессия	4
Разделение данных на множество категорий	Многопеременная логистическая регрессия	4
Выявление скрытых причин наблюдаемых явлений, нахождение наиболее вероятной скрытой причины для серии выходных данных	Скрытые марковские модели (алгоритм Витерби)	5
Кластеризация данных по фиксированному числу категорий, автоматическое разделение точек данных на различные классы	Метод *k*-средних	6

Таблица 1.1 (*окончание*)

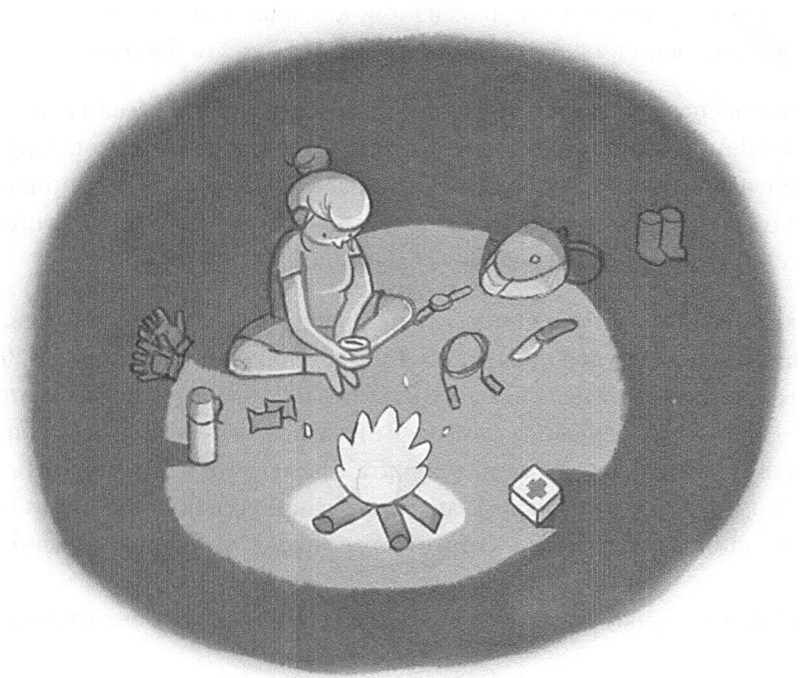
Реальные задачи	Алгоритм	Глава
Кластеризация данных по произвольным категориям, визуализация данных большой размерности с помощью низкоразмерного представления	Самоорганизующаяся карта	6
Уменьшение размерности данных, изучение скрытых переменных, отвечающих за данные с высокой размерностью	Автокодировщик	7
Планирование действий в среде с использованием нейронных сетей (обучение с подкреплением)	Q-обучение в нейронных сетях	8
Классификация данных с помощью контролируемых нейронных сетей	Перцептрон	9
Классификация реальных изображений с использованием контролируемых нейронных сетей	Сверточная нейронная сеть	9
Создание образов, которые совпадают с наблюдениями, с использованием нейронных сетей	Рекуррентная нейронная сеть	10
Прогнозирование ответов на естественном языке на запросы на естественном языке	Модель seq2seq	11
Обучение ранжированию элементов в результате изучения их полезности	Ранжирование	12

СОВЕТ Если вас интересуют сложные детали архитектуры TensorFlow, наилучшим источником для ознакомления с ними является официальная документация, представленная на сайте www.tensorflow.org/ extend/architecture. Эта книга делает рывок вперед и использует TensorFlow, не показывая всю настройку на низком уровне работы системы. Тем, кто заинтересован в облачных службах, можно посоветовать рассмотреть эту библиотеку Google для масштаба и скорости профессионального уровня: https://cloud.google.com/products/machine-learning/.

1.7. Краткие итоги

- TensorFlow стал инструментом, который выбирают профессионалы и исследователи для применения методов машинного обучения.
- Машинное обучение использует примеры для создания экспертной системы, которая может делать заключения по новым входным данным.
- Основной характеристикой машинного обучения является то, что эффективность обучения растет при добавлении новых обучающих данных.
- На протяжении многих лет ученые разрабатывали три основные архитектуры, которые подходят для решения большинства задач: обучение с учителем, обучение без учителя и обучение с подкреплением.
- После формулировки реальных задач для машинного обучения появилось несколько алгоритмов. Из многих библиотек программного обеспечения и прикладной среды для реализации проектов мы выбрали TensorFlow как наиболее подходящий вариант библиотеки. Разработанная Google и поддерживаемая сообществом пользователей интернета, TensorFlow предоставляет наиболее простой вариант использования программ промышленного стандарта.

Основы TensorFlow



Эта глава охватывает следующие темы:

- ✓ Ознакомление с рабочей схемой TensorFlow
- ✓ Создание интерактивных блокнотов Jupyter
- ✓ Визуализация алгоритмов с помощью TensorBoard

Прежде чем внедрять алгоритмы машинного обучения, давайте сначала познакомимся с тем, как использовать TensorFlow. Вам придется засучить рукава и написать простой код прямо сейчас! В этой главе описываются некоторые значительные преимущества TensorFlow, которые убедят вас в том, что для машинного обучения лучше всего подходит именно эта библиотека.

В качестве эксперимента давайте посмотрим, что получится, если использовать код на Python без доступной компьютерной библиотеки. Это все равно что использовать новый смартфон без установки дополнительных приложений. Функциональность будет обеспечена, но эффективность работы с правильно подобранными приложениями была бы намного выше.

Предположим, что вы, будучи владельцем частного бизнеса, отслеживаете интенсивность продаж своей продукции. Каталог продукции состоит из 100 пунктов, и вы представляете цену каждого пункта в виде вектора под названием цена (`price`). Другой 100-мерный вектор носит название объем (`amounts`) и представляет собой подсчет запаса каждого пункта. Вы можете написать часть кода на Python, как показано в следующем листинге, чтобы рассчитать доход от продажи всей продукции. Не забывайте, что этот код не импортирует ни одну библиотеку.

Листинг 2.1. Вычисление скалярного произведения двух векторов без использования библиотеки

```
revenue = 0
for price, amount in zip(prices, amounts):
    revenue += price * amount
```

Этот код только вычисляет скалярное произведение двух векторов. Представьте, какой большой код потребуется для чего-либо более сложного, например для решения системы линейных уравнений или расчета расстояния между двумя векторами.

При установке библиотеки TensorFlow вы также устанавливаете хорошо известную и надежную библиотеку Python под названием NumPy, которая облегчает математические манипуляции в Python. Использовать Python без его библиотек (NumPy и TensorFlow) — все равно что использовать камеру без автоматического режима съемки: с ней вы приобретаете больше универсальности, но при этом можете совершать случайные ошибки (для протокола: мы ничего не имеем против фотографов, которые настраивают апертуру, диафрагму и ISO). При машинном обучении легко допустить ошибку, поэтому мы будем использовать фотокамеру с автофокусировкой и библиотеку TensorFlow, чтобы помочь автоматизировать утомительную разработку программного обеспечения.

Следующий листинг показывает, как быстро можно написать это же скалярное произведение с помощью NumPy.

Листинг 2.2. Вычисление скалярного произведения с помощью NumPy

```
import numpy as np
revenue = np.dot(prices, amounts)
```

Python является лаконичным языком. Это означает, что в книге нет многочисленных страниц с кодами. С другой стороны, лаконичность языка Python также подразумевает, что вам придется познакомиться при прочтении этой главы с тем, что происходит после ввода каждой строки.

Алгоритмы машинного обучения требуют выполнения многих математических операций. Часто алгоритм сводится к нескольким простым функциям, выполняемым итеративно до сходимости. Поэтому можно использовать любой стандартный язык программирования для выполнения этих вычислений, но основой управляемого и высокопроизводительного кода является хорошо написанная библиотека, такая как TensorFlow (которая официально поддерживает языки Python и C++).

СОВЕТ Подробную документацию о различных функциях для интерфейсов прикладных программ на Python и C++ APIs можно найти на сайте www.tensorflow.org/api_docs/.

Навыки, которые можно приобрести в этой главе, направлены на использование для вычислений библиотеки TensorFlow, так как машинное обучение основано на математических формулировках. После знакомства с примерами и листингами вы сможете использовать TensorFlow для произвольных задач, таких как статистические расчеты с большим объемом данных. Акцент в этой главе делается на использовании TensorFlow, а не на машинном обучении. Это похоже на плавное трогание автомобиля с места, верно?

Позже в этой главе вы будете использовать основные функции TensorFlow, которые важны для машинного обучения. Они включают в себя представление вычислений в виде блок-схем, разделение проектирования и выполнения, частичные вычисления подграфа и автоматическое дифференцирование. Не мудрствуя лукаво, давайте напишем ваш первый фрагмент кода с помощью библиотеки TensorFlow!

2.1. Убедитесь, что TensorFlow работает

Прежде всего, необходимо проверить, все ли работает правильно. Проверьте уровень масла, замените сгоревший предохранитель в подвале и убедитесь, что остаток по кредиту нулевой. Это все шуточки, а мы продолжаем разговор о TensorFlow.

Прежде чем начать, ознакомьтесь с инструкциями процедур пошаговой установки, описанными в приложении. Для начала необходимо создать новый файл с названием test.py. Импортируйте библиотеку TensorFlow, запустив следующий скрипт:

```
import tensorflow as tf
```

ВОЗНИКЛИ ТЕХНИЧЕСКИЕ ЗАТРУДНЕНИЯ?

На этом шаге ошибки обычно возникают, если вы установили версию для GPU (графического процессора) и библиотека не может найти драйверы CUDA. Помните, если вы скомпилировали библиотеку с CUDA, необходимо обновить переменные среды, добавив пути к CUDA. Посмотрите инструкцию CUDA для TensorFlow (дополнительную информацию см. по ссылке: http://mng.bz/QUMh).

Эта процедура однократного импорта подготавливает TensorFlow для дальнейшей работы. Если интерпретатор Python не «ругается», можно начинать использование TensorFlow!

ПРИДЕРЖИВАЯСЬ СОГЛАШЕНИЙ TENSORFLOW

TensorFlow импортируется вместе с алиасами `tf`. В общем случае использование библиотеки TensorFlow вместе с `tf` позволяет поддерживать согласованность с другими разработчиками и проектами на TensorFlow с открытым исходным кодом. Конечно, можно использовать другие алиасы (или не использовать их совсем), но тогда применять фрагменты кода TensorFlow других людей в своих собственных проектах станет значительно сложнее.

2.2. Представление тензоров

Теперь вы знаете, как импортировать TensorFlow в исходный файл Python; давайте начнем использовать этот файл! Как было сказано в предыдущей главе, удобным способом описать объект реального мира является перечисление его свойств или признаков. Например, можно описать машину с помощью ее цвета, модели, типа двигателя, пробега и т. п. Упорядоченный перечень признаков носит название *вектора признаков*, и именно его нужно будет представить в коде.

Векторы признаков являются одним из самых полезных компонентов машинного обучения, что обусловлено их простотой (они являются списком чисел). Каждый элемент данных обычно состоит из вектора признаков, а хорошая база данных содержит сотни, если не тысячи, таких векторов. Нет сомнений, что вам часто придется работать одновременно более чем с одним вектором признаков. Матрица лаконично представляет список всех векторов, при этом каждый ее столбец является вектором признаков.

Синтаксисом представления матриц в библиотеке TensorFlow является вектор векторов, каждый из которых имеет одну и ту же длину. На рис. 2.1 приведен пример матрицы с двумя строками и тремя столбцами, такими как [[1, 2, 3], [4, 5, 6]]. Обратите внимание, что этот вектор содержит два элемента, а каждый элемент соответствует строке этой матрицы.

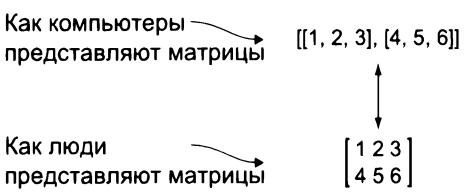


Рис. 2.1. Матрица в нижней половине диаграммы представляет собой визуализацию из ее компактной нотации кода в верхней половине диаграммы. Эта форма обозначения является общей парадигмой в большинстве научных вычислительных библиотек

Мы обращаемся к элементу матрицы, указав номер строки и столбца, на которых он находится. Например, первая строка и первый столбец указывают самый первый элемент матрицы в ее левом верхнем углу. Иногда удобно использовать более двух указателей, как при обращении к пикселу цветного изображения, указывая не только его строку и столбец, но и его канал (красный/зеленый/синий). *Тензором* называют обобщение матрицы, которое определяет элемент по его произвольному числу указателей.

ПРИМЕР ТЕНЗОРА

Представим себе начальную школу, в которой за каждым учеником строго закреплено определенное место. Вы — директор школы и плохо запоминаете имена. К счастью, для каждого класса составлена сетка мест, и вы

можете без труда присвоить учащимся псевдонимы по индексам строк и столбцов их мест в этой сетке.

В школе несколько классов, так что вы не можете просто сказать: «Доброе утро, номер 4,10! Продолжай так же хорошо учиться!» Вам необходимо также указать класс: «Привет, номер 4,10 из 2-го класса». В отличие от матрицы, использующей два индекса для определения элемента, учащимся в этой школе необходимо три индекса. Все они являются частью тензора 3-го ранга!

Синтаксическая структура тензора является еще большим вложенным вектором. Например, как показано на рис. 2.2, тензор 2 × 3 × 2 представляет собой сочетание [[[1,2], [3,4], [5,6]], [[7,8], [9,10], [11,12]]], которое можно рассматривать как две матрицы, каждая размером 3 × 2. Следовательно, мы можем сказать, что этот тензор имеет *ранг* 3. В общем случае рангом тензора является число указателей, требуемых для идентификации того или иного элемента. Алгоритмы машинного обучения в библиотеке TensorFlow действуют на тензоры, поэтому важно понять, как их следует использовать.

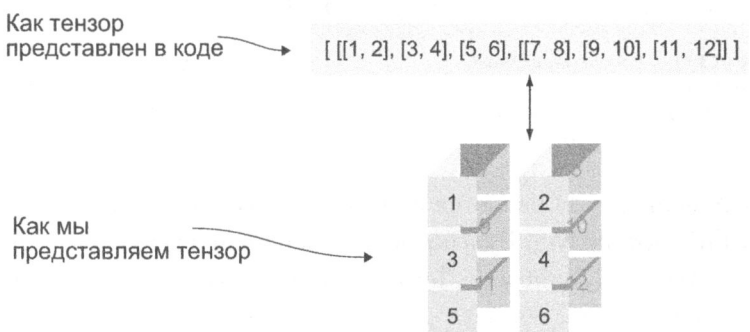


Рис. 2.2. Этот тензор можно представить как множество матриц, уложенных друг на друга. Чтобы указать элемент, необходимо задать номер строки и столбца, а также то, в какой именно матрице он расположен. Поэтому ранг этого тензора равен 3

Очень легко заблудиться в многочисленных способах представления тензора. Можно догадаться, что три строчки кода в листинге 2.3 пытаются представить одну и ту же матрицу 2 × 2. Эта матрица содержит два вектора признаков каждой размерности по два. Вектор может, например, представлять оценки двух фильмов двумя людьми. Человек, пронумерованный в строке матрицы, дает

оценку фильму, который имеет свой номер в столбце. Запустим код, чтобы увидеть, как создается матрица в TensorFlow.

Листинг 2.3. Различные способы представления тензоров

```
import tensorflow as tf
import numpy as np

m1 = [[1.0, 2.0],
      [3.0, 4.0]]

m2 = np.array([[1.0, 2.0],
               [3.0, 4.0]], dtype=np.float32)

m3 = tf.constant([[1.0, 2.0],
                  [3.0, 4.0]])

print(type(m1))
print(type(m2))
print(type(m3))

t1 = tf.convert_to_tensor(m1, dtype=tf.float32)
t2 = tf.convert_to_tensor(m2, dtype=tf.float32)
t3 = tf.convert_to_tensor(m3, dtype=tf.float32)

print(type(t1))
print(type(t2))
print(type(t3))
```

Теперь матрицы NumPy видны в библиотеке TensorFlow

Задает тремя способами матрицу 2 × 2

Выводит тип каждой матрицы

Создает тензорные объекты разного типа

Обратите внимание, что типы будут теперь теми же

Первая переменная (`m1`) является списком, вторая переменная (`m2`) является `ndarray` из библиотеки NumPy, а последняя переменная (`m3`) является константой объекта `Tensor` библиотеки TensorFlow, который вы инициализировали, используя оператор `tf.constant`.

Все операторы в библиотеке TensorFlow, такие как `negative`, предназначены для работы с тензорными объектами. Удобная функция `tf.convert_to_tensor( ... )`, которую можно вставлять где угодно, чтобы быть уверенным, что вы работаете с тензорами, а не с другими типами. Большинство функций в библиотеке TensorFlow уже выполняют эту функцию (избыточно), даже если вы забыли сделать это. Использование `tf.convert_to_tensor( ... )` не обязательно, но мы показываем ее здесь, потому что она помогает понять систему неявных типов, используемых в этой библиотеке. Листинг 2.3 трижды выдает следующее:

```
<class 'tensorflow.python.framework.ops.Tensor'>
```

СОВЕТ Вы можете найти листинги кодов на веб-сайте книги, чтобы упростить копирование и вставку кода: www.manning.com/books/machine-learning-with-tensorflow.

Давайте посмотрим на задание тензоров в коде еще раз. После импорта библиотеки TensorFlow вы можете пользоваться оператором `tf.constant` следующим образом (в листинге приводятся два тензора разной размерности).

Листинг 2.4. Создание тензоров

```
import tensorflow as tf

m1 = tf.constant([[1., 2.]])  ◄—— Задает матрицу 2 × 1

m2 = tf.constant([[1],
                  [2]])  ◄—— Задает матрицу 1 × 2

m3 = tf.constant([ [[1,2],
                    [3,4],
                    [5,6]],
                    [[7,8],
                    [9,10],
                    [11,12]] ])  ◄—— Задает тензор 3-го ранга

print(m1)
print(m2)   Попробуем вывести тензор
print(m3)
```

Выполнение кода из листинга 2.4 дает следующий результат:

```
Tensor( "Const:0",
        shape=TensorShape([Dimension(1), Dimension(2)]),
        dtype=float32 )
Tensor( "Const_1:0",
        shape=TensorShape([Dimension(2), Dimension(1)]),
        dtype=int32 )
Tensor( "Const_2:0",
        shape=TensorShape([Dimension(2), Dimension(3), Dimension(2)]),
        dtype=int32 )
```

Из приведенного выхода видно, что каждый тензор представлен объектом с подходящим названием `Tensor`. Каждый объект `Tensor` имеет принадлежащую только ему метку (`name`), размер (`shape`) (для определения его структуры)

и тип данных (`dtype`) (для указания типа величин, которыми вы собираетесь манипулировать). Поскольку в явном виде имя не приводится, библиотека автоматически генерирует имена: `Const:0`, `Const_1:0` и `Const_2:0`.

ТИПЫ ТЕНЗОРОВ

Обратите внимание, что каждый элемент `m1` заканчивается десятичным знаком. Этот знак указывает транслятору Python, что тип этих элементов не является целочисленным и что они относятся к типу чисел с плавающей запятой. Вы можете перейти к величинам явного типа `dtype`. Во многом аналогично массивам NumPy, тип данных тензора указывает тип величин, которыми вы собираетесь манипулировать в этом тензоре.

TensorFlow импортируется вместе с несколькими конструкторами простых тензоров. Например, `tf.zeros(shape)` создает тензор определенной формы, все элементы которого равны нулю. Аналогично этому, `tf.ones(shape)` создает тензор определенной формы, все элементы которого равны единице. Аргумент `shape` является одномерным (1D) тензором типа `int32` (список целых чисел), описывающим размеры тензора.

УПРАЖНЕНИЕ 2.1

Инициализируйте тензор 500 × 500 со всеми элементами, равными 0,5.

ОТВЕТ

```
tf.ones([500,500]) * 0.5
```

2.3. Создание операторов

Теперь, когда у вас уже есть несколько готовых к использованию тензоров, можно применять более интересные операторы, такие как сложение или умножение. Пусть каждая строка матрицы представляет собой перевод денег одному человеку (положительная величина) от другого человека (отрицательная величина). Операция изменения знака элементов этой матрицы является

способом представить историю финансовых операций денежного потока другого человека. Давайте начнем с простого и выполним операцию изменения знака на тензоре `m1` из листинга 2.4. Операция изменения знака элементов матрицы превращает положительные числа в отрицательные и наоборот, не изменяя их по модулю.

Операция изменения знака является одной из наиболее простых операций. Как показано в листинге 2.5, операция изменения знака берет на входе один тензор и создает другой со всеми измененными знаками. Попробуем выполнить код. Если вы освоили, как задать операцию изменения знака, этот навык можно будет применить ко всем операциям в библиотеке TensorFlow.

> **ПРИМЕЧАНИЕ** *Задать* операцию, такую как изменение знака, и *выполнить* ее — совсем не одно и то же. Пока что вы *задали*, как операция должна выполняться. В разделе 2.4 вы сможете *выполнить* эти операции, чтобы получить определенное значение.

Листинг 2.5. Использование оператора изменения знака

```
import tensorflow as tf

x = tf.constant([[1, 2]])      ◄── Задает произвольный тензор
negMatrix = tf.negative(x)     ◄── Меняет знак элементов тензора
print(negMatrix)               ◄── Выводит объект
```

Листинг 2.5 выдает следующее:

```
Tensor("Neg:0", shape=TensorShape([Dimension(1), Dimension(2)]), dtype=int32)
```

Обратите внимание, что на выходе будет не `[[-1, -2]]`. Это связано с тем, что вы печатаете определение операции изменения знака, а не фактический результат выполнения операции. Напечатанные выходные данные показывают, что операция изменения знака является классом `Tensor` с именем, формой и типом данных. Это имя присваивается автоматически, но вы также могли бы присвоить его в явном виде при применении оператора `tf.negative` в листинге 2.5. Аналогичным образом форма и тип данных были получены из `[[1, 2]]`, которые вы ввели.

ПОЛЕЗНЫЕ ОПЕРАТОРЫ TENSORFLOW

В официальной документации приводятся все доступные математические операторы: www.tensorflow.org/api_guides/python/math_ops. Ниже приведены примеры используемых операторов:

`tf.add(x, y)` — складывает два тензора одного типа, x + y;

`x + y tf.subtract(x, y)`— вычитает тензоры одного типа, x – y;

`x – y tf.multiply(x, y)`— перемножает поэлементно два тензора;

`tf.pow(x, y)`— возводит поэлементно *x* в степень *y;*

`tf.exp(x)`— аналогичен *pow(e, x)*, где *e* — число Эйлера (2,718 ...);

`tf.sqrt(x)`— аналогичен *pow(x, 0,5)*;

`tf.div(x, y)`— выполняет поэлементно деление x и y;

`tf.truediv(x, y)`— то же самое, что и `tf.div`, кроме того, что приводит аргументы как числа с плавающей запятой;

`tf.floordiv(x, y)`— то же самое, что и `truediv`, но с округлением окончательного результата до целого числа;

`tf.mod(x, y)`— берет поэлементно остаток от деления.

УПРАЖНЕНИЕ 2.2

Используйте операторы TensorFlow, с которыми вы уже познакомились, чтобы получить гауссово распределение (также известное как нормальное распределение). См. рис. 2.3 в качестве подсказки. Дополнительную информацию по плотности вероятности нормального распределения вы можете найти в Сети: https://ru.wikipedia.org/wiki/Нормальное_распределение.

ОТВЕТ

Большинство математических выражений, таких как ×, –, +, и т. п., являются сокращениями их эквивалентов в библиотеке TensorFlow, которые используются для краткости изложения. Гауссова функция содержит много операторов, поэтому лучше использовать следующие краткие обозначения:

```
from math import pi
mean = 0.0
sigma = 1.0
(tf.exp(tf.negative(tf.pow(x – mean, 2.0) /
                    (2.0 * tf.pow(sigma, 2.0) ))) *
 (1.0 / (sigma * tf.sqrt(2.0 * pi) )))
```

2.4. Выполнение операторов во время сеанса

Сеанс (session) является средой системы программного обеспечения, которая описывает, как должны выполняться строки кода. В библиотеке TensorFlow во время сеанса определяется, как аппаратные средства (такие, как центральный процессор и графический процессор) взаимодействуют между собой. Таким образом, можно спроектировать алгоритм машинного обучения без необходимости полного контроля над аппаратными средствами, с помощью которых этот алгоритм выполняется. Вы можете затем настроить конфигурацию сеанса, чтобы изменить его выполнение, не меняя строк кода машинного обучения.

Для выполнения операций и для получения обратно рассчитанных значений библиотеке TensorFlow требуется сеанс. Только зарегистрированный сеанс может заполнить значения объекта `Tensor`. Для этого необходимо создать определенный класс сеанса, используя `tf.Session()`, и сообщить о необходимости выполнить оператор, как это показано в следующем листинге. Результатом будет значение, которое затем можно будет использовать для дальнейших вычислений.

Листинг 2.6. Использование сеанса

```
import tensorflow as tf

x = tf.constant([[1., 2.]])  ◄── Задает произвольную матрицу
neg_op = tf.negative(x)  ◄── Выполняет на ней оператор изменения знака

with tf.Session() as sess:  ◄── Начинает сеанс, чтобы можно было выполнить операции
    result = sess.run(negMatrix)  ◄── Сообщает в сеансе о необходимости оценить negMatrix
print(result)  ◄── Выводит результирующую матрицу
```

Наши поздравления! Вы только что написали целиком свой первый код TensorFlow. Хотя все, что он делает, — это меняет знаки у элементов матрицы, чтобы получить в результате `[[-1, -2]]`, при этом ядро и фреймворк у него такие же, как и все остальное в библиотеке TensorFlow. Сеанс не только определяет, *где* ваш код будет выполняться на машине, но и также планирует, *как* будут выполняться соответствующие вычисления для возможности их распараллеливания.

ИСПОЛНЕНИЕ КОДА КАЖЕТСЯ НЕСКОЛЬКО МЕДЛЕННЫМ

Возможно, вы заметили, что код выполняется на несколько секунд дольше, чем ожидалось. Может казаться неестественным, что TensorFlow тратит лишние секунды для изменения знака элементов небольшой матрицы. Но значительная предварительная обработка выполняется для оптимизации библиотеки для крупных и сложных вычислений.

Каждый объект `Tensor` имеет функцию `eval()` для выполнения математических операций, которые определяют это значение. Но функция `eval()` требует определить объект сеанса для библиотеки, чтобы понять, как лучше всего использовать базовое аппаратное обеспечение. В листинге 2.6 мы использовали `sess.run(…)`, который является эквивалентом вызова функции `Tensor eval()` в ходе выполнения сеанса.

При выполнении кода TensorFlow с помощью интерактивной среды (для отладки или представления) часто легче создать сеанс в интерактивном режиме, в котором сеанс в неявном виде является частью любого вызова функции `eval()`. Таким образом, переменную сеанса можно не пропускать через весь код, что дает возможность сосредоточиться на значимой части алгоритма, как это видно из следующего листинга.

Листинг 2.7. Применение интерактивного режима сеанса

```
import tensorflow as tf
sess = tf.InteractiveSession()

x = tf.constant([[1., 2.]])
negMatrix = tf.negative(x)

result = negMatrix.eval()
print(result)

sess.close()
```

Начинает интерактивный сеанс, поэтому переменную сеанса больше не требуется упоминать

Задает произвольную матрицу и изменяет знак ее элементов

Теперь можно вычислить negMatrix, не задавая явным образом сеанс

Выводит матрицу с измененным знаком ее элементов

Не забудьте закрыть сеанс, чтобы освободить ресурсы

2.4.1. Представление кода как графа

Представьте врача, который предсказал, что вес новорожденного составит 3,4 килограмма. Вам хотелось бы определить, насколько это отличается от фактического веса новорожденного. Будучи инженером с аналитическим складом ума, вы построите функцию для описания вероятности всех возможных вариантов веса новорожденного. Например, 3,6 килограмма более вероятно, чем 4,5 килограмма.

Вы можете остановить свой выбор на гауссовой функции (или нормальной) распределения вероятности. Она берет в качестве входного значения число и выдает неотрицательное число, описывающее вероятность наблюдать это входное значение. Эта функция используется в течение всего времени машинного обучения, и ее легко задать в библиотеке TensorFlow. Она использует операции умножения, деления, изменения знака и еще пару других базовых операций.

Представим каждый оператор как узел графа. Всякий раз, когда вы видите символ «плюс» (+) или любой другой математический символ, просто нарисуйте его как один из многих узлов. Ребра между этими узлами представляют содержание математических функций. В частности, оператор `negative`, который мы изучали, является узлом, а входящие и выходящие ребра этого узла описывают, как преобразуется тензор. Тензор проходит через этот граф, и нам становится понятно, почему эту библиотеку назвали TensorFlow!

Важно отметить, что каждый оператор является строго типизированной функцией, которая берет входные тензоры определенной размерности и создает на выходе тензоры такой же размерности. На рис. 2.3 приведен пример того, как можно создать гауссову функцию, использовав библиотеку TensorFlow. Эта функция представлена как граф, в котором операторы являются узлами, а ребра представляют взаимодействия между узлами. Этот граф в целом представляет собой сложную математическую функцию (гауссову функцию). Небольшие фрагменты этого графа представляют собой простые математические понятия, такие как изменение знака или удвоение.

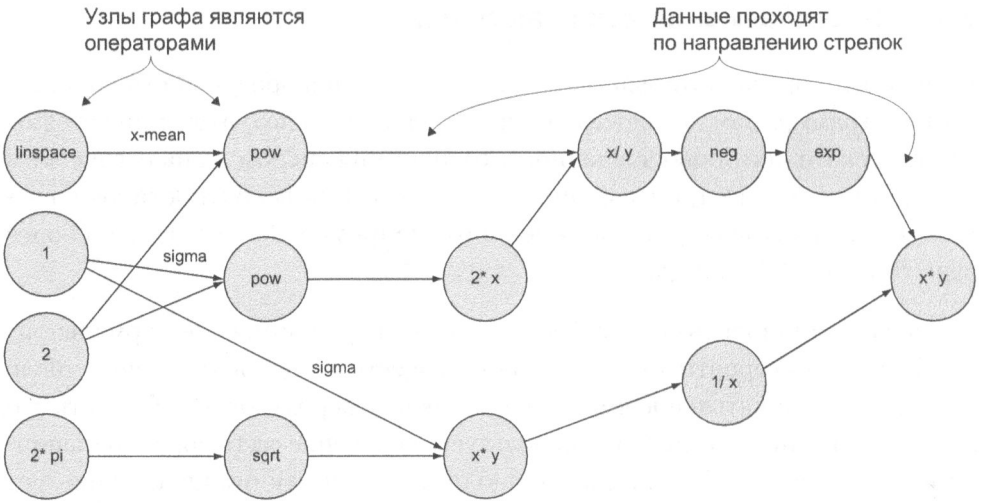


Рис. 2.3. Граф, представляющий операции, необходимые для получения гауссова распределения. Связи между узлами представляют то, как данные переходят от одного оператора к следующему. Сами операции простые, а сложности возникают в результате их переплетений

Алгоритмы библиотеки TensorFlow легко визуализировать. Их можно описать с помощью блок-схем. Техническим термином (и более правильным) для такой блок-схемы является *граф потока данных* (dataflow graph). Каждую стрелку графа потока данных называют *ребром*. А каждое состояние потока данных называют узлом. Целью сеанса является представить вашу программу на языке Python в виде графа потока данных, а затем привязать вычисление каждого узла графа к центральному или графическому процессору.

2.4.2. Настройка конфигурации сеансов

Вы также можете передать все возможные варианты `tf.Session`. Например, TensorFlow автоматически определяет лучший вариант привязки центрального или графического процессора к операции в зависимости от того, какой из них доступней. Вы можете также передать при создании сеанса дополнительный вариант, `log_device_placements=True`, что покажет вам, на каком именно устройстве будут выполняться вычисления (следующий листинг).

Листинг 2.8. Регистрация сеанса

```
import tensorflow as tf

x = tf.constant([[1., 2.]])
negMatrix = tf.negative(x)

with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
    result = sess.run(negMatrix)

print(result)
```

Задает произвольную матрицу и изменяет знак ее элементов

Начинает сеанс в специальной конфигурации, встроенной в конструктор, чтобы можно было провести регистрацию

Вычисляет negMatrix

Выводит результирующее значение

В выходных данных сообщается о том, какие именно устройства (ЦП или ГП) используются в сеансе для выполнения каждой операции. Например, выполнение кода из листинга 2.8 приводит к записям выходных данных (приведены далее). Это показывает, какое именно устройство было использовано для выполнения операции по изменению знака:

```
Neg: /job:localhost/replica:0/task:0/cpu:0
```

Сеансы являются важной частью TensorFlow. Вам необходимо вызвать сеанс, чтобы выполнить математические операции. На рис. 2.4 показано, как компоненты библиотеки TensorFlow взаимодействуют между собой в процессе машинного обучения. Во время сеанса не только выполняются операции графа, но и в качестве входных данных используются также переменные-заполнители, обычные переменные и константы. До сих пор мы использовали только константы, но в следующих разделах мы начнем использовать переменные и переменные-заполнители. Далее приводится краткий обзор этих трех типов переменных:

- *Переменная-заполнитель* — переменная, которой не присвоено значение, но которая будет инициализирована в активном сеансе. Обычно переменные-заполнители используют в качестве входных и выходных данных модели.
- *Переменная* — величина, которая может меняться, так же как параметры модели машинного обучения. Переменные можно инициализировать в сеансе до того, как их начнут использовать.
- *Константа* — величина, значение которой не меняется (как гиперпараметры или настройки).

Вся схема машинного обучения с библиотекой TensorFlow представлена на рис. 2.4. Большая часть кода в TensorFlow состоит из настройки графа и сеанса. После того как спроектирован граф и подключен сеанс для его выполнения, ваш код готов к использованию!

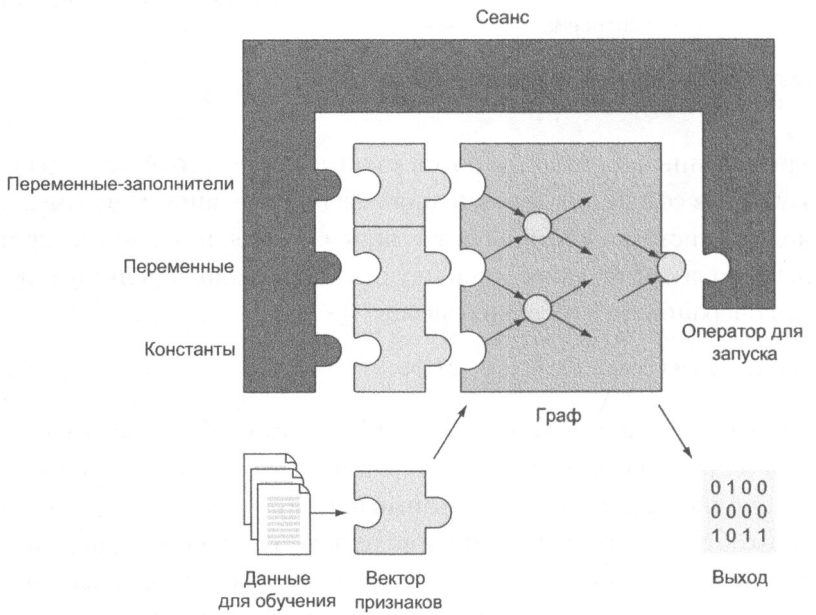


Рис. 2.4. Сеанс указывает, как использовать аппаратные средства для наиболее эффективной обработки графа. После начала работы сеанса к каждому узлу привязывается центральный или графический процессор. После обработки сеанс выдает данные в пригодном для использования формате, таком как, например, массив NumPy. Сеанс дополнительно может использовать переменные-заполнители, обычные переменные и константы

2.5. Написание кода в Jupyter

Поскольку TensorFlow является в первую очередь библиотекой Python, вам следует в полном объеме использовать интерпретатор Python. *Jupyter Notebook* является полноценной средой для использования интерактивной природы этого языка. Это веб-приложение, которое элегантно отображает вычисления, чтобы вы могли поделиться аннотированными интерактивными алгоритмами с другими пользователями, изучить технику или продемонстрировать код.

Вы можете делиться блокнотами Jupyter с другими пользователями для обмена идеями и для загрузки других блокнотов с целью изучения кода. В приложении есть инструкции, позволяющие выполнить установку Jupyter.

С нового терминала измените директорию того места, где вы хотели бы попрактиковаться в использовании кода TensorFlow, и запустите сервер Jupyter Notebook:

```
$ cd ~/MyTensorFlowStuff
$ jupyter notebook
```

Выполнение этой команды должно запустить новое окно с панелью инструментов Jupyter Notebook. Если никакое окно не откроется автоматически, необходимо будет вручную перейти на сайт http://localhost:8888. Вы увидите страничку, аналогичную представленной на рис. 2.5.

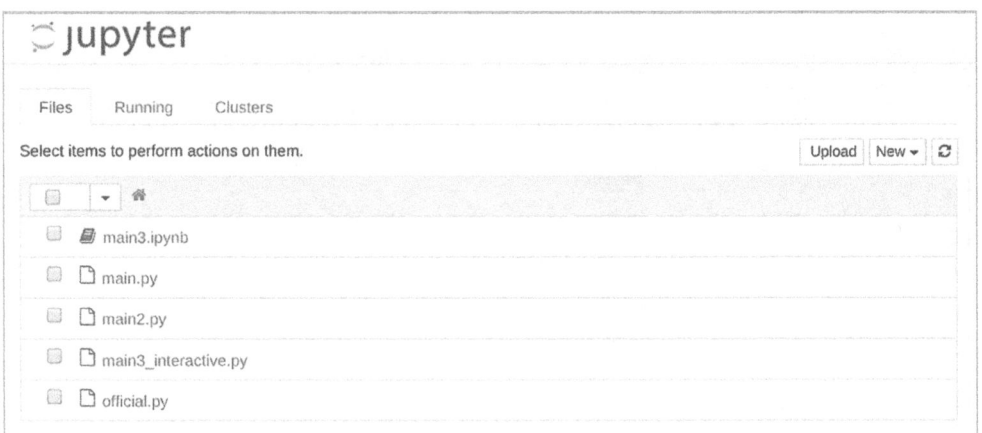


Рис. 2.5. Выполнение Jupyter Notebook запустит интерактивный блокнот Jupyter на http://localhost:8888.

Создайте новую интерактивную среду, щелкнув по находящемуся в правом верхнем углу раскрывающемуся меню New (Новый); затем выберите Notebooks > Python 3. Это создаст новый файл с названием Untitled.ipynb, который можно сразу же начать редактировать, используя интерфейс системы просмотра (браузера). Можно изменить имя Jupiter notebook, щелкнув на текущем названии (Untitled) и набрав что-нибудь запоминающееся, например «Тестовый блокнот TensorFlow».

Все, что есть в Jupyter Notebook, является независимой частью кода или текста, которая называется *ячейкой* (cell). Ячейки помогают делить длинные блоки кода на управляемые фрагменты и документации. Ячейки можно выполнять независимо или запускать все одновременно, но в определенном порядке. Обычно используют три способа выполнить ячейку:

- Нажатие Shift-Enter на ячейке приводит к выполнению ячейки и выделяет следующую за ней ячейку.
- Нажатие Ctrl-Enter сохраняет курсор на текущей ячейке после ее выполнения.
- Нажатие Alt-Enter выполняет ячейку и затем вставляет прямо под ней новую пустую ячейку.

Тип ячейки можно изменить, вызвав раскрывающееся меню на панели инструментов, как показано на рис. 2.6. В качестве альтернативы можно нажать Esc и выйти из режима редактирования, использовать кнопки со стрелками для

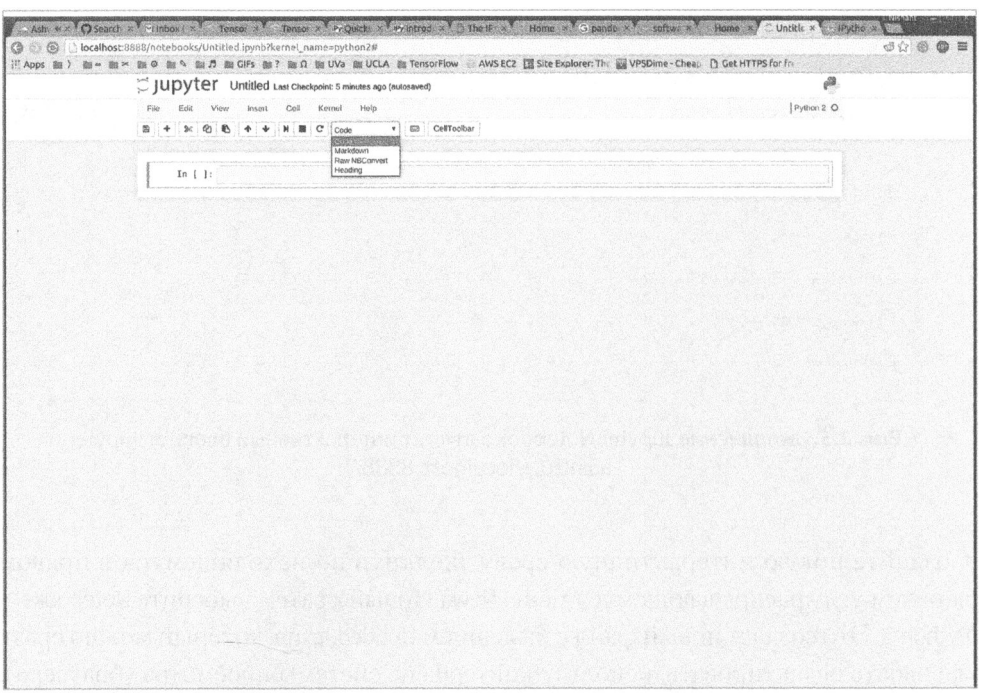


Рис. 2.6. Раскрывающееся меню для замены типа ячейки в Jupiter Notebook. Ячейка кода Python, когда для описания текста используется режим разметки

выделения ячейки и нажать Y, чтобы изменить режим программы, или M для режима разметки. И наконец, можно создать блокнот Jupyter, который удобным образом демонстрирует код TensorFlow с помощью чередования ячеек программы и текста, как показано на рис. 2.7.

УПРАЖНЕНИЕ 2.3

Если вы внимательно посмотрите на рис. 2.7, то заметите, что вместо `tf.neg` там указано `tf.negative`. Это странно. Можете объяснить, для чего это было сделано?

ОТВЕТ

Вам следует знать, что в библиотеке TensorFlow было изменено соглашение об именах, и с этим можно столкнуться, следуя старым обучающим инструкциям TensorFlow в Сети.

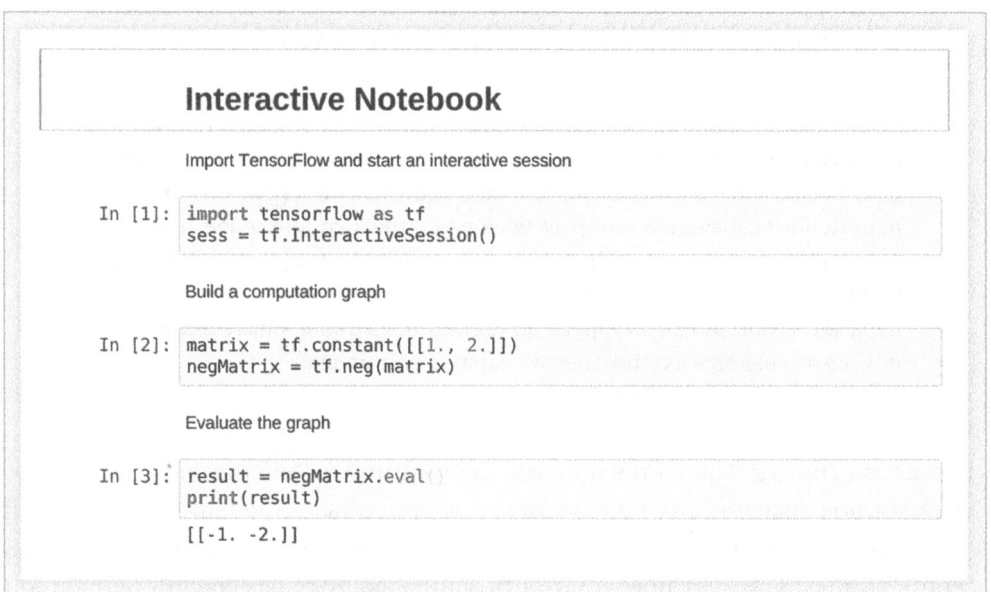


Рис. 2.7. Интерактивный блокнот Python представляет собой код и комментарии, которые были сгруппированы для удобства чтения

2.6. Использование переменных

Использование констант в TensorFlow является хорошим началом, но наиболее интересные приложения требуют изменения данных. Например, нейробиолог может заинтересоваться выявлением нейроактивности на основе измерений датчика. Пик нейроактивности может быть булевой переменной, которая меняется с течением времени. Для того чтобы это захватить в TensorFlow, можно использовать класс `Variable` (переменная) для указания узла, значение которого со временем меняется.

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОБЪЕКТА VARIABLE (ПЕРЕМЕННАЯ) В МАШИННОМ ОБУЧЕНИИ

В следующей главе будет подробно описана классическая задача машинного обучения — найти уравнение линии, которая является наилучшим приближением множества точек данных. Алгоритм начинает работать с начального приближения, которое является уравнением, задаваемым параметрами (такими, как наклон и точка пересечения с осью *Y*). Спустя некоторое время алгоритм генерирует значения этих чисел (их также называют параметрами), обеспечивая все более точную аппроксимацию кривой этих точек.

До сих пор мы манипулировали только константами. Программы только с константами не так интересны для реальных приложений, поэтому TensorFlow дает возможность использовать более «богатый» инструмент, такой как переменные, значение которых может меняться со временем. Алгоритм машинного обучения подстраивает эти параметры модели, пока не будет найдено оптимальное значение каждого из них. При машинном обучении параметры обычно флуктуируют до тех пор, пока в конечном счете процесс поиска не сведется к оптимальным параметрам модели.

Код из листинга 2.9 является простой программой из библиотеки TensorFlow, которая показывает, как использовать переменные. Значение переменной обновляется всякий раз, когда внезапно увеличивается какое-то из значений серии данных. Следует предусмотреть запись данных измерений нейронной активности на определенном промежутке времени. Это позволит обнаруживать,

когда именно внезапно возрастает активность нейронов. Конечно, этот алгоритм чрезмерно упрощен для учебных целей.

Начнем импортировать библиотеку TensorFlow. TensorFlow позволяет декларировать сеанс с помощью `tf.InteractiveSession()`. Когда декларируется интерактивный сеанс, функции TensorFlow не требуют атрибута сеанса (который потребовался бы в противном случае), что облегчает программирование с использованием Jupyter Notebooks.

Листинг 2.9. Использование переменных

```
import tensorflow as tf
sess = tf.InteractiveSession()

raw_data = [1., 2., 8., -1., 0., 5.5, 6., 13]
spike = tf.Variable(False)
spike.initializer.run()

for i in range(1, len(raw_data)):
    if raw_data[i] - raw_data[i-1] > 5:
        updater = tf.assign(spike, True)
        updater.eval()
    else:
        tf.assign(spike, False).eval()
    print("Spike", spike.eval())

sess.close()
```

Пусть у вас есть некоторый набор исходных данных, такой как этот

Начните сеанс в интерактивном режиме, чтобы исключить необходимость упоминать sess

Создайте булеву переменную spike, чтобы обнаруживать внезапные повышения числового ряда

Поскольку все переменные необходимо инициализировать, инициализируйте переменную, вызвав run() на ее инициализаторе

Для обновления переменной присвойте ей новое значение с помощью tf.assign(<var name>, <new value>). Вычислите ее, чтобы увидеть произошедшее изменение

Не забудьте закрыть сеанс, если он больше не будет использоваться

Пройдите в цикле по всем данным (пропуская первый элемент) и обновите spike, когда обнаружится значительное повышение активности

Ожидаемыми выходными данными выполнения листинга 2.9 является список значений переменной spike:

```
('Spike', False)
('Spike', True)
('Spike', False)
('Spike', False)
('Spike', True)
('Spike', False)
('Spike', True)
```

2.7. Сохранение и загрузка переменных

Представьте, что вы пишете монолитный блок кода, из которого вы хотите отдельно протестировать крошечный сегмент. В запутанных ситуациях с машинным обучением сохранение и загрузка данных в известных контрольных точках облегчают проверку кода. TensorFlow обеспечивает удобный интерфейс для сохранения и загрузки значений переменных на диск; давайте посмотрим, как его можно использовать для этой цели.

Давайте изменим код, созданный вами и приведенный в листинге 2.9, чтобы он сохранял пиковые данные (spike) на диск для дальнейшего их использования в любом другом месте. Нам придется изменить переменную `spike` на вектор булевых переменных, в котором собраны архивные данные пиковых значений (листинг 2.10). Обратите внимание, что вам придется явным образом дать имена переменным, чтобы их можно было загрузить позже с этим же именем. Присваивать имя переменной необязательно, однако очень рекомендуется для надлежащей организации вашего кода.

Попробуйте выполнить этот код, чтобы увидеть результаты.

Листинг 2.10. Сохранение переменных

```
import tensorflow as tf
sess = tf.InteractiveSession()

raw_data = [1., 2., 8., -1., 0., 5.5, 6., 13]
spikes = tf.Variable([False] * len(raw_data), name='spikes')
spikes.initializer.run()

saver = tf.train.Saver()

for i in range(1, len(raw_data)):
    if raw_data[i] - raw_data[i-1] > 5:
        spikes_val = spikes.eval()
        spikes_val[i] = True
        updater = tf.assign(spikes, spikes_val)
        updater.eval()
```

Задает булевый вектор spike для определения места внезапных всплесков в исходных данных

Импортирует TensorFlow и разрешает интерактивные сеансы

Пусть у вас есть некоторый набор исходных данных, такой как этот

Не забудьте инициализировать переменную

При обнаружении значительного повышения значений переменной пройдите в цикле по всем данным и обновите переменную spike

Обновите значение переменной spike, использовав функцию tf.assign

Оператор сохранения позволяет сохранить и восстановить переменные. Если ни один словарь не был введен в конструктор, тогда он сохраняет все переменные в текущей программе

Не забудьте вычислить поправку; в противном случае переменная spike не будет обновлена

```
save_path = saver.save(sess, "spikes.ckpt")
print("spikes data saved in file: %s" % save_path)

sess.close()
```

Сохраняет переменные на диске

Выводит директорию соответствующего файла с сохраненными переменными

Обратите внимание на пару сгенерированных файлов, один из которых spikes.ckpt, находящийся в той же директории, что и исходный код. Это компактно сохраненный бинарный файл, поэтому его невозможно изменить текстовым редактором. Для восстановления этих данных можно использовать функцию `restore` из оператора `saver`, как показано в следующем листинге.

Листинг 2.11. Загрузка переменных

```
import tensorflow as tf
sess = tf.InteractiveSession()

spikes = tf.Variable([False]*8, name='spikes')
# spikes.initializer.run()
saver = tf.train.Saver()

saver.restore(sess, "./spikes.ckpt")
print(spikes.eval())

sess.close()
```

Создает переменную с таким же размером и именем, как в сохраненных данных

Теперь больше нет необходимости инициализировать эту переменную, поскольку она будет загружена непосредственно

Восстанавливает данные из файла spikes.ckpt

Выводит загруженные данные

Создает оператор сохранения для возможности восстановления сохраненных данных

2.8. Визуализация данных с помощью TensorBoard

В машинном обучении наиболее затратной по времени частью является не программирование, а ожидание завершения выполнения кода. Например, знаменитый набор данных *ImageNet* содержит более 14 млн изображений, подготовленных для использования в машинном обучении. Иногда может потребоваться несколько дней или недель для завершения алгоритма обучения, использующего такую большую базу данных. Удобная информационная панель TensorBoard библиотеки TensorFlow дает возможность быстро взглянуть на то, как в каждом узле или графе меняются данные, давая некоторое представление о том, как выполняется код.

Давайте посмотрим, как можно визуализировать временные тренды переменных на реальном примере. В этом разделе будет применен алгоритм скользящего

среднего из TensorFlow, после чего можно будет внимательно проследить за представляющей интерес переменной для визуализации ее поведения с помощью TensorBoard.

2.8.1. Использование метода скользящего среднего

В этом разделе будет использована информационная панель TensorBoard для визуализации изменения данных. Предположим, что вас интересует вычисление среднего курса акций компании. Обычно вычисление среднего сводится к простому суммированию всех значений и делению этой суммы на число значений: среднее равно = $(x_1 + x_2 + ... + x_n)/n$. Если суммарное число переменных неизвестно, можно использовать метод, который носит название *экспоненциальное усреднение* (exponential averaging), чтобы оценить среднее значение неизвестного числа точек данных. Алгоритм экспоненциального усреднения вычисляет текущую оценку среднего как функцию предыдущей оценки среднего и текущего значения.

В более сжатом виде: $Avg_t = f(Avg_{t-1}, x_t) = (1-\alpha)\, Avg_{t-1} + \alpha\, x_t$. Альфа ($\alpha$) является настраиваемым параметром, который показывает, как сильно текущие значения должны быть смещены при вычислении среднего. Чем выше значение α, тем сильнее вычисленное среднее будет отличаться от предыдущей его оценки. На рис. 2.8 после листинга 2.16 показано, как панель TensorBoard визуализирует в зависимости от времени значения, а также соответствующее скользящее среднее.

Когда вы записали это в виде кода, хорошо бы подумать об основной части вычислений, которые выполняются на каждой итерации. В данном случае на каждой итерации вычисляется $Avg_t = (1 - \alpha)\, Avg_{t-1} + \alpha\, x_t$. В результате можно написать оператор TensorFlow (листинг 2.12), который в точности выполнит все то, что предусмотрено формулой. Для выполнения этого кода необходимо будет в конечном счете определить `alpha`, `curr_value` и `prev_avg`.

Листинг 2.12. Задает оператор обновления среднего

```
update_avg = alpha * curr_value + (1 - alpha) * prev_avg
```

Альфа tf.constant, curr_value является переменной-заполнителем, а prev_avg — простой переменной

Позже необходимо будет задать неопределенную переменную. Причина, по которой мы пишем код в обратном порядке, состоит в том, что сразу после интерфейса следует внедрить код периферийной настройки для выполнения требований этого интерфейса. Давайте сразу перейдем к той части сеанса, в которой можно посмотреть, как ведет себя код. Следующий листинг задает первичный цикл и вызовы оператора `update_avg` на каждой итерации. Выполнение оператора `update_avg` зависит от `curr_value`, которая подается с помощью аргумента `feed_dict`.

Листинг 2.13. Выполнение итераций алгоритма экспоненциального среднего

```
raw_data = np.random.normal(10, 1, 100)

with tf.Session() as sess:
    for i in range(len(raw_data)):
        curr_avg = sess.run(update_avg, feed_dict={curr_value:raw_data[i]}
        sess.run(tf.assign(prev_avg, curr_avg))
```

Отлично, общая картина понятна, теперь все, что осталось сделать, — это выписать неопределенные переменные. Давайте заполним пробелы и используем работающий фрагмент кода TensorFlow. Скопируйте следующий листинг, теперь его можно выполнить.

Листинг 2.14. Заполнение пробелов в коде для завершения алгоритма экспоненциального среднего

```
import tensorflow as tf
import numpy as np

raw_data = np.random.normal(10, 1, 100)

alpha = tf.constant(0.05)
curr_value = tf.placeholder(tf.float32)
prev_avg = tf.Variable(0.)
update_avg = alpha * curr_value + (1 - alpha) * prev_avg

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for i in range(len(raw_data)):
        curr_avg = sess.run(update_avg, feed_dict={curr_value: raw_data[i]})
        sess.run(tf.assign(prev_avg, curr_avg))
        print(raw_data[i], curr_avg)
```

Создает вектор из 100 чисел со средним значением 10 и со среднеквадратичным отклонением 1

Определяет альфу константой

Инициализирует предыдущее среднее нулем

Переменная-заполнитель — такая же переменная, но ее значение вводится во время сеанса

Проходит циклически по всем данным для обновления среднего

2.8.2. Визуализация метода скользящего среднего

Теперь, когда у нас есть работающий вариант алгоритма скользящего среднего, попробуем визуализировать результаты с помощью панели TensorBoard. Визуализация обычно выполняется в два этапа:

1. Выбираются узлы, за которыми необходимо следить при выполнении измерений, аннотировав их оператором `summary`.
2. Вызывается `add_summary`, чтобы составить очередь из данных для записи на диск.

Например, пусть есть переменная-заполнитель `img` и оператор `cost`, как показано в приведенном ниже листинге. Их следует аннотировать (дав каждому имя, такое как `img` или `cost`), чтобы затем визуализировать с помощью панели TensorBoard. Сделаем это на примере скользящего среднего.

Листинг 2.15. Аннотирование оператором summary

```
img = tf.placeholder(tf.float32, [None, None, None, 3])
cost = tf.reduce_sum(...)

my_img_summary = tf.summary.image("img", img)
my_cost_summary = tf.summary.scalar("cost", cost)
```

В более общем случае для установления связи с панелью TensorBoard необходим оператор `summary`, который создает упорядоченные строки, используемые `SummaryWriter` для сохранения обновлений в указанной директории.

Всякий раз при вызове метода `add_summary` из `SummaryWriter` TensorFlow будет сохранять данные на диск для их использования панелью TensorBoard.

> **ПРЕДУПРЕЖДЕНИЕ** Постарайтесь не вызывать функцию `add_summary` слишком часто! Хотя это и дает визуализацию высокого разрешения используемых переменных, но делается за счет дополнительных вычислений и замедления процесса обучения.

Выполните следующую команду, чтобы директория с логами была в той же папке, что и исходный код:

```
$ mkdir logs
```

Запустите TensorBoard с директорией размещения логов, введенной в качестве аргумента:

```
$ tensorboard --logdir=./logs
```

Откройте браузер и перейдите на сайт http://localhost:6006, который является предусмотренным по умолчанию адресом панели TensorBoard. В следующем листинге показано, как подключать `SummaryWriter` к коду. Выполните код и обновите TensorBoard для просмотра результатов визуализации.

Листинг 2.16. Написание аннотаций для просмотра панели TensorBoard

```
import tensorflow as tf
import numpy as np

raw_data = np.random.normal(10, 1, 100)

alpha = tf.constant(0.05)
curr_value = tf.placeholder(tf.float32)
prev_avg = tf.Variable(0.)
update_avg = alpha * curr_value + (1 - alpha) * prev_avg

avg_hist = tf.summary.scalar("running_average", update_avg)
value_hist = tf.summary.scalar("incoming_values", curr_value)
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter("./logs")
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    sess.add_graph(sess.graph)
    for i in range(len(raw_data)):
        summary_str, curr_avg = sess.run([merged, update_avg],
        feed_dict={curr_value: raw_data[i]})
        sess.run(tf.assign(prev_avg, curr_avg))
        print(raw_data[i], curr_avg)
        writer.add_summary(summary_str, i)
```

Создает итоговый узел для средних

Создает итоговый узел для этих значений

Объединяет аннотации, чтобы облегчить их выполнение сразу

Проходит директорию логов для записывающего устройства

Выполняет одновременно оператор merged и оператор update_avg

Добавляет аннотацию для пишущего устройства

Это не обязательно, но дает возможность визуализации графа вычислений в TensorBoard

СОВЕТ Перед началом работы TensorBoard у вас может возникнуть необходимость убедиться, что сеанс TensorFlow закончился. Если код из листинга 2.16 будет выполняться повторно, необходимо очистить директорию с логами.

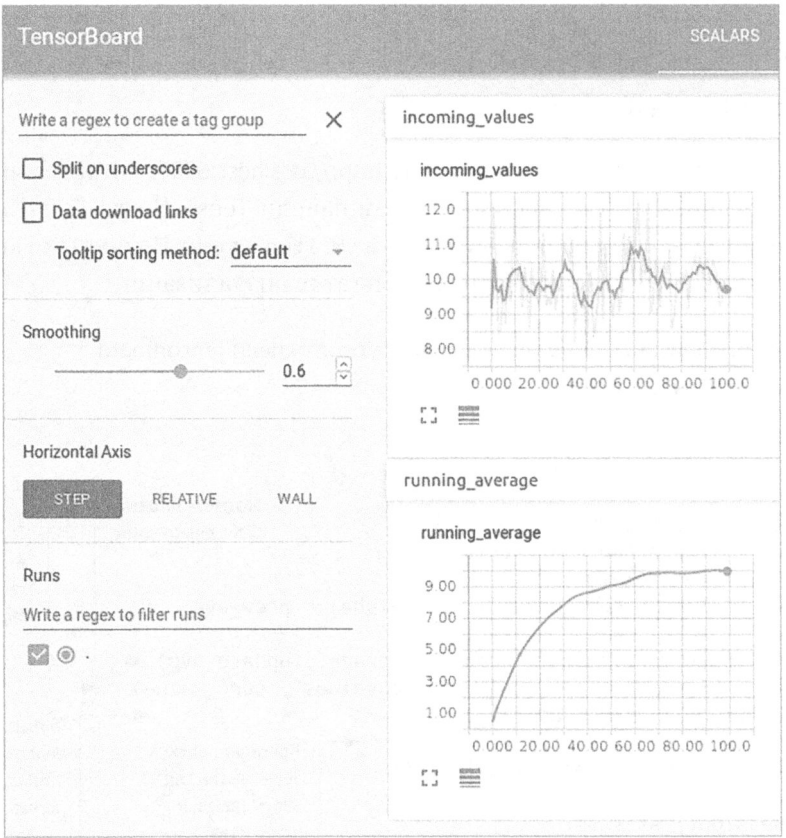


Рис. 2.8. Отображение на панели TensorBoard аннотации, созданной в листинге 2.16. TensorBoard предоставляет удобный интерфейс для визуализации данных, созданных в TensorFlow

2.9. Краткие итоги

- Теперь можно продумать представление математических алгоритмов в виде блок-схем вычислений. При рассмотрении каждого узла как операции, а ребер как потока данных написание кода TensorFlow становится проще. После задания графа во время сеанса происходит его вычисление, и вы получаете определенный результат.
- Нет сомнений, что TensorFlow дает значительно больше, чем представление вычислений в виде графа. В следующих главах будет видно, что некоторые

из встроенных функций заточены под применение в машинном обучении. Действительно, TensorFlow обеспечивает значительную поддержку сверточных нейронных сетей, популярных в настоящее время типов моделей для обработки изображений (также с многообещающими результатами обработки аудиоданных и текстовой информации).

- TensorBoard обеспечивает простой способ визуализации изменения данных в коде TensorFlow, а также обнаруживает баги с помощью проверки трендов данных.
- TensorFlow отлично работает с блокнотами Jupyter, которые представляют собой элегантный интерактивный посредник для обмена и документирования кода на Python.

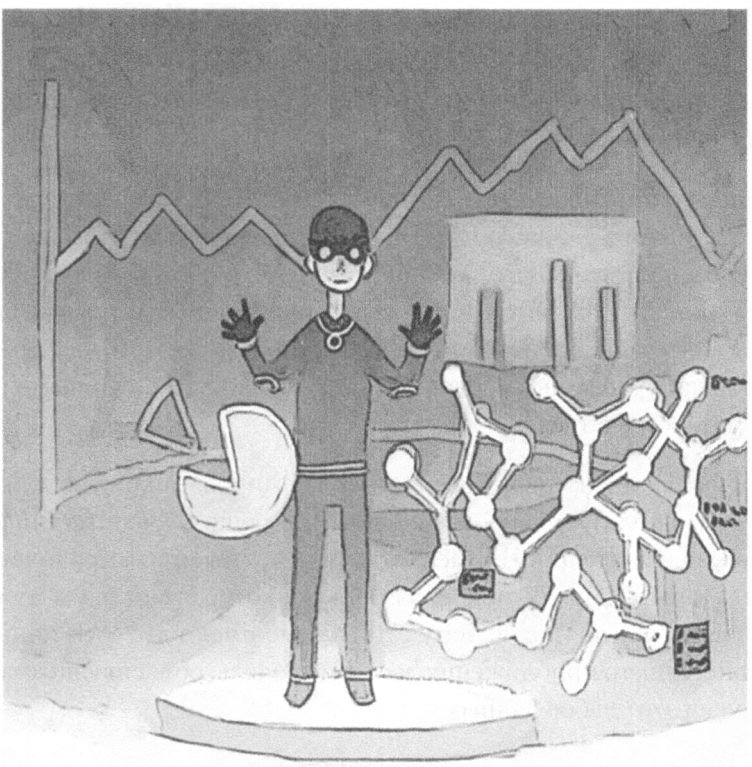
Часть II

Основные алгоритмы обучения

Когда бывший президент США Барак Обама во время предвыборной речи сказал: «Вы можете накрасить свинье губы помадой, но она все равно останется свиньей», он не имел в виду то, что большинство сложных идей в машинном обучении можно свести к нескольким фундаментальным идеям, хотя это так и есть. Например, к основным алгоритмам относятся регрессия, классификация, кластеризация и скрытые марковские модели. Эти идеи подробно описываются в соответствующих главах в том же порядке, в каком они перечислены.

После того как вы осилите эти четыре главы, то увидите, как можно решить большинство задач реального мира, используя аналогичные методы. То, что когда-то казалось чужеродным и запутанным, теперь с помощью формул из этих основных обучающих алгоритмов можно легко распутать и понять.

Линейная и нелинейная регрессия



Эта глава охватывает следующие темы:

- ✓ Аппроксимация линии к точкам данных
- ✓ Аппроксимация произвольных кривых к точкам данных
- ✓ Проверка эффективности алгоритмов регрессии
- ✓ Применение регрессии к реальным данным

Помните курсы естественных наук в старшей школе? Возможно, это было давно или (кто знает?), может быть, вы и сейчас в старшей школе и просто рано отправляетесь в путешествие по пути машинного обучения. В любом случае независимо от того, выбрали вы биологию, химию или физику, общий метод анализа данных один — это определение того, как изменение одной переменной влияет на другую.

Представьте построение корреляции между частотой выпадения осадков и сбором урожая. Вы можете наблюдать, что повышение частоты осадков приводит к росту урожайности. Подгонка линии к этим точкам данных дает возможность делать прогнозы об урожайности при различных условиях выпадения осадков. Если удается обнаружить лежащую в основе этого зависимость, она позволит делать успешные прогнозы значений данных, с которыми раньше не приходилось встречаться.

Регрессия изучает, как лучше всего аппроксимировать кривую для сведения данных. Она использует наиболее мощные и хорошо изученные типы алгоритмов обучения с учителем. Применяя регрессию, мы стараемся понять смысл, скрытый в точках данных, подбирая кривую, которая могла бы их породить. Так мы ищем объяснение, почему эти данные имеют тот или иной разброс. Кривая наилучшего приближения дает нам модель, объясняющую то, как мог быть получен этот набор данных.

В этой главе показано, как формулировать задачи реального мира для использования регрессии. Вы убедитесь, что TensorFlow является именно тем инструментом, который дает наиболее эффективные средства прогнозирования.

3.1. Формальные обозначения

Когда у вас в руках молоток, все задачи кажутся гвоздями. В этой главе представлен первый основной метод машинного обучения — регрессия, который формально определяется с помощью точных математических символов. Необходимость первоочередного знакомства с регрессией обусловлена тем, что многие из приобретенных навыков помогут при решении других задач, описанных в следующих главах. В конце этой главы регрессия станет наиболее часто используемым инструментом вашего набора средств машинного обучения.

Пусть у вас имеются данные о том, сколько денег люди тратят на пиво. Элис тратит 4 доллара на 2 бутылки, Боб — 6 долларов на 3 бутылки, а Клэр тратит 8 долларов на 4 бутылки. Вы хотели бы найти уравнение, которое описывает, как число бутылок влияет на итоговые затраты. Например, если линейное уравнение $y = 2x$ описывает стоимость покупки определенного числа бутылок, тогда можно найти, сколько стоит одна бутылка пива.

Когда линия хорошо аппроксимирует точки данных, можно утверждать, что использованная линейная модель эффективна. Но можно было бы попробовать использовать много разных наклонов линии, вместо того чтобы просто выбрать коэффициент наклона, равный двум. Выбираемый наклон является *параметром*, а уравнение, содержащее параметр, называют *моделью*. Выражаясь в терминах машинного обучения, уравнение кривой наилучшего приближения появляется в результате определения параметров модели в процессе машинного обучения.

Другой пример, уравнение $y = 3x$ также описывает линию, но с более крутым наклоном. Вы можете заменить этот коэффициент любым действительным числом, назовем его w, и это уравнение снова будет описывать прямую линию: $y = wx$. На рис. 3.1 показано, как изменение параметра w влияет на модель. Множество всех уравнений, обобщенных таким образом, обозначается как $M = \{y = wx \mid w \in \mathbb{R}\}$. Это следует понимать как «все уравнения $y = wx$, у которых w является действительным числом».

M является множеством всех возможных моделей. Выбор значения w генерирует модель-кандидата $M(w)$: $y = wx$. Алгоритмы регрессии, которые вы будете писать в TensorFlow, итеративно сходятся к значениям модельного параметра w, которые повышают эффективность модели. Оптимальный параметр, назовем его w^*, дает уравнение наилучшего приближения $M(w^*) : y = w^*$.

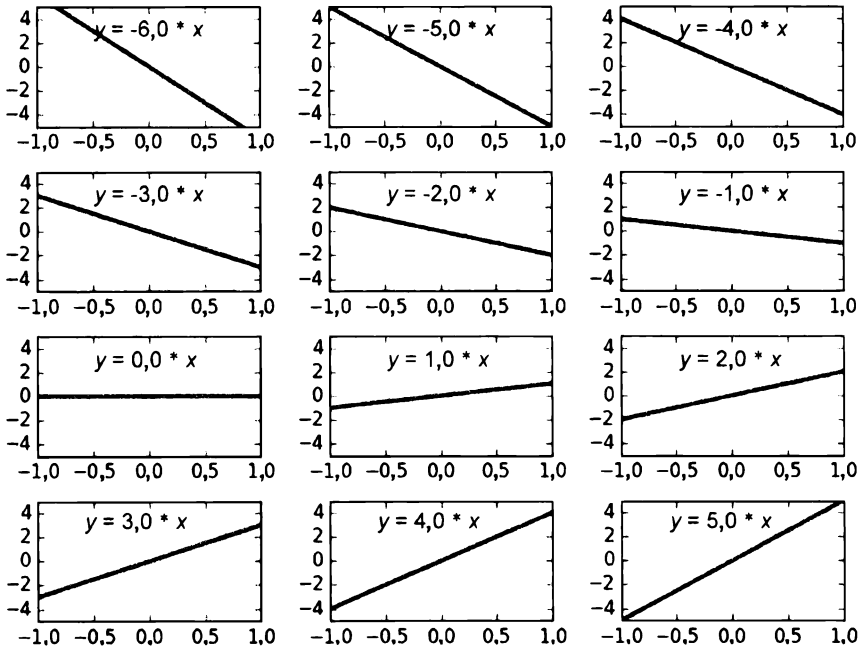


Рис. 3.1. Разные значения параметра w приводят к различным линейным уравнениям. Множество всех линейных уравнений образует линейную модель M

В самом общем случае алгоритм регрессии стремится создать функцию, назовем ее f, которая преобразует входные данные в выходные. Областью значений этой функции является вектор действительных значений R^d, а его областью значений является множество действительных чисел R.

ПРИМЕЧАНИЕ Регрессия может выдать множество выходных данных, а не только одно действительное число. В этом случае она носит название *многомерной регрессии* (multivariate regression).

Входные данные функции могут быть непрерывными или дискретными. Но ее выходные данные должны быть непрерывными, как это показано на рис. 3.2.

Мы хотели бы найти функцию f, которая хорошо согласуется с имеющимися точками данных, являющимися, по сути, парами вход/выход. К сожалению,

число возможных функций не ограничено, поэтому использование их друг за другом не приведет к успеху. Наличие слишком большого количества вариантов для выбора — это, как правило, плохая идея. Это заставляет нас сжать границы всех функций, с которыми мы хотели бы иметь дело. Например, если рассматривать только прямые линии для приближения точек данных, поиск значительно облегчается.

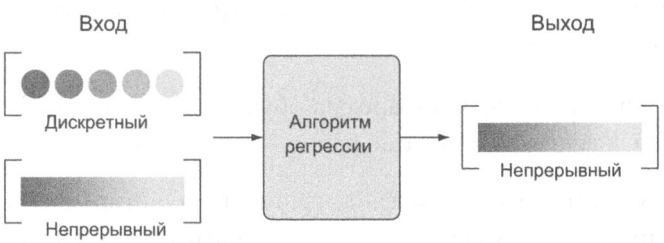


Рис. 3.2. Алгоритм регрессии предполагает непрерывный вывод. Вход может быть дискретным или непрерывным. Это различие входных и выходных данных имеет значение, так как дискретные выходные данные лучше классифицируются, что будет подробно рассматриваться в следующей главе

ПРИМЕЧАНИЕ Регрессия прогнозирует непрерывные выходные данные, но иногда это требование является чрезмерным. Иногда мы хотим спрогнозировать дискретный выход, такой как 0 или 1, и ничего между ними. Классификация — это метод, который лучше подходит для таких задач. Это будет подробно описано в главе 4.

УПРАЖНЕНИЕ 3.1

Сколько существует функций, которые преобразуют 10 целых чисел в 10 целых чисел? Например, пусть $f(x)$ будет функцией, которая может брать числа от 0 до 9 и давать числа от 0 до 9. Одним из примеров является функция тождественности, которая подражает собственному входу: например, $f(0) = 0$, $f(1) = 1$ и т. д. Сколько еще существует таких функций?

ОТВЕТ

$10^{10} = 10\,000\,000\,000$

3.1.1. Как понять, что алгоритм регрессии работает?

Предположим, вы пытаетесь продать компании по недвижимости алгоритм, способный составлять прогнозы касательно рынка жилья. Этот алгоритм прогнозирует цены на жилье по заданным параметрам, таким как число спальных комнат и общая площадь. Компании, которые занимаются недвижимостью, могли бы с легкостью заработать на этой информации миллионы, однако перед тем, как купить такой алгоритм у вас, они потребуют подтверждения, что он действительно работает.

Для оценки эффективности алгоритма обучения необходимо использовать такие понятия, как дисперсия и смещение.

- **Дисперсия** (variance) показывает, насколько чувствителен прогноз к использованному обучающему набору. В идеале выбор обучающего набора не должен играть существенной роли, то есть дисперсия должна быть низкой.
- **Смещение** (bias) показывает эффективность сделанных предположений относительно обучающего набора данных. Слишком большое число предположений может привести к невозможности обобщения моделью, поэтому предпочтительно добиваться низкого значения смещения.

Если модель слишком гибкая, она может случайно запомнить данные обучения вместо разрешения полезных шаблонов. Можно представить себе изогнутую функцию, проходящую через каждую точку набора данных и, казалось бы, не приводящую к ошибкам. Если такое происходит, то говорят, что обучающий алгоритм *чрезмерно аппроксимировал* данные, то есть оказался переобучен (overfit). В этом случае кривая наилучшего приближения будет хорошо согласовываться с обучающими данными, но при этом плохо аппроксимировать проверочные данные (рис. 3.3).

С другой стороны, недостаточно гибкая модель может лучше обобщать ранее не использованные проверочные данные, но будет менее эффективна при аппроксимации обучающих данных. Эту ситуацию принято называть *недостаточной обученностью* (underfitting). Слишком гибкая (универсальная) модель имеет высокую дисперсию и низкое смещение, тогда как «жесткая» модель имеет низкую дисперсию и высокое смещение. В идеале необходима модель с низкой случайной ошибкой (дисперсией) и низкой систематической

ошибкой (смещением). Это позволит обобщать модель для ранее не использованных данных и улавливать закономерности в используемых данных. На рис. 3.4 приведены примеры недостаточно точной аппроксимации и чрезмерно близкой подгонки в двумерной модели.

Рис. 3.3. В идеальном случае кривая наилучшего приближения хорошо аппроксимирует обучающие и проверочные данные. Если оказалось, что она плохо аппроксимирует проверочные и обучающие данные, то есть вероятность, что модель недостаточно хорошо обучена. Если же она плохо аппроксимирует проверочные данные, но хорошо — обучающие, говорят, что модель избыточно обучена (переобучена)

Рис. 3.4. Примеры недостаточной и чрезмерной аппроксимации данных

Проще говоря, дисперсия модели является мерой того, насколько плохо флуктуирует реакция модели, а смещение является мерой того, насколько плохо реакция модели отклоняется от контрольных данных. Наша модель должна давать результаты, характеризующиеся точностью (низким смещением), а также воспроизводимостью (низкой дисперсией).

УПРАЖНЕНИЕ 3.2

Пусть у вас есть модель $M(w) : y = wx$. Сколько возможных функций можно получить, если параметр w может принимать целочисленные значения в интервале от 0 до 9 (включительно)?

ОТВЕТ

Только 10: $\{y = 0, y = x, y = 2x, \ldots, y = 9x\}$.

Таким образом, измерение того, насколько хорошо ваша модель аппроксимирует обучающие данные, не является хорошим показателем ее способности к обобщениям. Вместо этого модель необходимо оценить на отдельном наборе тестовых данных. Может оказаться, что модель хорошо аппроксимирует данные, с помощью которых она обучалась, но ужасно аппроксимирует проверочные данные, и в этом случае модель, похоже, чрезмерно аппроксимирует обучающие данные. Если ошибка модели на проверочных данных приблизительно такая же, как и на обучающих данных, и обе ошибки одного типа, тогда можно считать, что модель хорошо обучена или переобучена, если ошибка на проверочных данных выше.

Поэтому для оценки эффективности машинного обучения набор данных следует разделить на две группы: обучающие и проверочные данные. Модель обучается, используя обучающие данные, а эффективность обучения проверяют на проверочных данных (как именно оценивается эффективность обучения модели, описано в следующем разделе). Из множества возможных значений параметра необходимо найти один, который лучше всего согласуется с данными. Способом оценки *наилучшего соответствия* является использование функции стоимости, которая будет подробно описана в следующем разделе.

3.2. Линейная регрессия

Начнем с создания имитации реальных данных, чтобы проникнуть в самое сердце линейной регрессии. Создадим исходный файл Python, назовем его regression.py, а затем используем следующий листинг для инициализации данных. Этот код будет создавать выходные данные, аналогичные представленным на рис. 3.5.

Листинг 3.1. Визуализация исходных входных данных

Импортирует NumPy с целью инициализации входных данных

Использует библиотеку matplotlib для визуализации данных

```
import numpy as np
import matplotlib.pyplot as plt

x_train = np.linspace(-1, 1, 101)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

plt.scatter(x_train, y_train)
plt.show()
```

Входными значениями являются равноотстоящие между –1 и 1 числа 101

Использует функцию библиотеки matplotlib для генерации диаграммы рассеяния данных

Выходные значения пропорциональны входным, но с добавленным шума

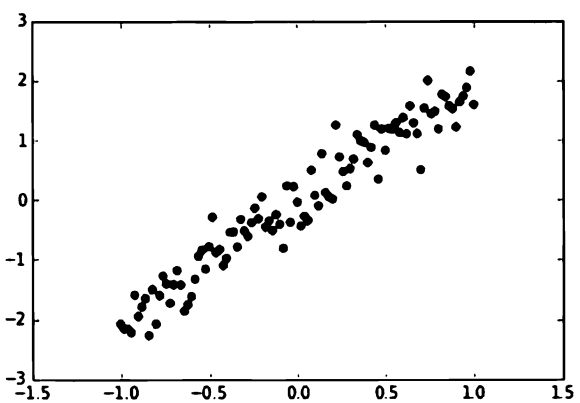


Рис. 3.5. Диаграмма рассеяния y = x + (noise)

Теперь, когда есть некоторые точки данных, можно попытаться аппроксимировать их линией. По крайней мере, для библиотеки TensorFlow необходимо предоставить оценку каждого испробованного варианта параметра. Эти оценки обычно называют *функцией стоимости* (cost function). Она используется для вычисления ошибки значения, или «стоимости», заданной целевой функции. Чем выше стоимость, тем хуже окажется выбор параметра модели. Например, если линия наилучшего приближения $y = 2x$, то выбор параметра 2,01 будет давать низкую стоимость, а выбор параметра −1 даст высокую стоимость.

После того как задача была сформулирована как задача минимизации стоимости, в соответствии с обозначениями на рис. 3.6 TensorFlow принимается за обновление значений параметра, чтобы найти наилучшее из возможных значений. Каждый шаг циклического перебора данных для обновления параметра носит название *эпохи* (epoch).

$$w* = \arg\min_w \underbrace{\mathrm{cost}(Y_{\text{модель}}, Y_{\text{идеал}})}_{|\underbrace{Y_{\text{модель}}}_{M(w,\,X)} - Y_{\text{идеал}}|}$$

Рис. 3.6. Для любого минимизированного параметра *w* стоимость будет оптимальной. Стоимость определяется как норма разности между истинным значением и откликом модели. И наконец, значение отклика рассчитывается по функции, использованной в модели

В этом примере затраты определены как сумма ошибок. Ошибка прогнозирования x часто вычисляется как квадрат разности между фактическим значением $f(x)$ и прогнозированным значением $M(w, x)$. Поэтому затраты являются суммой квадратов разности между фактическими и прогнозированными значениями, что можно увидеть на рис. 3.7.

Обновите предыдущий код, чтобы он выглядел так же, как следующий листинг. Этот код определяет функцию стоимости и делает запрос TensorFlow на запуск оптимизации, чтобы найти искомые параметры модели.

Листинг 3.2. Решение задачи линейной регрессии

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

learning_rate = 0.01
training_epochs = 100

x_train = np.linspace(-1, 1, 101)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

def model(X, w):
    return tf.multiply(X, w)

w = tf.Variable(0.0, name="weights")

y_model = model(X, w)
cost = tf.square(Y-y_model)

train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
    for (x, y) in zip(x_train, y_train):
        sess.run(train_op, feed_dict={X: x, Y: y})

w_val = sess.run(w)

sess.close()
plt.scatter(x_train, y_train)
y_learned = x_train*w_val
plt.plot(x_train, y_learned, 'r')
plt.show()
```

Импортирует TensorFlow для работы алгоритма обучения. Чтобы задать начальные данные, необходимо использовать NumPy. Для визуализации данных потребуется matplotlib

Определяет константы, используемые алгоритмом обучения. Они получили название гиперпараметров

Определяет входные и выходные узлы в виде переменных-заполнителей, поскольку значение будет вводиться с помощью x_train и y_train

Определяет модель y = w*X

Определяет фиктивные данные, которые потребуются, чтобы найти линию наилучшего приближения

Определяет значения переменных

Определяет функцию стоимости

Задает сеанс и инициализирует все переменные

Многократно циклически перебирает данные набора

Циклически перебирает каждый элемент базы данных

Обновляет параметр модели с целью попытаться минимизировать функцию затрат

Получает окончательное значение параметра

Закрывает сеанс

Наносит на график исходные данные

Строит линию наилучшего приближения

Задает операцию, которая будет вызываться на каждой итерации алгоритма обучения

На рис. 3.8 показаны результаты решения задачи линейной регрессии с помощью TensorFlow. Остальные разделы задачи регрессии являются незначительными модификациями кода, приведенного в листинге 3.2. Весь процесс

включает в себя обновление параметров модели с помощью TensorFlow, как это подытожено на рис. 3.9.

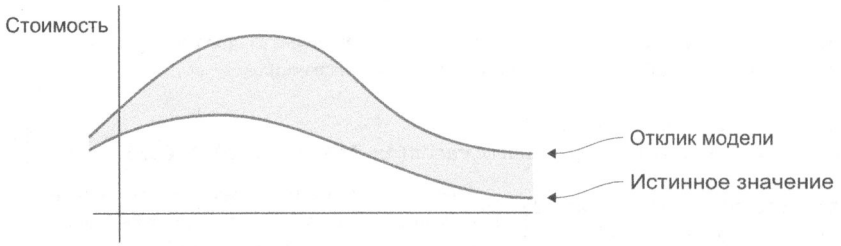


Рис. 3.7. Стоимость является нормой точечной разницы между откликом модели и истинным значением

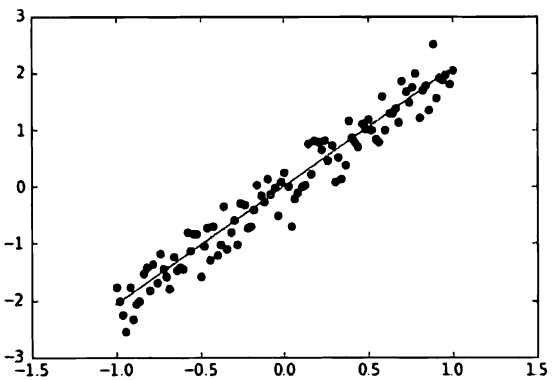


Рис. 3.8. Оценка линейной регрессии, показанная в листинге 3.2

Вы только что научились применять простую модель регрессии. Все дальнейшие усовершенствования сводятся к простому повышению качества модели с помощью подходящего сочетания дисперсии и смещения, как уже упоминалось ранее. Например, только что разработанная модель линейной регрессии обременена большим смещением; это обусловлено только ограниченным набором функций, а именно линейных функций. В следующем разделе будет рассмотрена более универсальная модель. Обратите внимание, что для этого потребуется только заменить граф TensorFlow, тогда как все остальное (предварительная обработка, обучение, оценка эффективности) останется прежним.

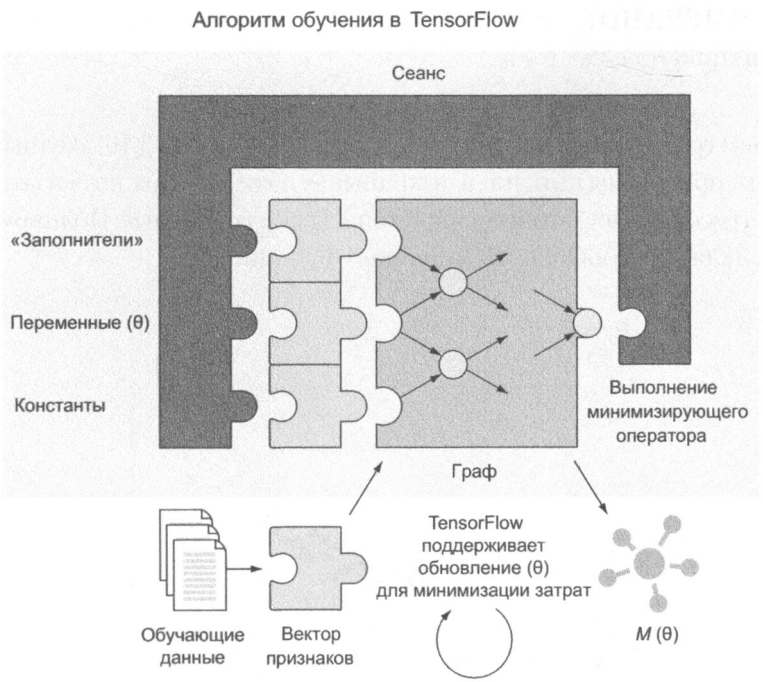


Рис. 3.9. Алгоритм обучения обновляет параметры модели для минимизации заданной функции стоимости

3.3. Полиномиальная модель

Линейная модель может быть первым приближением, но редко когда корреляции реального мира носят такой простой характер. Например, траектория ракеты в космическом пространстве искривляется по отношению к наблюдателю на Земле. Интенсивность сигнала Wi-Fi падает по закону обратных квадратов. Изменение высоты цветка на протяжении его жизни меняется определенно не линейным образом.

Если точки данных формируют сглаженную кривую, а не прямую линию, модель регрессии необходимо изменить с прямой линии на какое-то иное представление. Один из таких методов использует полиномиальную модель. *Полином* является обобщением линейной функции. N-я степень полинома выглядит следующим образом:

$$f(x) = w_n x^n + \ldots + w_1 x + w_0.$$

ПРИМЕЧАНИЕ Когда $n = 1$, полином превращается в линейную функцию $f(x) = w_1 x + w_0$.

Рассмотрим график разброса данных, приведенный на рис. 3.10, входные данные на котором приведены на оси x, а выходные — на оси y. Можно сказать, что линейная регрессия недостаточно хорошо описывает все данные. Полиномиальная функция является обобщением линейной функции.

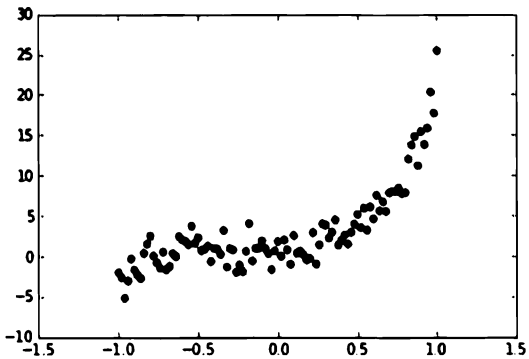


Рис. 3.10. Точки данных, такие как эти, не подходят для линейной модели

Попытаемся использовать для приближения этих данных полином. Создадим новый файл и назовем его polynomial.py, а затем будем действовать в соответствии с приведенным листингом.

Листинг 3.3. Использование полиномиальной модели

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

learning_rate = 0.01
training_epochs = 40
```

Импортирует необходимые библиотеки и инициализирует гиперпараметры

```
trX = np.linspace(-1, 1, 101)
```

Задает на входе фейковые исходные данные

```
num_coeffs = 6
trY_coeffs = [1, 2, 3, 4, 5, 6]
trY = 0
for i in range(num_coeffs):
    trY += trY_coeffs[i] * np.power(trX, i)
```

Задает исходные выходные данные на основе полинома 5-й степени

```
trY += np.random.randn(*trX.shape) * 1.5
```

Добавляет шум

```
plt.scatter(trX, trY)
plt.show()
```

Представляет диаграмму рассеяния исходных данных

```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
```

Задает узлы для пар входных и выходных данных

```
def model(X, w):
    terms = []
    for i in range(num_coeffs):
        term = tf.multiply(w[i], tf.pow(X, i))
        terms.append(term)
    return tf.add_n(terms)
```

Задает полиномиальную модель

```
w = tf.Variable([0.] * num_coeffs, name="parameters")
y_model = model(X, w)
```

Задает вектор параметров нулями

```
cost = (tf.pow(Y-y_model, 2))
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Как и раньше, задает функцию стоимости

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
    for (x, y) in zip(trX, trY):
        sess.run(train_op, feed_dict={X: x, Y: y})

w_val = sess.run(w)
print(w_val)
```

Задает сеанс и выполняет, как и раньше, алгоритм обучения

```
sess.close()
```

После этого завершает сеанс

```
plt.scatter(trX, trY)
trY2 = 0
for i in range(num_coeffs):
    trY2 += w_val[i] * np.power(trX, i)

plt.plot(trX, trY2, 'r')
plt.show()
```

Наносит результаты на график

Конечным результатом этого кода является полином 5-й степени, который аппроксимирует данные (рис. 3.11).

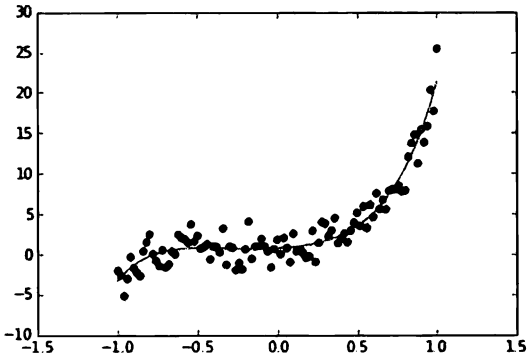


Рис. 3.11. Кривая наилучшего приближения плавно вписывается в данные нелинейной зависимости

3.4. Регуляризация

Не дайте себя одурачить удивительной универсальностью полиномов, которая была показана в предыдущем разделе. То, что полиномы высокого порядка являются расширением полиномов низкого порядка, еще не означает, что предпочтение следует всегда отдавать универсальным моделям.

В реальном мире исходные данные редко образуют сглаженную кривую, похожую на полиномиальную. Допустим, что вы наносите на график цены на недвижимость в зависимости от времени. В этих данных, скорее всего, будут флуктуации. Цель регрессии — представить сложные данные в виде простого математического уравнения. Если выбранная модель слишком универсальная, это может усложнить интерпретацию входных данных.

Возьмем, например, данные, представленные на рис. 3.12. Вы пытаетесь аппроксимировать полином 8-й степени в точки, которые, как оказалось, следуют уравнению $y = x^2$. Этот процесс заканчивается неудачей, поскольку алгоритм пытается сделать все возможное для обновления девяти коэффициентов полинома.

Регуляризацией (regularization) называют метод структурирования параметров в удобной форме, часто применяемый для решения проблем переобучения. В этом случае следует ожидать, что все полученные в результате машинного

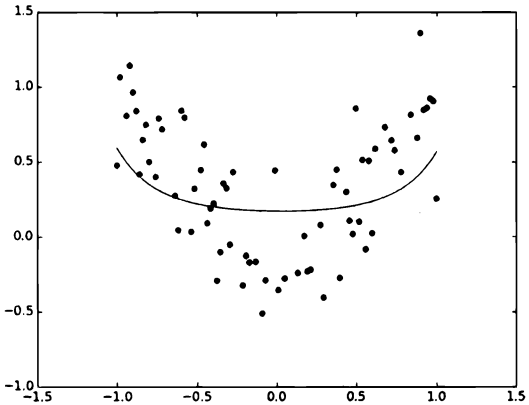


Рис. 3.12. Если модель слишком гибкая, кривая наилучшего приближения может выглядеть запутанной или непонятной. Нам нужно использовать регуляризацию для улучшения подгонки, чтобы изученная модель хорошо работала с проверочными данными

обучения коэффициенты будут нулевыми, кроме второго, что соответствует кривой $y = x^2$. Алгоритм регрессии этого не учитывает, поэтому он может выдавать кривые, которые будут обеспечивать низкую погрешность аппроксимации точек данных, но выглядеть непривычно сложными.

Чтобы повлиять на алгоритм обучения для создания меньшего вектора коэффициентов (назовем его w), необходимо к члену, приводящему к потере точности, добавить штрафной коэффициент. Для контроля веса штрафного коэффициента необходимо умножить его на положительную константу λ:

$$\text{Cost}(X, Y) = \text{Loss}\,(X, Y) + \lambda|\omega|\,.$$

Если λ равна 0, регуляризация не работает. Если задавать λ все большие и большие значения, параметры, приводящие к большим нормам, будут штрафоваться сильнее. Выбор нормы меняется от случая к случаю, но параметры обычно измеряются с помощью их L1- или L2-нормы. Проще говоря, регуляризация несколько уменьшает универсальность модели, которая в противном случае может легко превратиться в запутанную модель.

Для оценки того, какое значение параметра регуляризации λ дает лучший результат, необходимо разбить набор данных на две не связанные между собой

части. Около 70 % случайно выбранных входных и выходных пар будут состоять из обучающего набора данных. Остальные 30 % будут использоваться для проверки. Вы будете использовать функцию, приведенную в следующем листинге, чтобы разбить данные на части.

Листинг 3.4. Разделение набора данных на обучающий и проверочный наборы

```
def split_dataset(x_dataset, y_dataset, ratio):
    arr = np.arange(x_dataset.size)
    np.random.shuffle(arr)
    num_train = int(ratio * x_dataset.size)
    x_train = x_dataset[arr[0:num_train]]
    x_test = x_dataset[arr[num_train:x_dataset.size]]
    y_train = y_dataset[arr[0:num_train]]
    y_test = y_dataset[arr[num_train:x_dataset.size]]
    return x_train, x_test, y_train, y_test
```

Берет входные и выходные данные, а также требуемое соотношение разделения данных

Перетасовывает список номеров

Рассчитывает числа примеров обучения

Использует перетасованный список для разделения x_dataset

Иначе разделяет y_dataset

Возвращает части x и y базы данных

УПРАЖНЕНИЕ 3.3

Библиотека Python, получившая название *scikit-learn*, поддерживает много полезных алгоритмов обработки данных. Вы можете вызвать функцию в scikit-learn, чтобы точно выполнить листинг 3.4. Сможете найти эту функцию в документации библиотеки? Подсказка: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection.

ОТВЕТ

Она называется `sklearn.model_selection.train_test_split`.

С помощью этого инструмента можно проверить, какое из значений λ лучше работает с вашими данными. Откройте новый файл Python и введите код из следующего листинга.

Листинг 3.5. Оценка параметров регуляризации

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

learning_rate = 0.001
training_epochs = 1000
reg_lambda = 0.
```

Импортирует необходимые библиотеки и инициализирует гиперпараметры

```
x_dataset = np.linspace(-1, 1, 100)

num_coeffs = 9
y_dataset_params = [0.] * num_coeffs
y_dataset_params[2] = 1
y_dataset = 0
for i in range(num_coeffs):
    y_dataset += y_dataset_params[i] * np.power(x_dataset, i)
y_dataset += np.random.randn(*x_dataset.shape) * 0.3
```

Создает фиктивные данные, $y = x^2$

```
(x_train, x_test, y_train, y_test) = split_dataset(x_dataset,
                                                   y_dataset, 0.7)
```

Разбивает набор данных на обучающие данные (70 %) и проверочные данные, используя программу, приведенную в листинге 3.4

```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
```

Определяет входные и выходные переменные-заполнители

```
def model(X, w):
terms = []
    for i in range(num_coeffs):
        term = tf.multiply(w[i], tf.pow(X, i))
        terms.append(term)
    return tf.add_n(terms)
```

Определяет модель

```
w = tf.Variable([0.] * num_coeffs, name="parameters")
y_model = model(X, w)
cost = tf.div(tf.add(tf.reduce_sum(tf.square(Y-y_model)),
                     tf.multiply(reg_lambda,
                     tf.reduce_sum(tf.square(w)))),
              2*x_train.size)
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Определяет регуляризованную функцию затрат

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
```

Определяет сеанс

```
for reg_lambda in np.linspace(0,1,100):
    for epoch in range(training_epochs):
        sess.run(train_op, feed_dict={X: x_train, Y: y_train})
    final_cost = sess.run(cost, feed_dict={X: x_test, Y:y_test})
    print('reg lambda', reg_lambda)
print('final cost', final_cost)

sess.close()
```

Пробует разные параметры регуляризации

Закрывает сеанс

Если вы выберете соответствующий выход для каждого параметра регуляризации из листинга 3.5, вы увидите, как изменяется кривая по мере увеличения λ. Если λ равна 0, алгоритм использует члены высокого порядка для соответствия данным. Если вы начинаете штрафовать параметры с высокой нормой L2, стоимость уменьшается, указывая на то, что происходит восстановление после переобучения, как это показано на рис. 3.13.

3.5. Применение линейной регрессии

Выполнять линейную регрессию на фейковых данных — это все равно что купить машину и никогда на ней не ездить. Любая машина надеется проявить себя в реальном мире! К счастью, чтобы испробовать ваши новоприобретенные знания о регрессии, в интернете есть различные базы данных для этого:

- Массачусетский университет в городе Амхерсте предоставляет небольшие базы данных разного типа: www.umass.edu/statdata/statdata.
- Kaggle содержит все типы крупномасштабных данных для машинного обучения: www.kaggle.com/datasets.
- Сайт Data.gov является базой данных, которая была открыта по инициативе правительства США, и в ней содержится много интересных и представляющих практическую ценность наборов данных: https://catalog.data.gov.

Большое число баз данных содержат даты. Например, есть набор данных всех телефонных неэкстренных вызовов 3-1-1 в Лос-Анджелесе, Калифорния. Его можно найти на сайте http://mng.bz/6vHx. Хорошей характеристикой для отслеживания может быть частота вызовов за день, неделю или месяц. Следующий листинг позволяет получить число вызовов за неделю.

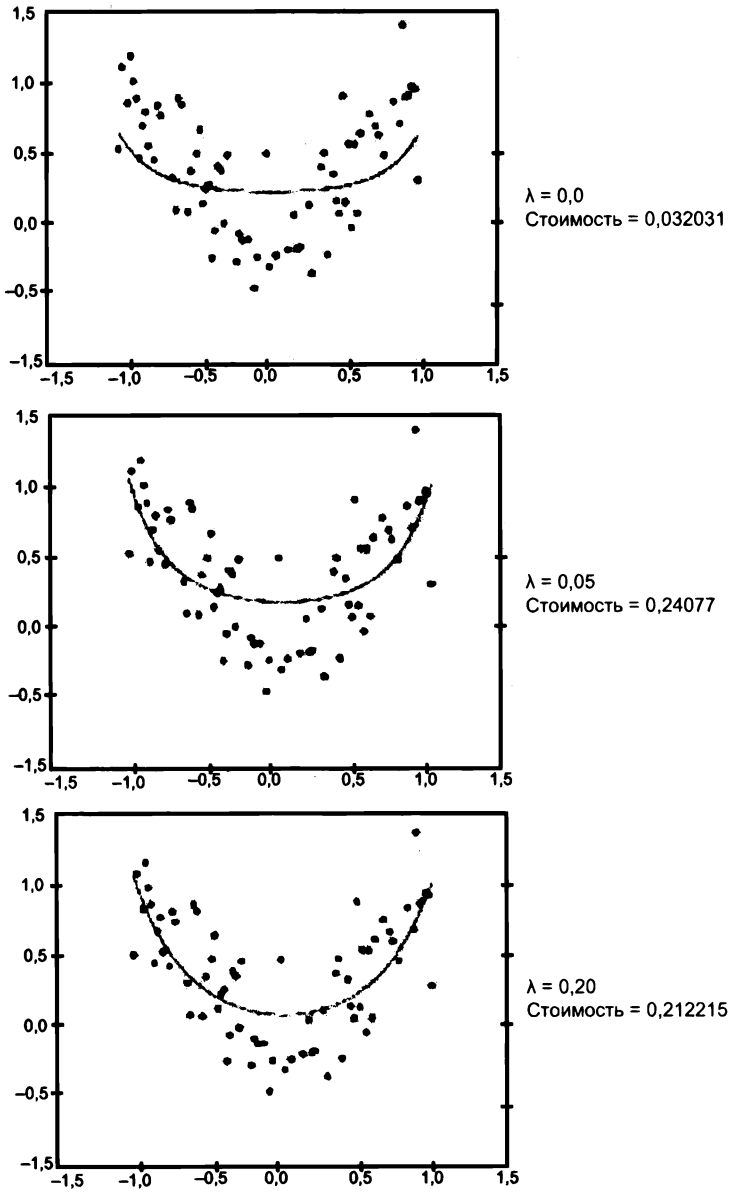


Рис. 3.13. При некотором увеличении параметра регуляризации стоимость начинает снижаться. Это позволяет предположить, что модель первоначально была переобучена, а регуляризация помогла внести новую структуру

Листинг 3.6. Обработка исходных баз данных CSV

```
import csv      ◄── Для облегчения считывания CSV-файлов
import time     ◄── Для использования полезных функций данных

def read(filename, date_idx, date_parse, year, bucket=7):

    days_in_year = 365

    freq = {}   ◄── Задает начальную карту частоты вызовов
    for period in range(0, int(days_in_year / bucket)):
        freq[period] = 0

    with open(filename, 'rb') as csvfile:   ◄── Считывает данные и суммирует
        csvreader = csv.reader(csvfile)         счет за период
        csvreader.next()
        for row in csvreader:
            if row[date_idx] == '':
                continue
            t = time.strptime(row[date_idx], date_parse)
            if t.tm_year == year and t.tm_yday < (days_in_year-1):
freq[int(t.tm_yday / bucket)] += 1

    return freq

freq = read('311.csv', 0, '%m/%d/%Y', 2014)   ◄── Получает еженедельный подсчет
                                                  частоты вызовов 3-1-1 за 2014 г.
```

Этот код использует обучающие данные для линейной регрессии. Переменная `freq` является словарем, который переводит период (например, неделю) в подсчет частоты. В году 52 недели, поэтому у нас есть 52 точки, если оставить `bucket=7` таким, какой он есть.

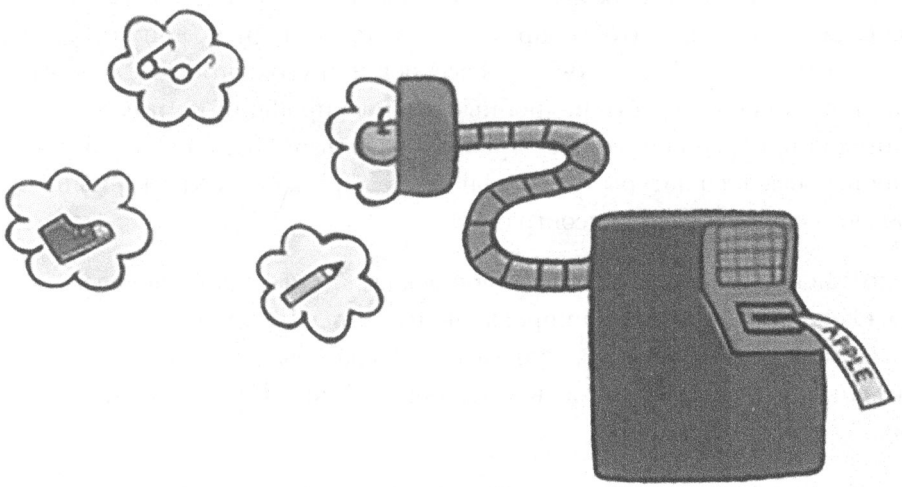
Теперь, когда есть точки данных, входные и выходные данные, необходимо аппроксимировать модель регрессии, используя метод, описанный в этой главе. В более практическом смысле обученную модель можно использовать для интерполяции или экстраполяции подсчетов частоты.

3.6. Краткие итоги

- Регрессия является одним из методов машинного обучения с учителем для прогнозирования выходных данных непрерывного типа.

- Определяя набор моделей, вы значительно сокращаете пространство поиска возможных функций. Кроме того, TensorFlow использует дифференцируемое свойство функций, выполняя при машинном обучении эффективную оптимизацию градиентным спуском.
- Линейную регрессию можно легко модифицировать на обучение модели, используя полиномы или другие, более сложные кривые.
- Во избежание переобучения для функции стоимости следует применить регуляризацию штрафованием параметров с большим значением.
- Если выход этой функции не непрерывный, тогда рекомендуется использовать алгоритм классификации (см. следующую главу).
- TensorFlow позволяет рационально и эффективно решать задачи машинного обучения на основе линейной регрессии, а также делать полезные прогнозы по важным вопросам, таким как сельскохозяйственное производство, сердечные заболевания, цены на жилье и многое другое.

Краткое введение в классификацию



Эта глава охватывает следующие темы:

- ✓ Запись формальных обозначений
- ✓ Применение логистической регрессии
- ✓ Работа с матрицей неточностей
- ✓ Знакомство с многоклассовой классификацией

Представьте, что рекламное агентство собирает информацию о пользовательских взаимодействиях, чтобы определить, какой тип объявлений показывать. Это не редкость. У Google, Twitter, Facebook и других гигантов, которые полагаются на рекламу, есть подробные личные профили их пользователей, помогающие персонализировать рекламу. Пользователь, который недавно искал игровые клавиатуры или графические карты, скорее всего, кликнет на объявлении о новейших видеоиграх.

Предоставление специально созданной рекламы для каждого человека может быть сложным, поэтому группировка пользователей по категориям является общей практикой. Например, пользователь может быть классифицирован как «геймер» и будет получать соответствующие объявления, связанные с видеоигрой.

Машинное обучение является инструментом перехода для выполнения этой задачи. На фундаментальном уровне машинного обучения специалисты стремятся создать инструмент, который способствовал бы пониманию данных. Классификация элементов данных, принадлежащих к разным категориям, является превосходным способом охарактеризовать данные для определенных целей.

Предыдущая глава была посвящена применению регрессии с целью аппроксимации кривой данных. Как вы помните, кривая наилучшего приближения является функцией, которая использует в качестве входных данных элемент базы данных и присваивает ему номер. Метод машинного обучения, который

вместо этого присваивает входным данным отдельные классы, носит название *классификации* (classification). Этот алгоритм обучения с учителем используется для дискретного выхода (каждое дискретное значение носит название *класс*). Входом обычно является вектор признаков, а выходом — класс. Если есть только два класса (например, такие, как Истина и Ложь, Вкл./Выкл., Да или Нет), то такой алгоритм обучения называют *бинарным классификатором* (binary classifier). Иначе он называется *многоклассовым классификатором* (multiclass classifier).

Существует много типов классификаторов, но в этой главе акцент делается на тех, которые представлены в табл. 4.1. У каждого из них есть свои преимущества и недостатки, в которых мы начнем разбираться лучше после их использования в TensorFlow.

Таблица 4.1. Классификаторы

Тип	Достоинства	Недостатки
Линейная регрессия	Простота реализации	Нет гарантии работоспособности. Поддержка только бинарных меток
Логистическая регрессия	Высокая точность. Универсальный метод регуляризации модели для самостоятельной настройки. Отклики модели являются мерой вероятности. Модель можно легко обновить, используя новые данные	Поддержка только бинарных меток
Многопеременная логистическая регрессия	Поддерживает многоклассовую классификацию. Отклики модели являются мерой вероятности	Сложнее в реализации

Линейную регрессию использовать проще всего, поскольку в главе 3 мы уже выполнили самую трудную часть работы, но, как будет видно в дальнейшем, эта регрессия является ужасным классификатором. Намного лучшим классификатором является алгоритм логистической регрессии. Чтобы определить функцию лучших затрат, он использует основное свойство логарифмов. И наконец, многопеременная логистическая регрессия представлет собой прямой метод решения задачи многоклассовой классификации. Она является естественным

обобщением логистической регрессии. Название «*многоклассовая логистическая регрессия*» этот метод получил из-за функции `softmax`, которая используется на последнем шаге его работы.

4.1. Формальные обозначения

Используя математические обозначения, классификатор является функцией $y = f(x)$, в которой аргумент x обозначает входные данные, а y — выходные данные, в частности категорию (рис. 4.1). Используя терминологию научной литературы, можно входной вектор x называть *независимой переменной*, а выходные данные y — *зависимой переменной*.

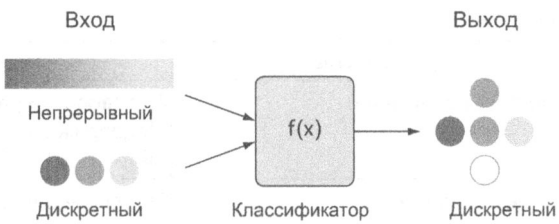


Рис. 4.1. Классификатор создает дискретные выходные данные, используя при этом непрерывные или дискретные входные данные

Строго говоря, метки классов ограничены диапазоном возможных значений. Можно представить себе двухзначные метки, такие же, как используемые в Python булевы переменные. Если входные признаки имеют только фиксированный набор возможных значений, необходимо убедиться, что используемая модель может понимать, как с ними обращаться. Поскольку набор функций в модели обычно связан с непрерывными действительными числами, набор данных необходимо обработать так, чтобы учесть дискретные переменные, которые делятся на два типа: ординальные и номинальные (рис. 4.2).

Значения ординального типа, в соответствии с их названием, можно упорядочить. Например, значения набора четных чисел от 1 до 10 являются ординальными, так как целые числа можно сравнивать между собой. С другой стороны, элементы из набора фруктов `{банан, яблоко, апельсин}` не могут быть

Й

упорядочены естественным образом. Мы называем элементы из этих наборов номинальными, потому что они могут быть описаны только по именам.

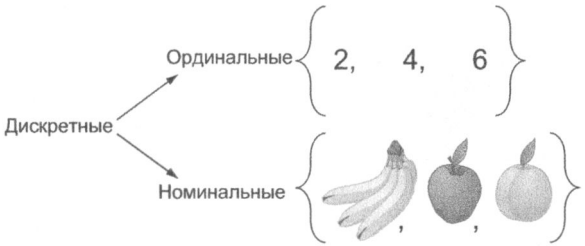


Рис. 4.2. Есть два типа дискретных наборов: имеющие значения, которые можно упорядочить (ординальные), и значения, которые упорядочить нельзя (номинальные)

Простой метод представления номинальных элементов в базе данных состоит в присваивании каждой метке номера. Наш набор `{банан, яблоко, апельсин}` может вместо этого рассматриваться как `{0, 1, 2}`. Но некоторые модели классификации могут быть очень предвзятыми. Это связано с поведением базы данных. Например, линейная регрессия может интерпретировать яблоко как нечто среднее между бананом и апельсином, что не имеет какого-то определенного смысла.

Обходной путь представления номинальных категорий зависимой переменной состоит в добавлении *фиктивных переменных* (dummy variables) для каждой номинальной переменной. В этом примере переменная `фрукт` будет удалена и заменена тремя различными переменными: `банан`, `яблоко` и `апельсин`. Каждая переменная принимает значение 0 или 1 (рис. 4.3) в зависимости от категории, для которой это значение является истиной. Этот процесс часто называют прямым кодированием.

Так же как линейная регрессия из главы 3, алгоритм обучения должен перебирать все возможные функции, поддерживаемые моделью, которую мы обозначим через M. В линейной регрессии модель использует параметр w. Функция $y = M(w)$ может пытаться оценить стоимость. В итоге мы выбираем w с наименьшей стоимостью. Единственным отличием регрессии от классификации является то, что выходные значения больше не являются непрерывными, а составляют набор дискретных меток классов.

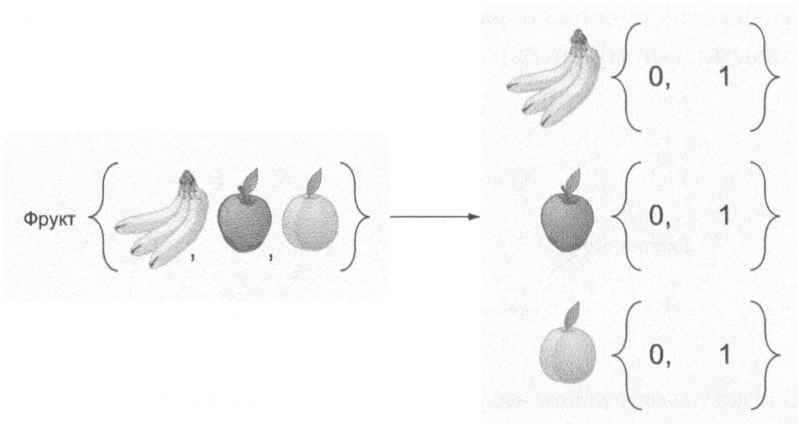


Рис. 4.3. Если переменные имеют номинальные значения, их следует предварительно обработать. Один из методов состоит в обращении с каждой номинальной величиной как с булевой переменной, как это показано на рисунке справа: банан, яблоко и апельсин являются тремя добавленными переменными, принимающими значение 0 или 1. Исходную переменную фрукт удаляют

УПРАЖНЕНИЕ 4.1

Что удобнее — рассматривать приведенные далее задачи как регрессию или как классификацию? (А) Прогнозирование курса акций. (Б) Принятие решения, какие из акций продавать, покупать или удерживать. (В) Оценка качества компьютера по 10-балльной шкале

ОТВЕТ

(А) Регрессия. (Б) Классификация. (В) Любая из них.

Поскольку типы ввода/вывода для регрессии еще более общие, чем типы классификации, ничто не мешает вам использовать алгоритм линейной регрессии для задач классификации. Фактически это именно то, что вы сделаете в разделе 4.3. Прежде чем приступать к реализации кода TensorFlow, важно оценить эффективность классификатора. В следующем разделе рассматриваются самые современные подходы к измерению успешности классификатора.

4.2. Оценка эффективности

Перед тем как приступать к составлению алгоритма классификации, необходимо проверить успешность его результатов. В этом разделе описываются основные методы оценки эффективности решения задач классификации.

4.2.1. Правильность

Помните ли вы примеры многовариантного выбора в средней школе или в вузе? Задачи классификации в машинном обучении аналогичны им. Если задан оператор, вашей задачей является классифицировать его как один из заданных многовариантных «ответов». Если есть только два варианта выбора, как в случае с примером, в котором использовались «истина» или «ложь», мы называем этот случай *бинарной классификацией*. Если бы это были выпускные экзамены в школе, типичным способом измерения вашей оценки был бы подсчет количества правильных ответов и деление их на общее количество вопросов.

Машинное обучение использует такую же стратегию оценки и называет эту оценку *правильностью* (accuracy). Правильность оценивается по следующей формуле:

$$\text{правильность} = \frac{\#\text{правильные заключения}}{\#\text{все заключения}}.$$

Эта формула дает грубую оценку эффективности, которая может быть достаточной, если вас не беспокоит итоговая правильность алгоритма. Но оценка правильности не подразумевает разделения результатов для каждого класса на правильные и неправильные.

Для учета этого ограничения применяется *матрица неточностей* (confusion matrix), которая дает более детальную оценку эффективности метода классификации. Полезно также получить эффективность классификатора, проверив его для каждого класса.

Например, рассмотрим бинарный классификатор с «положительными» и «отрицательными» метками классов. Как показано на рис. 4.4, матрица неточностей представляет собой таблицу, которая сравнивает прогнозируемый отклик с фактическим. Если элементы данных были правильно спрогнозированы как

положительные, такие заключения называют *истинно положительными* (ИП). Если элементы были спрогнозированы неправильно как положительные, такие заключения называют *ложноположительными* (ЛП). Если алгоритм случайно прогнозирует элемент как отрицательный, когда он на самом деле положительный, то это заключение называют *ложноотрицательным* (ЛО). И наконец, когда прогноз и действительность совпадают в том, что элементу данных присвоена метка отрицательного элемента, это заключение называют *истинно отрицательным* (ИО). Обратите внимание на то, что эта матрица носит название *матрицы неточностей*, потому что позволяет оценить, как часто модель путает два класса при попытке их дифференцирования.

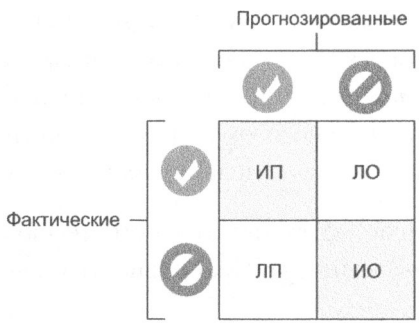


Рис. 4.4. Можно сравнивать спрогнозированные результаты с фактическими результатами, используя матрицу положительных (галочка) и отрицательных (запрещающий знак) меток

4.2.2. Точность и полнота

Хотя истинно положительные (ИП), ложноположительные (ЛП), истинно отрицательные (ИО) и ложноотрицательные (ЛО) заключения имеют значение по отдельности, эффективность классификации определяется их взаимодействием.

Отношение истинно положительных заключений к суммарно положительным носит название *точности* (precision). Оно является мерой вероятности того, что положительный прогноз окажется правильным. Левый столбец на рис. 4.4

является суммарным числом прогнозов (ИП + ЛП), поэтому точность определяется следующим уравнением:

$$\text{точность} = \frac{\text{ИП}}{\text{ЛП}}.$$

Отношение истинно положительных заключений ко всем возможным положительным заключениям носит название *полноты* (recall). Это отношение является мерой частоты истинно положительных заключений. А именно того, как много истинно положительных заключений было успешно спрогнозировано. Верхняя строка на рис. 4.4 является суммарным числом положительных заключений (ИП + ЛП), поэтому полнота определяется следующим уравнением:

$$\text{полнота} = \frac{\text{ИП}}{\text{ЛО}}.$$

Проще говоря, точность является мерой правильных прогнозов алгоритма, а полнота является мерой правильных результатов, полученных алгоритмом на окончательном наборе. Если точность выше полноты, модель лучше идентифицирует правильные элементы, чем не идентифицирует некоторые неправильные элементы, и наоборот.

Давайте быстро рассмотрим пример. Предположим, вы пытаетесь идентифицировать кошек в наборе из 100 изображений; 40 фотографий — кошки, а 60 — собаки. Когда вы запускаете классификатор, 10 из кошек идентифицируются как собаки, а 20 из собак идентифицируются как кошки. Ваша матрица неточностей выглядит как на рис. 4.5.

Матрица неточностей		Прогнозируемые	
		Кошка	Собака
Фактические	Кошка	30 Истинно положительные	20 Ложноположительные
	Собака	10 Ложноотрицательные	40 Истинно отрицательные

Рис. 4.5. Пример матрицы неточностей при оценке эффективности алгоритма классификации

Вы можете увидеть общее количество кошек в левой части столбца прогноза: 30 определены правильно, а 10 нет, общее число составляет 40.

УПРАЖНЕНИЕ 4.2

Какова точность и полнота для кошек? Какова правильность системы?

ОТВЕТ

Для кошек точность составляет 30 / (30 + 20), или 3/5. Полнота: 30 / (30 + 10), или 3/4. Правильность: (30 + 40) / 100, или 70 %.

4.2.3. Кривая ошибок

Поскольку бинарная классификация является одним из наиболее популярных инструментов, существует много проверенных методов оценки ее эффективности, таких как, например, кривая ошибок (ROC-кривая, receiver operating characteristic). ROC-кривая дает возможность сравнивать соотношения между ложноположительными и истинно положительными заключениями. По оси X откладываются ложноположительные, а по оси Y — истинно положительные значения.

Бинарная классификация уменьшает вектор признаков до одного числа и затем описывает класс на основании того, больше или меньше это число указанной пороговой величины. При изменении пороговой величины классификации в машинном обучении на график зависимости наносятся различные значения частоты ложноположительных и истинно положительных заключений.

Надежным способом сравнения различных алгоритмов классификации является сравнение их ROC-кривых. Если две кривые не пересекаются, то один из методов определенно лучше другого. Хорошие алгоритмы находятся значительно выше базовой линии. Количественное сравнение алгоритмов классификации осуществляется измерением площади по ROC-кривой. Если у модели есть область под кривой, значению выше 0,9 соответствует превосходный алгоритм классификации. Модель, случайным образом определяющая выходное значение, будет иметь площадь под кривой, приблизительно равную 0,5. Пример приведен на рис. 4.6.

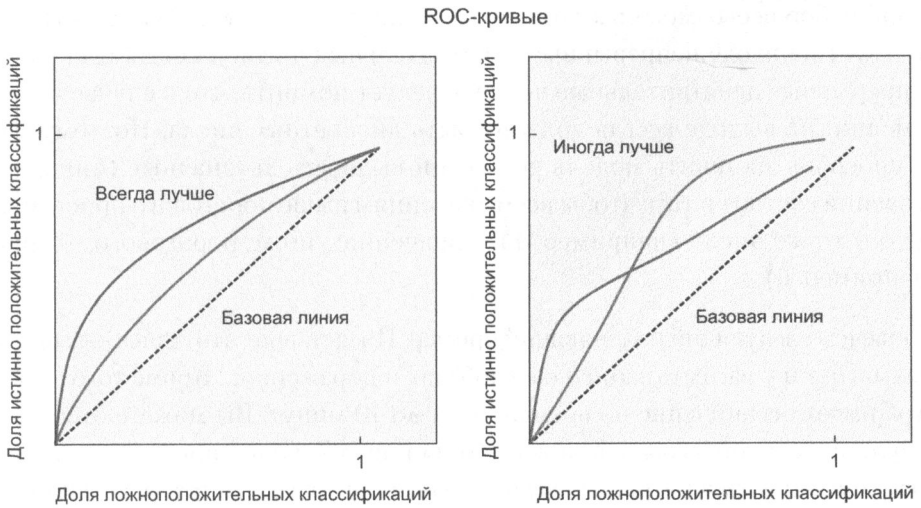


Рис. 4.6. Принципиальным способом сравнения алгоритмов является проверка ROC-кривых. Когда во всех случаях относительное число истинно положительных заключений выше, чем ложноположительных, можно сделать вывод, что используемый алгоритм доминирует по своей эффективности. Когда относительное число истинно положительных заключений меньше, чем ложноположительных, график оказывается значительно ниже базовой линии, отмеченной пунктиром

УПРАЖНЕНИЕ 4.3

Как будет выглядеть доля 100 % правильных заключений (все истинно положительные, без ложноположительных) в виде точки на ROC-кривой?

ОТВЕТ

Точка для 100 % правильного относительного числа будет расположена на положительной части оси *Y* ROC-кривой.

4.3. Использование для классификации линейной регрессии

Одним из несложных вариантов применения алгоритма классификации является соответствующая настройка алгоритма линейной регрессии, такая же, как была описана в главе 3. Напомним, модель линейной регрессии представляет

собой набор всевозможных линейных функций, $f(x) = wx$. Функция $f(x)$ использует на входе непрерывные действительные числа и создает на выходе непрерывные действительные числа. Следует помнить, что в случае классификации на выходе всегда должны быть дискретные числа. Поэтому один из способов заставить модель регрессии выдавать двузначные (бинарные) значения состоит в том, чтобы всем значениям выше порогового присваивать одно и то же число (например, 1), а значениям ниже порогового — другое (например, 0).

Приведем следующий поясняющий пример. Представьте, что Алиса — азартный шахматист и у вас есть записи о ее победах и поражениях. Кроме того, каждая игра имеет ограничение по времени от 1 до 10 минут. Вы можете отобразить результат каждой игры, как показано на рис. 4.7. Ось x представляет собой временной интервал игры, а ось y показывает, выиграла Алиса ($y = 1$) или проиграла ($y = 0$).

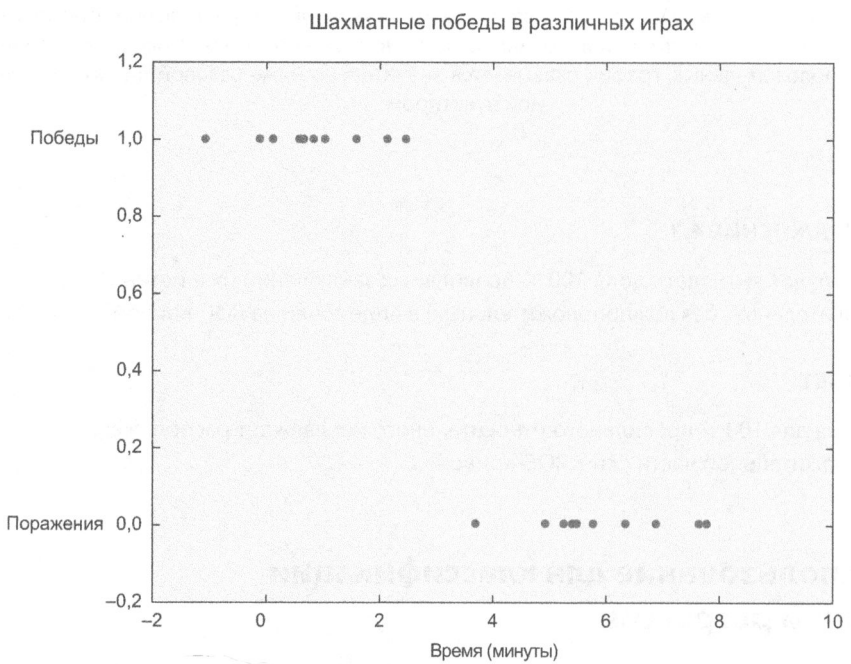


Рис. 4.7. Визуальное представление обучающего набора данных для бинарной классификации. Все значения разбиты на два класса: все точки с $y = 1$ и все точки с $y = 0$

Как видно из приведенных данных, Алиса быстро думает: она всегда выигрывает в блиц-турнирах. Но она обычно проигрывает в матчах, которые длятся дольше. По построенному графику можно спрогнозировать критическую продолжительность матча, позволяющую решить, сможет ли Алиса победить.

Вы хотите вызвать ее на игру, в которой сможете с уверенностью победить. Если выбирать заведомо продолжительные матчи, которые займут 10 минут, она откажется играть. Поэтому давайте выберем как можно более короткое время игры, на которое Алиса согласится, одновременно смещая баланс в вашу пользу.

Линейная аппроксимация данных обеспечит то, с чем можно будет работать. На рис. 4.8 показана линия наилучшего приближения, полученная с помощью алгоритма линейной регрессии из листинга 4.1 (ниже). Значение линии ближе к 1, чем к 0, для тех игр, которые Алиса может выиграть. Оказывается, что, если выбрать время, соответствующее тому, при котором значение линии меньше, чем 0,5 (то есть когда Алиса с большей вероятностью проиграет, чем выиграет), у вас появится хороший шанс выиграть.

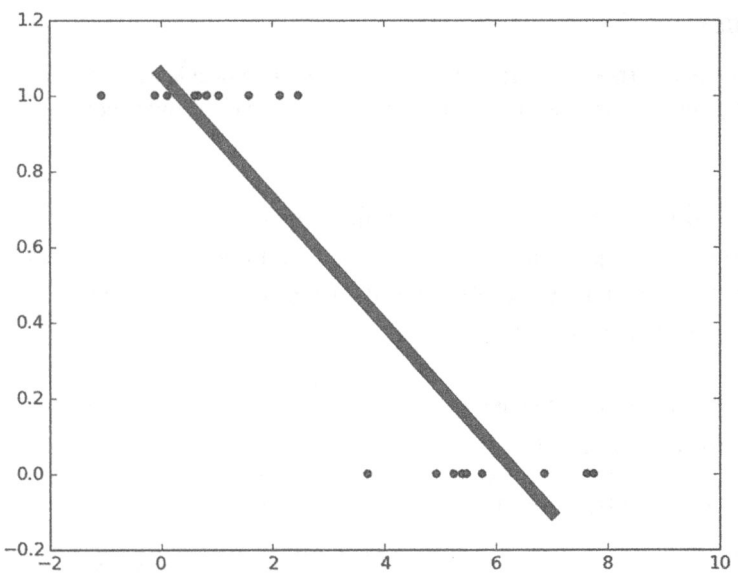


Рис. 4.8. Диагональная линия является линией наилучшего приближения для данных, используемых при классификации. Очевидно, эта линия не аппроксимирует надлежащим образом эти данные, однако она предоставляет неточный метод классификации новых данных

Эта линия стремится аппроксимировать данные насколько это возможно. Благодаря природе обучающих данных эта модель будет давать на выходе значения, близкие к 1, для положительных примеров и значения, близкие к 0, для отрицательных примеров. Поскольку моделирование этих данных выполняется с помощью прямой линии, то некоторые входные данные могут давать значения между 0 и 1. Как вы можете себе представить, значения, заходящие слишком глубоко в одну из категорий, приведут к значениям, большим 1 или меньшим 0. Вам нужно решить, когда элемент принадлежит к одной категории в большей степени, чем к другой. Как правило, вы выбираете среднюю точку 0,5 в качестве решающей границы (также называемой *порогом*, threshold). Вы можете видеть, что эта процедура использует линейную регрессию для выполнения классификации.

УПРАЖНЕНИЕ 4.4

Какие недостатки у линейной регрессии при ее применении для классификации? (См. подсказку в листинге 4.4.)

ОТВЕТ

Линейная регрессия чувствительна к резко отклоняющимся значениям в ваших данных, поэтому она не является точным классификатором.

Давайте напишем ваш первый классификатор! Откроем новый файл Python и назовем его linear.py. Запустите код из следующего листинга. В TensorFlow следует задать узлы переменной-заполнителя, а затем ввести в них переменные из оператора `session.run()`.

Листинг 4.1. Использование линейной регрессии для классификации

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x_label0 = np.random.normal(5, 1, 10)
x_label1 = np.random.normal(2, 1, 10)
xs = np.append(x_label0, x_label1)
labels = [0.] * len(x_label0) + [1.] * len(x_label1)

plt.scatter(xs, labels)
```

Импортирует библиотеку TensorFlow для использования обучающего алгоритма NumPy для обработки данных и matplotlib для их визуализации

Инициализирует фиктивные данные, 10 экземпляров каждой метки

Инициализирует соответствующие метки

Вычисляет данные

```
learning_rate = 0.001
training_epochs = 1000

X = tf.placeholder("float")
Y = tf.placeholder("float")

def model(X, w):
    return tf.add(tf.multiply(w[1], tf.pow(X, 1)),
                  tf.multiply(w[0], tf.pow(X, 0)))

w = tf.Variable([0., 0.], name="parameters")
y_model = model(X, w)
cost = tf.reduce_sum(tf.square(Y-y_model))

train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Декларирует гиперпараметры

Определяет узлы для пар входных/выходных данных

Определяет модель прямой линии y = w1 * x + w0

Определяет переменные

Определяет функцию затрат

Определяет вспомогательную переменную, так как обращаться к ней придется много раз

Задает правило определения параметров при обучении

После создания графа TensorFlow в следующем листинге вы увидите, как открыть новый сеанс и построить граф. Оператор `train_op` обновляет параметры модели для все более точных предположений. Оператор `train_op` запускается циклически много раз, итеративно улучшая оценку параметра. Следующий листинг генерирует график, аналогичный рис. 4.8.

Листинг 4.2. Выполнение графа

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
    sess.run(train_op, feed_dict={X: xs, Y: labels})
    current_cost = sess.run(cost, feed_dict={X: xs, Y: labels})
    if epoch % 100 == 0:
        print(epoch, current_cost)

w_val = sess.run(w)
print('learned parameters', w_val)

sess.close()

all_xs = np.linspace(0, 10, 100)
plt.plot(all_xs, all_xs*w_val[1] + w_val[0])
plt.show()
```

Записываются затраты, полученные для текущих параметров

Открывает новый сеанс и инициализирует переменные

Многократно выполняет операцию обучения

Выводит сведения из лога в процессе выполнения кода

Печатает параметры, полученные в ходе обучения

Закрывает сеанс, если он больше не используется

Показывает линию наилучшего приближения

Для оценки эффективности можно подсчитать число правильных прогнозов и определить долю успешных. В следующем листинге добавляются еще два узла к предыдущему коду в файле linear.py, которые называются `correct_prediction` и `accuracy`. Теперь можно вывести оценку правильности, чтобы увидеть долю успешных прогнозов. Код может быть выполнен непосредственно перед закрытием сеанса.

Листинг 4.3. Оценка правильности

Если модель дает на выходе значение больше, чем 0,5, в этом случае метка должна быть положительной, и наоборот

```
correct_prediction = tf.equal(Y, tf.to_float(tf.greater(y_model, 0.5)))
accuracy = tf.reduce_mean(tf.to_float(correct_prediction))

print('accuracy', sess.run(accuracy, feed_dict={X: xs, Y: labels}))
```

Рассчитывается доля успешных прогнозов

Выводит оценку успешности по предоставленным входным данным

Предыдущий код выводит следующий результат:

```
('learned parameters', array([ 1.2816, -0.2171], dtype=float32))
('accuracy', 0.95)
```

Если бы классификацию можно было осуществлять так просто, эту главу можно было бы уже завершить. К сожалению, метод линейной регрессии не справляется, если обучение проводить по более экстремальным данным, которые включают *резко отклоняющиеся значения*, или *выбросы* (outliers).

Например, пусть Алиса проиграла партию, которая длилась 20 минут. Вы обучаете классификатор на данных, которые включают точку выброса. В следующем листинге заменяется время одной из партий со значением 20. Давайте посмотрим, как появление выброса влияет на работу классификатора.

Листинг 4.4. Линейная регрессия не справляется с классификацией

```
x_label0 = np.append(np.random.normal(5, 1, 9), 20)
```

Когда вы повторно запустите код с этими изменениями, вы увидите результат, аналогичный приведенному на рис. 4.9.

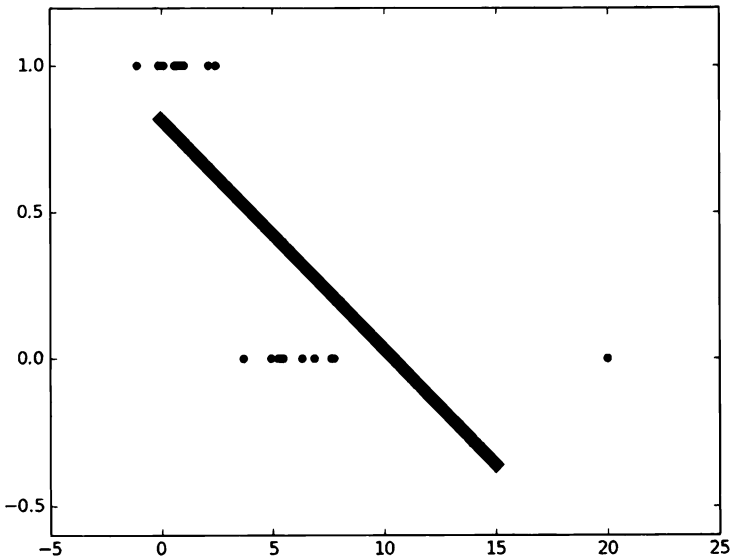


Рис. 4.9. Новый элемент в обучающих данных со значением 20 сильно влияет на линию наилучшего приближения. Эта линия слишком чувствительна к данным с выбросом, и поэтому линейная регрессия не справляется с классификацией

Исходный классификатор предположил, что вы можете обыграть Алису в трехминутной игре. Вероятно, она согласилась бы сыграть в такую короткую игру. Но пересмотренный классификатор, если вы придерживаетесь того же порога 0,5, теперь предполагает, что самая короткая игра, которую она проиграет, — это пятиминутная игра. Скорее всего, Алиса откажется играть в такую длинную игру!

4.4. Использование логистической регрессии

Логистическая регрессия использует аналитическую функцию с теоретическими гарантиями правильности и эффективности. Она аналогична линейной регрессии, за исключением того, что там используется другая функция стоимости и несколько изменена функция отклика модели.

Перейдем к приведенной здесь линейной функции:

$$y(x) = wx.$$

В линейной регрессии линия с отличным от нуля наклоном может быть в пределах от минус бесконечности до плюс бесконечности. Если единственным разумным результатом для классификации являются 0 или 1, то это будет очевидно и не потребуется подбор функции с такими свойствами. К счастью, сигмовидная функция, приведенная на рис. 4.10, хорошо справляется с этим, так как быстро сходится к 0 или к 1.

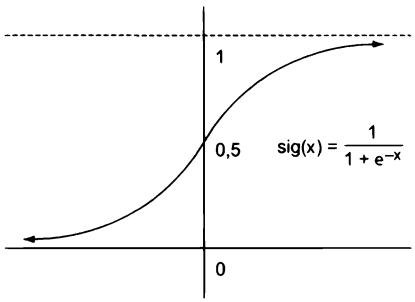


Рис. 4.10. Визуализация сигмовидной функции

Если $x = 0$, сигмовидная функция дает 0,5. По мере увеличения x функция стремится к 1. А по мере уменьшения x до минус бесконечности функция стремится к 0.

В логистической регрессии нашей моделью является *sig* (линейная зависимость (x)). Как оказалось, параметры наилучшей настройки этой функции предполагают линейное разделение двух классов. Эту разделяющую линию также называют *линейной границей решения* (linear decision boundary).

4.4.1. Решение одномерной логистической регрессии

Функция стоимости, используемая в логистической регрессии, несколько отличается от той, которая используется в линейной регрессии. Хотя можно было бы использовать ту же самую функцию затрат, что и раньше, но ее использование не даст такого же быстрого решения или гарантии, что это решение будет оптимальным. Виной этому является сигмовидная функция, которая приводит к тому, что у функции стоимости появляется множество «выступов». TensorFlow и большинство других библиотек машинного обучения лучше

всего работают с простыми функциями стоимости. Специалисты нашли, как изменить функцию затрат и использовать сигмовидные кривые для логистической регрессии.

Новая функция стоимости между фактическим значением y и откликом модели h будет описываться следующим уравнением, состоящим из двух частей:

$$\mathrm{Cost}(y,\ h)=\begin{cases}-\log(h), & \text{если } y=1\\ \log(1-h), & \text{если } y=0\end{cases}.$$

Эти два уравнения можно собрать в одно длинное уравнение:

$$\mathrm{Cost}(y,h)=-y\log(h)-(1-y)\log(1-h).$$

Эта функция обладает именно теми качествами, которые необходимы для эффективного и оптимального обучения. Точнее говоря, она является выпуклой, но не стоит беспокоиться о том, что из этого следует. Мы стремимся свести к минимуму затраты: представьте затраты высотой, а функцию затрат поверхностью Земли. Мы стараемся найти самую низкую точку на поверхности Земли. Намного легче найти самую низкую точку на поверхности Земли, если для этого не потребуется забираться в горы. Такую область называют *выпуклой поверхностью* (convex). Там нет горбов.

Процесс поиска можно представить как шар, который скатывается вниз с возвышенности. В конце концов шар остановится в нижней части, которая является *оптимальной точкой*. Выпуклая функция может иметь неровную поверхность, что затруднит прогноз, куда покатится шар. Он может даже не закончить скатывание в самой низкой точке. Ваша функция является выпуклой, поэтому алгоритм может легко определить, как минимизировать эти затраты и «скатить шар с возвышения».

Выпуклость является хорошим свойством, но возможность получения правильных результатов также является важным критерием при выборе функции стоимости. Откуда вы знаете, что эта функция стоимости выполняет в точности то, для чего она предназначена? Для того чтобы ответить на этот вопрос наглядным образом, взгляните на рис. 4.11. Для расчета стоимости используется $-log(x)$, когда требуется получить 1 (обратите внимание, что $-log(1)=0$). Этот алгоритм всегда отклоняется от значения 0, потому что в этом случае стоимость устремляется к бесконечности. Сложение этих функций вместе дает кривую,

которая стремится к бесконечности при 0 и 1, при исключении отрицательных частей.

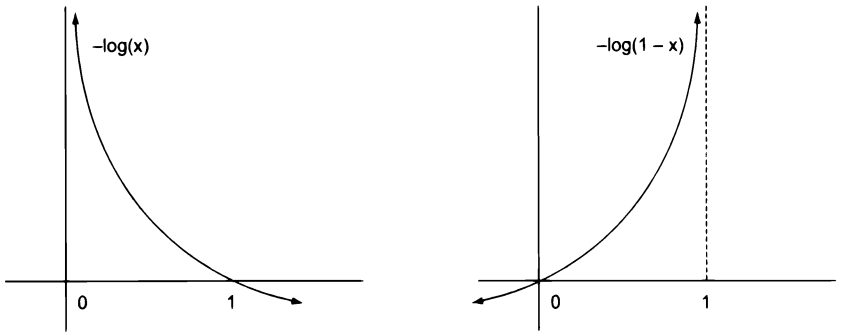


Рис. 4.11. Вот визуализация того, как две функции стоимости штрафуют значения при 0 и 1. Обратите внимание, что левая функция сильно штрафует при 0, но не имеет стоимости при 1. Функция стоимости справа отображает противоположные явления

Конечно, рисунки — это неформальный способ убедить вас, но технические пояснения того, почему функция стоимости является оптимальной, выходят за рамки этой книги. Если вас интересует находящаяся в основе этого математика, вам будет любопытно узнать, что функция стоимости основана на принципе максимальной энтропии, познакомиться с которым можно в интернете.

На рис. 4.12 приводятся результаты оптимальной настройки на основе логистической регрессии, использующей одномерный набор данных. Сигмовидная кривая обеспечивает лучшую линейную границу решения, чем та, которую дает линейная регрессия.

Вы начнете замечать шаблоны в листингах. При обычном использовании TensorFlow сначала создается набор фиктивных данных, задаются переменные-заполнители, обычные переменные, модель, функция стоимости для этой модели (обычно это означает квадратичную ошибку или логарифмическую среднюю квадратичную ошибку), создается оператор `train_op` с помощью градиентного спуска, итеративно подаются примеры данных (возможно, с меткой или выходными данными) и, наконец, собираются оптимизированные значения.

Создайте новый исходный файл logistic_1d.py и скопируйте в него код из листинга 4.5, который будет генерировать график с рис. 4.12.

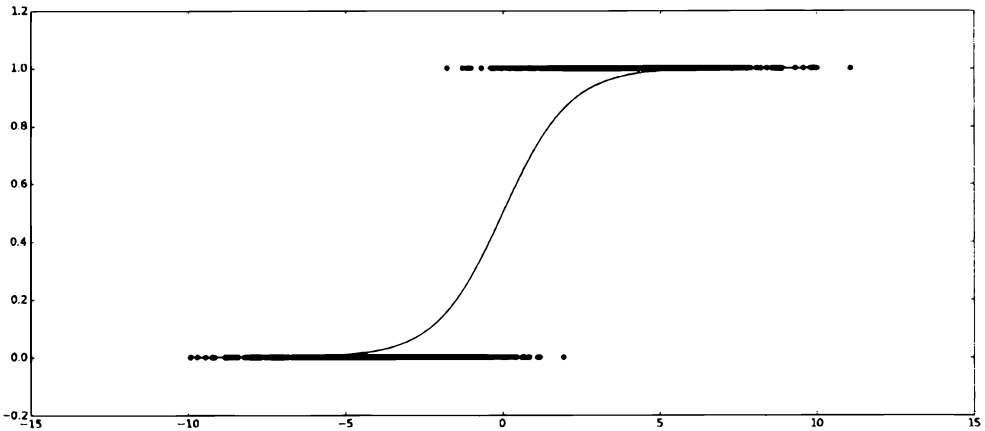


Рис. 4.12. На этом рисунке приведена оптимальная сигмовидная кривая для бинарной классификации набора данных. Обратите внимание, что кривая целиком находится в области между y = 0 и y = 1. Поэтому эта кривая нечувствительна к выбросам

Листинг 4.5. Использование одномерной логистической регрессии

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
learning_rate = 0.01
training_epochs = 1000

def sigmoid(x):
    return 1. / (1. + np.exp(-x))

x1 = np.random.normal(-4, 2, 1000)
x2 = np.random.normal(4, 2, 1000)
xs = np.append(x1, x2)
ys = np.asarray([0.] * len(x1) + [1.] * len(x2))

plt.scatter(xs, ys)

X = tf.placeholder(tf.float32, shape=(None,), name="x")
Y = tf.placeholder(tf.float32, shape=(None,), name="y")
w = tf.Variable([0., 0.], name="parameter", trainable=True)
y_model = tf.sigmoid(w[1] * X + w[0])
cost = tf.reduce_mean(-Y * tf.log(y_model) - (1 - Y) * tf.log(1 - y_model))
```

Определяет узел параметра

Импортирует необходимые библиотеки

Определяет гиперпараметры

Определяет вспомогательную функцию для вычислений сигмовидной функции

Инициализирует фиктивные данные

Визуализация данных

Определяет входные/выходные переменные-заполнители

Задает кросс-энтропийную функцию потерь

Задает модель, используя сигмовидную функцию из библиотеки

Определяет минимизирующую переменную

```
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    prev_err = 0
    for epoch in range(training_epochs):
        err, _ = sess.run([cost, train_op], {X: xs, Y: ys})
        print(epoch, err)
        if abs(prev_err - err) < 0.0001:
            break
        prev_err = err
    w_val = sess.run(w, {X: xs, Y: ys})

all_xs = np.linspace(-10, 10, 100)
plt.plot(all_xs, sigmoid((all_xs * w_val[1] + w_val[0])))
plt.show()
```

Открывает новый сеанс и инициализирует все переменные

Проверяет сходимость: при изменении менее чем на 0,01 % за итерацию считается, что алгоритм сошелся

Строит полученную в результате машинного обучения сигмовидную функцию

Получает после машинного обучения значения параметров

Обновляет предыдущее значение ошибки

Определяет затраты и обновляет параметры обучения

Повторяет итерации до сходимости или до достижения максимального числа итераций

Определяет переменную, чтобы отслеживать предыдущую ошибку

КРОСС-ЭНТРОПИЙНЫЕ ПОТЕРИ В TENSORFLOW

Как можно увидеть из листинга 4.5, кросс-энтропийные потери усредняются по каждой паре входных/выходных данных с помощью оператора `tf.reduce_mean`. Другой удобной и более общей функцией, предоставляемой TensorFlow, является функция с названием `tf.nn.softmax_cross_entropy_with_logits`. Дополнительные сведения о ней можно найти в официальной документации: http://mng.bz/8mEk.

И вот оно! Если вы играли в шахматы против Алисы, то теперь у вас есть бинарный классификатор для порога принятия решений, который указывает, в каких случаях партию можно выиграть, а в каких нет.

4.4.2. Решение двумерной логистической регрессии

Теперь давайте посмотрим, как можно использовать логистическую регрессию со многими независимыми переменными. Число независимых переменных соответствует размерности. В нашем случае с помощью решения задачи двумерной логистической регрессии будет выполнена классификация пары независимых переменных. Идеи, с которыми вы познакомитесь в этом разделе, можно экстраполировать на произвольную размерность.

ПРИМЕЧАНИЕ Предположим, вы подумываете о покупке нового телефона. Единственными характеристиками, которые вам нужны, являются: 1) операционная система, 2) размер и 3) стоимость. Цель состоит в том, чтобы решить, стоит ли покупать телефон. В этом случае есть три независимые переменные (характеристики телефона) и одна зависимая переменная (стоит ли покупать). Поэтому мы рассматриваем это как проблему классификации, в которой входной вектор является трехмерным.

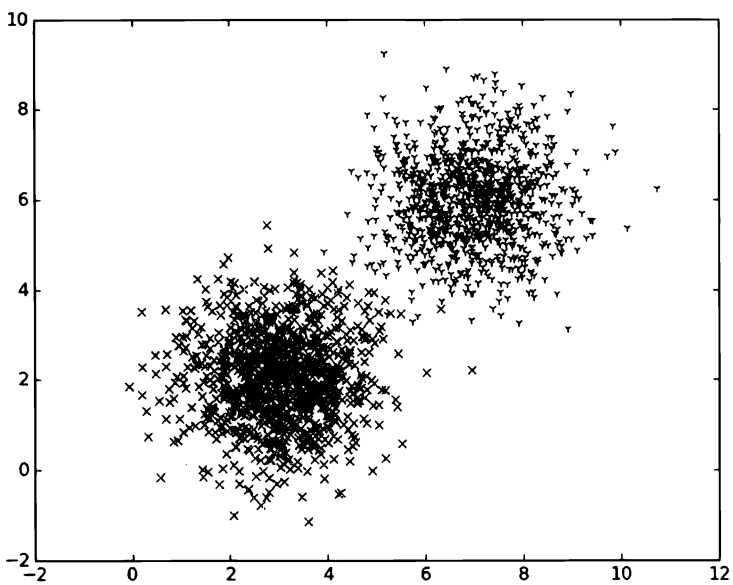


Рис. 4.13. Оси *x* и *y* представляют две независимые переменные. Зависимая переменная содержит два возможных класса, различающихся формой нанесенных точек

Рассмотрим базу данным, приведенную на рис. 4.13. Она представляет криминальную активность двух банд в городе. Первым измерением является ось x, которую можно рассматривать как широту, а вторым измерением является ось y, представляющая долготу. Один кластер образован вокруг точки (3, 2), а второй — вокруг точки (7, 6). Ваша задача — решить, какая из банд вероятнее всего несет ответственность за новое преступление, произошедшее в точке (6, 4).

Создадим новый файл и назовем его logistic_2d.py, а затем будем действовать в соответствии с листингом 4.6.

Листинг 4.6. Настройка данных для двумерной логистической регрессии

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

learning_rate = 0.1
training_epochs = 2000

def sigmoid(x):
    return 1. / (1. + np.exp(-x))

x1_label1 = np.random.normal(3, 1, 1000)
x2_label1 = np.random.normal(2, 1, 1000)
x1_label2 = np.random.normal(7, 1, 1000)
x2_label2 = np.random.normal(6, 1, 1000)
x1s = np.append(x1_label1, x1_label2)
x2s = np.append(x2_label1, x2_label2)
ys = np.asarray([0.] * len(x1_label1) + [1.] * len(x1_label2))
```

Импортирует необходимые библиотеки

Определяет гиперпараметры

Задает вспомогательную сигмовидную функцию

Инициализирует фейковые данные

У вас две независимые переменные ($x1$ и $x2$). Самый простой способ смоделировать соответствие между входным значением x и выходным $M(x)$ состоит в использовании следующего уравнения, где w — параметр, который определяется с помощью TensorFlow:

$$M(x, w) = sig(w_2 x_2 + w_1 x_1 + w_0).$$

В следующем листинге вы реализуете уравнение и соответствующую функцию стоимости, чтобы узнать параметры.

Листинг 4.7. Использование TensorFlow для решения задачи многомерной логистической регрессии

Определяет сигмовидную модель, используя обе входные переменные

```
X1 = tf.placeholder(tf.float32, shape=(None,), name="x1")
X2 = tf.placeholder(tf.float32, shape=(None,), name="x2")
Y = tf.placeholder(tf.float32, shape=(None,), name="y")
w = tf.Variable([0., 0., 0.], name="w", trainable=True)

y_model = tf.sigmoid(w[2] * X2 + w[1] * X1 + w[0])
cost = tf.reduce_mean(-tf.log(y_model * Y + (1 - y_model) * (1 - Y)))
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    prev_err = 0
    for epoch in range(training_epochs):
        err, _ = sess.run([cost, train_op], {X1: x1s, X2: x2s, Y: ys})
        print(epoch, err)
        if abs(prev_err - err) < 0.0001:
            break
        prev_err = err
    w_val = sess.run(w, {X1: x1s, X2: x2s, Y: ys})

x1_boundary, x2_boundary = [], []
for x1_test in np.linspace(0, 10, 100):
    for x2_test in np.linspace(0, 10, 100):
    z = sigmoid(-x2_test*w_val[2] - x1_test*w_val[1] - w_val[0])
    if abs(z - 0.5) < 0.01:
        x1_boundary.append(x1_test)
        x2_boundary.append(x2_test)

plt.scatter(x1_boundary, x2_boundary, c='b', marker='o', s=20)
plt.scatter(x1_label1, x2_label1, c='r', marker='x', s=20)
plt.scatter(x1_label2, x2_label2, c='g', marker='1', s=20)

plt.show()
```

Определяет узлы входных/выходных переменных-заполнителей

Определяет узел параметра

Определяет шаг обучения

Задает массив для граничных точек

Проходит циклом по всем точкам

Показывает граничную линию вместе с данными

Получает в процессе обучения значения параметров, перед тем как закрыть сеанс

Если отклик модели близок к 0,5, обновляются граничные точки

Создает новый сеанс, инициализирует переменные и добивается в процессе обучения сходимости параметров

На рис. 4.14 приводится линия границы, полученная в результате обучения с помощью обучающих данных. Преступление, которое ложится на эту линию, с равной вероятностью могло быть совершено каждой из банд.

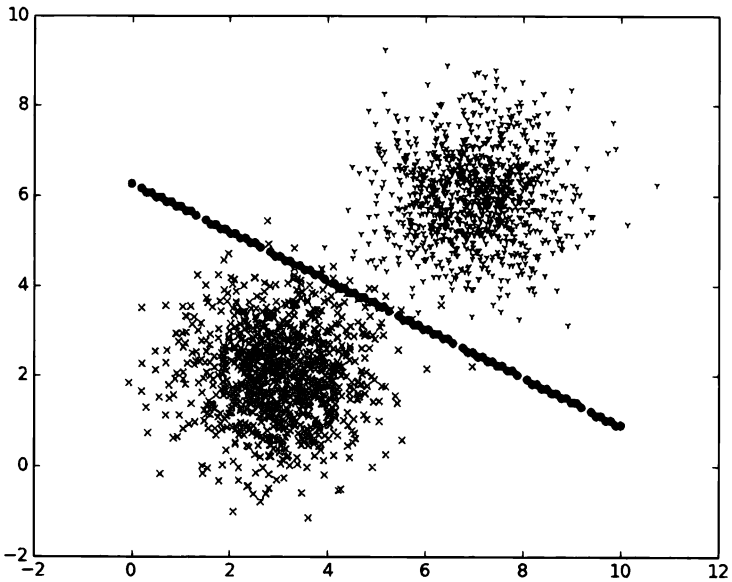


Рис. 4.14. Диагональная пунктирная линия представляет собой случай, когда вероятность между двумя решениями разделяется поровну. Уверенность в принятии решения возрастает, поскольку данные находятся дальше от линии

4.5. Многоклассовая классификация

До сих пор мы имели дело с многомерными входными данными, а не с многовариантными выходными данными, как это показано на рис. 4.15. А что, если теперь вместо бинарных классов данных у вас три, четыре или 100 классов? Логистическая регрессия требует не более двух меток (label).

Представьте классификацию: для примера, это популярная многовариантная задача, целью которой является заключение относительно класса изображения из множества возможных вариантов. Фотографию можно отнести к одной из сотен категорий.

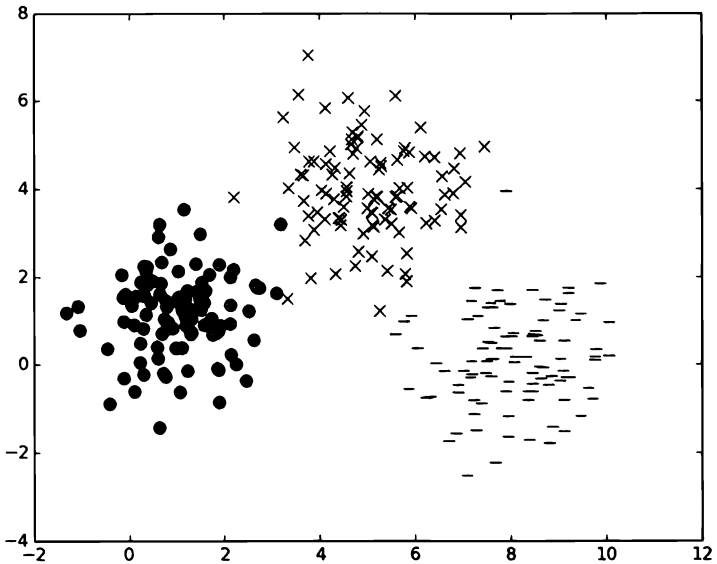


Рис. 4.15. Независимая переменная является двумерной, определяемой осью *x* и *y*. Зависимая переменная может принадлежать к одному из трех классов, различающихся формой точек данных

Для того чтобы обрабатывать более двух меток, можно повторно разумно использовать логистическую регрессию (используя метод «один против всех» или «каждый против каждого») или разработать новый метод (многоклассовой логистической регрессии). В следующем разделе мы рассмотрим каждый из этих вариантов решения. Методы логистической регрессии требуют значительных усилий по специальному проектированию, поэтому давайте сосредоточимся на многоклассовой логистической регрессии.

4.5.1. Один против всех

Прежде всего, следует обучить классификатор каждой метке, как показано на рис. 4.16. Если их три, необходимо иметь три доступных классификатора: `f1`, `f2` и `f3`. Для проверки на новых данных следует запускать каждый классификатор, чтобы установить тот, который дает наиболее уверенный отклик. Интуитивным образом вы относите новую точку к определенной метке классификатора, который дает наиболее уверенный отклик.

Рис. 4.16. «Один против всех» является методом многоклассовой классификации, от которого требуют определить каждую метку

4.5.2. Каждый против каждого

Затем необходимо обучить классификатор каждой паре меток, как показано на рис. 4.17. Если классов три, то требуются три разные пары. Но в случае k меток требуются $k(k-1)/2$ пар меток. На новых данных вы запускаете все классификаторы и выбираете класс с наибольшими выигрышами.

Рис. 4.17. В многоклассовой классификации «каждый против каждого» для каждой пары меток используется свой детектор

4.5.3. Многоклассовая логистическая регрессия

Многоклассовая логистическая регрессия (softmax regression) получила свое название от традиционной функции `max`, которая берет вектор и возвращает максимальное значение; но многоклассовая регрессия не является в точности функцией `max`, так как у нее есть дополнительное преимущество — непрерывность и дифференцируемость. В результате у нее есть полезные свойства для стохастического градиентного спуска, позволяющего работать эффективно.

В этом типе многоклассовой классификации каждый класс имеет оценку достоверности (или вероятности) для каждого входного вектора. В этой классификации пиковым значениям присваиваются максимальные оценки.

Откройте новый файл softmax.py и следуйте приведенному далее листингу. Прежде всего, вам необходимо визуализировать фейковые данные для воспроизведения рис. 4.15 (которые также воспроизведены на рис. 4.18).

Листинг 4.8. Визуализация многоклассовых данных

```
import numpy as np
import matplotlib.pyplot as plt

x1_label0 = np.random.normal(1, 1, (100, 1))
x2_label0 = np.random.normal(1, 1, (100, 1))
x1_label1 = np.random.normal(5, 1, (100, 1))
x2_label1 = np.random.normal(4, 1, (100, 1))
x1_label2 = np.random.normal(8, 1, (100, 1))
x2_label2 = np.random.normal(0, 1, (100, 1))

plt.scatter(x1_label0, x2_label0, c='r', marker='o', s=60)
plt.scatter(x1_label1, x2_label1, c='g', marker='x', s=60)
plt.scatter(x1_label2, x2_label2, c='b', marker='_', s=60)
plt.show()
```

Импортирует библиотеки NumPy и matplotlib

Генерирует точки вблизи (1, 1)

Генерирует точки вблизи (5, 4)

Генерирует точки вблизи (8, 0)

Визуализирует три класса на диаграмме разброса данных

Затем в листинге 4.9 необходимо задать обучающие и проверочные данные для подготовки к шагу многоклассовой регрессии. Метки должны быть представлены как вектор, в котором только один элемент равен 1, а все остальные — 0. Это представление носит название *кодирования с одним активным состоянием* (one-hot encoding). Например, при наличии трех меток они будут представлены как следующие векторы: [1, 0, 0], [0, 1, 0] и [0, 0, 1].

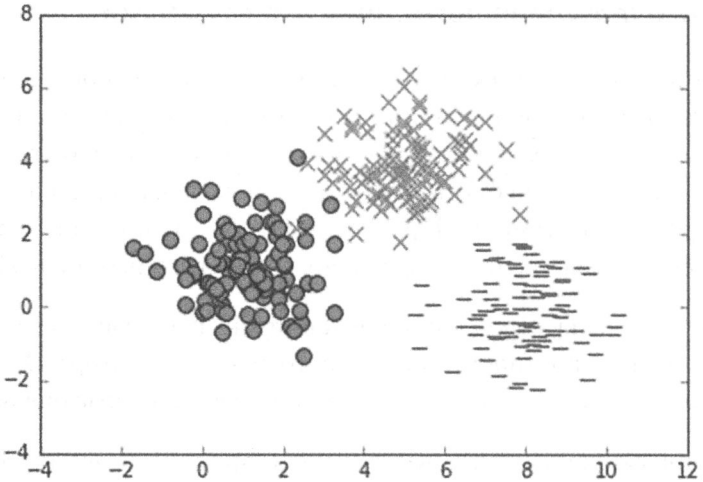


Рис. 4.18. Двумерные обучающие данные для классификации со многими выходами

УПРАЖНЕНИЕ 4.5

Кодирование с одним активным состоянием может показаться ненужным. Почему бы просто не иметь одномерные выходные данные со значениями 1, 2 и 3, представляющими эти три класса?

ОТВЕТ

Регрессия может порождать семантическую структуру в выходных данных. Если выходные данные имеют сходный вид, регрессия предполагает, что соответствующие им входные данные также имели сходный вид. При использовании одной размерности предполагается, что классы с метками 2 и 3 имеют большее сходство между собой, чем с метками 1 и 3. Необходимо проявлять осторожность, чтобы не сделать ненужных или неправильных предположений, поэтому безопаснее применять кодирование с одним активным состоянием.

Листинг 4.9. Настройка обучающих и проверочных данных для многоклассовой классификации

```
xs_label0 = np.hstack((x1_label0, x2_label0))
xs_label1 = np.hstack((x1_label1, x2_label1))
xs_label2 = np.hstack((x1_label2, x2_label2))
xs = np.vstack((xs_label0, xs_label1, xs_label2))

labels = np.matrix([[1., 0., 0.]] * len(x1_label0) + [[0., 1., 0.]] *
    len(x1_label1) + [[0., 0., 1.]] * len(x1_label2))

arr = np.arange(xs.shape[0])
np.random.shuffle(arr)
xs = xs[arr, :]
labels = labels[arr, :]

test_x1_label0 = np.random.normal(1, 1, (10, 1))
test_x2_label0 = np.random.normal(1, 1, (10, 1))
test_x1_label1 = np.random.normal(5, 1, (10, 1))
test_x2_label1 = np.random.normal(4, 1, (10, 1))
test_x1_label2 = np.random.normal(8, 1, (10, 1))
test_x2_label2 = np.random.normal(0, 1, (10, 1))
test_xs_label0 = np.hstack((test_x1_label0, test_x2_label0))
test_xs_label1 = np.hstack((test_x1_label1, test_x2_label1))
test_xs_label2 = np.hstack((test_x1_label2, test_x2_label2))

test_xs = np.vstack((test_xs_label0, test_xs_label1, test_xs_label2))
test_labels = np.matrix([[1., 0., 0.]] * 10 + [[0., 1., 0.]] * 10 +
    [[0., 0., 1.]] * 10)
train_size, num_features = xs.shape
```

Объединяет все входные данные в одну большую матрицу

Создает соответствующие метки с одним активным состоянием

Перемешивает набор данных

Подготавливает проверочные данные и метки

Форма набора данных говорит о числе примеров и признаков в расчете на один пример

И наконец, в листинге 4.10 будет использована многоклассовая регрессия. В отличие от сигмовидной функции, используемой в логистической регрессии, здесь будет использоваться функция `softmax` из библиотеки TensorFlow. Функция `softmax` аналогична функции `max`, которая выдает максимальное значение из списка чисел. Эта функция носит название softmax (мягкий максимум), потому что она является «мягкой», или «гладкой», аппроксимацией функции `max`, которая не является гладкой или непрерывной (и это плохо). Непрерывная и гладкая функция способствует получению в процессе обучения правильных весовых коэффициентов нейронной сети с помощью обратного распространения ошибки обучения.

УПРАЖНЕНИЕ 4.6

Какая из следующих функций непрерывная?

```
f(x) = x2
f(x) = min(x, 0)
f(x) = tan(x)
```

ОТВЕТ

Первые две являются непрерывными. Последняя из них, `tan(x)`, имеет периодические асимптоты, поэтому есть определенные значения, для которых нет приемлемых результатов.

Листинг 4.10. Применение многоклассовой регрессии

```
import tensorflow as tf

learning_rate = 0.01
training_epochs = 1000
num_labels = 3
batch_size = 100

X = tf.placeholder("float", shape=[None, num_features])
Y = tf.placeholder("float", shape=[None, num_labels])

W = tf.Variable(tf.zeros([num_features, num_labels]))
b = tf.Variable(tf.zeros([num_labels]))
y_model = tf.nn.softmax(tf.matmul(X, W) + b)

cost = -tf.reduce_sum(Y * tf.log(y_model))
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

correct_prediction = tf.equal(tf.argmax(y_model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Определяет гиперпараметры

Определяет узлы входных/выходных переменных-заполнителей

Определяет параметры модели

Создает алгоритм модели многоклассовой регрессии

Определяет оператор для измерения доли успешных прогнозов

Определяет алгоритм обучения

Теперь, после того как вы задали граф вычисления TensorFlow, выполните его, открыв сеанс. В этот раз мы используем новую форму итеративного обновления параметров, названную *пакетным обучением* (batch learning). Вместо передачи данных по одному элементу за раз алгоритм оптимизации будет выполнять свою работу с пакетами данных. Это ускоряет процесс оптимизации,

но вносит риск сходимости к локальному, а не глобальному оптимальному значению. Воспользуемся следующим листингом для выполнения оптимизации данных пакетами.

Листинг 4.11. Выполнение графа

Находит поднабор данных, соответствующий текущему пакету

Выполняется циклически необходимое число раз, чтобы пройти по всем данным набора

```
with tf.Session() as sess:
    tf.global_variables_initializer().run()

    for step in range(training_epochs * train_size // batch_size):
        offset = (step * batch_size) % train_size
        batch_xs = xs[offset:(offset + batch_size), :]
        batch_labels = labels[offset:(offset + batch_size)]
        err, _ = sess.run([cost, train_op], feed_dict={X: batch_xs, Y:
    batch_labels})
        print (step, err)

    W_val = sess.run(W)
    print('w', W_val)
    b_val = sess.run(b)
    print('b', b_val)
    print("accuracy", accuracy.eval(feed_dict={X: test_xs, Y: test_labels}))
```

Открывает новый сеанс и инициализирует переменные

Многократно выполняет оптимизацию этого пакета

Выводит полученные результаты

Выводит параметры, полученные в ходе обучения

Выводит долю успешных прогнозов

Окончательные выходные данные работы алгоритма регрессии многопеременной логистической функции на базе данных следующие:

```
('w', array([[-2.101, -0.021, 2.122],
            [-0.371, 2.229, -1.858]], dtype=float32))
('b', array([10.305, -2.612, -7.693], dtype=float32))
Accuracy 1.0
```

Вы только что узнали про весовые коэффициенты и коэффициенты смещения модели. Параметры, полученные в результате обучения, можно использовать повторно для того, чтобы сделать вывод от обучающих данных. Простой способ это сделать — сохранить и загрузить переменные с помощью объекта `Saver` из библиотеки TensorFlow (www.tensorflow.org/programmers_ guide/saved_model). Для получения отклика модели на проверочные данные ее можно запустить (вызвав `y_model` в коде).

4.6. Применение классификации

Эмоции являются сложным понятием для практического применения. Счастье, печаль, гнев, возбуждение и страх — это эмоции, представляющие собой субъективные понятия. То, что кажется привлекательным одному, может вызвать насмешку у другого. Текст, который может одного человека привести в гнев, у другого вызовет страх. Если у людей столько проблем с этим, то на что вообще может рассчитывать компьютер?

По крайней мере, специалисты по машинному обучению сообразили, как классифицировать положительные и отрицательные эмоции, которые выражены в определенном тексте. Например, вы разрабатываете сайт, аналогичный Amazon.com, в котором каждый раздел содержит отзывы пользователей. Вы хотели бы, чтобы используемая поисковая система предпочитала разделы с положительными отзывами. Вероятно, что наилучшей метрикой является средняя оценка, или число лайков. Но что, если большинство отзывов не содержат явно выраженных оценок?

Анализ настроений можно рассматривать как проблему двоичной классификации. Входными данными является текст на естественном языке, а на выходе — бинарные решения, содержащие заключение о позитивных или негативных эмоциях. Далее приводятся базы данных, которые можно найти в интернете для точного решения этой задачи.

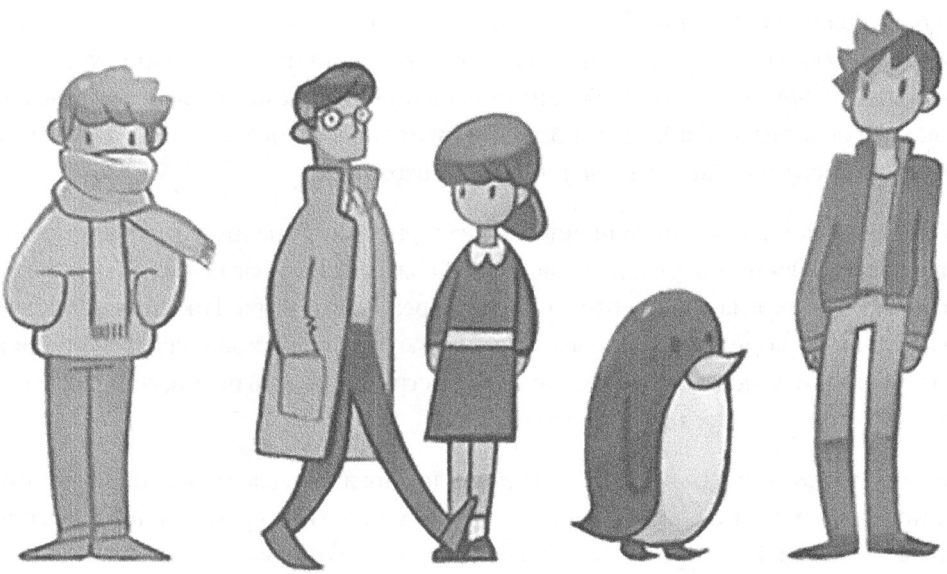
- База данных анализа полнометражных фильмов: http://mng.bz/60nj
- Размеченный набор данных на тему эмоций: http://mng.bz/CzSM
- База данных анализа эмоций в Twitter: http://mng.bz/2M4d

Самая большая проблема состоит в том, чтобы понять, как представлять исходный текст в качестве входных данных алгоритма классификации. В этой главе в качестве входных данных алгоритма классификации всегда использовался вектор признаков. Одним из старейших методов преобразования исходного текста в вектор признаков является так называемый *мешок слов* (bag-of-words). Вы можете найти превосходный обучающий курс и программу для реализации этого метода на сайте: http://mng.bz/K8yz.

4.7. Краткие итоги

- Есть множество методов решения задач классификации, но логистическая и многоклассовая регрессия относятся к наиболее надежным по полноте и эффективности методам.
- Важно предварительно обрабатывать данные, перед тем как приступать к процедуре классификации. Например, дискретные независимые переменные могут быть преобразованы в бинарные переменные.
- До сих пор вы подходили к классификации с точки зрения регрессии. В следующих главах мы вернемся к классификации с помощью нейронных сетей.
- Имеется много методов многоклассовой классификации. И нет определенного ответа на вопрос, какой метод следует выбрать в первую очередь: «один против всех», «каждый против каждого» или многоклассовую регрессию. Но многоклассовая регрессия несколько более автоматизирована и позволяет манипулировать большим числом гиперпараметров.

Автоматическая кластеризация данных



Эта глава охватывает следующие темы:

- ✓ Базовая кластеризация методом *k*-средних
- ✓ Представление аудио
- ✓ Сегментация аудио
- ✓ Кластеризация на основе самоорганизующихся карт

Предположим, что у вас на жестком диске есть подборка непиратских, легальных записей формата MP3. Все песни собраны в одной папке. Возможно, автоматическая группировка песен по категориям, таким как кантри, рэп и рок, поможет их организовать. Действие присоединения элемента к группе (так же, как присоединение файлов MP3 к списку аудиофайлов) неконтролируемым образом носит название *кластеризация* (clustering).

В предыдущей главе, посвященной классификации, предполагалось наличие набора размеченных обучающих данных. К сожалению, у вас не всегда есть такая возможность, когда собираются данные из реального мира. Предположим, что вы хотели бы разделить большое количество музыкальных записей на подборки интересных вам исполнителей. Как можно сгруппировать песни, если у вас нет прямого доступа к их метаданным?

Spotify, *SoundCloud*, *Google Music*, *Pandora* и многие другие музыкальные стриминговые сервисы пытаются решить эту проблему, чтобы рекомендовать песни пользователям. Их метод включает соединение различных методик машинного обучения, но в основе этих методик всегда лежит кластеризация.

Кластеризация является процессом разумной категоризации элементов в вашем наборе данных. Идея состоит в том, что два элемента в одном кластере «ближе» друг к другу, чем элементы, принадлежащие разным кластерам. Это общее определение, в котором остается открытым вопрос толкования *близости*. Например, возможно ли, что гепарды и леопарды принадлежат к одному и тому же кластеру, в то время как слоны принадлежат к другому, если близость

оценивается сходством двух видов в иерархии биологической классификации (семейство, род и вид).

Вы можете представить, как много алгоритмов кластеризации могут использоваться здесь. В этой главе акцент делается на двух: *метод k-средних* и *самоорганизующиеся карты*. Эти методы полностью *неконтролируемые* (без учителя), так как они обучают модель без контрольных примеров.

Сначала вы узнаете, как загружать аудиофайлы в TensorFlow и представлять их как векторы признаков. Затем вы будете внедрять различные методы кластеризации для решения проблем реального мира.

5.1. Обход файлов в TensorFlow

К распространенным типам входных данных, используемых в машинном обучении, относятся файлы со звуковыми (аудио) и графическими данными. Это не должно быть сюрпризом, потому что звуковые записи и фотографии являются необработанными, избыточными и часто содержащими шумы представлениями семантических понятий. Машинное обучение — это инструмент, помогающий справиться с этими осложнениями.

Эти файлы с данными имеют различные форматы: например, изображение может быть закодировано как PNG или JPEG, а звуковые файлы могут иметь формат MP3 или WAV. В этой главе вы узнаете, как считывать аудиофайлы для их использования в качестве входных данных алгоритма кластеризации, чтобы все музыкальные записи, которые имеют аналогичное звучание, можно было группировать автоматически.

УПРАЖНЕНИЕ 5.1

Какие преимущества и недостатки у форматов MP3 и WAV? А у форматов PNG и JPEG?

ОТВЕТ

Форматы MP3 и JPEG предполагают значительную степень сжатия данных, поэтому их легче хранить или передавать. Но из-за того, что сжатие сопровождается потерями, форматы WAV и PNG ближе к исходным данным.

Чтение файлов с диска не является способностью, относящейся к машинному обучению. Можно использовать различные библиотеки Python (такие, как NumPy или SciPy) для загрузки файлов в память. Некоторым разработчикам нравится выполнять этап обработки данных отдельно от этапа машинного обучения. Не существует абсолютно правильного или абсолютно неправильного способа управления этим конвейером, но мы попробуем использовать TensorFlow как для обработки данных, так и для обучения.

В библиотеке TensorFlow есть оператор `tf.train.match_filenames_once(…)`, который можно использовать для перечисления файлов в директории. Можно затем передать эту информацию прямо оператору очереди `tf.train.string_input_producer(…)`. Таким образом можно получить доступ к именам файлов по отдельности, не загружая все сразу. Учитывая имя файла, вы можете декодировать файл для извлечения полезных данных. На рис. 5.1 показан весь процесс использования очереди.

В следующем листинге показано считывание файлов с диска в TensorFlow.

Листинг 5.1. Обход директории для просмотра данных

```
import tensorflow as tf

filenames = tf.train.match_filenames_once('./audio_dataset/*.wav')
count_num_files = tf.size(filenames)
filename_queue = tf.train.string_input_producer(filenames)
reader = tf.WholeFileReader()
file name, file_contents = reader.read(filename_queue)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_files = sess.run(count_num_files)

    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)

    for i in range(num_files):
        audio_file = sess.run(filename)
        print(audio_file)
```

Хранит имена файлов, которые соответствуют шаблону восстановления

Задает конвейер для имен файлов случайным образом

По умолчанию считывает файл из библиотеки TensorFlow

Выполняет операцию считывания для извлечения данных файла

Подсчитывает число файлов

Инициализирует потоки для очереди имен файлов

Проходит циклически и поочередно по всем данным

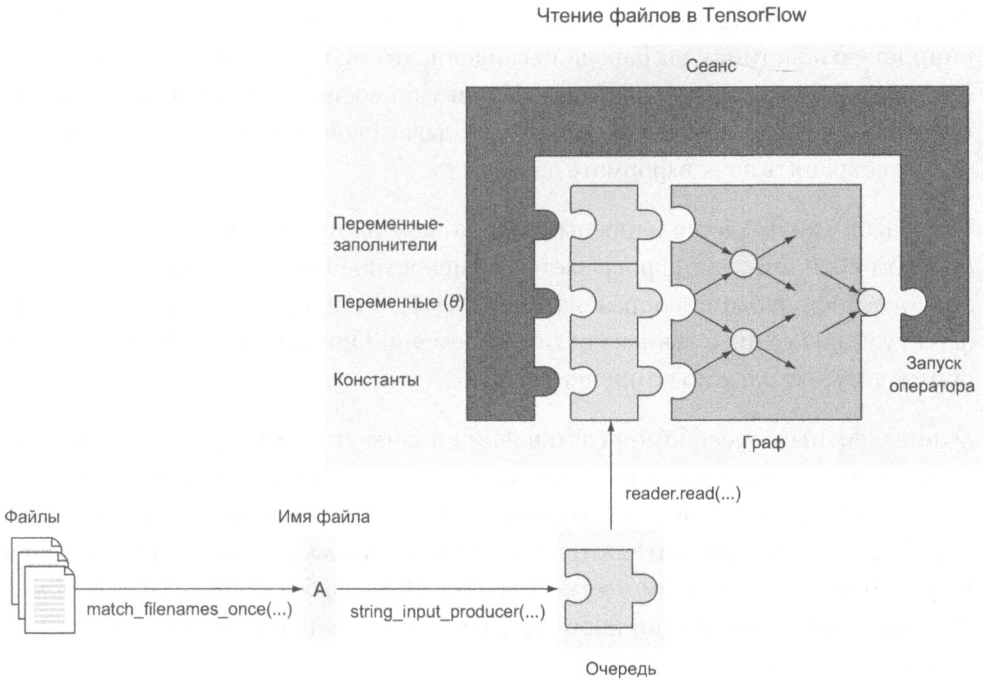


Рис. 5.1. Для считывания файлов можно воспользоваться очередью в TensorFlow. Эта очередь встроена в инфраструктуру TensorFlow, и для получения к ней доступа можно воспользоваться функцией reader.read(…)

СОВЕТ Если листинг 5.1 у вас не сработал, вы можете воспользоваться советом, размещенным на официальном форуме этой книги: http://mng.bz/Q9aD.

5.2. Извлечение признаков из звукозаписи

Алгоритмы машинного обучения в качестве входных данных обычно используют векторы признаков; однако аудиофайлы используют другой формат. Необходим способ, который позволил бы извлекать признаки из таких файлов, чтобы создавать векторы признаков.

Это помогает понять, как устроены аудиофайлы. Если вам приходилось видеть виниловую пластинку, вы наверняка видели, что звук записывается с помощью канавок, вырезанных на пластинке. Наши уши воспринимают звук как серию колебаний, передаваемых в воздухе. Записывая свойства колебаний, алгоритм может сохранять звук в формате данных.

Реальный мир является непрерывным, но компьютеры хранят данные в дискретном виде. Звук оцифровывается в дискретное представление с помощью аналого-цифрового преобразователя (АЦП). Можно представить звук как флуктуацию волны в зависимости от времени. Однако эти данные содержат шумы, которые сложно компенсировать.

Эквивалентным способом представления волны является проверка ее частот за каждый интервал времени. Это представление носит название *частотной области*. Не представляет труда преобразовать временную область в частотную и наоборот с помощью математической операции *дискретного преобразования Фурье* (обычно применяемого с помощью алгоритма *быстрого преобразования Фурье*). Этот метод можно использовать для извлечения вектора признаков из звука.

Библиотека Python позволит просматривать звуковые данные в этой частотной области. Ее можно загрузить с сайта https://github.com/BinRoot/BregmanToolkit/archive/master.zip. Разархивируйте ее и запустите команду для установки:

```
$ python setup.py install
```

ТРЕБУЕТСЯ PYTHON 2

BregmanToolkit официально поддерживается на Python 2. Если вы используете Jupyter Notebook, вы можете иметь доступ к обеим версиям Python, следуя указаниям, изложенным в официальных документах Jupyter: http://mng.bz/ebvw.

В частности, Python 2 можно включить с помощью двух следующих команд:

```
$ python2 -m pip install ipykernel
$ python2 -m -ipykernel install --user
```

Звук можно разделить на 12 тонов. В музыке эти тона обозначают через C, C#, D, D#, E, F, F#, G, G#, A, A# и B. В листинге 5.2 показано, как можно

определить вклад каждого тона в 0,1-секундном интервале, используя матрицу из 12 строк. Число столбцов растет по мере увеличения длины звуковой записи. В частности, для звуковой записи продолжительностью t секунд в матрице будет $10 \times t$ столбцов. Эту матрицу также называют *хромограммой* звуковой записи.

Листинг 5.2. Представление звуковой записи в Python

```
from bregman.suite import *

def get_chromagram(audio_file):
    F = Chromagram(audio_file, nfft=16384, wfft=8192, nhop=2205)
    return F.X
```

Передает имя файла

Представляет значения 12-мерного вектора 10 раз в секунду

Использует эти параметры для описания 12 тонов каждую 0,1 с

Хромограмме соответствует матрица, представленная на рис. 5.2. Звуковую запись можно прочитать как хромограмму, а сама хромограмма является рецептом получения звуковой записи. Теперь у нас есть способ преобразования звуковой записи в матрицу и обратно. Как вы уже знаете, большинство алгоритмов машинного обучения применяют векторы признаков как удобную форму записи данных. А первый алгоритм машинного обучения, с которым вы познакомитесь, — это кластеризация методом k-средних.

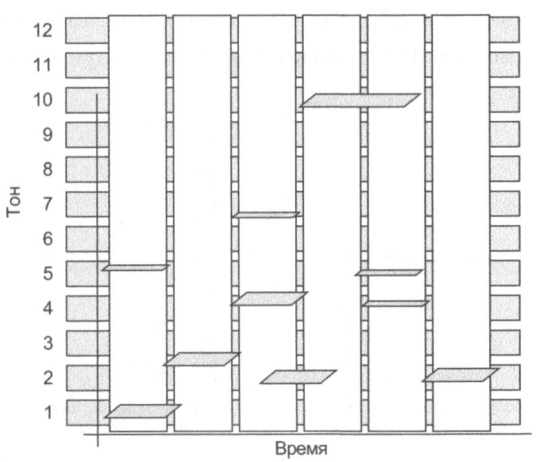


Рис. 5.2. Матрица хромограммы, в которой ось x представляет время, а ось y — тона. Параллелограммы указывают на наличие определенного тона в определенное время

Для выполнения алгоритма обучения по приведенной хромограмме необходимо принять решение, как будет представлен вектор признаков. Можно упростить аудиозапись, учитывая только наиболее значимые тона за интервал времени, как показано на рис. 5.3.

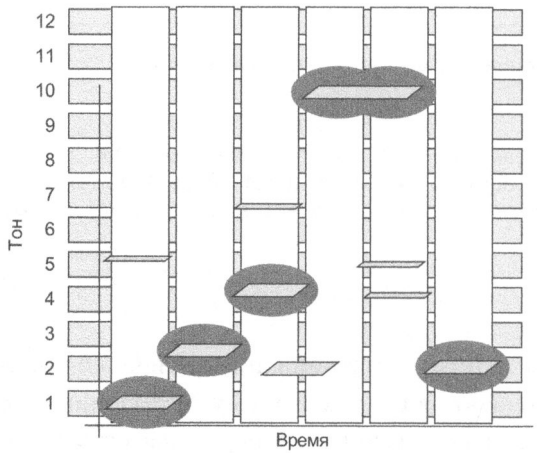


Рис. 5.3. Выделен наиболее значимый тон для каждого интервала времени. Его можно считать самым громким тоном

Затем можно подсчитать, сколько раз каждый тон проявляется в файле звуковых данных. На рис. 5.4 приведена гистограмма, образующая 12-мерный вектор. Если этот вектор нормировать так, чтобы все результаты подсчета по тонам в сумме давали 1, можно будет легко сравнивать звукозаписи разной длины.

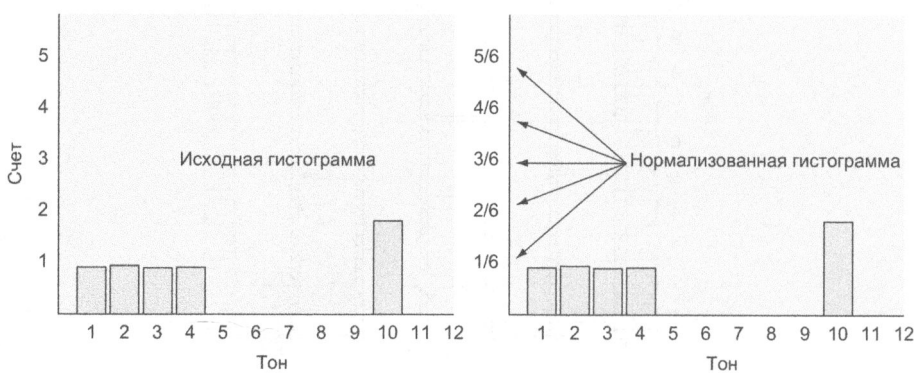


Рис. 5.4. Для получения этой гистограммы, которая играет роль вектора признаков, была определена частота самых громких тонов, прозвучавших в каждом интервале

УПРАЖНЕНИЕ 5.2

Какие еще есть способы представить аудиозапись в виде вектора признаков?

ОТВЕТ

Вы можете представить аудиозапись как изображение (например, как спектрограмму) и использовать методы анализа изображений для извлечения характеристик.

Посмотрите на следующий листинг, который позволяет получить гистограмму с рис. 5.4, которая и является вектором признаков.

Листинг 5.3. Получение базы данных для метода k-средних

```
import tensorflow as tf
import numpy as np
from bregman.suite import *


filenames = tf.train.match_filenames_once('./audio_dataset/*.wav')
count_num_files = tf.size(filenames)
filename_queue = tf.train.string_input_producer(filenames)
reader = tf.WholeFileReader()
filename, file_contents = reader.read(filename_queue)

chroma = tf.placeholder(tf.float32)
max_freqs = tf.argmax(chroma, 0)

def get_next_chromagram(sess):
    audio_file = sess.run(filename)
    F = Chromagram(audio_file, nfft=16384, wfft=8192, nhop=2205)
    return F.X

def extract_feature_vector(sess, chroma_data):

    num_features, num_samples = np.shape(chroma_data)
    freq_vals = sess.run(max_freqs, feed_dict={chroma: chroma_data})
    hist, bins = np.histogram(freq_vals, bins=range(num_features + 1))
    return hist.astype(float) / num_samples
def get_dataset(sess):
    num_files = sess.run(count_num_files)
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)
    xs = []
    for _ in range(num_files):
```

Определяет оператор для выявления тона с максимальным вкладом

Преобразует хромограмму в вектор признаков

Составляет матрицу, в которой каждая строка является элементом данных

```
        chroma_data = get_next_chromagram(sess)
        x = [extract_feature_vector(sess, chroma_data)]
        x = np.matrix(x)
        if len(xs) == 0:
            xs = x
        else:
            xs = np.vstack((xs, x))
    return xs
```

ПРИМЕЧАНИЕ Все листинги можно найти на сайте книги www.manning.com/books/machine-learning-with-tensorflow и на GitHub в https://github.com/BinRoot/TensorFlow-Book/tree/master/ch05_clustering.

5.3. Кластеризация методом k-средних

Алгоритм k-средних является одним из старейших, но все еще наиболее надежным способом кластеризации данных. Символ k в методе k-средних обозначает переменную, являющуюся натуральным числом. Эта переменная обозначает число используемых при кластеризации средних. Поэтому на первом этапе кластеризации методом k-средних необходимо выбрать значение k. Пусть, например, $k = 3$. С учетом этого цель кластеризации тремя средними состоит в делении набора данных на три категории (или *кластера*).

ВЫБОР ЧИСЛА КЛАСТЕРОВ

Выбор подходящего числа кластеров часто зависит от задачи. Например, представьте, что вы планируете мероприятие для сотен как молодых, так и пожилых людей. Если ваш бюджет рассчитан на мероприятие только с двумя опциями, то можно использовать кластеризацию методом k-средних с $k = 2$, что предполагает деление гостей на две возрастные группы. В других случаях выбор значения k не является таким очевидным. Автоматическое оценивание значения k несколько более сложная задача, поэтому мы не будем рассматривать ее в этом разделе. Проще говоря, наиболее прямым способом определить наилучшее значение k является последовательный перебор всего диапазона моделей k-средних с использованием функции стоимости, чтобы определить, какое значение k приводит к наилучшему различению кластеров между собой при самом низком значении k.

Алгоритм k-средних рассматривает точки данных как точки в пространстве. Если вашей базой данных является совокупность гостей на мероприятии, можно каждого из них представить через его возраст. Таким образом, эта база данных является совокупностью векторов признаков. В этом случае каждый вектор признаков имеет размерность 1, потому что учитывается только возраст человека.

При кластеризации музыки по аудиоданным точками данных являются векторы признаков из файлов звуковых данных. Если две точки близки друг другу, их звуковые характеристики считаются аналогичными. Мы хотим установить, какие аудиофайлы относятся к одному типу, так что кластеры — хороший вариант организации музыкальных файлов.

Середину всех точек в кластере называют их *центроидом* (centroid). В зависимости от характеристик звуковых данных, которые необходимо извлечь, центроид может соответствовать таким понятиям, как громкость звука, звук высокой тональности или саксофоноподобный звук. Важно отметить, что алгоритм k-средних присваивает классы неопределенного вида, такие как кластер 1, кластер 2 и кластер 3. На рис. 5.5 приведены примеры звуковых данных.

Алгоритм k-средних присваивает вектор признаков одному из k кластеров, выбирая при этом тот, чей центроид находится ближе всего к нему. Алгоритм k-средних начинает работу с предположения о местоположении кластера. Со временем начальное приближение итеративно улучшается. Алгоритм либо сходится, если он больше не улучшает начальное приближение, либо останавливается после выполнения заданного максимального числа итераций.

В основе алгоритма лежат два этапа: присваивания и повторного центрирования.

1. На этапе присваивания каждому элементу данных (вектору признаков) присваивается определенная категория ближайшего центроида.
2. На этапе повторного центрирования вычисляется средняя точка обновленных кластеров.

Два этих этапа повторяют для улучшения результатов кластеризации, а алгоритм завершает работу после требуемого числа повторений или если последующие присвоения больше ничего не меняют. Рисунок 5.6 иллюстрирует работу этого алгоритма.

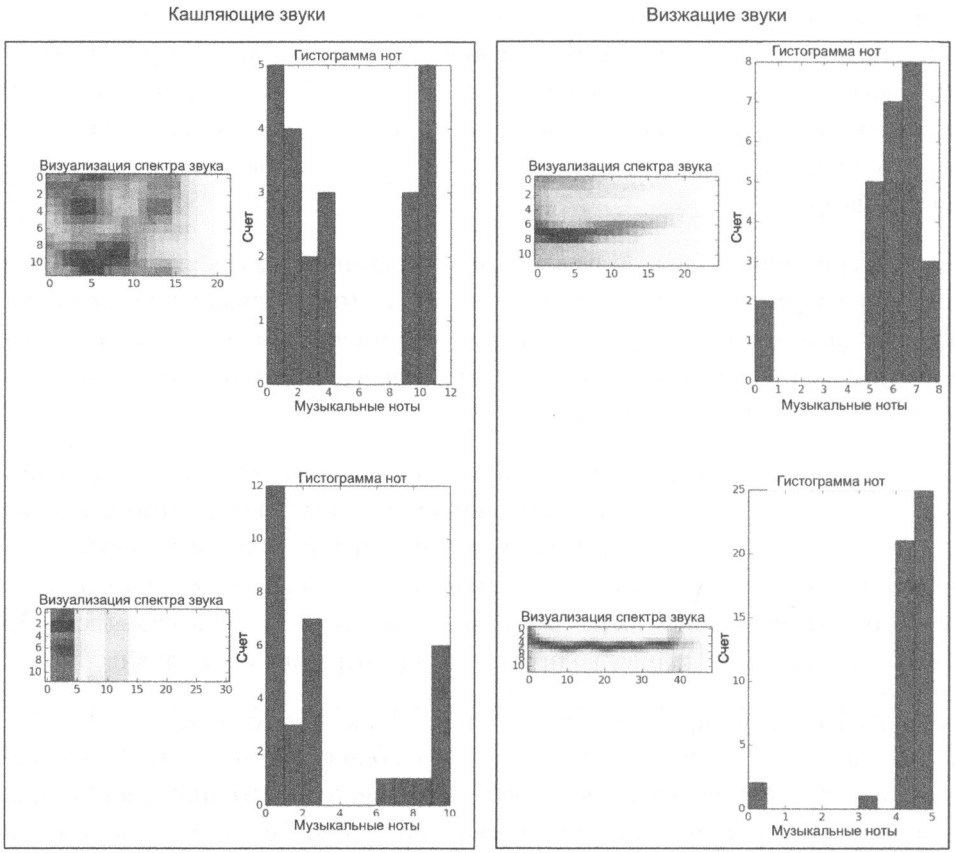


Рис. 5.5. Четыре примера аудиофайлов. Как можно заметить, два примера справа имеют аналогичные гистограммы. Две гистограммы слева также являются сходными. Эти алгоритмы кластеризации способны группировать эти звуки вместе

Листинг 5.4 показывает, как можно использовать алгоритм k-средних, применяя базу данных, полученную в результате выполнения кода, приведенного в листинге 5.3. Для упрощения задачи выберем $k = 2$, так что тот факт, что используемый алгоритм делит аудиофайлы на несхожие между собой категории, можно легко проверить. Используем первые k векторов в качестве начального приближения положения центроидов.

Рис. 5.6. Одна итерация алгоритма *k*-средних. Представим, что данные необходимо разбить на три сегмента (или *категории*). Процесс можно запустить с начальным приближением деления на красный, зеленый и синий сегменты и приступить к шагу присваивания. Затем выполняется обновление цветов в сегментах с усреднением цветов, принадлежащих к каждому сегменту. Итерации продолжаются, пока сегменты заметным образом не меняют цвет, приходя к окончательному представлению цвета каждого кластера

Листинг 5.4. Применение k-средних

```
k = 2
max_iterations = 100

def initial_cluster_centroids(X, k):
    return X[0:k, :]

def assign_cluster(X, centroids):
    expanded_vectors = tf.expand_dims(X, 0)
    expanded_centroids = tf.expand_dims(centroids, 1)
    distances = tf.reduce_sum(tf.square(tf.subtract(expanded_vectors,
expanded_centroids)), 2)
    mins = tf.argmin(distances, 0)
    return mins

def recompute_centroids(X, Y):
```

Принимает решение по числу кластеров

Определяет максимальное число итераций для алгоритма k-средних

Выбирает начальное приближение центроидов кластеров

Присваивает каждому элементу данных ближайший к нему центроид

Обновляет центроиды кластеров, используя их средние точки

```
    sums = tf.unsorted_segment_sum(X, Y, k)
    counts = tf.unsorted_segment_sum(tf.ones_like(X), Y, k)
    return sums / counts

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    X = get_dataset(sess)
    centroids = initial_cluster_centroids(X, k)
    i, converged = 0, False
    while not converged and i < max_iterations:
        i += 1
        Y = assign_cluster(X, centroids)
        centroids = sess.run(recompute_centroids(X, Y))
    print(centroids)
```

Выполняет итерации для поиска положения кластеров

Вот и всё! Если вы знаете количество кластеров и представление вектора признаков, вы можете использовать листинг 5.4 для кластеризации чего угодно. В следующем разделе вы примените кластеризацию к аудиофрагментам в аудиофайле.

5.4. Сегментация звуковых данных

В предыдущем разделе вы кластеризировали различные аудиофайлы для автоматической группировки. Этот раздел посвящен использованию алгоритмов кластеризации только для одного звукового файла. В то время как первый способ называется *кластеризацией*, последний носит название *сегментации*. Сегментация — это еще одно название кластеризации, но мы часто говорим о *сегменте* вместо *кластера* при разделении одного изображения или аудиофайла на отдельные компоненты. Это похоже на то, как разделение предложения на слова отличается от деления слова на буквы. Хотя оба они основаны на делении целого на составляющие, слова отличаются от букв.

Допустим, у вас есть аудиофайл большой длительности, например подкаст или ток-шоу. Для идентификации того, кто из двух человек говорит в тот или иной момент времени, необходимо написать алгоритм машинного обучения. Целью сегментации аудиофайла является выявление тех частей файла, которые принадлежат одной и той же категории. В этом случае необходимо предусмотреть категорию для каждого человека, а выражения, произнесенные каждым из них, должны сводиться к соответствующим категориям, как это показано на рис. 5.7.

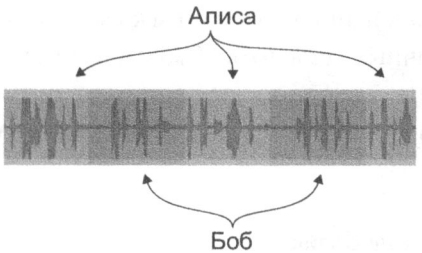


Рис. 5.7. Сегментация звуковой записи является автоматическим процессом маркировки сегментов

Откройте новый файл и следуйте листингу 5.5, чтобы запустить процесс организации звуковых данных для сегментации. В результате аудиофайл будет разбит на множество сегментов размера `segment_size`. Большой файл может содержать сотни, если не тысячи, сегментов.

Листинг 5.5. Организация данных для сегментации

```
import tensorflow as tf
import numpy as np
from bregman.suite import *

k = 2                                  ◄── Принимает решение относительно числа кластеров
segment_size = 50                      ◄── Чем меньше размер сегмента, тем лучше результаты (но медленнее выполнение)
max_iterations = 100                   ◄── Принимает решение, когда прекратить итерации

chroma = tf.placeholder(tf.float32)
max_freqs = tf.argmax(chroma, 0)

def get_chromagram(audio_file):
    F = Chromagram(audio_file, nfft=16384, wfft=8192, nhop=2205)
    return F.X

def get_dataset(sess, audio_file):     ◄── Получает базу данных путем извлечения сегментов звуковых данных в виде отдельных элементов данных
    chroma_data = get_chromagram(audio_file)
    print('chroma_data', np.shape(chroma_data))
    chroma_length = np.shape(chroma_data)[1]
    xs = []
    for i in range(chroma_length / segment_size):
        chroma_segment = chroma_data[:, i*segment_size:(i+1)*segment_size]
        x = extract_feature_vector(sess, chroma_segment)
        if len(xs) == 0:
            xs = x
        else:
            xs = np.vstack((xs, x))
    return xs
```

Теперь, используя эту базу данных, выполним кластеризацию методом k-средних для выявления идентичных сегментов. Смысл в том, что метод k-средних будет присваивать сегментам сходного звучания одну и ту же метку. Если у двух людей голоса звучат по-разному, их звуковые фрагменты будут также иметь разные метки.

Листинг 5.6. Сегментация аудиофайлов

```
with tf.Session() as sess:
    X = get_dataset(sess, 'TalkingMachinesPodcast.wav')
    print(np.shape(X))
    centroids = initial_cluster_centroids(X, k)
    i, converged = 0, False
    while not converged and i < max_iterations:  ←— Выполняет алгоритм k-средних
        i += 1
        Y = assign_cluster(X, centroids)
        centroids = sess.run(recompute_centroids(X, Y))
        if i % 50 == 0:
            print('iteration', i)
    segments = sess.run(Y)
    for i in range(len(segments)):  ←— Выводит метки для каждого временного интервала
        seconds = (i * segment_size) / float(10)
        min, sec = divmod(seconds, 60)
        time_str = '{}m {}s'.format(min, sec)
        print(time_str, segments[i])
```

Результатом запуска листинга 5.6 является список меток времени и идентификаторов кластера, которые соответствуют тому, кто говорит во время подкаста:

```
('0.0m 0.0s', 0)
('0.0m 2.5s', 1)
('0.0m 5.0s', 0)
('0.0m 7.5s', 1)
('0.0m 10.0s', 1)
('0.0m 12.5s', 1)
('0.0m 15.0s', 1)
('0.0m 17.5s', 0)
('0.0m 20.0s', 1)
('0.0m 22.5s', 1)
('0.0m 25.0s', 0)
('0.0m 27.5s', 0)
```

УПРАЖНЕНИЕ 5.3

Как можно установить, сошелся алгоритм кластеризации или нет (чтобы можно было прервать выполнение алгоритма раньше)?

ОТВЕТ

Один из способов — следить за тем, как изменяются центроиды кластера, и объявлять о сходимости, когда больше не нужны обновления (например, если ошибка значительно не уменьшается от одной итерации к другой). Для этого необходимо вычислить величину ошибки и решить, что считать «значительным».

5.5. Кластеризация с самоорганизующимися картами

Самоорганизующаяся карта (SOM — self-organizing map) является моделью для представления данных в пространстве низкой размерности. При этой операции происходит автоматическое сближение сходных элементов данных. Предположим, что вы заказываете пиццу для большого числа гостей. Вы не хотите заказывать один и тот же тип пиццы всем гостям, потому что кому-то нравится причудливое сочетание ананаса с грибами и перцем, а вы предпочитаете анчоусы с рукколой и репчатым луком.

Предпочтительный топпинг для каждого гостя может быть представлен в виде трехмерного вектора. Метод SOM позволяет вместить эти трехмерные векторы в пространство двух измерений (при условии, что определены метрики расстояния между пиццами). Затем с помощью визуализации двумерной диаграммы выявляются подходящие варианты числа кластеров.

Хотя при использовании метода SOM может потребоваться больше времени, чтобы модель сошлась, чем для алгоритма k-средних, в методе SOM не делается предположений о числе кластеров. В реальном мире сложно выбрать число кластеров. Представим группу людей, показанных на рис. 5.8, в которой со временем кластеры могут меняться.

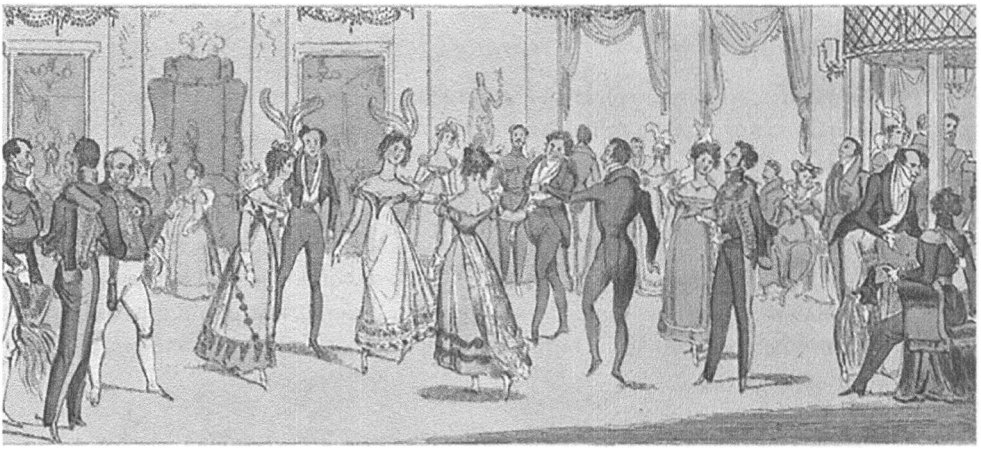


Рис. 5.8. В реальном мире мы видим группы людей, состоящие из кластеров. Применение метода k-средних требует знания числа кластеров. Более универсальной является самоорганизующаяся карта, которая не использует предварительные сведения о числе кластеров

Метод SOM просто по-новому истолковывает данные в виде структуры, приводящей к кластеризации. Этот алгоритм работает следующим образом. Сначала строится сетка узлов; каждый узел содержит вектор веса той же размерности, что и элемент данных. Веса каждого узла инициализируются случайными числами, обычно имеющими нормальное распределение.

Затем все элементы данных поочередно передаются в сеть. Для каждого элемента сеть подбирает узел, вектор веса которого ближе всего подходит к нему. Этот узел называют *лучшей единицей соответствия* (BMU – best matching unit).

После того как сеть идентифицировала BMU, все соседние узлы обновляются таким образом, что их векторы веса перемещаются ближе к значению BMU. На расположенные ближе узлы влияние оказывается сильнее, чем на удаленные узлы. Кроме того, число расположенных вокруг BMU соседних узлов сокращается со временем со скоростью, которая обычно определяется эмпирически. На рис. 5.9 иллюстрируется работа этого алгоритма.

В листинге 5.7. показано, как можно использовать метод SOM в TensorFlow. Откройте новый исходный файл и повторите код из листинга.

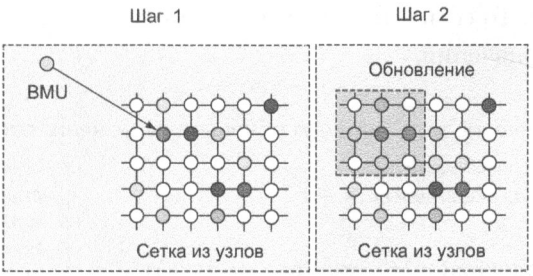


Рис. 5.9. Одна итерация алгоритма SOM. На первом шаге идентифицируется BMU, а на втором обновляются соседние узлы. Эти два шага итеративно повторяются с обучающими данными, пока не будет выполнен заданный критерий сходимости

Листинг 5.7. Настройка алгоритма SOM

```
import tensorflow as tf
import numpy as np

class SOM:
    def  init (self, width, height, dim):
        self.num_iters = 100
        self.width = width
        self.height = height
        self.dim = dim
        self.node_locs = self.get_locs()

        nodes = tf.Variable(tf.random_normal([width*height, dim]))
        self.nodes = nodes

        x = tf.placeholder(tf.float32, [dim])
        iter = tf.placeholder(tf.float32)

        self.x = x
        self.iter = iter

        bmu_loc = self.get_bmu_loc(x)

        self.propagate_nodes = self.get_propagation(bmu_loc, x, iter)
```

Каждый узел является вектором размерности dim. Для двумерной сетки число узлов определяется как произведение ширины на высоту; get_locs задается в листинге 5.10

Эти два оператора являются входными для каждой итерации

Вам потребуется получить к ним доступ из другого метода

Обновляет значения узлов (в листинге 5.8)

Находит узел, который ближе всего к входному узлу (в листинге 5.9)

В листинге 5.8. указано то, как обновлять соседние узлы для текущего интервала времени и положения BMU. С течением времени соседние с BMU веса будут

меняться меньше. То есть с течением времени веса постепенно приобретают установившиеся значения.

Листинг 5.8. Определение того, как будут обновляться значения соседних узлов

Распаковывает bmu_loc, чтобы можно было его попарно сравнивать с каждым элементом node_locs

Скорость снижается по мере увеличения числа выполненных итераций. Это значение влияет на параметры альфа и сигма

```
def get_propagation(self, bmu_loc, x, iter):
    num_nodes = self.width * self.height
    rate = 1.0 - tf.div(iter, self.num_iters)
    alpha = rate * 0.5
    sigma = rate * tf.to_float(tf.maximum(self.width, self.height)) / 2.
    expanded_bmu_loc = tf.expand_dims(tf.to_float(bmu_loc), 0)
    sqr_dists_from_bmu = tf.reduce_sum(
        tf.square(tf.subtract(expanded_bmu_loc, self.node_locs)), 1)
    neigh_factor =
        tf.exp(-tf.div(sqr_dists_from_bmu, 2 * tf.square(sigma)))
    rate = tf.multiply(alpha, neigh_factor)
    rate_factor =
        tf.stack([tf.tile(tf.slice(rate, [i], [1]),
                [self.dim]) for i in range(num_nodes)])
    nodes_diff = tf.multiply(
        rate_factor =
        tf.subtract(tf.stack([x for i in range(num_nodes)]), self.nodes))
    update_nodes = tf.add(self.nodes, nodes_diff)
    return tf.assign(self.nodes, update_nodes)
```

Проверяет, чтобы все близкие к BMU узлы изменялись меньше

Задает обновления

Возвращает оператор для выполнения обновлений

В следующем листинге показано, как найти положение BMU, имея входные данные. Поиск выполняется по сетке узлов, чтобы найти наиболее близкий узел. Это аналогично шагу присваивания в кластеризации методом k-средних, в котором каждый узел сетки является потенциальным центроидом кластера.

Листинг 5.9. Получение данных о местоположении ближайшего узла

```
def get_bmu_loc(self, x):
    expanded_x = tf.expand_dims(x, 0)
    sqr_diff = tf.square(tf.subtract(expanded_x, self.nodes))
    dists = tf.reduce_sum(sqr_diff, 1)
    bmu_idx = tf.argmin(dists, 0)
    bmu_loc = tf.stack([tf.mod(bmu_idx, self.width), tf.div(bmu_idx,
        self.width)])
    return bmu_loc
```

В следующем листинге создается вспомогательный метод для генерации списка координат (x, y) всех узлов сетки.

Листинг 5.10. Генерация матрицы точек

```
def get_locs(self):
    locs = [[x, y]
            for y in range(self.height)
            for x in range(self.width)]
    return tf.to_float(locs)
```

И наконец, зададим метод, названный `train`, чтобы выполнить алгоритм, приведенный в листинге 5.11. Сначала необходимо задать сеанс и выполнить оператор `global_variables_initializer`. Затем определенное число раз выполняется оператор цикла `num_iters`, чтобы обновить веса, используя поочередно входные данные. После завершения цикла производится запись об окончательных весах узлов и их положениях.

Листинг 5.11. Выполнение алгоритма SOM

```
def train(self, data):
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(self.num_iters):
            for data_x in data:
            sess.run(self.propagate_nodes, feed_dict={self.x: data_x,
            ➥self.iter: i})
    centroid_grid = [[] for i in range(self.width)]
    self.nodes_val = list(sess.run(self.nodes))
    self.locs_val = list(sess.run(self.node_locs))
    for i, l in enumerate(self.locs_val):
        centroid_grid[int(l[0])].append(self.nodes_val[i])
    self.centroid_grid = centroid_grid
```

Вот и всё! Теперь посмотрим, как это работает. Проверьте алгоритм, используя для этого SOM некоторых входных данных. В листинге 5.12 входными данными является список трехмерных векторов признаков. Используя для SOM обучающие данные, получим в этих данных кластеры. Будет использована сетка 4×4, но лучше всего попробовать различные значения для кросс-валидации оптимального размера сетки. На рис. 5.10 показаны выходные данные выполняемого кода.

Листинг 5.12. Тестирование реализации и визуализация результатов

```
from matplotlib import pyplot as plt
import numpy as np
from som import SOM

colors = np.array(
     [[0., 0., 1.],
      [0., 0., 0.95],
      [0., 0.05, 1.],
      [0., 1., 0.],
      [0., 0.95, 0.],
      [0., 1, 0.05],
      [1., 0., 0.],
      [1., 0.05, 0.],
      [1., 0., 0.05],
      [1., 1., 0.]])

som = SOM(4, 4, 3)
som.train(colors)

plt.imshow(som.centroid_grid)
plt.show()
```

Размер сетки — 4 × 4, а размерность входных данных равна трем

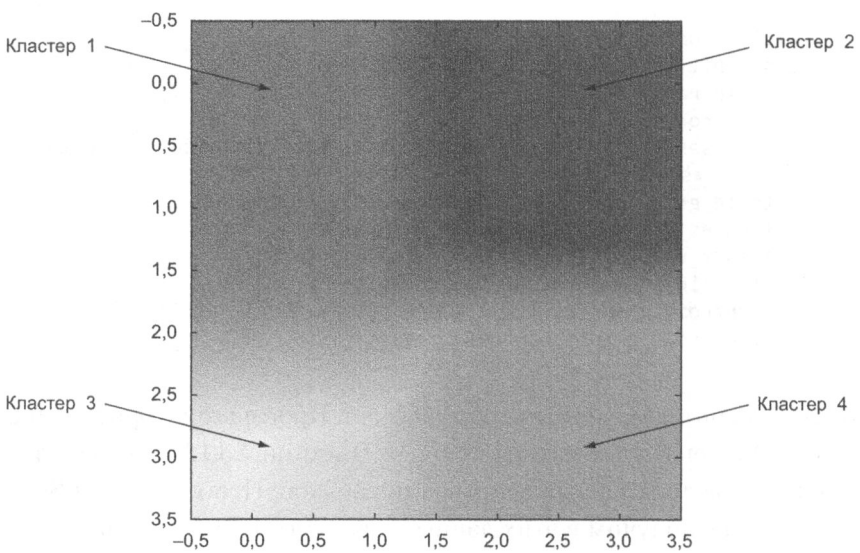


Рис. 5.10. Метод SOM помещает все точки трехмерных данных в сетку двумерного пространства. В ней можно выбрать центроиды кластеров (автоматически или вручную) и добиться кластеризации в наглядном пространстве двух измерений

SOM помещает данные большой размерности в пространство двух измерений, что позволяет значительно облегчить процесс кластеризации. Это удобный шаг для предварительной обработки. Его можно выполнить вручную и указать центроиды кластеров, наблюдая за выходными данными SOM. Возможные варианты центроидов можно также найти автоматически, наблюдая за градиентом весов. Интересующимся мы рекомендуем прочесть статью «Clustering of the Self-Organizing Map», Юхи Весанто (Juha Vesanto) и Изы Алхониеми (Esa Alhoniemi): http://mng.bz/XzyS.

5.6. Применение кластеризации

Вы уже познакомились с двумя практическими применениями кластеризации: с упорядочением музыкальной коллекции и сегментацией аудиоклипа для маркировки сходных звуков. Кластеризация особенно полезна, когда обучающий набор данных не содержит соответствующих меток. Как известно, подобная ситуация характерна для обучения без учителя. Иногда используемые данные просто очень неудобно аннотировать.

Предположим, вы хотите разобраться с данными датчика акселерометра телефона или умных часов. На каждом временном этапе акселерометр выдает трехмерный вектор, но вы не понимаете, идет ли при этом человек, стоит, сидит, танцует или делает что-то еще. Набор данных для решения этой задачи можно найти по адресу http://mng.bz/rTMe.

Для того чтобы кластеризовать данные временного ряда, необходимо преобразовать список векторов акселерометра в компактный вектор признаков. Одним из способов кластеризации является построение гистограммы различий между последовательными значениями ускорения. Производная ускорения носит название *рывка* (jerk), и соответствующую гистограмму можно получить, оконтуривая разности значений рывка.

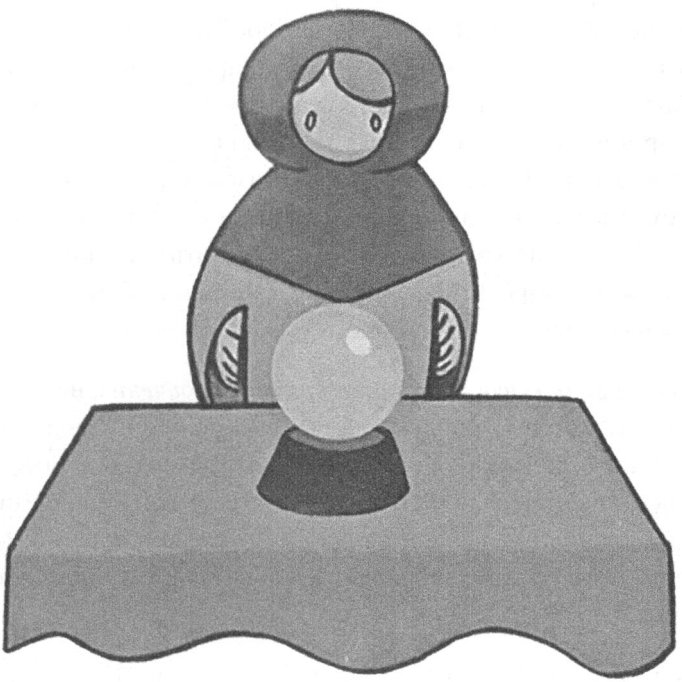
Этот процесс построения гистограммы по данным напоминает шаги предварительной обработки звуковых данных, которые были описаны в этой главе. После преобразования гистограммы в векторы признаков можно воспользоваться кодом из листингов, которые приводились ранее (как для метода k-средних в TensorFlow).

ПРИМЕЧАНИЕ В то время как предыдущие главы описывали обучение с учителем, в этой главе акцент делается на обучении без учителя. В следующей главе представлен алгоритм машинного обучения, который не относится ни к одному из них. Это фреймворк моделирования, который не привлекает в наши дни большого внимания программистов, но является важным инструментом для специалистов в области статистики с целью выявления скрытых составных элементов данных.

5.7. Краткие итоги

- Кластеризация относится к алгоритмам обучения без учителя и используется для обнаружения структур в данных.
- Кластеризация методом k-средних является одним из наиболее простых методов для понимания и реализации алгоритмов, который демонстрирует высокую скорость и точность.
- Если число кластеров не определено, можно использовать алгоритм самоорганизующихся карт, который позволяет представить данные в упрощенном виде.

Скрытое марковское моделирование



Эта глава охватывает следующие темы:

- ✓ Определение интерпретационных моделей
- ✓ Использование цепей Маркова для моделирования данных
- ✓ Получение заключений о скрытом состоянии с помощью скрытой модели Маркова

Если взрывается ракета, то кто-то, вероятно, будет уволен. Поэтому специалисты по ракетостроению должны быть способны давать убедительные заключения о надежности всех компонентов и конфигураций. Они осуществляют это с помощью физического моделирования и математической дедукции из первичных принципов. Вы, наверное, тоже решали научные проблемы с помощью логических размышлений. Рассмотрим закон Бойля—Мариотта: давление и объем газа обратно пропорциональны друг другу при фиксированной температуре. Из этого простого закона можно делать глубокие умозаключения об окружающем нас мире. С недавних пор машинное обучение начало играть важную роль в поддержке дедуктивного умозаключения.

Такие понятия, как *ракетостроение* и *машинное обучение*, не относятся к тем фразам, которые обычно говорят вместе. Но в наши дни моделирование показаний реального датчика с помощью управляемых данными алгоритмов становится все более часто используемым средством в аэрокосмической промышленности. Кроме того, используемые методы машинного обучения находят широкое применение в здравоохранении и автомобильной индустрии. Но почему это происходит?

Этот бум частично может быть обусловлен лучшим пониманием *интерпретируемых моделей* (interpretable models), к которым относятся модели машинного обучения и при использовании которых получаемые в результате обучения параметры имеют понятное истолкование. Например, при взрыве ракеты интерпретируемая модель может помочь проследить основную причину.

УПРАЖНЕНИЕ 6.1

Что делает интерпретируемую модель несколько субъективной? Что является вашим критерием оценки интерпретируемой модели?

ОТВЕТ

Нам нравится ссылаться на математические доказательства как на метод объяснения де-факто. Если кто-то должен был убедить другого в истинности математической теоремы, то доказательство, в котором неопровержимо прослеживаются шаги рассуждения, является достаточным.

Эта глава посвящена раскрытию скрытых объяснений, лежащих в основе данных наблюдений. Представим кукловода, привязывающего веревки для того, чтобы кукла казалась живой. Анализ одних только движений куклы-марионетки может привести к чрезмерно сложным выводам о том, как можно заставить двигаться неодушевленный предмет. После того как были обнаружены привязанные веревки, становится понятно, что кукловод является наилучшим объяснением ее движений.

Эта глава знакомит со *скрытыми марковскими моделями* (HMM, hidden Markov models), которые наглядно раскрывают характеристики исследуемой проблемы. Метод HMM является «кукловодом», действия которого объясняют результаты наблюдений. Вы моделируете наблюдения, используя марковские цепи, описание которых приводится в разделе 6.2.

Перед тем как начать подробное знакомство с марковскими цепями и HMM, рассмотрим альтернативные модели. В следующем разделе представлены модели, которые могут не относиться к интерпретируемым.

6.1. Пример неинтерпретируемой модели

Одним из классических примеров алгоритма машинного обучения типа черный ящик, результаты которого сложно интерпретировать, является классификация изображений. В задаче классификации изображений целью является разметка каждого входного изображения. Проще говоря, классификация изображений часто позиционируется как вопрос с многовариантным ответом: какая из

перечисленных категорий лучше всего описывает это изображение? Специалисты по машинному обучению достигли значительных успехов в решении этой задачи, дойдя до того уровня, когда лучшие современные классификаторы изображений приближаются по эффективности на определенных наборах данных к человеку.

Вы научитесь решать задачу классификации изображений, приведенную в главе 9, с помощью сверточных нейронных сетей (CNN, convolutional neural networks), относящихся к классу моделей машинного обучения, которые выполняют обучение при наличии многих параметров. Но эти же параметры являются и проблемой для CNN: чему соответствует каждый из тысячи, если не из миллиона, параметров? Сложно спросить алгоритм классификации изображений, почему он принял такое решение. Нам доступны лишь полученные в результате обучения параметры, которые не могут в точности объяснить логические выводы, лежащие в основе классификации.

Машинное обучение иногда пользуется дурной славой черного ящика, который решает определенную задачу, не раскрывая, как он пришел к этому решению. Цель этого раздела — снять завесу тайны с машинного обучения с интерпретируемыми моделями. В частности, вы узнаете о HMM и воспользуетесь TensorFlow для их реализации.

6.2. Модель Маркова

Андрей Марков был русским математиком, который изучал способы изменения систем во времени в условиях появления случайности. Представьте, что частицы газа подпрыгивают в воздухе. Отслеживание положения каждой частицы по ньютоновской физике может усложниться, поэтому введение случайности помогает немного упростить физическую модель.

Марков понял, что упростить модель еще больше поможет моделирование только ограниченной области вокруг газовых частиц. Например, газовая частица в Европе едва ли оказывает какое-либо влияние на частицу в США. Тогда почему бы не воспользоваться этим? Математические расчеты упрощаются, если рассматривать только ближайшее окружение, а не всю систему. Эту точку зрения теперь называют *марковским свойством*.

Рассмотрим моделирование погоды. Метеорологи оценивают разные условия с помощью термометров, барометров и анемометров для прогноза погоды. Для этого они используют свою проницательность и многолетний опыт.

Давайте воспользуемся марковским свойством, чтобы начать с простой модели прогноза. Прежде всего, необходимо определить возможные ситуации, или *состояния*, которые мы будем принимать во внимание. На рис. 6.1 показаны три состояния погоды, являющиеся узлами графа: облачно, дождливо и солнечно.

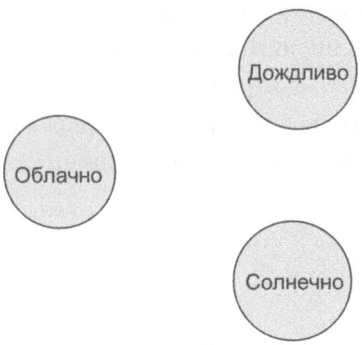


Рис. 6.1. Погодные условия (состояния), представленные в виде узлов графа

Теперь, когда у нас есть состояния, необходимо определить, как одно состояние переходит в другое. Моделирование погоды как детерминистской системы является сложной задачей. Вывод, что если сегодня солнечно, то будет солнечно и завтра, не очевиден. Вместо этого можно внести случайность и сказать, что если сегодня солнечно, то завтра с вероятностью 90 % будет солнечно, а с вероятностью 10 % — облачно. Марковское свойство вступает в игру, когда используются только сегодняшние погодные условия, чтобы спрогнозировать завтрашние (вместо использования всей предшествующей истории).

УПРАЖНЕНИЕ 6.2

Говорят, что робот, который принимает решение, какое действие выполнять на основе только своего текущего состояния, следует марковскому свойству. В чем преимущества и недостатки такого процесса принятия решений?

ОТВЕТ

С марковским свойством легко работать в области вычислений. Но эти модели нельзя обобщать в ситуациях, которые требуют знания исторических данных. Примером являются модели, в которых важна временная тенденция или в которой знание более чем одного состояния в прошлом дает лучшее представление о том, чего следует ожидать в дальнейшем.

На рис. 6.2 показаны переходы в виде ориентированных ребер, проведенных между узлами, стрелки которых направлены на следующие состояния. Каждое ребро имеет вес, представляющий вероятность (например, равную 30 % того, что, если сегодня дождливо, завтра будет облачно). Отсутствие ребра между двумя узлами указывает, что вероятность этого перехода близка к нулю. Вероятности переходов можно узнать из архивных данных, но сейчас будем считать, что они нам известны.

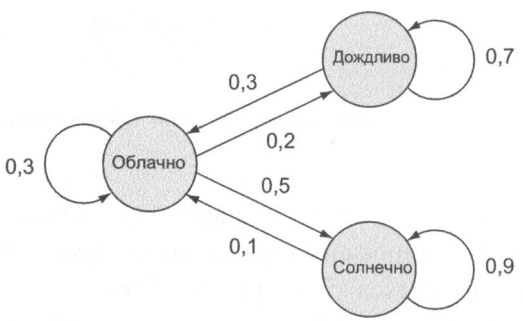


Рис. 6.2. Вероятности переходов между состояниями погоды представлены направленными ребрами

Если есть три состояния, то переходы можно представить как матрицу 3 × 3. Каждый элемент этой матрицы (в строке i и столбце j) соответствует вероятности, связанной с ребром из узла i в узел j. В общем случае, если есть N состояний, то *матрица переходов* будет иметь размерность $N \times N$ (рис. 6.4).

Мы называем эту систему *марковской моделью*. С течением времени состояние изменяется в соответствии с вероятностью переходов, определенной на рис. 6.2. В нашем примере состояние «солнечно» сегодня с вероятность 90 % приведет

к тому, что завтра снова будет «солнечно», поэтому можно провести ребро с маркировкой 0,9, которое возвращается в исходное состояние. Вероятность того, что за солнечным днем последует облачный день, равна 10 %, что на схеме указано ребром с маркировкой 0,1, направленным из состояния «солнечно» в состояние «облачно».

На рис. 6.3 показан другой вариант визуализации того, как меняются состояния, если известны вероятности переходов. Это так называемая *решетчатая диаграмма* (trellis diagram), которая оказалась важным инструментом, как будет видно позже, когда мы реализуем алгоритмы TensorFlow.

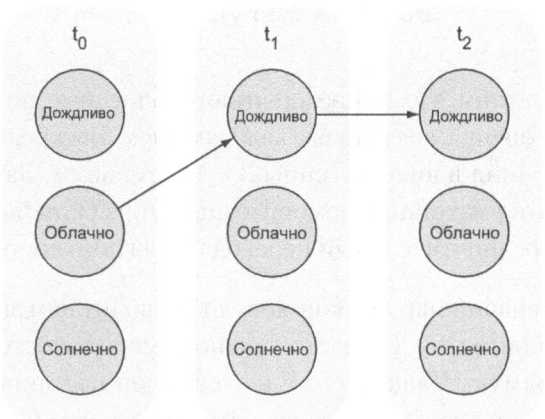


Рис. 6.3. Решетчатое представление марковской системы изменения состояний с течением времени

В предыдущих разделах вы уже видели, как код TensorFlow строит граф, чтобы представить процесс вычислений. Может показаться заманчивым рассматривать каждый узел в марковской модели как узел TensorFlow. Но даже несмотря на то, что рис. 6.2 и 6.3 превосходно иллюстрируют переходы между состояниями, есть более эффективный способ использовать их в программе, и он показан на рис. 6.4.

Помните, что узлы графов в TensorFlow являются тензорами, поэтому можно представить матрицу переходов (назовем ее T) в TensorFlow в виде узла. Затем можно применить математические операции к узлу TensorFlow, которые позволяют получать представляющие интерес результаты.

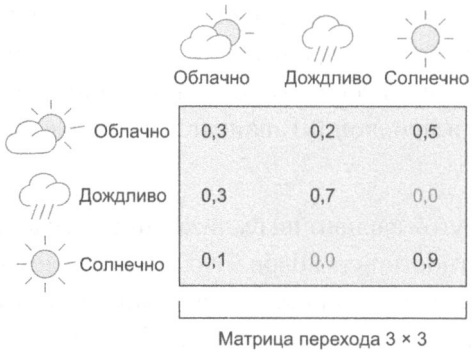


Рис. 6.4. Матрица переходов передает вероятности перехода состояния слева (строки) в состояние вверху (столбцы)

Например, предположим, что вы предпочитаете солнечные дни, а не дождливые, поэтому у вас есть оценка, связанная с каждым днем. Вы представляете оценки для каждого состояния в виде матрицы 3 × 1 с именем *s*. Затем, перемножив в TensorFlow две этих матрицы с помощью оператора `tf.matmul(T*s)`, мы получим ожидаемый предпочтительный переход из каждого состояния.

Представление сценария марковской модели позволит нам значительно упростить то, как мы видим мир. Но часто трудно измерить состояние мира непосредственно напрямую. Чаще всего у нас есть данные многочисленных наблюдений для оценки скрытого смысла. Именно это нам и предстоит решить в следующем разделе!

6.3. Скрытое марковское моделирование

Марковской моделью, определенной в предыдущем разделе, удобно пользоваться, когда все состояния можно наблюдать, но это не всегда так. Представим, что у нас есть доступ только к показаниям температуры в городе. Температура не является погодой, но она связана с ней. Как в таком случае мы сможем сделать заключение о погоде из этого набора непрямых измерений?

Дождливая погода, вероятнее всего, приводит к низким показаниям температуры, тогда как в солнечный день будет более высокая температура. Имея только температурные измерения и вероятности переходов, все же можно делать разумные заключения о наиболее вероятной погоде. Проблемы, похожие на эту, достаточно распространены в реальном мире. Любое состояние

может оставлять после себя некоторые признаки, а эти признаки — все, что нам доступно.

Подобные этим модели относятся к классу HMM, так как истинные состояния в окружающем нас мире (такие, как дождливая или солнечная погода) не могут наблюдаться непосредственно. Эти скрытые состояния подчиняются марковской модели, при этом каждое состояние порождает с определенной вероятностью определенные результаты измерений. Например, скрытое состояние «солнечно» может порождать высокие показания температуры, но, по той или иной причине, также и низкие показания.

В модели HMM следует задать вероятность эмиссии каждого состояния, которая обычно представляется матрицей под названием *матрица эмиссии* (emission matrix). Число строк в этой матрице равно числу состояний (солнечно, облачно, дождливо), а число столбцов равно числу типов наблюдений (жарко, умеренно, холодно). Каждый элемент матрицы является вероятностью эмиссии.

Каноническим методом визуализации HMM является добавление решетки с наблюдениями, как показано на рис. 6.5.

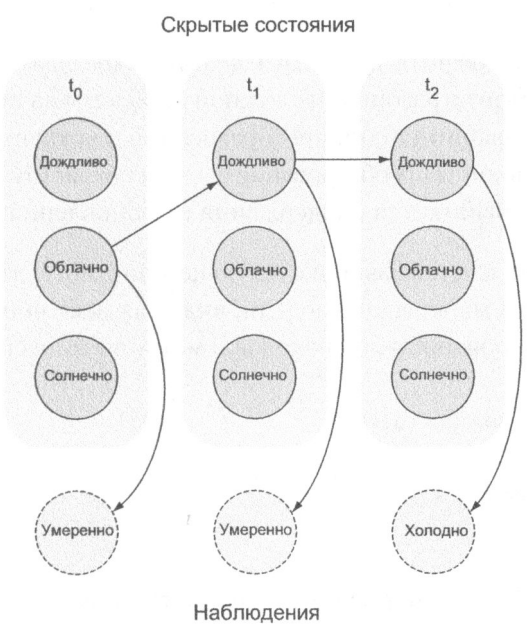


Рис. 6.5. Решетка скрытой марковской модели показывает, как погодные условия могут порождать температурные показания

Итак, это *почти* все. HMM является описанием вероятностей переходов, вероятностей эмиссий, а также еще одной вероятности: *начальной вероятности* (initial probabilities). Начальной вероятностью называют вероятность появления каждого состояния без априорных знаний. Если моделировать погоду в Барселоне, то, вероятно, у состояния «солнечно» начальная вероятность будет больше. Но при моделировании погоды в Лондоне значению начальной вероятности «дождливо» можно присвоить несколько более высокое значение.

HMM позволяет понять последовательность наблюдений. В этом сценарии моделирования погоды вы можете задать вопрос, какова вероятность наблюдения определенной последовательности температурных показаний? Мы ответим на этот вопрос с помощью *алгоритма прямого хода* (forward algorithm).

6.4. Алгоритм прямого хода

С помощью этого алгоритма вычисляется вероятность наблюдения. Многие изменения могут вызвать определенное наблюдение, поэтому простой способ перечисления всех вероятностей требует значительного времени для вычислений.

Эту же задачу можно решить, используя *динамическое программирование*, стратегия которого состоит в разбиении сложной проблемы на небольшие простые задачи и в использовании таблиц преобразований для кэширования результатов. В своем коде таблицы преобразования следует сохранять как массив NumPy и передавать в TensorFlow для поддержания их обновления.

Создайте класс `HMM`, как показано в следующем листинге, чтобы фиксировать скрытые параметры марковской модели, включая вектор начальных вероятностей, матрицы вероятностей переходов и матрицы эмиссии.

Листинг 6.1. Объявление класса HMM

```
import numpy as np
import tensorflow as tf
```

Импорт необходимых библиотек

```
class HMM(object):
    def  init (self, initial_prob, trans_prob, obs_prob):
```

```
self.N = np.size(initial_prob)
self.initial_prob = initial_prob
self.trans_prob = trans_prob
self.emission = tf.constant(obs_prob)

assert self.initial_prob.shape == (self.N, 1)
assert self.trans_prob.shape == (self.N, self.N)
assert obs_prob.shape[0] == self.N

self.obs_idx = tf.placeholder(tf.int32)
self.fwd = tf.placeholder(tf.float64)
```

Сохраняет параметры как переменные метода

Двойная проверка того, что форма всех матриц имеет смысл

Задает переменные-заполнители, используемые для алгоритма прямого логического вывода

Затем необходимо задать вспомогательную функцию для доступа к строке из матриц эмиссии. Приведенный в следующем листинге код описывает вспомогательную функцию для быстрого получения данных из произвольной матрицы. Функция `slice` извлекает часть исходного тензора. Эта функция требует в качестве входных данных соответствующий тензор, начальное положение задается тензором, так же как и размер среза.

Листинг 6.2. Создание вспомогательной функции для доступа к вероятности эмиссии тех или иных наблюдений

```
def get_emission(self, obs_idx):
    slice_location = [0, obs_idx]
    num_rows = tf.shape(self.emission)[0]
    slice_shape = [num_rows, 1]
    return tf.slice(self.emission, slice_location, slice_shape)
```

То место, где нужно сделать срез матрицы эмиссии

Форма среза

Выполняются срезы

Необходимо задать два оператора TensorFlow. Первый, представленный в следующем листинге, будет выполняться только один раз для инициализации кэша алгоритма прямого хода.

Листинг 6.3. Инициализация кэша

```
def forward_init_op(self):
    obs_prob = self.get_emission(self.obs_idx)
    fwd = tf.multiply(self.initial_prob, obs_prob)
    return fwd
```

Следующий оператор при каждом наблюдении будет обновлять кэш (листинг 6.4). Выполнение этого кода часто называют *выполнением шага вперед*. Хотя оно и имеет вид `forward_op`, функция не использует входные данные, так как зависит от переменных-заполнителей, которые необходимо задать для сеанса. Точнее говоря, `self.fwd` и `self.obs_idx` являются входами в эту функцию.

Листинг 6.4. Обновление кэша

```
def forward_op(self):
    transitions = tf.matmul(self.fwd,
  tf.transpose(self.get_emission(self.obs_idx)))
    weighted_transitions = transitions * self.trans_prob
    fwd = tf.reduce_sum(weighted_transitions, 0)
    return tf.reshape(fwd, tf.shape(self.fwd))
```

Зададим за пределами класса `HMM` функцию, которая будет запускать алгоритм прямого хода, как показано в следующем листинге. Алгоритм прямого хода выполняет для каждого наблюдения шаг вперед. После завершения работы алгоритм выдает вероятность наблюдений.

Листинг 6.5. Объявление алгоритма прямого хода дает HMM

```
def forward_algorithm(sess, hmm, observations):
    fwd = sess.run(hmm.forward_init_op(), feed_dict={hmm.obs_idx:
     observations[0]})
    for t in range(1, len(observations)):
        fwd = sess.run(hmm.forward_op(), feed_dict={hmm.obs_idx:
     observations[t], hmm.fwd: fwd})
    prob = sess.run(tf.reduce_sum(fwd))
    return prob
```

Зададим в главной функции класс HMM, введя вектор начальных вероятностей, матрицу вероятностей переходов и матрицу вероятностей эмиссии. Приведенный в листинге 6.6 пример взят прямо из статьи, посвященной HMM: http://mng.bz/8ztL[1] (рис. 6.6).

В общем случае эти три понятия определяются следующим образом:

- *Вектор начальной вероятности* — исходная вероятность состояний.

[1] https://ru.wikipedia.org/wiki/Скрытая_марковская_модель

- *Матрица вероятностей переходов* — вероятности, связанные с переходом из текущего состояния в следующее.
- *Матрица вероятностей эмиссии* — вероятность наблюдаемого состояния при условии того, что интересующее вас состояние уже было.

Имея эти матрицы, можно вызвать алгоритм прямого хода, который был только что задан.

```
states = ('Rainy', 'Sunny')

observations = ('walk', 'shop', 'clean')

start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

transition_probability = {
   'Rainy  : {'Rainy': 0.7, 'Sunny': 0.3},
   'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6},
}

emission_probability = {
   'Rainy  : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
   'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},
}
```

Рис. 6.6. Скриншот примера сценария HMM, взятого из Википедии

Листинг 6.6. Объявление HMM и вызов алгоритма прямого логического заключения

```
if  name == ' main ':
    initial_prob = np.array([[0.6],
                             [0.4]])

    trans_prob = np.array([[0.7, 0.3],
                           [0.4, 0.6]])

    obs_prob = np.array([[0.1, 0.4, 0.5],
                         [0.6, 0.3, 0.1]])

    hmm = HMM(initial_prob=initial_prob, trans_prob=trans_prob,
     obs_prob=obs_prob)

    observations = [0, 1, 1, 2, 1]
    with tf.Session() as sess:
        prob = forward_algorithm(sess, hmm, observations)
        print('Probability of observing {} is {}'.format(observations, prob))
```

При выполнении кода из листинга 6.6 на выходе будет следующее:

```
Probability of observing [0, 1, 1, 2, 1] is 0.0045403
```

6.5. Декодирование Витерби

Алгоритм декодирования Витерби находит наиболее вероятную последовательность скрытых состояний при наличии ряда наблюдений. Для него необходима схема кэша, аналогичного используемому в алгоритме прямого хода. Назовем его кэшом `viterbi`. В конструкторе HMM добавьте строку, показанную в следующем листинге.

Листинг 6.7. Добавление кэша Витерби в качестве переменной экземпляра

```
def  init (self, initial_prob, trans_prob, obs_prob):
  ...
  ...
  ...
  self.viterbi = tf.placeholder(tf.float64)
```

В следующем листинге будет задан оператор TensorFlow для обновления кэша `viterbi`. Этот метод относится к классу HMM.

Листинг 6.8. Объявление оператора для обновления кэша алгоритма прямого хода

```
def decode_op(self):
        transitions = tf.matmul(self.viterbi,
     tf.transpose(self.get_emission(self.obs_idx)))
        weighted_transitions = transitions * self.trans_prob
        viterbi = tf.reduce_max(weighted_transitions, 0)
        return tf.reshape(viterbi, tf.shape(self.viterbi))
```

Кроме этого, потребуется оператор обновления обратных указателей.

Листинг 6.9. Объявление оператора для обновления обратных указателей

```
def backpt_op(self):
    back_transitions = tf.matmul(self.viterbi, np.ones((1, self.N)))
    weighted_back_transitions = back_transitions * self.trans_prob
    return tf.argmax(weighted_back_transitions, 0)
```

И наконец, в следующем листинге за пределами HMM задается функция декодирования Витерби.

Листинг 6.10. Объявление алгоритма декодирования Витерби

```
def viterbi_decode(sess, hmm, observations):
    viterbi = sess.run(hmm.forward_init_op(), feed_dict={hmm.obs:
     observations[0]})
    backpts = np.ones((hmm.N, len(observations)), 'int32') * -1
    for t in range(1, len(observations)):
        viterbi, backpt = sess.run([hmm.decode_op(), hmm.backpt_op()],
                                   feed_dict={hmm.obs: observations[t],
                                              hmm.viterbi: viterbi})
        backpts[:, t] = backpt
    tokens = [viterbi[:, -1].argmax()]
    for i in range(len(observations) - 1, 0, -1):
        tokens.append(backpts[tokens[-1], i])
    return tokens[::-1]
```

Мы можем запустить код, представленный в следующем листинге, для оценки декодирования данных измерений Витерби.

Листинг 6.11. Выполнение декодирования Витерби

```
seq = viterbi_decode(sess, hmm, observations)
print('Most likely hidden states are {}'.format(seq))
```

6.6. Применение скрытых марковских моделей

Теперь, после того как были реализованы алгоритм прямого хода и алгоритм Витерби, рассмотрим интересные области их применения.

6.6.1. Моделирование видео

Представьте, что вы можете опознать человека, основываясь исключительно на том, как он ходит. Идентификация людей по походке — довольно крутая идея, но для ее реализации необходима модель, позволяющая распознавать манеру ходьбы. Рассмотрим HMM, в которой последовательностью скрытых состояний походки являются: 1) положение покоя, 2) правая нога вперед, 3) положение покоя, 4) левая нога вперед, и, наконец, 5) положение покоя. Наблюдаемые

состояния являются силуэтами человека, идущего, бегущего трусцой, бегущего обычным способом, и взяты из видеозаписи (набор таких данных доступен по адресу http://mng.bz/Tqfx).

6.6.2. Моделирование ДНК

ДНК является последовательностью нуклеотидов, и мы с каждым годом все больше узнаем о ее структурах. Одним из представляющих интерес способов изучения длинной цепочки ДНК является моделирование этих областей, если нам известны порядки величин вероятностей их проявления. Так же как облачный день обычно следует за дождливым, так и определенные области цепочки ДНК (*инициирующий кодон*) чаще всего расположены перед другими областями ДНК (*стоп-кодон*).

6.6.3. Моделирование изображения

При распознавании рукописного текста мы пытаемся извлечь простой текст из изображений рукописных слов. Один из методов состоит в поочередном распознавании символов с последующим соединением результатов. Чтобы построить НММ, можно воспользоваться тем, что символы записаны в последовательностях или словах. Знание предыдущего символа, возможно, поможет вам исключить вероятность следующего символа. Скрытыми состояниями является простой текст, а наблюдениями — кадрированные изображения, содержащие отдельные символы.

6.7. Применение скрытых марковских моделей

Скрытые марковские модели лучше всего работают тогда, когда у вас есть представление о том, что такое скрытые состояния и как они меняются со временем. К счастью, в области обработки естественного языка разметка частей речи предложения может быть решена с использованием НММ:

- Последовательность слов в предложении соответствует наблюдениям НММ. Например, предложение «Открой шлюзовые двери, ХАЛ» имеет шесть наблюдаемых слов.

- Скрытыми состояниями являются части речи, такие как глагол, существительное, прилагательное и т. д. Наблюдаемое слово *открой* в предыдущем примере должно соответствовать скрытому состоянию *глагол*.
- Вероятности перехода могут быть спроектированы программистом или получены с помощью данных. Эти вероятности представляют правила частей речи. Например, вероятность появления двух глаголов друг за другом должна быть низкой. Настраивая вероятность перехода, можно избежать необходимости заставлять алгоритм использовать все возможности.

Вероятности эмиссии каждого слова могут быть получены из данных. Традиционный размеченный набор данных частей речи носит название Moby; его можно найти на сайте www.gutenberg.org/ebooks/3203.

> **ПРИМЕЧАНИЕ** Теперь у вас есть все, что нужно, для разработки собственных экспериментов с использованием скрытых марковских моделей! Это мощный инструмент, и мы призываем вас попробовать использовать его для своих собственных данных. Предварительно определите некоторые переходы и выбросы и посмотрите, сможете ли вы восстановить скрытые состояния. Надеемся, эта глава поможет вам начать это делать.

6.8. Краткие итоги

- Сложную, запутанную систему можно упростить, использовав марковскую модель.
- Скрытая марковская модель особенно полезна при применении в реальном мире, поскольку большинство наблюдений являются измерениями скрытых состояний.
- Алгоритмы прямого хода и декодирования Витерби относятся к наиболее распространенным алгоритмам, используемым в HMM.

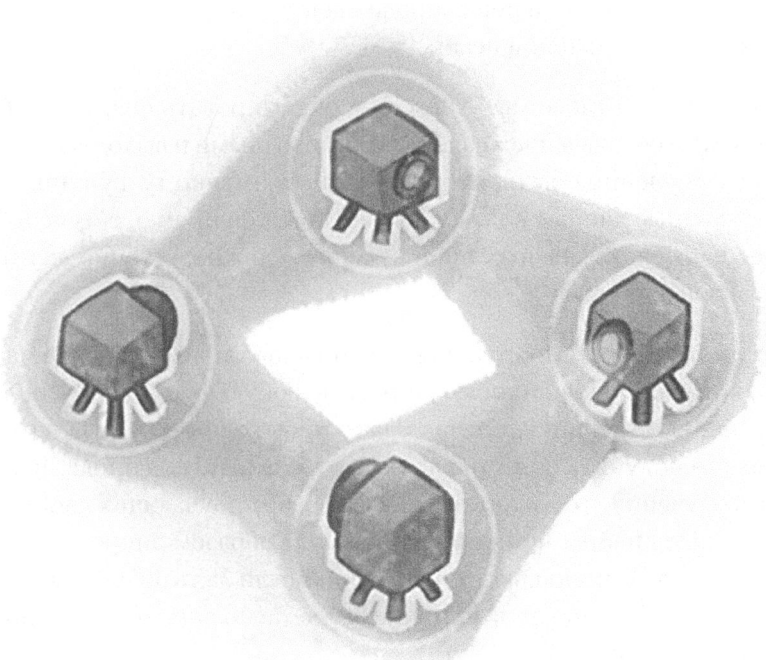
Часть III

Парадигма нейронных сетей

В настоящий момент мы наблюдаем, как многие отрасли промышленности пытаются возвести нейронные сети на пъедестал. Исследование в области глубокого обучения стало символом статуса корпорации, а теория, на которой оно основано, окутана туманом и дымовой завесой. Компаниями, включая такие гиганты, как *NVIDIA*, *Facebook*, *Amazon*, *Microsoft* и, конечно же, *Google*, были брошены огромные деньги на продвижение этого метода. Оказалось, что глубокое обучение исключительно хорошо работает при решении некоторых проблем и использование TensorFlow как раз то, что позволяет их решать.

Главы в этой части книги с самых основ знакомят читателей с нейронными сетями и приводят примеры применения этих сетей для решения практических задач реального мира. В порядке следования эти главы посвящены автокодировщикам, обучению с подкреплением, сверточным нейронным сетям, рекуррентным нейронным сетям, моделям seq2seq и ранжированию. Полный вперед!

Знакомство с автокодировщиками



Эта глава охватывает следующие темы:

- ✓ Знакомство с нейронными сетями
- ✓ Проектирование автокодировщиков
- ✓ Представление изображений с использованием автокодировщика

Приходилось ли вам слышать, как человек напевает мелодию без слов, и сразу же узнавать песню? Может быть, для вас это не составляет труда, но лично у меня напрочь отсутствует музыкальный слух. Напев без слов является аппроксимацией песни. Лучшей аппроксимацией может быть пение. А если подключить инструментальное сопровождение, то тогда кавер-версия будет неотличима от оригинальной песни.

Вместо песен в этой главе мы будем аппроксимировать функции. Функции являются общим обозначением связей между входными и выходными данными. В машинном обучении обычно требуется найти именно ту функцию, которая связывает входные данные с выходными. Найти функцию, обеспечивающую наилучшее из всех возможных приближений, сложно, но аппроксимировать функцию значительно легче.

Искусственные нейронные сети являются моделью, применяемой в машинном обучении, которая может аппроксимировать любую функцию. Используемая модель является функцией, которая дает те выходные данные, которые вы рассчитываете получить для имеющихся у вас входных данных. В терминах машинного обучения это означает, что, имея набор обучающих данных, можно построить модель нейронной сети, наилучшим образом аппроксимирующую неявную функцию. С помощью этой функции могли быть получены обучающие данные, и хотя она не может дать точного ответа, но вполне может приносить определенную пользу.

До сих пор мы создавали модели, в явном виде проектируя функцию, независимо от того, является она линейной, полиномиальной или чем-то более

сложным. Нейронные сети дают некоторую свободу действий, когда дело доходит до выбора подходящей функции, а следовательно, и подходящей модели. Теоретически нейронная сеть может моделировать типы преобразований общего назначения, для использования которых не требуется многое знать о моделируемой функции!

После того как в разделе 7.1 будут введены нейронные сети, в разделе 7.2 мы узнаем, как пользоваться автокодировщиками, которые кодируют данные в меньшие и более быстрые представления.

7.1. Нейронные сети

Если вы слышали о нейронных сетях, то наверняка видели схемы узлов и ребер, соединенных в сложную сеть. Эта визуализация в большей степени используется в биологии, в особенности для отображения нейронов мозга. Оказалось, что это также удобный способ визуально представить функцию, такую как $f(x) = w \times x + b$, приведенную на рис. 7.1.

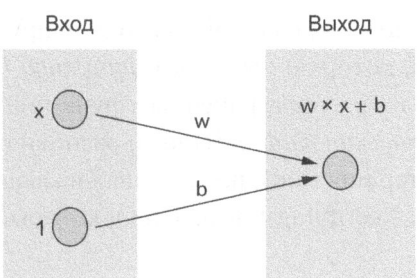


Рис. 7.1. Графическое представление линейного уравнения $f(x) = w \times x + b$. Узлы представлены кружками, а ребра — стрелками. Приведенные на ребрах символы называют *весами*, на которые нужно умножить входное значение. Если две стрелки направлены на один узел, то они обозначают суммирование значений на входах с учетом весов (в данном случае используются два входа)

Напомним, что линейная модель — это набор линейных функций: например, $f(x) = w \times x + b$, где (w, b) является вектором параметров. Алгоритм обучения переходит от одних значений w и b к другим, пока не находит комбинацию, которая наилучшим образом согласуется с данными. После успешной сходимости

алгоритма выбирается наилучшая из всех возможных линейная функция для описания данных.

Линейная функция является тем вариантом, с которого можно начать поиск, но в реальном мире не всегда все так просто. И поэтому мы приступаем к изучению такого типа машинного обучения, с помощью которого TensorFlow формирует отправную точку. Это глава является введением в тип модели, который носит название *искусственная нейронная сеть* (artificial neural network) и который может аппроксимировать совершенно произвольные функции (а не только линейные).

УПРАЖНЕНИЕ 7.1

Является ли $f(x) = |x|$ линейной функцией?

ОТВЕТ

Нет. Это две линейные функции, объединенные в нуле, а не одна прямая линия.

Чтобы внести нелинейность, к каждому выходу нейрона следует применить нелинейную функцию, которую называют *функцией активации* (activation function). К трем наиболее распространенным функциям активации относятся *сигмоид* (sig), *гиперболический тангенс* (tan) и разновидность — *линейно изменяющаяся функция* (ramp function), получившая название *усеченное линейное преобразование* (ReLU, Rectifying Linear Unit), которые представлены в виде графика на рис. 7.2.

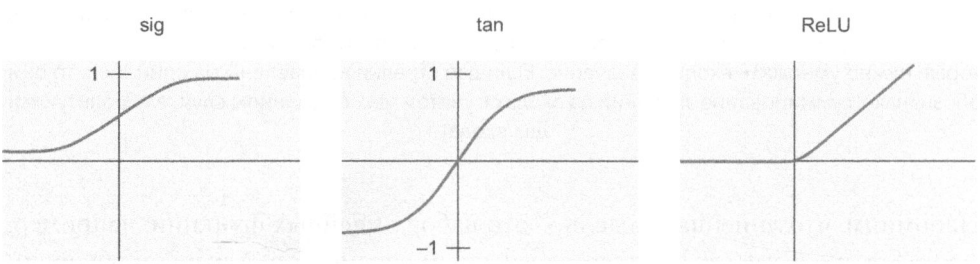


Рис. 7.2. Используйте нелинейную функцию, такую как sig, tan и ReLU, чтобы внести в модели нелинейность

Не следует слишком беспокоиться о том, какая из функций активации лучше и при каких обстоятельствах. Этот вопрос все еще активно исследуется. Можно свободно экспериментировать со всеми тремя представленными на рис. 7.2 функциями. Обычно наилучшая из них выбирается с помощью перекрестной проверки того, какая из них дает наилучшую модель, с учетом тех баз данных, с которыми вы работаете. Помните матрицу неточностей в главе 4? Можно проверить, какая модель дает меньше всего ложноположительных или ложноотрицательных заключений, или использовать любой другой критерий, лучше всего подходящий для решаемых задач.

Сигмовидная функция для нас не является новой. Как вы помните, система классификации под названием *логистическая регрессия* из главы 4 применяет эту сигмовидную функцию к линейной функции $w \times x + b$. Модель нейронной сети на рис. 7.3 представляет функцию $f(x) = \text{sig}(w \times x + b)$. Ей соответствует сеть с одним входом и одним выходом, в которой w и b являются параметрами этой модели.

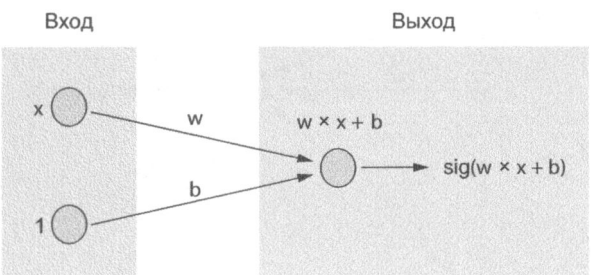


Рис. 7.3. На выходе узла применена нелинейная функция, аналогичная сигмовидной

Если имеется два входа ($x1$ и $x2$), можно изменить нейронную сеть, чтобы она выглядела как на рис. 7.4. При наличии обучающих данных и функции стоимости параметрами, получаемыми при обучении, являются $w1$, $w2$ и b. Когда пытаются моделировать данные, имеющие множество входов, то функция для всех входов является общей. Например, при классификации изображений в качестве входа использует все изображение (пиксел за пикселом).

Естественно, эту задачу можно обобщить для произвольного числа входов ($x1$, $x2$, ..., xn). Соответствующая нейронная сеть представляет функцию $f(x1, ..., xn) = \text{sig}(wn \times xn + ... + w1 \times x1 + b)$, как показано на рис. 7.5.

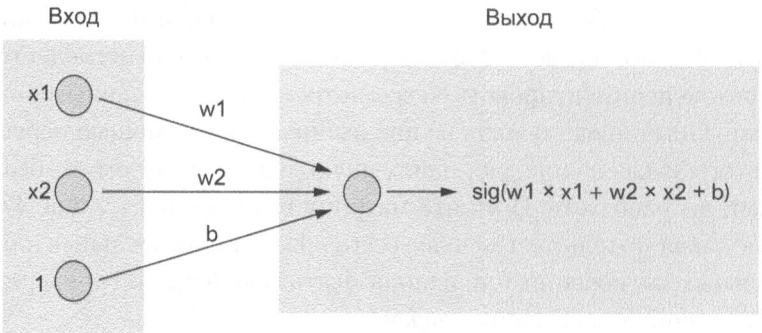


Рис. 7.4. Сеть с двумя входами имеет три параметра (*w*1, *w*2 и *b*). Помните, что несколько линий, ведущих в один узел, указывают на сложение значений на входах с учетом весовых коэффициентов

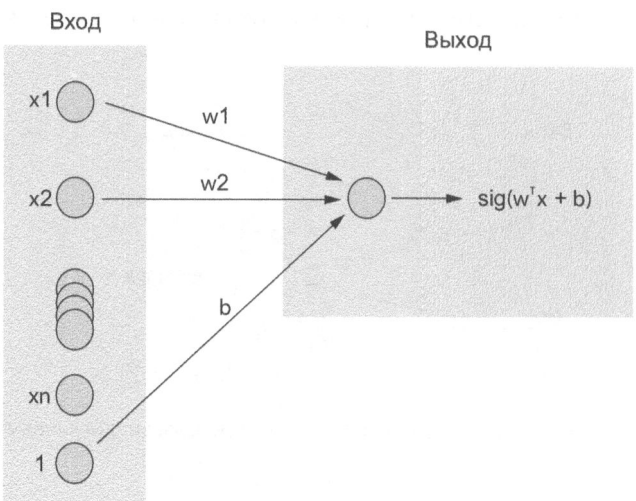


Рис. 7.5. Размерность входа может быть произвольной. Например, каждый пиксел в полутоновом изображении может иметь соответствующий вход *xi*. Все входы нейронной сети используются для одного выхода, который можно применять в регрессии или классификации. Обозначение w^T указывает на транспонирование *w*, то есть на преобразование вектора $n \times 1$ в вектор $1 \times n$. Таким образом можно перемножить его с *x* (который имеет размерность $n \times 1$). Такое матричное перемножение называют *скалярным произведением*, и оно дает скалярную (одномерную) величину

До сих пор мы занимались только входным и выходным слоем. Ничто не мешает нам добавить между ними произвольное число нейронов. Нейроны, которые не используются ни как входные, ни как выходные, носят название *скрытых нейронов*. Они скрыты от входного и выходного интерфейсов нейронной сети, так что ни один из них не может влиять на них напрямую. *Скрытый слой* является любым скоплением скрытых нейронов, которые не связаны друг с другом, как это показано на рис. 7.6. Добавление скрытых слоев значительно улучшает выразительную силу сети.

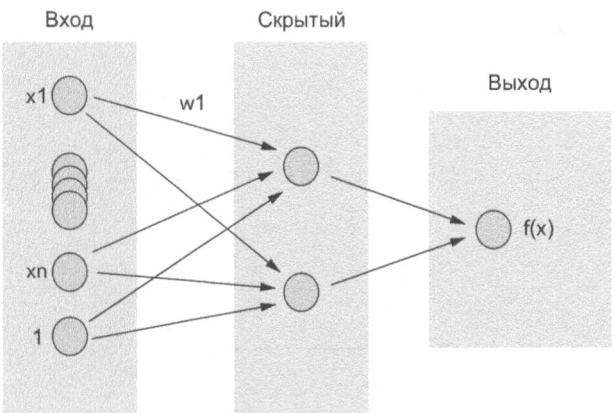


Рис. 7.6. Те узлы, которые не относятся ни к входу, ни к выходу, называются *скрытыми нейронами*. Скрытый слой является скоплением не связанных между собой скрытых нейронов

Поскольку функция активации является нелинейной, то нейронная сеть даже с одним скрытым слоем может аппроксимировать произвольные функции. В линейных моделях, независимо от того, какие именно параметры получаются при обучении, функция остается линейной. С другой стороны, нелинейная нейронная сеть со скрытым слоем является достаточно универсальной, чтобы приблизительно представлять любую функцию!

TensorFlow комплектуется многими вспомогательными функциями, способствующими эффективному получению параметров нейронной сети. В этой главе мы покажем, как можно вызвать эти функции, когда приступим к описанию использования автокодировщика.

7.2. Автокодировщики

Автокодировщик (autoencoder) является одним из типов нейронной сети, которая пытается получить в результате обучения параметры, приближающие, насколько это возможно, выходные данные к входным. Очевидный способ сделать это — вернуть входные данные, как это показано на рис. 7.7.

Но автокодировщик гораздо интереснее. Ведь он содержит небольшой скрытый слой! Если этот скрытый слой имеет меньшую размерность, чем входной слой, то он является сжатием исходных данных, а соответствующую процедуру называют *кодированием*.

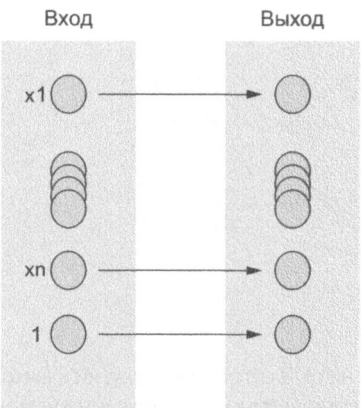


Рис. 7.7. Если требуется создать сеть, в которой вход совпадает с выходом, можно соединить соответствующие узлы и задать вес каждого параметра как равный 1

КОДИРОВАНИЕ ДАННЫХ В РЕАЛЬНОМ МИРЕ

Существует несколько форматов аудио, но самым популярным является формат MP3 из-за относительно небольшого размера файлов. Возможно, вы уже догадались, что такое эффективное хранилище кроме достоинств имеет и определенные недостатки. Алгоритм генерации файла MP3 получает оригинальный несжатый звук и сжимает его в гораздо меньший файл, который для вас звучит практически так же. Но это потеря означает, что вы не сможете полностью восстановить исходный несжатый звук из кодированной версии. Аналогичным образом в этой главе мы хотим уменьшить размерность данных, чтобы сделать их более работоспособными, при этом нам не обязательно создавать идеально точное воспроизведение.

Этот процесс преобразования входных данных из скрытого слоя носит название *декодирования* (decoding). На рис. 7.8 приведен утрированный пример автокодировщика.

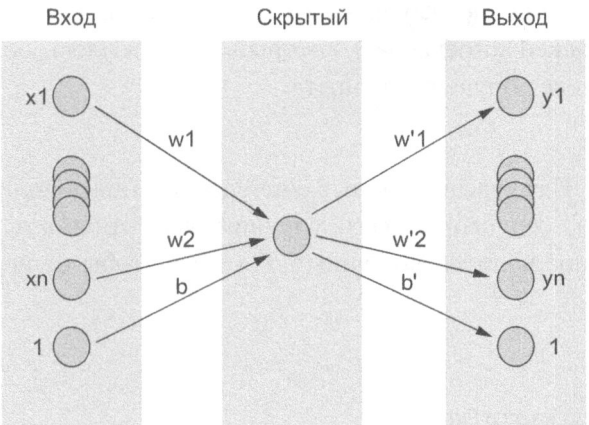


Рис. 7.8. На этом рисунке показано наложение на сеть ограничения, с помощью которого делается попытка преобразовать ее входные данные. Данные будут проходить через узкий канал, которым является скрытый слой. В этом примере в скрытом слое есть только один узел. Эта сеть пытается кодировать (и декодировать) *n*-мерный входной сигнал в одномерный, что, вероятно, будет сложно осуществить на практике

Кодирование является превосходным способом уменьшить размеры входных данных. Например, если можно представить изображение 256 × 256 всего лишь в виде сотни скрытых узлов, то каждый элемент данных будет при этом уменьшен в тысячи раз!

УПРАЖНЕНИЕ 7.2

Пусть *x* обозначает входной вектор (*x*1, *x*2, …, *xn*), а *y* обозначает выходной вектор (*y*1, *y*2, …, *yn*). И наконец, пусть *w* и *w'* обозначают веса кодировщика и декодеровщика соответственно. Какой может быть функция стоимости для обучения этой нейронной сети?

ОТВЕТ

Функция потерь приведена в листинге 7.3.

Имеет смысл воспользоваться объектно-ориентированным программированием, чтобы реализовать автокодировщик. При этом позже можно будет использовать этот класс повторно в других приложениях, не беспокоясь относительно сильносвязанного кода. Создание своего кода, приведенного в листинге 7.1, помогает построить более глубокие архитектуры, такие как *каскадный автокодировщик* (stacked autoencoder), который, как оказалось, более эффективно работает на основе полученного опыта.

> **СОВЕТ** В случае нейронных сетей добавление дополнительных скрытых слоев, скорее всего, улучшит эффективность, если есть достаточное количество данных для того, чтобы не переобучить модель.

Листинг 7.1. Схема класса Python

```
class Autoencoder:
    def  init (self, input_dim, hidden_dim):  ◄—— Инициализирует переменные

    def train(self, data):  ◄—— Обучение на базе данных

    def test(self, data):  ◄—— Обучение на новых данных
```

Откройте новый исходный файл Python и назовите его autoencoder.py. Этот файл определит класс автокодировщика, который вы будете использовать из отдельного фрагмента кода.

Конструктор установит все переменные TensorFlow, переменные-заполнители, оптимизаторы и операторы. Все, что не требует немедленного запуска сеанса, может идти в конструкторе. Поскольку вы работаете с двумя наборами коэффициентов, весами и коэффициентами смещения (один для шага кодирования, а другой для шага декодирования), то для устранения неоднозначности имени переменной можно использовать области имен TensorFlow.

Например, в следующем листинге показан пример определения переменной в области имен. Теперь вы можете легко сохранить и восстановить эту переменную, не беспокоясь о конфликте имен.

Листинг 7.2. Использование областей имен

```
with tf.name_scope('encode'):
    weights = tf.Variable(tf.random_normal([input_dim, hidden_dim],
     dtype=tf.float32), name='weights')
    biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
```

Двигаясь дальше, попробуем реализовать конструктор, который показан в следующем листинге.

Листинг 7.3. Класс автокодировщика

Задает набор данных входного слоя

Определяет веса и коэффициенты смещения в области имен, которые им можно сообщать, кроме весовых коэффициентов и коэффициентов сдвига декодировщика

```
import tensorflow as tf
import numpy as np

class Autoencoder:
    def __init__(self, input_dim, hidden_dim, epoch=250,
     learning_rate=0.001):
        self.epoch = epoch
        self.learning_rate = learning_rate

        x = tf.placeholder(dtype=tf.float32, shape=[None, input_dim])

        with tf.name_scope('encode'):
            weights = tf.Variable(tf.random_normal([input_dim, hidden_dim],
     dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
            encoded = tf.nn.tanh(tf.matmul(x, weights) + biases)
        with tf.name_scope('decode'):
            weights = tf.Variable(tf.random_normal([hidden_dim, input_dim],
     dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]), name='biases')
            decoded = tf.matmul(encoded, weights) + biases

        self.x = x
        self.encoded = encoded
        self.decoded = decoded

        self.loss = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
     self.decoded))))
        self.train_op =
     tf.train.RMSPropOptimizer(self.learning_rate).minimize(self.loss)
        self.saver = tf.train.Saver()
```

Число циклов обучения

Гиперпараметр модуля оптимизации

Переменные метода

Веса и коэффициенты смещения кодировщика задаются в этой области имен

Определяет затраты преобразования

Выбирает модуль оптимизации

Определяет модуль сохранения, чтобы сохранять параметры модели, после того как процесс обучения был завершен

Теперь, в следующем листинге, вы определите метод класса `train`, который получит набор данных и в процессе обучения, используя эти данные, выдаст параметры для минимизации потерь.

Листинг 7.4. Обучение автокодировщика

Итеративно выполняет определенное число циклов, заданных в конструкторе

Начинает сеанс TensorFlow и инициализирует все переменные

```
    def train(self, data):
        num_samples = len(data)
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            for i in range(self.epoch):
                for j in range(num_samples):
                    l, _ = sess.run([self.loss, self.train_op],
                    feed_dict={self.x: [data[j]]})
                if i % 10 == 0:
                    print('epoch {0}: loss = {1}'.format(i, l))
                    self.saver.save(sess, './model.ckpt')
            self.saver.save(sess, './model.ckpt')
```

Процесс обучения нейронной сети по элементам данных выполняется по одной выборке за раз

Сохраняет полученные в результате обучения параметры в файл

Выводит ошибку преобразования через каждые 10 циклов

Теперь у вас есть достаточное количество кода для разработки алгоритма, который обучает автокодировщик на произвольных данных. Прежде чем начать использовать этот класс, давайте создадим еще один метод. Как показано в следующем листинге, метод `test` позволит вам оценить автокодировщик на новых данных.

Листинг 7.5. Проверка модели по данным

Загружает параметры, полученные в ходе обучения

```
def test(self, data):
    with tf.Session() as sess:
        self.saver.restore(sess, './model.ckpt')
        hidden, reconstructed = sess.run([self.encoded, self.decoded],
 feed_dict={self.x: data})
    print('input', data)
    print('compressed', hidden)
    print('reconstructed', reconstructed)
    return reconstructed
```

Преобразует входные данные

И наконец, создадим новый файл под названием main.py и воспользуемся вашим классом Autoencoder, как показано в следующем листинге.

Листинг 7.6. Использование класса Autoencoder

```
from autoencoder import Autoencoder
from sklearn import datasets

hidden_dim = 1
data = datasets.load_iris().data
input_dim = len(data[0])
ae = Autoencoder(input_dim, hidden_dim)
ae.train(data)
ae.test([[8, 4, 6, 2]])
```

Запуск функции train выдает отладочную информацию о том, как уменьшаются потери за эпохи. Функция test показывает информацию о процессе кодирования и декодирования:

```
('input', [[8, 4, 6, 2]])
('compressed', array([[ 0.78238308]], dtype=float32))
('reconstructed', array([[ 6.87756062, 2.79838109, 6.25144577,
     2.23120356]], dtype=float32))
```

При этом можно сжать четырехмерный вектор до вектора одной размерности и затем декодировать его обратно в четырехмерный вектор с некоторой потерей данных.

7.3. Пакетное обучение

Обучение сети по одному образцу является беспроигрышным вариантом, если нет ограничений по времени. Но если обучение сети занимает времени больше, чем можно себе позволить, то одним из решений этой проблемы является обучение с одновременным использованием многочисленных входных данных, получившее название *пакетного обучения* (batch training).

Обычно по мере увеличения размера пакета алгоритм ускоряется, однако имеет низкий уровень вероятности успешной сходимости. Это обоюдоострый меч.

Рассмотрим сказанное на примере следующего листинга. Приведенной там вспомогательной функцией воспользуемся позже.

Листинг 7.7. Вспомогательная функция пакетного обучения

```
def get_batch(X, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a]
```

Чтобы использовать пакетное обучение, потребуется изменить метод `train` из листинга 7.4. Версия пакетного обучения приведена в следующем листинге. В ней используется дополнительный внутренний цикл для каждого пакета данных. Обычно числа итераций для пакета должно хватить, чтобы все данные сошлись на одной и той же эпохе.

Листинг 7.8. Пакетное обучение

```
def train(self, data, batch_size=10):
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            for i in range(self.epoch):
                for j in range(500):
                    batch_data = get_batch(data, self.batch_size)
                    l, _ = sess.run([self.loss, self.train_op],
     feed_dict={self.x: batch_data})
                if i % 10 == 0:
                    print('epoch {0}: loss = {1}'.format(i, l))
                    self.saver.save(sess, ./model.ckpt')
            self.saver.save(sess, ./model.ckpt')
```

В цикле выбирает различные пакеты

Выполняет оптимизацию случайно выбранного пакета

7.4. Работа с изображениями

Большинство нейронных сетей, таких как наш автокодировщик, используют только одномерный вход. С другой стороны, пикселы изображений индексированы по строкам и столбцам. Кроме того, если пиксел представляет цветное изображение, то он имеет насыщенность красного, зеленого и синего цвета, как показано на рис. 7.9.

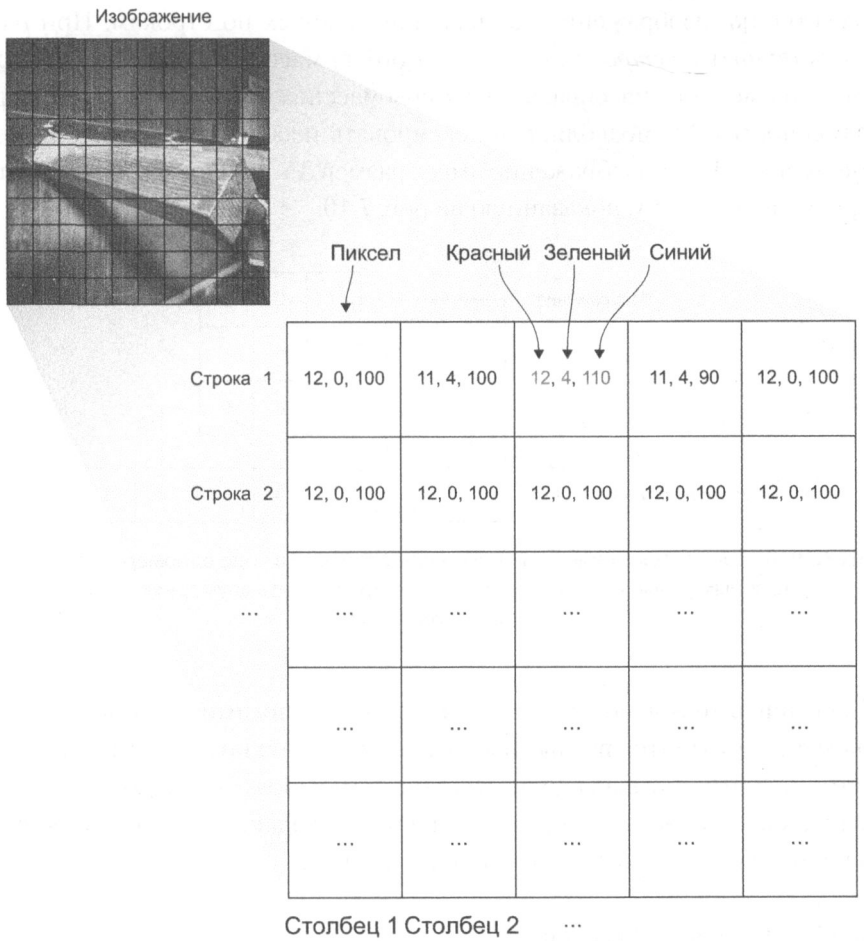


Рис. 7.9. Цветное изображение состоит из пикселов, причем каждый из них имеет определенные значения насыщенности красного, зеленого и синего цвета

Удобный способ управлять большой размерностью изображений включает два шага:

1. Преобразование изображения в градации серого, которое позволяет соединить значение насыщенности красного, зеленого и синего цвета в *интенсивность пиксела* (pixel intensity), которая является взвешенным средним насыщенностей цвета.

2. Перестройка изображения размещением данных по строкам. При *размещении данных по строкам* (row-major order) массив хранится в виде длинного одномерного массива; все размеры массива помещают в конец первой размерности. Это позволяет индексировать изображение одним номером вместо двух. Если изображение имеет размер 3 × 3 пиксела, оно будет перестроено в структуру, показанную на рис. 7.10.

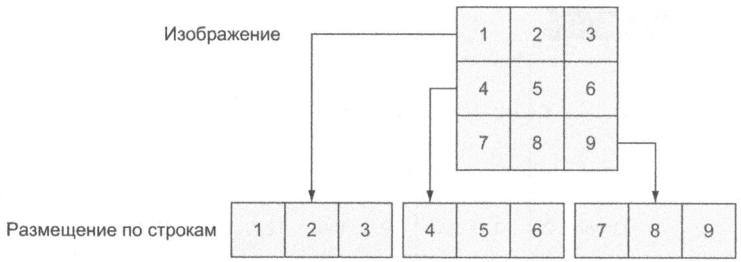


Рис. 7.10. Любое изображение может быть представлено в виде одномерного массива с построчным размещением. Это позволяет представить двумерную структуру как одномерную

Изображение в TensorFlow можно использовать многими способами. Если на жестком диске хранятся разные изображения, их можно загрузить с помощью библиотеки SciPy, которая входит в состав TensorFlow. В следующем листинге показано, как загружать изображение в шкале градаций серого, изменять его размер и переводить в построчное представление.

Листинг 7.9. Загрузка изображений

```
from scipy.misc import imread, imresize

gray_image = imread(filepath, True)
small_gray_image = imresize(gray_image, 1. / 8.)
x = small_gray_image.flatten()
```

Загружает изображения в шкале уровней серого цвета

Преобразует в одномерную структуру

Изменяет на несколько меньший размер

Обработка изображений — это излюбленная область исследований, поэтому наборы данных легко доступны и вы можете использовать их для собственных

ограниченных по количеству изображений. Например, набор данных под названием CIFAR-10 содержит 60 000 размеченных изображений, каждое размером 32 × 32.

УПРАЖНЕНИЕ 7.3

Можете ли вы назвать другие наборы данных с изображениями? Погуглите в интернете, чтобы узнать больше!

ОТВЕТ

В глубоком обучении чаще всего используется набор данных ImageNet (www.image-net.org). Много наборов данных также можно найти по адресу http://deeplearning.net/datasets.

Загрузите набор данных Python с сайта www.cs.toronto.edu/~kriz/cifar.html. Поместите папку cifar-10-batches-py в рабочий каталог. Следующий листинг получен со страницы сайта CIFAR-10; добавьте код в новый файл main_ imgs.py.

Листинг 7.10. Чтение из извлеченного набора данных CIFAR-10

```
import pickle

def unpickle(file):    ◄—— Считывает файл CIFAR-10, возвращаясь в директорию загрузки
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict
```

Давайте прочитаем каждый из файлов набора данных, используя только что созданную функцию `unpickle`. Набор данных CIFA-10 содержит шесть файлов, каждый из которых имеет префикс data_batch, и за ним следует номер. Каждый файл содержит информацию о данных изображения и соответствующей метке. В следующем листинге показан циклический обход всех файлов и добавление данных в память.

Листинг 7.11. Считывание всех файлов CIFAR-10 в память

```
import numpy as np

names = unpickle('./cifar-10-batches-py/batches.meta')['label_names']
data, labels = [], []
for i in range(1, 6):
    filename = './cifar-10-batches-py/data_batch_' + str(i)
    batch_data = unpickle(filename)
    if len(data) > 0:
        data = np.vstack((data, batch_data['data']))
        labels = np.hstack((labels, batch_data['labels']))
    else:
        data = batch_data['data']
        labels = batch_data['labels']
```

Циклически проходит по шести файлам

Загружает файл, чтобы перейти в директорию Python

Классы являются одномерными, поэтому их можно укладывать горизонтально

Строка данных представляет каждый пример, поэтому их можно размещать вертикально

Каждое изображение представлено как серия красных пикселов, за которой следует серия зеленых, а затем синих пикселов. В листинге 7.12 показано создание вспомогательной функции для преобразования цветного изображения в изображение в шкале уровней серого цвета путем усреднения насыщенностей красного, зеленого и синего цвета.

> **ПРИМЕЧАНИЕ** Вы можете добиться более реалистичного оттенка серого другими способами, но делается это за счет подхода усреднения трех значений. Человеческое восприятие более чувствительно к зеленому цвету, поэтому в некоторых других вариантах серого зеленые значения могут иметь более высокий вес при усреднении.

Листинг 7.12. Преобразования изображений из набора CIFAR-10 в изображение в градациях серого

```
def grayscale(a):
    return a.reshape(a.shape[0], 3, 32, 32).mean(1).reshape(a.shape[0], -1)

data = grayscale(data)
```

И наконец, соберем все изображения определенного класса, например `horse` (лошадь). Теперь запустим автокодировщик на всех изображениях лошадей, как показано в следующем листинге.

Листинг 7.13. Настройка автокодировщика

```
from autoencoder import Autoencoder

x = np.matrix(data)
y = np.array(labels)

horse_indices = np.where(y == 7)[0]

horse_x = x[horse_indices]

print(np.shape(horse_x)) # (5000, 3072)

input_dim = np.shape(horse_x)[1]
hidden_dim = 100
ae = Autoencoder(input_dim, hidden_dim)
ae.train(horse_x)
```

Теперь вы можете кодировать изображения, похожие на ваши обучающие данные, всего в 100 чисел. Эта модель автокодировщика является одной из самых простых, поэтому ясно, что это будет кодирование с потерями. Внимание! Выполнение этого кода может занять до 10 минут. Выход будет отслеживать значения потерь каждые 10 эпох:

```
epoch 0: loss = 99.8635025024
epoch 10: loss = 35.3869667053
epoch 20: loss = 15.9411172867
epoch 30: loss = 7.66391372681
epoch 40: loss = 1.39575612545
epoch 50: loss = 0.00389165547676
epoch 60: loss = 0.00203850422986
epoch 70: loss = 0.00186171964742
epoch 80: loss = 0.00231492402963
epoch 90: loss = 0.00166488380637
epoch 100: loss = 0.00172081717756
epoch 110: loss = 0.0018497039564
epoch 120: loss = 0.00220602494664
epoch 130: loss = 0.00179589167237
epoch 140: loss = 0.00122790911701
epoch 150: loss = 0.0027100709267
epoch 160: loss = 0.00213225837797
epoch 170: loss = 0.00215123943053
epoch 180: loss = 0.00148373935372
epoch 190: loss = 0.00171591725666
```

Посмотрите сайт книги или архив GitHub, где приводится полный пример выходных данных: https:// www.manning.com/books/machine-learning-with-tensorflow или http://mng.bz/D0Na.

7.5. Применение автокодировщиков

В этой главе приводится описание наиболее эффективного типа автокодировщика, но также рассмотрены и другие варианты, каждый со своими преимуществами и областями применения. Давайте рассмотрим некоторые из них:

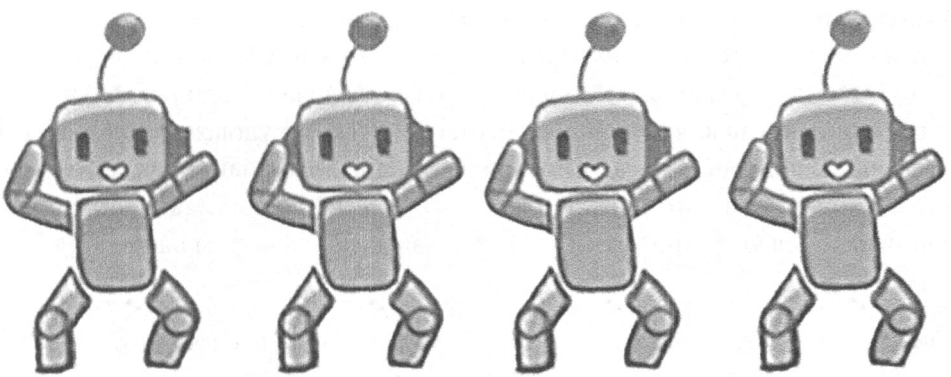
- *Каскадный автокодировщик* начальный этап работы выполняет как обычный кодировщик. Он обучается кодированию входных данных в небольшой скрытый слой путем минимизации ошибки преобразования. Скрытый слой теперь рассматривается как вход для нового кодировщика, который пытается кодировать первый слой скрытых нейронов в слой меньшего размера (второй слой скрытых нейронов). Это может быть продолжено произвольным образом. Часто полученные в процессе обучения веса кодирования используются как начальные значения для решения задач регрессии или классификации в глубокой архитектуре нейронной сети.
- *Шумоподавляющий автокодировщик* (denoising autoencoder) получает входные данные с шумами вместо исходных входных данных и пытается этот шум подавить. Для минимизации ошибки преобразования функция стоимости больше не используется. Теперь попытаемся минимизировать отклонение изображения после шумоподавления от исходного. Интуитивно мы все еще можем воспринимать фотографию даже с царапинами или с нанесенной маркировкой. Если машина тоже видит сквозь шумы входные данные, то, возможно, она лучше понимает их. Модели шумоподавления, как было показано, лучше улавливают характерные признаки изображения.
- *Вариационный автокодировщик* (variational autoencoder) может создавать новые естественные изображения, непосредственно используя скрытые переменные. Представим, что кодируется изображение мужчины, представленного 100-мерным вектором, а затем изображение женщины как другой 100-мерный вектор. Можно взять среднее двух векторов, пропустить через декодировщик и получить некое изображение, визуально представляющее

человека, являющегося чем-то средним между мужчиной и женщиной. Эта генеративная способность вариационного автокодировщика была унаследована у вероятностных моделей, или *байесовских сетей*.

7.6. Краткие итоги

- Нейронная сеть полезна, если для описания набора данных неэффективна линейная модель.
- Автокодировщики относятся к алгоритмам обучения без учителя, которые пытаются воспроизвести свои входные данные, при этом выявляя представляющие интерес структуры в данных.
- Изображения можно без труда подавать в качестве входных данных в нейронную сеть с помощью сглаживания или преобразования в градации серого.

Обучение с подкреплением



Эта глава охватывает следующие темы:

- ✓ Определение обучения с подкреплением
- ✓ Реализация обучения с подкреплением

Люди учатся на прошлом опыте (или, по крайней мере, *должны* учиться). Вы познавали мир не сразу. Удачи и поражения способствовали вашему формированию. И эта глава посвящена разработке системы машинного обучения, которая приводится в действие критикой и поощрениями.

Вы узнавали, что делает людей счастливыми: например, общение с друзьями, семьей или даже с незнакомцами, вы научились ездить на велосипеде, пробуя различные движения мышц, пока езда не стала получаться автоматически. При выполнении тех или иных действий успех иногда приходит сразу. Например, поиск хорошего ресторана поблизости может дать мгновенное удовлетворение. В другой раз награда появляется не сразу, как в случае продолжительной поездки в поисках исключительного места, где можно с удовольствием поесть. Обучение с подкреплением основано на выборе правильных действий в любом состоянии, как это представлено на рис. 8.1, где показано, как человек принимает решение прибыть к месту назначения и при этом найти короткий маршрут.

Так и при поездке из дома на работу вы всегда выбираете один и тот же маршрут. Но однажды ваша любознательность берет верх, и вы решаетесь попробовать другой маршрут. Эта дилемма выбора — попробовать новый маршрут или продолжать пользоваться хорошо известным старым маршрутом — является примером дилеммы *исследовать* или *использовать* (exploration versus exploitation).

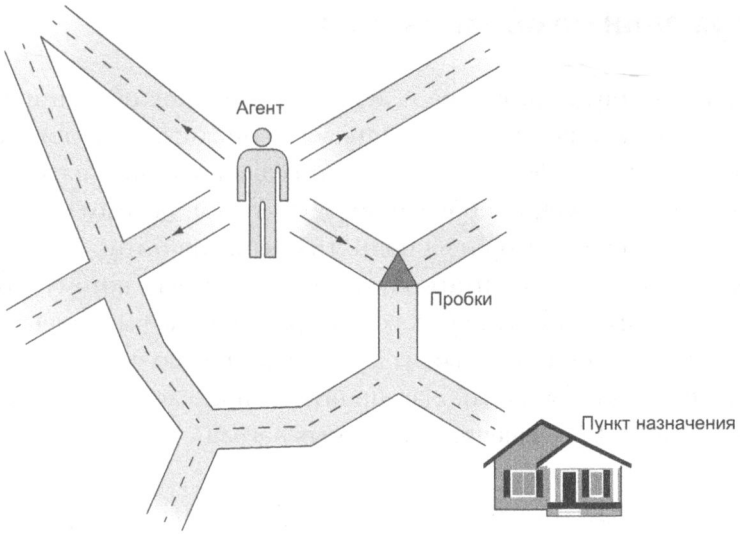


Рис. 8.1. Агент (человек) ищет оптимальный маршрут, чтобы добраться до места назначения в самый разгар движения и при неожиданных ситуациях; это именно та задача, которую можно поставить для обучения с подкреплением

ПРИМЕЧАНИЕ Почему компромисс между использованием нового или сохранением старого называют проблемой исследования или использования? Исследование вносит смысл, но и под использованием можно понимать применение ваших знаний о положении вещей, оставаясь с тем, что вы хорошо знаете.

Все эти примеры могут быть объединены под общим названием: выполнение определенного сценария действия может принести определенную награду. Выражаясь техническим языком, сценарий является *состоянием*. И мы называем совокупность всех возможных состояний *пространством состояний* (state space). Выполнение того или иного действия приводит к изменению состояния. Но вопрос состоит в том, чтобы узнать, какая последовательность действий приносит максимально ожидаемый успех.

8.1. Формальные обозначения

В то время как контролируемое и неконтролируемое обучение находятся в противоположных концах диапазона всевозможных методов машинного обучения, *обучение с подкреплением* (reinforcement learning) находится где-то посредине. Оно не является обучением с учителем, потому что обучающие данные появляются из алгоритма принятия решения о том, исследовать или использовать. И его нельзя считать обучением без учителя, потому что алгоритм использует обратную связь с окружающей средой. Пока выполнение действия в том или ином положении приводит к успеху, можно пользоваться обучением с подкреплением, чтобы найти благоприятную последовательность действий для максимального обеспечения ожидаемых наград.

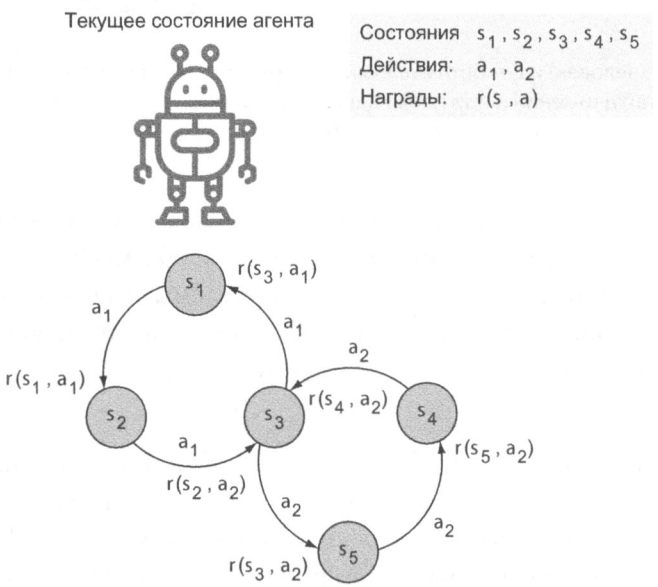


Рис. 8.2. Действия представлены стрелками, а состояния — кружками. Выполнение действия в состоянии приводит к награде. Если начать в состоянии $s1$, можно выполнить действие $a1$, чтобы получить награду $r(s1, a1)$

Нельзя не заметить, что фраза «обучение с подкреплением» содержит антропоморфизацию алгоритма, которая заключается в выполнении *действий*

в определенных *ситуациях* для *достижения успеха*. Этот алгоритм часто называют *агентом*, который *взаимодействует* с окружающей средой. Не следует удивляться тому, что большая часть теории обучения с подкреплением применяется в робототехнике. На рис. 8.2 демонстрируется взаимодействие между состояниями, действиями и результатами (успехом).

Для изменения состояния робот выполняет действия. Но как он решает, какое действие следует предпринять? Чтобы ответить на этот вопрос, в следующем подразделе вводится новая концепция, получившая название *политика* (policy).

ПОЛЬЗУЮТСЯ ЛИ ЛЮДИ ОБУЧЕНИЕМ С ПОДКРЕПЛЕНИЕМ?

Обучение с подкреплением представляется лучшим способом объяснить, как выполнять следующее действие, основываясь на текущей ситуации. Возможно, люди ведут себя так исходя из биологических причин. Но давайте не будем опережать самих себя и рассмотрим следующий пример.

Иногда люди действуют не задумываясь. Если мне хочется пить, я могу инстинктивно схватить стакан воды, чтобы утолить жажду. Я не перебираю в сознании все возможные движения и не выбираю оптимальный вариант, как именно поднести стакан, после тщательных вычислений.

Самое важное то, что выполняемые нами действия не характеризуются одними только наблюдениями в каждый момент их выполнения. Иначе получается, что мы не умнее бактерии, которая действует в зависимости от состояния окружающей среды. Кажется, что происходит что-то более сложное, и простая модель обучения с подкреплением не может полностью объяснить поведение человека.

8.1.1. Политика

Каждый приводит свою комнату в порядок по-разному. Некоторые начинают заправлять постель. Я предпочитаю приводить в порядок комнату по часовой стрелке, чтобы не пропустить ни одного угла. Вы когда-нибудь видели, как работает робот-пылесос, например, Roomba? Некоторые программируют стратегию, которой можно следовать, чтобы выполнить в комнате уборку. В стимулированном обучении процесс выбора агентом определенного действия

называется *политикой*: это набор действий, которые определяют следующее состояние (рис. 8.3).

Рис. 8.3. Стратегия предлагает, какое следует выполнить действие в данном состоянии

Цель обучения с подкреплением — выявить наилучшую стратегию. Распространенным способом разработки политик является анализ долговременной последовательности действий в каждом состоянии. *Награда* является мерой результативности выполнения какого-либо действия. Наилучшую из всех возможных стратегий называют *оптимальной политикой*, и она является Священным Граалем обучения с подкреплением. Оптимальная политика указывает оптимальное действие в любом состоянии, однако она может не обеспечивать в тот момент максимальную награду.

Если измерять награду по немедленным последствиям, то есть по состоянию вещей после предпринятого действия, то ее легко просчитать. Такой подход называют *жадной стратегией*, однако не всегда стоит выбирать действие с наилучшей *немедленной* наградой. Например, при уборке комнаты вы можете сначала заправить постель, потому что так комната сразу выглядит аккуратнее. Но если вам еще нужно постирать простыни, то уборку постели нельзя считать лучшей стратегией. Вам следует посмотреть на результаты следующих нескольких действий и на конечное состояние, чтобы прийти к оптимальному варианту. Аналогичным образом при игре в шахматы взятие ферзя у противника может дать преимущество фигурам на доске, но если через несколько ходов это приведет к шаху и мату, то сделанный ход не лучший из всех возможных.

Вы можете также выбирать действие произвольным образом, и эту стратегию называют *случайной стратегией*. Если вы нашли стратегию для решения задачи обучения с подкреплением, рекомендуется устроить двойную проверку того, что полученная в процессе обучения стратегия более эффективна, чем случайная стратегия или жадная стратегия.

ОГРАНИЧЕНИЯ (МАРКОВСКОГО) ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

Большинство вариантов обучения с подкреплением предполагают, что лучшее действие может быть установлено на основе знания текущего состояния, а не исходя из учета длительной истории состояний и действий, которые привели к этому состоянию. Этот метод принятия решений основан на текущем состоянии и называется марковским, а фреймворк часто называют *марковским процессом принятия решений* (МППР).

Ситуации, в которых в определенном состоянии достаточно сведений, чтобы сделать следующий шаг, можно моделировать с помощью алгоритмов обучения с подкреплением. Но большинство ситуаций реального мира нельзя считать марковскими и для этих ситуаций нужен более реалистичный метод, такой, например, как иерархическое представление состояний и действий. Проще говоря, иерархические модели похожи на контекстно-свободную грамматику, тогда как МППР напоминают конечные автоматы. Экспрессивный скачок в моделировании проблемы, как в случае перехода от МППР к иерархическим представлениям, может значительно повысить эффективность алгоритма планирования.

8.1.2. Выгода

Долгосрочную награду называют *выгодой*. Если известна выгода выполнения определенного действия в определенном состоянии, поиск оптимальной стратегии легко выполнить с помощью обучения с подкреплением. Например, чтобы решить, какое выполнить действие, выбирается действие с максимальной выгодой. Самое сложное, как можно догадаться, состоит в раскрытии значений этой выгоды.

Выгода выполнения действия a в состоянии s записывается как функция $Q(s, a)$, которую называют *функцией выгоды* (рис. 8.4).

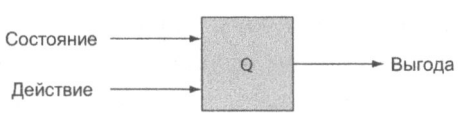


Рис. 8.4. При заданном состоянии и выполненном действии применение функции выгоды Q предсказывает ожидаемую и суммарную выгоды: ближайшая награда (следующее состояние) плюс награды, полученные позже, при следовании оптимальной стратегии

C

УПРАЖНЕНИЕ 8.1

Если у вас функция выгоды Q(s, a), как ее можно использовать, чтобы получить функцию политики?

ОТВЕТ

Policy(s) = argmax_a Q(s, a)

Элегантный способ рассчитать выгоду определенной пары «состояние — действие» (s, a) заключается в рекурсивном учете выгод будущих действий. На выгоду текущего действия влияет не только ближайшая награда, но и наилучшие действия (см. следующую формулу). В этой формуле s' обозначает следующее состояние, а a' — следующее действие. Награда за выполнение действия a в состоянии s обозначается $r(s, a)$:

$$Q(s, a) = r(s, a) + \gamma \max Q(s', a').$$

Здесь γ — гиперпараметр, который следует выбрать и который называют *дисконтирующим множителем* (discount factor). Если γ равен 0, агент выбирает действие, которое приводит к максимальной ближайшей награде. Более высокие значения γ заставят агента придать больше значения учету долгосрочных последствий. Эту формулу можно прочитать следующим образом: «выгодой этого действия является ближайшая награда при условии выполнения этого действия, добавленная к произведению дисконтирующего множителя на максимальную награду, которая возникла после этого».

Оценка будущих наград относится к одному из тех гиперпараметров, которыми можно манипулировать, но есть также и другой гиперпараметр. В некоторых вариантах использования обучения с подкреплением ставшая доступной новая информация может оказаться важнее, чем архивные записи, и наоборот. Например, если робота пытаются обучить выполнять задания быстро, но не обязательно оптимально, то можно задать более высокую скорость обучения. Или, если роботу дали больше времени, чтобы изучить новые действия и варианты использования старых действий, скорость обучения придется уменьшить. Обозначим скорость обучения через α и изменим функцию выгоды (обратите внимание на то, что, когда $\alpha = 1$, уравнения становятся идентичными):

$$Q(s, a) \leftarrow Q(s, a) + \alpha\,(r(s, a) + \gamma \max Q(s', a') - Q(s, a)).$$

Обучение с подкреплением может быть выполнено, если известна Q-функция, $Q(s, a)$. *Нейронные сети* (глава 7) представляют собой способ аппроксимации функций, если имеется достаточно обучающих данных. TensorFlow служит превосходным инструментом для работы с нейронными сетями, так как содержит множество важных алгоритмов, упрощающих использование нейронных сетей.

8.2. Применение обучения с подкреплением

Для использования обучения с подкреплением необходимо определить способ получения награды после того, как выполнено определенное действие из определенного состояния. Биржевой трейдер без труда выполняет эти требования, потому что покупка и продажа ценных бумаг меняет состояние брокера (остаток денежных средств), при этом каждое действие обеспечивает награду или приводит к потере.

Состояния в этой ситуации представляют собой вектор, содержащий информацию о текущем состоянии бюджета, текущем объеме ценных бумаг и недавнюю историю цен ценных бумаг (последние 200 курсов акций). Каждое состояние является 202-мерным вектором.

УПРАЖНЕНИЕ 8.2

Назовите возможные недостатки использования обучения с подкреплением для задачи по покупкам и продаже акций.

ОТВЕТ

Выполняя действия на рынке, такие как покупка или продажа акций, вы можете оказать влияние на рынок, в результате чего он резко изменится по сравнению с вашими данными обучения.

Проще говоря, на фондовом рынке есть только три действия: покупка, продажа и удержание ценных бумаг.

- Покупка акций по текущей цене уменьшает бюджет, одновременно увеличивая запас ценных бумаг.
- Продажа акций превращает их в деньги по текущему курсу.
- Удержание не производит ни того ни другого. Это действие подразумевает выжидание в течение определенного периода времени и не приносит никакой награды.

На рис. 8.5 представлена возможная политика при наличии данных фондового рынка.

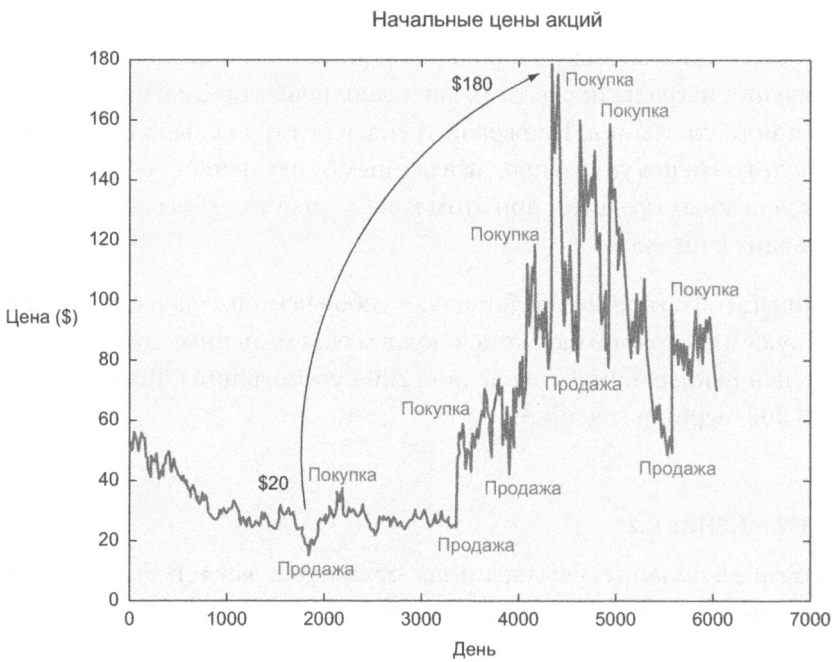


Рис. 8.5. В идеале наш алгоритм должен выполнять операции покупок при низких ценах и операции продаж при высоких. Однократное выполнение этой рекомендации может принести награду приблизительно в 160 долларов. Но реальная прибыль получается, если покупки и продажи выполнять чаще. Вы когда-нибудь слышали термин *высокочастотная торговля*? Предполагается покупать при низких ценах и продавать при высоких настолько часто, насколько это возможно, чтобы обеспечить максимально возможную прибыль за определенный период времени

Цель состоит в том, чтобы найти политику, применение которой приносит максимальную чистую прибыль на фондовом рынке. Разве это не круто? Давайте сделаем это!

8.3. Реализация обучения с подкреплением

Чтобы собрать курсы ценных бумаг, воспользуйтесь библиотекой yahoo_finance на Python. Ее можно установить с помощью оператора pip или следовать официальной инструкции (https://pypi.python.org/pypi/yahoo-finance). Команда для установки с помощью оператора pip выглядит следующим образом:

```
$ pip install yahoo-finance
```

После установки этой библиотеки следует импортировать все необходимые для дальнейшей работы библиотеки.

Листинг 8.1. Импортирование необходимых библиотек

```
from yahoo_finance import Share
from matplotlib import pyplot as plt
import numpy as np
import tensorflow as tf
import random
```

Для получения исходных данных биржевых цен

Для построения графика биржевых акций

Для числовых операций и машинного обучения

Напишите вспомогательную функцию для получения биржевых курсов с помощью библиотеки yahoo_finance. Этой библиотеке необходимо предоставить информацию, содержащую три параметра: символ акции, даты начала и окончания. Когда вы выберете каждый из этих параметров, получится список чисел, представляющий курс акций в этот период по дням.

Если выбрать даты начала и окончания, значительно отстоящие друг от друга, то для выборки этих данных потребуется некоторое время. Возможно, лучше сохранить эти данные на диск (то есть в кэш), чтобы их можно было в следующий раз загружать локально. В следующем листинге видно, как можно использовать эту библиотеку и записать эти данные в кэш.

Листинг 8.2. Вспомогательная функция для получения биржевых цен

```
def get_prices(share_symbol, start_date, end_date,
               cache_filename='stock_prices.npy'):
   try:  ◄── Попытка загрузить данные из файла, если они уже были вычислены
      stock_prices = np.load(cache_filename)
   except IOError:
      share = Share(share_symbol)  ◄── Извлекает курсы акций из библиотеки
      stock_hist = share.get_historical(start_date, end_date)
      stock_prices = [stock_price['Open'] for stock_price in stock_hist]
      np.save(cache_filename, stock_prices)  ◄── Запись результатов в кэш

   return stock_prices.astype(float)
```

Извлекает из исходных данных только необходимую информацию

Для проверки работоспособности можно попробовать визуализировать данные биржевых курсов. Построить график и сохранить его на диске.

Листинг 8.3. Вспомогательная функция для построения графика биржевого курса

```
def plot_prices(prices):
   plt.title('Opening stock prices')
   plt.xlabel('day')
   plt.ylabel('price ($)')
   plt.plot(prices)
   plt.savefig('prices.png')
   plt.show()
```

Вы можете получить некоторые данные и визуализировать их, используя код из следующего листинга.

Листинг 8.4. Отбор части данных для визуального представления

```
if __name__ == '__main__':
   prices = get_prices('MSFT', '1992-07-22', '2016-07-22')
   plot_prices(prices)
```

На рис. 8.6 показан график, полученный в результате выполнения кода из листинга 8.4.

Большинство алгоритмов обучения с подкреплением следуют аналогичной схеме применения. Поэтому можно попробовать создать класс соответствующих методов, чтобы на него можно было впоследствии ссылаться, наподобие абстрактного класса или интерфейса. В качестве примера можно обратиться

к следующему листингу и к рис. 8.7. Для обучения с подкреплением необходимы две четко определенные операции: выбор действия и улучшение функции прибыли или Q-функции.

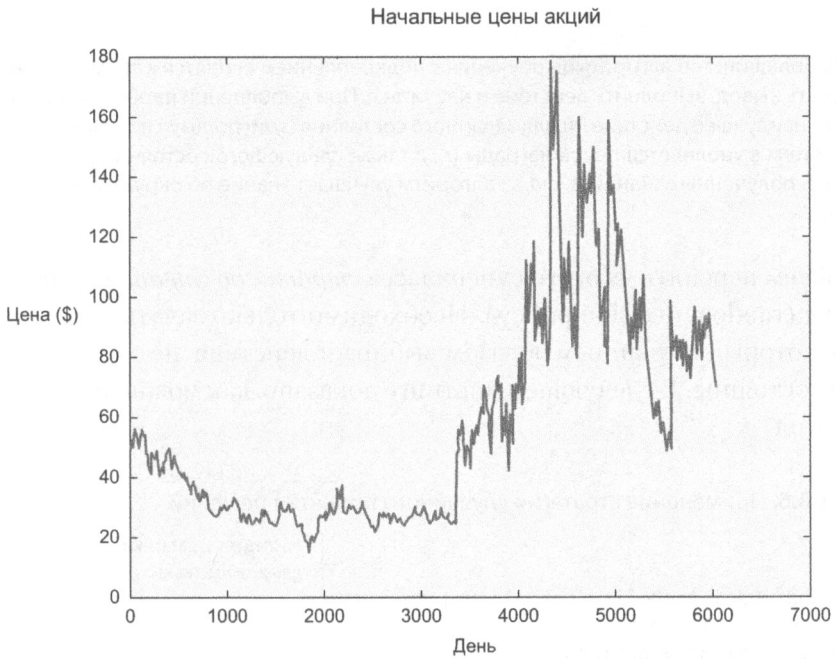


Рис. 8.6. Этот график представляет биржевые курсы акций Microsoft (MSFT) с 22 июля 1992 года по 22 июля 2016 года. Не правда ли, было бы здорово покупать за 3000 и продавать весь день за 5000? Давайте посмотрим, можно ли обучить наш код покупать, продавать и удерживать акции для получения оптимальной прибыли

Листинг 8.5. Задает суперкласс для всех стратегий принятия решений

```
class DecisionPolicy:
   def select_action(self, current_state):
      pass

   def update_q(self, state, action, reward, next_state):
      pass
```

Для заданного состояния используемая стратегия принятия решений определит следующее действие

Улучшение Q-функции из нового опыта или выполнение определенного действия

Infer(s) => a

Do(s, a) => r, s'

Learn(s, r, a, s')

Рис. 8.7. Большинство алгоритмов обучения с подкреплением сводятся к трем основным шагам: сделать вывод, выполнить действие и научиться. При выполнении первого шага алгоритм выбирает наилучшее действие (a) для заданного состояния (s), используя полученные до этого знания. Затем выполняется поиск награды (r), а также следующего состояния (s'). С помощью вновь полученных знаний (s, r, a, s') алгоритм улучшает знание об окружающем мире

Попробуем перенять из этого суперкласса *стратегию случайного принятия решений* (random decision policy). Необходимо только задать метод `select_action`, который случайным образом выбирает действия, не обращая внимания на состояние. В следующем листинге показано, как можно использовать этот метод.

Листинг 8.6. Применение стратегии случайного принятия решений

```
class RandomDecisionPolicy(DecisionPolicy):
   def  init (self, actions):
      self.actions = actions

   def select_action(self, current_state):
      action = random.choice(self.actions)
      return action
```

Наследует стратегию принятия решений для использования ее функций

Случайным образом выбирается следующее действие

В листинге 8.7 предполагается заданная стратегия (аналогичная приведенной в листинге 8.6), которая выполняется на реальных данных курсов акций. Эта функция учитывает изучение ситуации и применение полученных знаний в каждый интервал времени. На рис. 8.8 представлен алгоритм из листинга 8.7.

Для получения более надежных оценок успеха дважды запустим процесс моделирования и усредним результаты. Это займет некоторое время (возможно, 5 минут), но полученные результаты будут надежнее.

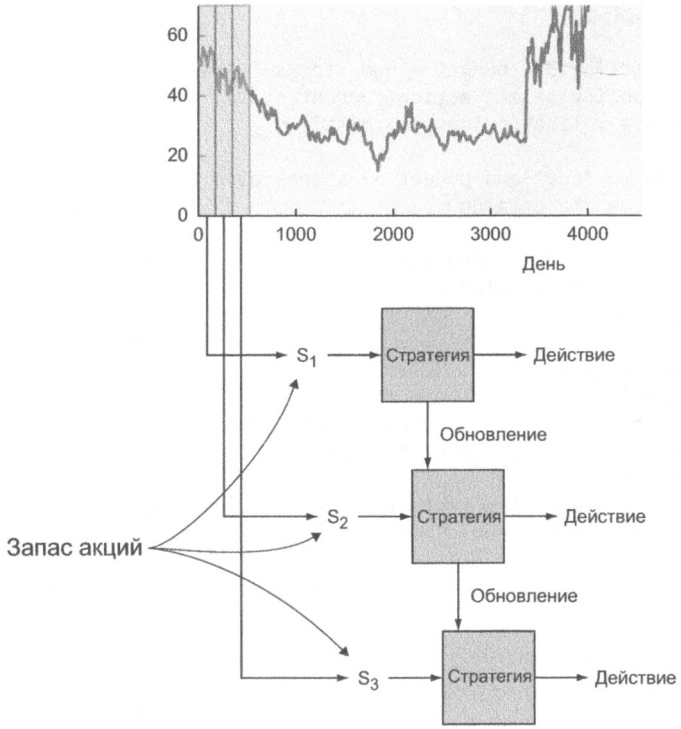


Рис. 8.8. Скользящее окно определенного размера итеративно проходит по курсам акций, как это показано на графике, сегментированном на части для выделения состояний S_1, S_2 и S_3. Используемая стратегия предлагает одно из действий: продолжить использовать уже выполненное действие или случайным образом выбрать другое действие. По мере получения награды за выполненное действие стратегию можно обновлять

Листинг 8.7. Использование заданной стратегии принятия решений и возвращение эффективности

Это состояние является вектором размерности hist + 2. Следует превратить его в матрицу NumPy

```
def run_simulation(policy, initial_budget, initial_num_stocks, prices, hist):
    budget = initial_budget
    num_stocks = initial_num_stocks
    share_value = 0
    transitions = list()
    for i in range(len(prices) - hist - 1):
        if i % 1000 == 0:
            print('progress {:.2f}%'.format(float(100*i) / (len(prices) -
    hist - 1)))
        current_state = np.asmatrix(np.hstack((prices[i:i+hist],
    budget, num_stocks)))
```

Инициализируются значения, которые зависят от полученной оценки чистой прибыли портфеля ценных бумаг

Вычисляет стоимость акций

```
    current_portfolio = budget + num_stocks * share_value
    action = policy.select_action(current_state, i)
    share_value = float(prices[i + hist])

    if action == 'Buy' and budget >= share_value:
       budget -= share_value
       num_stocks += 1
    elif action == 'Sell' and num_stocks > 0:
       budget += share_value
       num_stocks -= 1
    else:
       action = 'Hold'
    new_portfolio = budget + num_stocks * share_value
    reward = new_portfolio - current_portfolio
    next_state = np.asmatrix(np.hstack((prices[i+1:i+hist+1], budget,
  num_stocks)))
    transitions.append((current_state, action, reward, next_state))
    policy.update_q(current_state, action, reward, next_state)

  portfolio = budget + num_stocks * share_value
  return portfolio
```

Выбирает действие исходя из текущей стратегии

Обновляет стоимость акций на основании определенного действия

Обновляет стратегию после испытания нового действия

Оценивает окончательную стоимость акций

Оценивает награду в результате выполненного действия в заданном состоянии

Вычисляет новую стоимость акций после выполненного действия

Листинг 8.8. Многократное выполнение процесса имитационного моделирования для получения усредненной эффективности

```
def run_simulations(policy, budget, num_stocks, prices, hist):
   num_tries = 10
   final_portfolios = list()
   for i in range(num_tries):
      final_portfolio = run_simulation(policy, budget, num_stocks, prices,
    hist)
      final_portfolios.append(final_portfolio)
      print('Final portfolio: ${}'.format(final_portfolio))
   plt.title('Final Portfolio Value')
   plt.xlabel('Simulation #')
   plt.ylabel('Net worth')
   plt.plot(final_portfolios)
   plt.show()
```

Примается решение, сколько раз повторять моделирование

Сохраняет доходность каждого запуска в этом массиве

Выполняет этот алгоритм

В функции `main` добавьте следующие строки, чтобы определить политику принятия решений и выполнить моделирование. Таким образом вы увидите, как эта стратегия работает.

Листинг 8.9. Определяет стратегию принятия решений

```
if __name__ == '__main__':
    prices = get_prices('MSFT', '1992-07-22', '2016-07-22')
    plot_prices(prices)
    actions = ['Buy', 'Sell', 'Hold']
    hist = 3
    policy = RandomDecisionPolicy(actions)
    budget = 100000.0
    num_stocks = 0
    run_simulations(policy, budget, num_stocks, prices, hist)
```

Определяет перечень действий, которые может предпринять агент

Инициализирует стратегию случайного принятия решений

Определяет число уже имеющихся акций

Многократно выполняет процесс моделирования для вычисления ожидаемой окончательной величины чистой прибыли

Определяет начальную сумму денежных средств, которые можно использовать

Теперь, когда есть базовая линия для сравнения полученных результатов, воспользуемся методом нейронных сетей для получения в процессе обучения Q-функции. Стратегию принятия решений часто называют *Q-обучением*. Листинг 8.10 содержит новый гиперпараметр, `epsilon`, для исключения возможности «заедания» хода решения при многократном повторном выполнении одного и того же действия. Чем меньше значение этого параметра, тем чаще он будет случайным образом исследовать новые действия. Q-функция задается функцией, представленной на рис. 8.9.

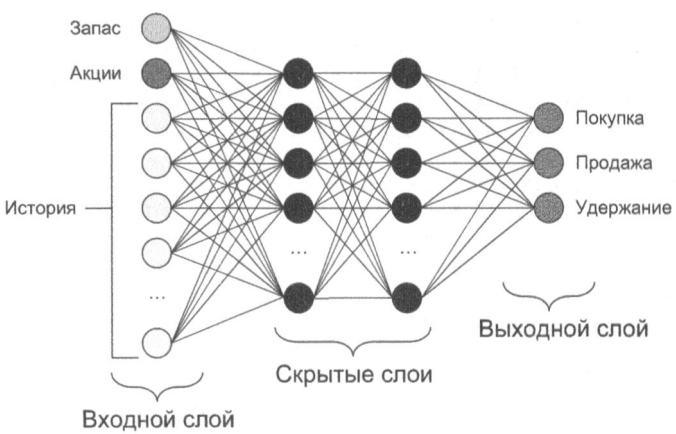


Рис. 8.9. Входом является вектор пространства состояний, все три выхода выдают по одному из значений выгод

УПРАЖНЕНИЕ 8.3

Существуют ли другие возможные факторы, которые игнорируются в представлении пространства состояний и которые могут влиять на курсы акций? Как можно их учитывать в моделировании?

ОТВЕТ

Цены на акции зависят от множества факторов, в том числе от общих тенденций рынка, новостей и конкретных отраслевых тенденций. Каждый из них, после количественного определения, может быть применен в качестве дополнительных измерений к модели.

Листинг 8.10. Применение более продуманной стратегии принятия решений

```
class QLearningDecisionPolicy(DecisionPolicy):
    def __init__(self, actions, input_dim):
        self.epsilon = 0.95
        self.gamma = 0.3
        self.actions = actions
        output_dim = len(actions)
        h1_dim = 20

        self.x = tf.placeholder(tf.float32, [None, input_dim])
        self.y = tf.placeholder(tf.float32, [output_dim])
        W1 = tf.Variable(tf.random_normal([input_dim, h1_dim]))
        b1 = tf.Variable(tf.constant(0.1, shape=[h1_dim]))
        h1 = tf.nn.relu(tf.matmul(self.x, W1) + b1)
        W2 = tf.Variable(tf.random_normal([h1_dim, output_dim]))
        b2 = tf.Variable(tf.constant(0.1, shape=[output_dim]))
        self.q = tf.nn.relu(tf.matmul(h1, W2) + b2)

        loss = tf.square(self.y - self.q)
        self.train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)
        self.sess = tf.Session()
        self.sess.run(tf.global_variables_initializer())

    def select_action(self, current_state, step):
        threshold = min(self.epsilon, step / 1000.)
        if random.random() < threshold:
```

Определяет оператор для вычисления выгоды

Задает гиперпараметры из Q-функции

Задает число скрытых узлов в нейронных сетях

Задает входные и выходные тензоры

Проектирует архитектуру нейронной сети

Определяет сеанс и инициализирует все переменные

Определяет потери в виде квадрата ошибки

Используется наилучший из вариантов с вероятностью ε

Использует процедуру оптимизации для обновления параметров модели для минимизации потерь

```
            # Exploit best option with probability epsilon
        action_q_vals = self.sess.run(self.q, feed_dict={self.x:
current_state})
        action_idx = np.argmax(action_q_vals)
        action = self.actions[action_idx]
    else:  ◄—— Исследует случайный вариант с вероятностью 1 — ε
        # Explore random option with probability 1 - epsilon
        action = self.actions[random.randint(0, len(self.actions) - 1)]
    return action

def update_q(self, state, action, reward, next_state):
    action_q_vals = self.sess.run(self.q, feed_dict={self.x: state})
    next_action_q_vals = self.sess.run(self.q, feed_dict={self.x:
next_state})
    next_action_idx = np.argmax(next_action_q_vals)
    current_action_idx = self.actions.index(action)
    action_q_vals[0, current_action_idx] = reward + self.gamma *
next_action_q_vals[0, next_action_idx]
    action_q_vals = np.squeeze(np.asarray(action_q_vals))
    self.sess.run(self.train_op, feed_dict={self.x: state, self.y:
action_q_vals})
```

В Q-функции выполняется обновление параметров ее модели

Выходные данные при выполнении всего скрипта приводятся на рис. 8.10.

Рис. 8.10. Этот алгоритм обучает оптимальной стратегии для торговли акциями Microsoft

8.4. Исследование других областей использования обучения с подкреплением

Обучение с подкреплением применяется чаще, чем вы можете ожидать. Можно забыть о том, что оно существует, если вы научились использовать обучение с учителем или без. Но следующие примеры откроют вам глаза на успешное использование обучения с подкреплением компанией *Google*:

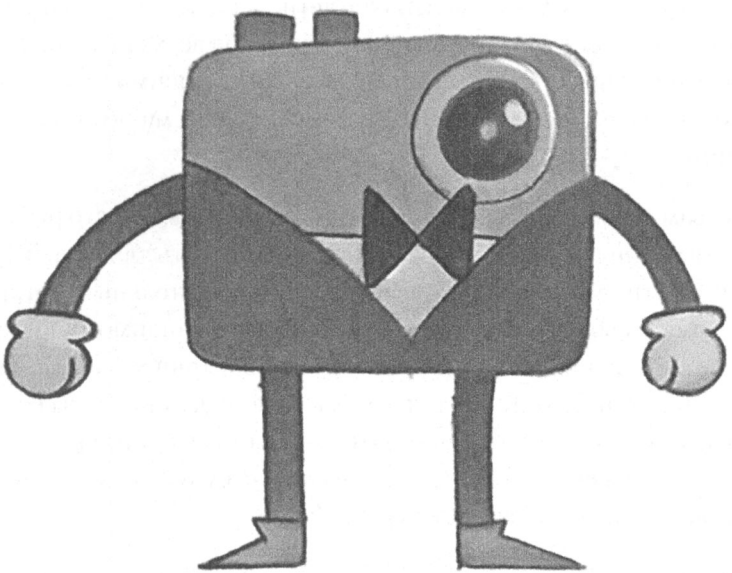
- *Ведение игры.* В феврале 2015 года *Google* разработала систему обучения с подкреплением, получившую название Deep RL, для обучения игры на консоли Atari 2600. В отличие от большинства вариантов обучения с подкреплением, этот алгоритм имеет вход высокой размерности: он по кадрам воспринимает исходные изображения видеоигры. Таким образом, один и тот же алгоритм может работать с любыми видеоиграми без значительного перепрограммирования или изменения конфигурации.
- *Ведение большего числа игр.* В январе 2016 года *Google* опубликовала статью об агенте искусственного интеллекта, способном выигрывать в игре Го. Эта игра известна как непредсказуемая из-за огромного числа возможных конфигураций (даже большего, чем в шахматах!). Этот алгоритм, использующий обучение с подкреплением, мог обыгрывать лучших игроков в Го. Последняя версия алгоритма, AlphaGo Zero, выпущенная в конце 2017 года, была способна неизменно обыгрывать более раннюю версию со счетом 100:0 всего лишь за 40 дней тренировок. И алгоритм будет играть значительно лучше к тому времени, когда вы прочтете эти строки.
- *Робототехника и управление.* В марте 2016 года *Google* продемонстрировала способ обучения роботов, использующий множество примеров, как правильно захватить объект. *Google* собрала более 800 000 попыток захвата и удержания объекта, используя множество роботов, и разработала модель для захвата произвольных объектов. Впечатляет то, что роботы были способны захватывать объект с помощью одних только данных, получаемых от видеокамеры. Обучение простому действию ухватить и удерживать объект требует объединения опыта многих роботов, пытавшихся в течение многих дней методом простого перебора возможных вариантов добиться достаточного числа удачных шаблонов. Понятно, что роботам необходимо было пройти длинный путь, чтобы обобщить полученный опыт, но все же это представляется интересным началом.

ПРИМЕЧАНИЕ А теперь, когда вы применили обучение с подкреплением на фондовом рынке, бросайте школу или увольняйтесь с работы, чтобы начать играть на бирже. Это ваше вознаграждение, дорогой читатель, за то, что вы так далеко зашли в чтении этой книги! Я, конечно же, шучу, ведь реальный фондовый рынок значительно более сложный зверь. Однако методы, описанные в этой главе, применимы ко многим ситуациям в реальном мире.

8.5. Краткие итоги

- Обучение с подкреплением является основным инструментом решения задач, которые можно ограничить состояниями, изменяющимися в результате действий, предпринятых агентом для получения награды.
- Алгоритмы обучения с подкреплением содержат три основных шага: заключение о лучшем действии на основе текущего состояния, выполнение этого действия и обучение исходя из полученных результатов.
- Q-обучение является методом обучения с подкреплением посредством разработки алгоритма аппроксимации функции выгоды (Q-функции). После того как найдена достаточно хорошая аппроксимация, можно делать заключение о наилучших действиях, которые следует выполнить на основе каждого состояния.

Сверточные нейронные сети



Эта глава охватывает следующие темы:

- ✓ Проверка компонентов сверточных нейронных сетей
- ✓ Классификация естественных изображений с помощью глубокого обучения
- ✓ Повышение эффективности нейронных сетей: советы и рекомендации

Покупки в магазинах после изнурительного дня — весьма обременительное занятие. Мои глаза атакует слишком большой объем информации. Распродажи, купоны, разнообразные цвета, маленькие дети, мерцающие огни и заполненные людьми проходы — вот только несколько примеров всех сигналов, которые направлены в зрительную кору головного мозга, независимо от того, хочу я или не хочу обращать на это внимание. Визуальная система поглощает изобилие информации.

Наверняка вам знакома фраза «лучше один раз увидеть, чем сто раз услышать». Это может быть справедливо для вас и для меня (то есть для людей), но сможет ли машина найти смысл в изображениях? Наши зрительные фоторецепторы подбирают длины волн света, но эта информация, по-видимому, не распространяется на наше сознание. В конце концов, я не могу точно сказать, какие длины световых волн наблюдаю. Точно так же камера получает пикселы изображения. Но мы хотим вместо этого получать что-то более высокого уровня, например названия или положения объектов. Как мы получаем из пикселов информацию, воспринимаемую на человеческом уровне?

Для получения определенного смысла из исходных данных потребуется спроектировать модель нейронной сети. В предыдущих главах было представлено несколько типов моделей нейронной сети, таких как полносвязные модели (глава 8) и автокодировщики (глава 7). В этой главе мы познакомимся с другим типом моделей, который называется *сверточная нейронная сеть*

(CNN – convolutional neural network). Эта модель отлично работает с изображениями и другими сенсорными данными, такими как звук. Например, модель CNN может надежно классифицировать, какой объект отображается на картинке.

Модель CNN, которая будет рассмотрена в этой главе, будет обучена классифицировать изображения по одной из 10 возможных категорий. В данном случае «картинка лучше только *одного* слова», так как у нас всего 10 возможных вариантов. Это крошечный шаг к восприятию на человеческом уровне, но с чего-то нам надо начинать, верно?

9.1. Недостатки нейронных сетей

Машинное обучение представляет собой вечную борьбу за разработку модели, которая обладала бы достаточной выразительностью для представления данных, но при этом не была такой универсальной, чтобы доходить до переобученности и запоминать паттерны. Нейронные сети предлагаются как способ повышения выразительности; хотя, как можно догадаться, они сильно страдают от ловушек переобучения.

> **ПРИМЕЧАНИЕ** Переобучение возникает, когда обученная модель исключительно точна на обучающем наборе данных и плоха на проверочном наборе данных. Эта модель, вероятно, чрезмерно универсальна для небольшого объема доступных данных, и в конце концов она просто запоминает обучающие данные.

Для сравнения универсальности двух моделей машинного обучения вы можете использовать быстрый и грубый эвристический алгоритм, чтобы подсчитать число параметров, которые требуется определить в результате обучения. Как показано на рис. 9.1, полносвязная нейронная сеть, которая берет изображение размером 256 × 256 и отображает его на слой из 10 нейронов, будет иметь 256 × 256 × 10 = 655 360 параметров! Сравните ее с моделью, содержащей только пять параметров. Можно предположить, что полносвязная нейронная сеть может представлять более сложные данные, чем модель с пятью параметрами.

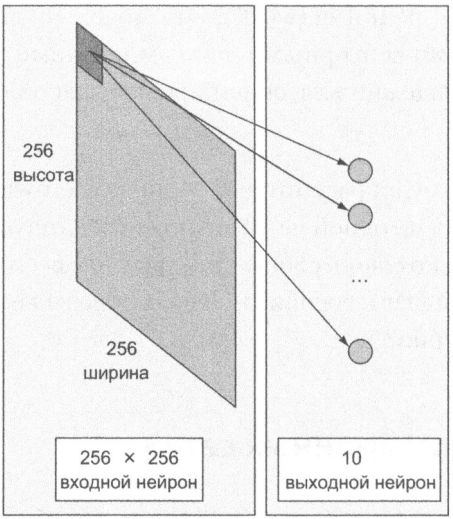


Рис. 9.1. В полносвязной нейронной сети каждый пиксел изображения рассматривается как вход. Для изображения в серых тонах размером 256 × 256 это составит 256 × 256 нейронов! Присоединение каждого нейрона к 10 выходам дает 256 × 256 × 10 = 655 360 весов

В следующем разделе рассматриваются сверточные нейронные сети, которые являются разумным способом снижения числа параметров. Вместо того чтобы заниматься полносвязными сетями, CNN многократно использует повторно те же параметры.

9.2. Сверточные нейронные сети

Главная идея, лежащая в основе сверточных нейронных сетей, состоит в том, что вполне достаточно локального осмысления изображения. Практическое преимущество сверточных нейронных сетей таково, что, имея несколько параметров, можно значительно сократить время на обучение, а также объем данных, необходимых для обучения модели.

Вместо полносвязных сетей с весами от каждого пиксела CNN имеет достаточное число весов, необходимых для просмотра небольшого фрагмента изображения. Это все равно что читать книгу с помощью лупы: в конечном счете вы прочитываете всю страницу, но в любой момент времени смотрите только на небольшой ее фрагмент.

Представьте изображение размером 256 × 256. Вместо использования кода TensorFlow, обрабатывающего все изображение сразу, можно сканировать изображение фрагмент за фрагментом, скажем, окном размером 5 × 5. Окно размером 5 × 5 скользит по изображению (обычно слева направо и сверху вниз), как показано на рис. 9.2. То, как «быстро» оно скользит, называют *длиной шага* (stride length). Например, длина шага 2 означает, что скользящее окно 5 × 5 перемещается на 2 пиксела за раз, пока не пройдет все изображение. В TensorFlow, как будет скоро показано, можно настроить длину шага и размер окна, используя встроенную библиотеку функций.

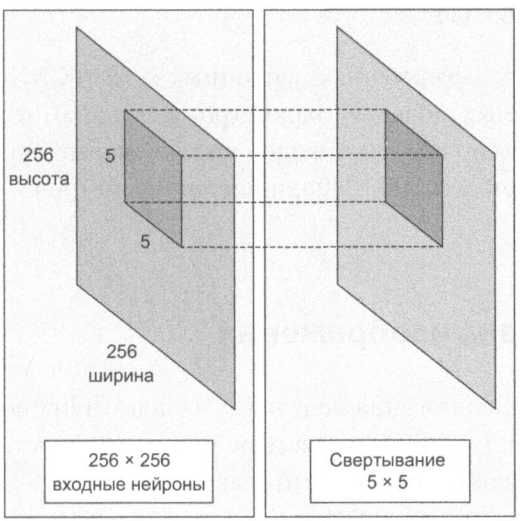


Рис. 9.2. Операция свертки 5 × 5 по изображению, как показано в левой части рисунка, создает другое изображение, как показано в правой части рисунка. В этом случае созданное изображение имеет такой же размер, как исходное. Для преобразования исходного изображения в свернутое изображение требуется всего 5 × 5 = 25 параметров!

Это окно размером 5 × 5 имеет связанную с ним матрицу весов 5 × 5.

ОПРЕДЕЛЕНИЕ *Сверткой* (convolution) называют взвешенное суммирование значений интенсивности пикселов изображения по мере прохождения окна по всему изображению. Оказывается, что этот процесс свертки изображения с матрицей весов создает другое изображение (такого же размера, который зависит от свертывания). *Свертыванием* называют процесс применения свертки.

Все манипуляции скользящего окна происходят в сверточном слое нейронной сети. Типичная сверточная нейронная сеть имеет несколько сверточных слоев. Каждый сверточный слой обычно создает много дополнительных сверток, поэтому матрица весовых коэффициентов является тензором $5 \times 5 \times n$, где n — число сверток.

В качестве примера пусть изображение проходит через сверточный слой с матрицей весовых коэффициентов размером $5 \times 5 \times 64$. Это создает 64 свертки скользящим окном 5×5. Поэтому соответствующая модель имеет $5 \times 5 \times 64 = 1600$ параметров, что значительно меньше числа параметров полносвязной сети: $256 \times 256 = 65\,536$.

Привлекательность сверточных нейронных сетей (CNN) состоит в том, что число используемых моделью параметров не зависит от размера исходного изображения. Можно выполнить одну и ту же сверточную нейронную сеть на изображения размером 300×300, и число параметров в сверточном слое не изменится!

9.3. Подготовка изображения

Перед началом использования модели CNN с TensorFlow подготовим несколько изображений. Листинги в этом разделе помогут вам установить обучающий набор данных для оставшейся части главы.

Прежде всего, следует загрузить набор данных CIFAR-10 с сайта www.cs.toronto.edu/~kriz/cifar-10- python.tar.gz. В этом наборе содержатся 60 000 изображений, равномерно распределенных по 10 категориям, что представляет достаточно большой ресурс для задач классификации. Затем файл с изображениями следует поместить в рабочую директорию. На рис. 9.3 приведены примеры изображений из этого набора данных.

Мы уже использовали набор данных CIFAR-10 в предыдущей главе, посвященной автокодировщикам, и теперь снова рассмотрим этот код. Следующий листинг взят прямо из документации CIFAR-10, находящейся на сайте www.cs.toronto.edu/~kriz/cifar.html. Поместите код в файл cifar_tools.py.

Рис. 9.3. Изображения из набора данных CIFAR-10. Так как все они имеют размер 32 × 32, их сложно рассмотреть, но можно распознать на них некоторые объекты

Листинг 9.1. Загрузка изображений из файла CIFAR-10 в Python

```
import pickle

def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict
```

Нейронные сети предрасположены к переобучению, поэтому важно сделать все возможное для минимизации этой ошибки. Для этого необходимо не забывать выполнять очистку данных перед их обработкой.

Очистка данных является основным процессом конвейера машинного обучения. Приведенный в листинге 9.2 код для чистки набора изображений использует следующие три шага:

1. Если у вас изображение в цвете, попробуйте преобразовать его в оттенки серого, чтобы уменьшить размерность входных данных и, следовательно, уменьшить количество параметров.
2. Подумайте об обрезке изображения по центру, потому что края изображения не предоставляют никакой полезной информации.
3. Нормализуйте входные данные вычитанием средней величины и делением на среднеквадратическое отклонение каждой выборки данных, чтобы градиенты во время обратного распространения не изменялись слишком резко.

В следующем листинге показано, как очистить набор данных с помощью этих методов.

Листинг 9.2. Очистка данных

```
import numpy as np

def clean(data):
    imgs = data.reshape(data.shape[0], 3, 32, 32)
    grayscale_imgs = imgs.mean(1)
    cropped_imgs = grayscale_imgs[:, 4:28, 4:28]
    img_data = cropped_imgs.reshape(data.shape[0], -1)
    img_size = np.shape(img_data)[1]
    means = np.mean(img_data, axis=1)
    meansT = means.reshape(len(means), 1)
    stds = np.std(img_data, axis=1)
    stdsT = stds.reshape(len(stds), 1)
    adj_stds = np.maximum(stdsT, 1.0 / np.sqrt(img_size))
    normalized = (img_data - meansT) / adj_stds
    return normalized
```

Преобразует изображение в оттенки серого осреднением насыщенности цвета

Перестраивает данные в матрицу 32 × 32 с тремя каналами

Обрезает изображение 32 × 32 в изображение 24 × 24

Нормализует интенсивности пикселов путем вычитания среднего и деления на среднеквадратическое отклонение

Сохраните все изображения из набора данных CIFAR-10 и запустите функцию очистки. В следующем листинге задается удобный метод считывания, очистки

и структурирования данных для использования в TensorFlow. Туда же следует включить код из файла cifar_tools.py.

Листинг 9.3. Предварительная обработка всех файлов CIFAR-10

```
def read_data(directory):
    names = unpickle('{}/batches.meta'.format(directory))['label_names']
    print('names', names)

    data, labels = [], []
    for i in range(1, 6):
        filename = '{}/data_batch_{}'.format(directory, i)
        batch_data = unpickle(filename)
        if len(data) > 0:
            data = np.vstack((data, batch_data['data']))
            labels = np.hstack((labels, batch_data['labels']))
        else:
            data = batch_data['data']
            labels = batch_data['labels']

    print(np.shape(data), np.shape(labels))

    data = clean(data)
    data = data.astype(np.float32)
    return names, data, labels
```

В файле using_cifar.py можно использовать метод, импортировав для этого `cifar_tools`. В листингах 9.4 и 9.5 показано, как делать выборку нескольких изображений из набора данных и визуализировать их.

Листинг 9.4. Использование вспомогательной функции `cifar_tools`

```
import cifar_tools

names, data, labels = \
    cifar_tools.read_data('your/location/to/cifar-10-batches-py')
```

Вы можете произвольно выбрать несколько изображений и отрисовать их в соответствии с меткой. Следующий листинг делает именно это, поэтому вы можете лучше понять тип данных, с которыми будете иметь дело.

Листинг 9.5. Визуализация изображений из набора данных

```
import numpy as np
import matplotlib.pyplot as plt
import random

def show_some_examples(names, data, labels):
    plt.figure()
    rows, cols = 4, 4
    random_idxs = random.sample(range(len(data)), rows * cols)
    for i in range(rows * cols):
        plt.subplot(rows, cols, i + 1)
        j = random_idxs[i]
        plt.title(names[labels[j]])
        img = np.reshape(data[j, :], (24, 24))
        plt.imshow(img, cmap='Greys_r')
        plt.axis('off')
    plt.tight_layout()
    plt.savefig('cifar_examples.png')

show_some_examples(names, data, labels)
```

Преобразует изображения до необходимого числа строк и столбцов

Случайным образом выбирает изображения из набора данных, чтобы их можно было показать

Запустив этот код, вы создадите файл cifar_examples.png, который будет похож на рис. 9.3.

9.3.1. Создание фильтров

В этом разделе мы покажем, как выполнять свертку изображения с помощью пары фрагментов окна размером 5 × 5, которые назовем *фильтрами*. Применение фильтров — важный шаг в работе сверточных нейронных сетей, поэтому мы тщательно изучим, как при этом преобразуются данные. Чтобы понять работу модели сверточных нейронных сетей при обработке изображений, проследим, как именно выбранный фильтр преобразует изображение. Фильтры используются для извлечения важных элементов изображения, таких как края и формы. Вы можете обучить модель этим функциям.

Помните: вектор функций указывает на то, как вы представляете точку данных. Когда вы применяете фильтр к изображению, соответствующая точка в преобразованном изображении представляет собой функцию, которая гласит: «Когда вы применяете этот фильтр к этой точке, она теперь имеет это новое значение». Чем больше фильтров вы применяете к изображению, тем большей размерности будет вектор признаков.

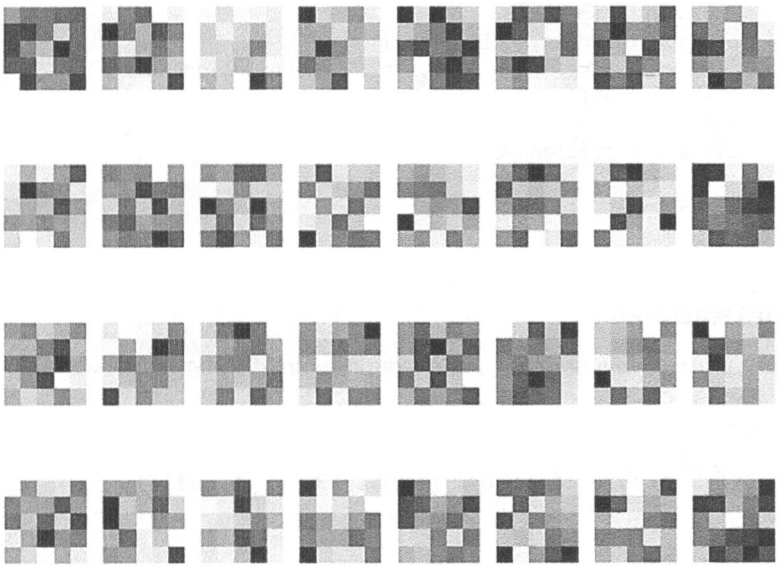


Рис. 9.4. Здесь представлены 32 случайным образом инициализированные матрицы, каждая размером 5 × 5. Это фильтры для выполнения свертки входного изображения

Откроем новый файл conv_visuals.py. Инициализируем случайным образом 32 фильтра. Это можно сделать, задав переменную W размером 5 × 5 × 1 × 32. Первые два числа соответствуют размеру фильтра. Последнее число равно числу сверток (32). Единица в размере переменной соответствует размерности входа, потому что функция conv2d способна выполнять свертку изображений с несколькими входами (в нашем примере внимание уделяется только изображению в градациях серого, поэтому канал один). В следующем листинге показан процесс получения фильтров, которые представлены на рис. 9.4.

Листинг 9.6. Создание и визуализация случайных фильтров

Определяет достаточное число строк и столбцов, чтобы показать 32 фрагмента (рис. 9.4)

Определяет тензор, представляющий случайные фильтры

```
W = tf.Variable(tf.random_normal([5, 5, 1, 32]))

def show_weights(W, filename=None):
    plt.figure()
    rows, cols = 4, 8
    for i in range(np.shape(W)[3]):
```

Визуально представляет каждую матрицу фильтров

```
        img = W[:, :, 0, i]
        plt.subplot(rows, cols, i + 1)
        plt.imshow(img, cmap='Greys_r', interpolation='none')
        plt.axis('off')
    if filename:
        plt.savefig(filename)
    else:
        plt.show()
```

УПРАЖНЕНИЕ 9.1

Измените код в листинге 9.6 для генерирования 64 фильтров размером 3 × 3.

ОТВЕТ

```
W = tf.Variable(tf.random_normal([3, 3, 1, 64]))
```

Используем сеанс, как показано в следующем листинге, и с помощью оператора `global_variables_initializer` инициализируем веса. Затем вызовем функцию `show_weights` для визуализации случайных фильтров (см. рис. 9.4).

Листинг 9.7. Использование сеанса для инициализации весов

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    w_val = sess.run(w)
    show_weights(W_val, 'step0_weights.png')
```

9.3.2. Свертывания с использованием фильтров

В предыдущем разделе мы подготовили фильтры для использования. В этом разделе вы будете использовать функцию свертки TensorFlow для случайно сгенерированных фильтров. В следующем листинге содержится код для визуализации выходов свертки. Вы будете использовать его позже, так же, как вы использовали `show_weight`.

Листинг 9.8. Демонстрация результатов процедуры свертки

```
def show_conv_results(data, filename=None):
    plt.figure()
    rows, cols = 4, 8
    for i in range(np.shape(data)[3]):
        img = data[0, :, :, i]  ←—— Форма тензора отличается от приведенной в листинге 9.6
        plt.subplot(rows, cols, i + 1)
        plt.imshow(img, cmap='Greys_r', interpolation='none')
        plt.axis('off')
    if filename:
        plt.savefig(filename)
    else:
        plt.show()
```

Теперь у нас есть пример входного изображения как на рис. 9.5. Для получения множества свернутых изображений можно выполнить свертку изображения размером 24 × 24, используя фильтры 5 × 5.

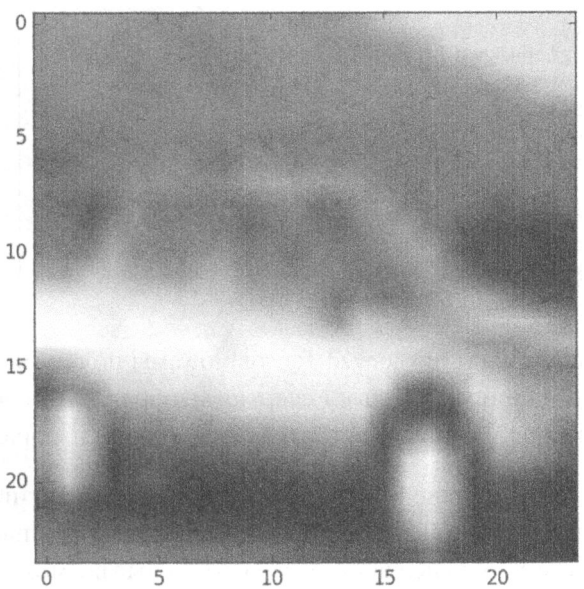


Рис. 9.5. Пример изображения 24 × 24 на основе набора данных CIFAR-10

Все эти свертки — уникальная перспектива взглянуть на одно и то же изображение. Различные перспективы работают совместно для распознавания образа объекта, содержащегося в изображении. Следующий листинг показывает, как это можно сделать шаг за шагом.

Листинг 9.9. Визуализация сверток

```
raw_data = data[4, :]
raw_img = np.reshape(raw_data, (24, 24))
plt.figure()
plt.imshow(raw_img, cmap='Greys_r')
plt.savefig('input_image.png')

x = tf.reshape(raw_data, shape=[-1, 24, 24, 1])

b = tf.Variable(tf.random_normal([32]))
conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
conv_with_b = tf.nn.bias_add(conv, b)
conv_out = tf.nn.relu(conv_with_b)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    conv_val = sess.run(conv)
    show_conv_results(conv_val, 'step1_convs.png')
    print(np.shape(conv_val))

conv_out_val = sess.run(conv_out)
    show_conv_results(conv_out_val, 'step2_conv_outs.png')
    print(np.shape(conv_out_val))
```

Берет изображение из набора данных CIFAR и визуализирует его

Определяет входной тензор для изображения размером 24 × 24

Определяет фильтры и соответствующие параметры

Выполняет свертку выбранного изображения

И наконец, вызвав функцию `conv2d` TensorFlow, мы получим 32 изображения, представленные на рис. 9.6. Смысл свертки изображений состоит в том, что каждая из 32 сверток улавливает различные признаки изображения.

С добавлением члена смещения (bias term) и функции активации, например `relu` (для примера листинг 9.12), сверточный слой сети начинает вести себя нелинейным образом, что повышает его выразительные возможности. На рис. 9.7 показано, каким становится результат каждой из 32 сверток.

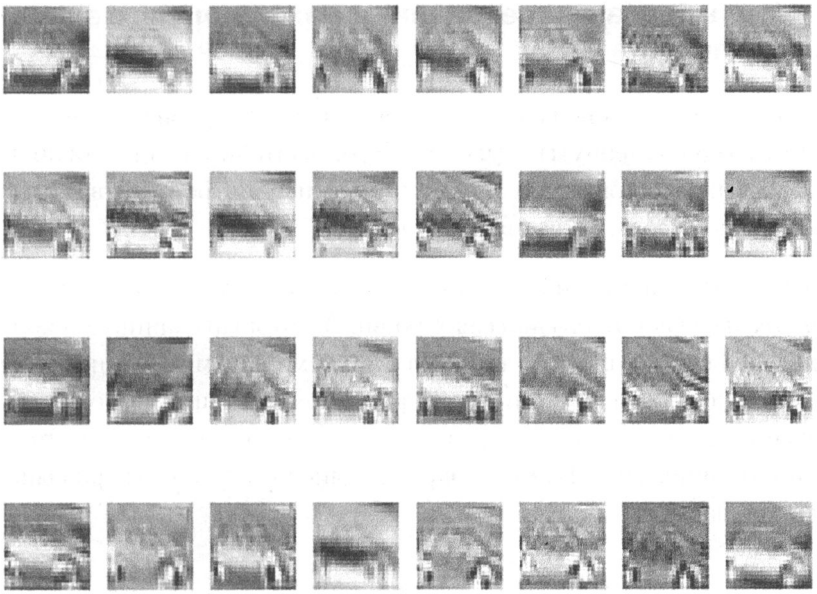


Рис. 9.6. Результирующие изображения, получившиеся в результате свертки случайным образом выбранными фильтрами изображения машины

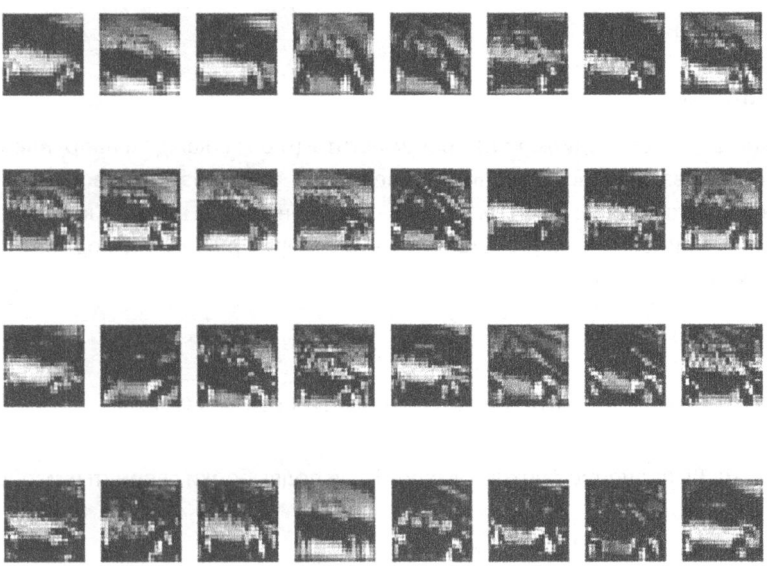


Рис. 9.7. После добавления члена смещения и функции активации результирующие свертки могут выделять на изображении более сложные структуры

9.3.3. Подвыборка с определением максимального значения (max pooling)

После того как сверточный слой извлекает полезные признаки, рекомендуется уменьшить размер свернутых выходов. Перемасштабирование или подвыборка свернутых выходов помогает сократить число параметров, что, в свою очередь, дает возможность исключить переобучение.

В этом состоит основная идея, лежащая в основе метода *подвыборки с определением максимального значения* (max-pooling), который сканирует скользящим окном изображение и выбирает пиксел с максимальным значением интенсивности. В зависимости от длины шага размер результирующего изображения уменьшается, составляя небольшую часть исходного изображения. Это снижает размерность данных и тем самым сокращает число параметров при выполнении следующих шагов.

УПРАЖНЕНИЕ 9.2

Допустим, мы хотим применить подвыборку с определением максимального значения изображения размером 32 × 32. Если размер окна 2 × 2, а длина шага 2, насколько большим будет получившееся после подвыборки изображение?

ОТВЕТ

Окно 2 × 2 необходимо будет переместить 16 раз в каждом направлении, чтобы охватить изображение размером 32 × 32, поэтому изображение уменьшится вдвое: 16 × 16. Так как оно уменьшится вдвое в каждом направлении, его размер составит одну четверть размера исходного изображения (½ × ½).

Поместите следующий листинг внутри контекста `Session`.

Листинг 9.10. Выполнение функции `maxpool` для подвыборки свернутых изображений

```
k = 2
maxpool = tf.nn.max_pool(conv_out,
                         ksize=[1, k, k, 1],
                         strides=[1, k, k, 1],
```

```
                                padding='SAME')

with tf.Session() as sess:
    maxpool_val = sess.run(maxpool)
    show_conv_results(maxpool_val, 'step3_maxpool.png')
    print(np.shape(maxpool_val))
```

В результате выполнения этого кода функция `maxpool` вдвое уменьшает размер изображения и создает свернутые выходы низкого разрешения, как показано на рис. 9.8.

У нас есть инструменты, необходимые для реализации полностью свернутой нейронной сети. В следующем разделе мы, наконец, займемся обучением классификатора изображений.

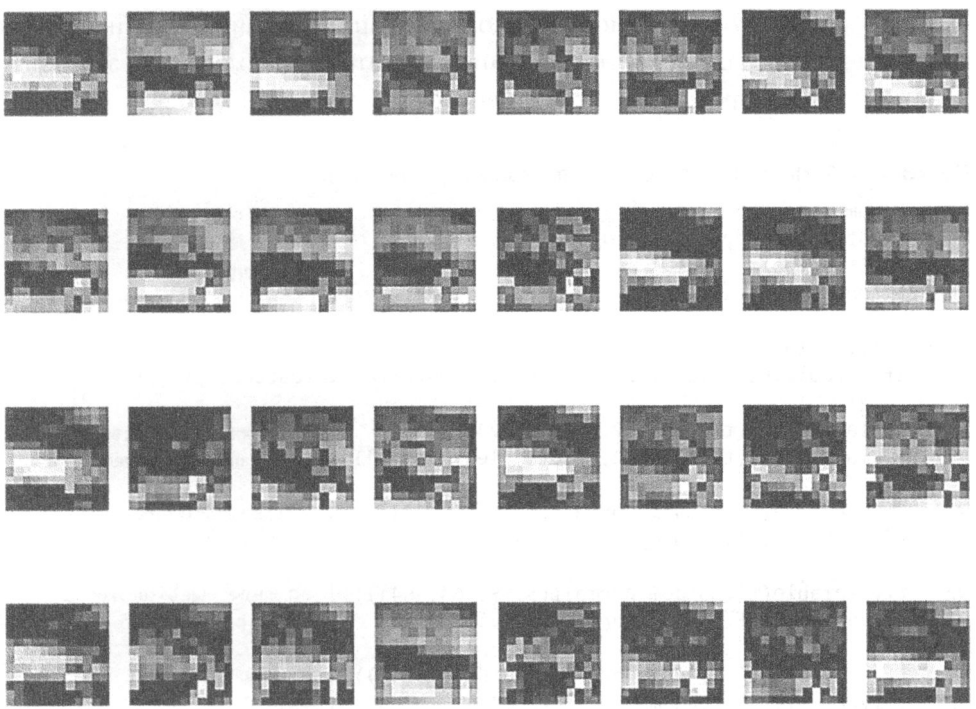


Рис. 9.8. После выполнения работы функции maxpool свернутые выходы уменьшаются вдвое, позволяя алгоритму быстрее выполнять свою работу без потери значительного объема информации

9.4. Использование сверточной нейронной сети в TensorFlow

Сверточная нейронная сеть состоит из нескольких слоев сверток и подвыборок. Слой свертки дает различные представления изображения, тогда как подвыборка слоя упрощает расчеты благодаря снижению размерности без значительной потери информации.

Рассмотрим изображение размером 256 × 256, которое было свернуто фильтром размером 5 × 5 в 64 свертки. Как показано на рис. 9.9, свертка подвергается подвыборке, чтобы в результате получить 64 свернутых изображения размером 128 × 128.

Теперь, когда вы знаете, как создавать фильтры и использовать операцию свертки, давайте создадим новый исходный файл. Вы начнете с объявления всех переменных. В листинге 9.11 импортируйте все библиотеки, загрузите набор данных и определите все переменные.

Листинг 9.11. Настройка весов сверточной нейронной сети

```
import numpy as np
import matplotlib.pyplot as plt
import cifar_tools
import tensorflow as tf

names, data, labels = \
    cifar_tools.read_data('/home/binroot/res/cifar-10-batches-py')

x = tf.placeholder(tf.float32, [None, 24 * 24])
y = tf.placeholder(tf.float32, [None, len(names)])

W1 = tf.Variable(tf.random_normal([5, 5, 1, 64]))
b1 = tf.Variable(tf.random_normal([64]))

W2 = tf.Variable(tf.random_normal([5, 5, 64, 64]))
b2 = tf.Variable(tf.random_normal([64]))

W3 = tf.Variable(tf.random_normal([6*6*64, 1024]))
b3 = tf.Variable(tf.random_normal([1024]))

W_out = tf.Variable(tf.random_normal([1024, len(names)]))
b_out = tf.Variable(tf.random_normal([len(names)]))
```

Загружает набор данных

Определяет входные и выходные переменные-заполнители

Использует 64 свертки с окном размером 5 × 5

Использует еще 64 свертки с окном размером 5 × 5

Вводит полносвязный слой

Определяет переменные для полносвязного линейного слоя

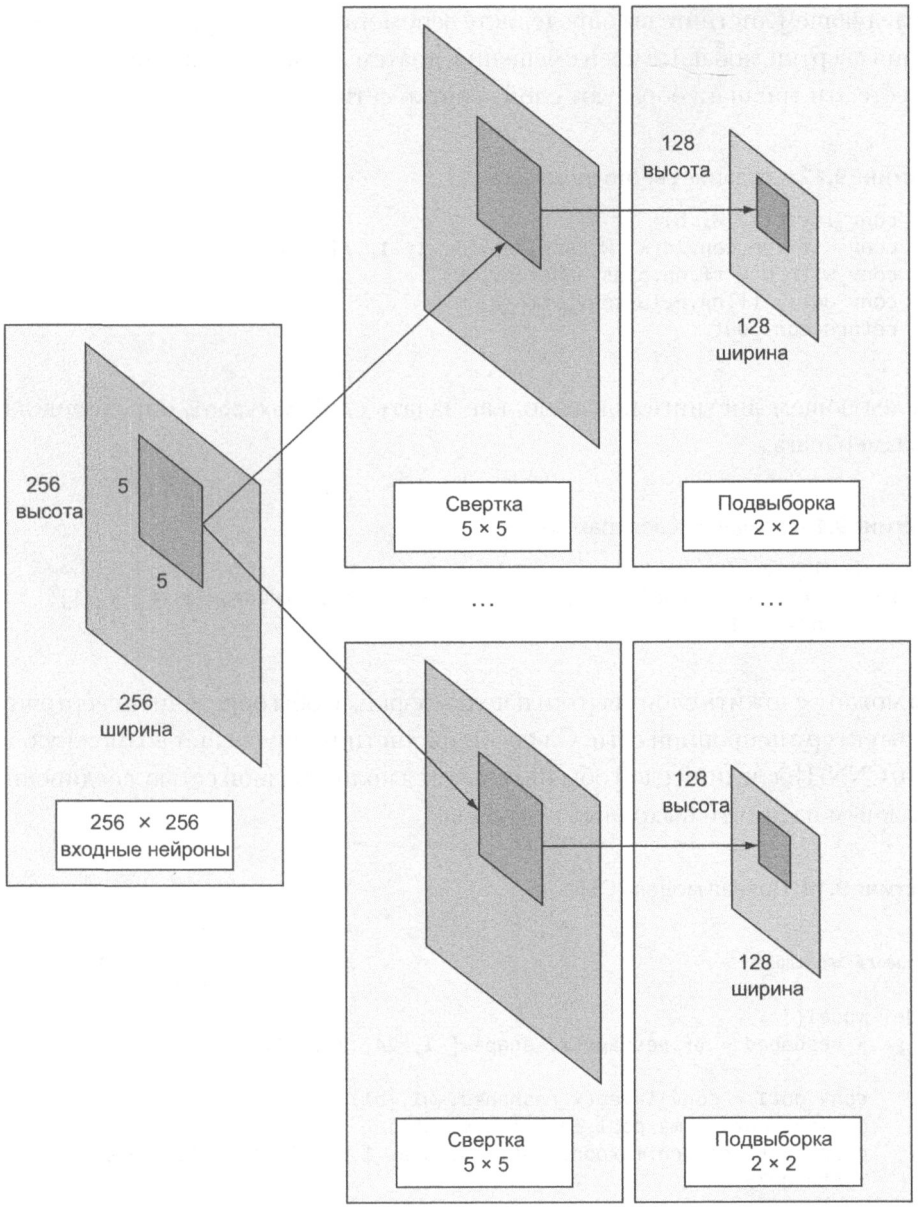


Рис. 9.9. Входное изображение, свернутое несколькими фильтрами размером 5 × 5. Слой свертки включает в себя дополнительный член смещения с функцией активации, что дает 5 × 5 + 5 = 30 параметров. Затем с помощью подвыборки уменьшается размерность данных (для этого не нужны дополнительные параметры)

В следующем листинге вы определиете вспомогательную функцию для выполнения свертки, добавите член смещения, а затем добавите функцию активации. Вместе эти три шага образуют слой свертки сети.

Листинг 9.12. Создание сверточного слоя

```
def conv_layer(x, W, b):
    conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
    conv_with_b = tf.nn.bias_add(conv, b)
    conv_out = tf.nn.relu(conv_with_b)
    return conv_out
```

В следующем листинге показано, как задать слой `max-pool`, определив ядро и размер шага.

Листинг 9.13. Создание слоя max-pool

```
def maxpool_layer(conv, k=2):
    return tf.nn.max_pool(conv, ksize=[1, k, k, 1], strides=[1, k, k, 1],
     padding='SAME')
```

Вы можете сложить слои свертки и подвыборки, чтобы определить сверточную архитектуру нейронной сети. Следующий листинг определяет возможную модель CNN. Последний слой обычно является полносвязной сетью, соединенной с каждым из десяти выходных нейронов.

Листинг 9.14. Полная модель CNN

Строит второй слой

Строит первый слой свертки и подвыборки с определением максимального значения

```
def model():
    x_reshaped = tf.reshape(x, shape=[-1, 24, 24, 1])

    conv_out1 = conv_layer(x_reshaped, W1, b1)
    maxpool_out1 = maxpool_layer(conv_out1)
    norm1 = tf.nn.lrn(maxpool_out1, 4, bias=1.0, alpha=0.001 / 9.0,
     beta=0.75)

    conv_out2 = conv_layer(norm1, W2, b2)
    norm2 = tf.nn.lrn(conv_out2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)
    maxpool_out2 = maxpool_layer(norm2)
```

```
maxpool_reshaped = tf.reshape(maxpool_out2, [-1,
 W3.get_shape().as_list()[0]])
local = tf.add(tf.matmul(maxpool_reshaped, W3), b3)
local_out = tf.nn.relu(local)

out = tf.add(tf.matmul(local_out, W_out), b_out)
return out
```

Строит итоговые полносвязные слои

9.4.1. Оценка эффективности

После построения архитектуры нейронной сети необходимо задать функцию стоимости, которую требуется минимизировать. Воспользуемся функцией `softmax_cross_entropy_with_ logits` из библиотеки TensorFlow, описание которой можно найти в официальной документации (http://mng.bz/8mEk):

> *[Функция* `softmax_cross_entropy_with_logits`*] оценивает вероятность ошибки в задачах дискретной классификации, в которых классы являются взаимоисключающими (каждый элемент относится в точности к одному классу). Например, каждое изображение в наборе данных CIFAR-10 имеет маркировку одной и только одной меткой: изображение может быть собакой или грузовиком, но не тем и другим.*

Поскольку изображение может принадлежать одному из десяти возможных классов, представим себе этот выбор как 10-мерный вектор. Все элементы этого вектора — нули, кроме элемента, соответствующего определенному классу, который имеет значение 1. Это представление, которое уже встречалось в предыдущих главах, носит название *кодирования с одним активным состоянием.*

Как показано в листинге 9.15, расчет стоимости осуществляется с помощью функции потерь перекрестной энтропии, которая упоминалась в главе 4. Эта функция выдает вероятность ошибки классификации. Обратите внимание, что этот метод работает только для вариантов простой квалификации, в которых все классы взаимоисключающие (например, грузовик не может быть еще и собакой). Можно использовать разные виды оптимизации, но в нашем примере попробуем продолжить работу с AdamOptimizer, достаточно простым и быстрым

методом оптимизации (подробное описание можно найти на сайте http://mng.bz/zW98). Может быть, он не вполне подходит для использования в реальном мире, но хорошо работает для категории методов, имеющихся в наличии.

Листинг 9.15. Определяющие операторы для измерения стоимости и правильности

```
model_op = model()

cost = tf.reduce_mean(  ←— Задает кросс-энтропийную функцию потерь
    tf.nn.softmax_cross_entropy_with_logits(logits=model_op, labels=y)
)

train_op = tf.train.AdamOptimizer(learning_rate=0.001).minimize(cost)  ←—

correct_pred = tf.equal(tf.argmax(model_op, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

Определяет оператор обучения для минимизации функции потерь

И наконец, в последнем разделе главы мы запустим оператор обучения для минимизации стоимости нейронной сети. Проделав эту операцию несколько раз по всему набору данных, мы получим оптимальные веса (или параметры модели).

9.4.2. Обучения классификатора

В следующем листинге представлен циклический перебор всех изображений небольшими пакетами для обучения нейронной сети. Со временем веса начинают медленно сходиться к локальному оптимуму, что позволит точно распознать изображения по обучающим данным.

Листинг 9.16. Обучение нейронной сети с помощью набора данных CIFAR-10

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    onehot_labels = tf.one_hot(labels, len(names), on_value=1., off_value=0.,
     axis=-1)
    onehot_vals = sess.run(onehot_labels)
    batch_size = len(data) // 200
    print('batch size', batch_size)
    for j in range(0, 1000):  ←— Циклически выполняет 1000 эпох
        print('EPOCH', j)
        for i in range(0, len(data), batch_size):  ←— Обучает сеть пакетами
            batch_data = data[i:i+batch_size, :]
```

```
            batch_onehot_vals = onehot_vals[i:i+batch_size, :]
            _, accuracy_val = sess.run([train_op, accuracy], feed_dict={x:
 batch_data, y: batch_onehot_vals})
        if i % 1000 == 0:
             print(i, accuracy_val)
print('DONE WITH EPOCH')
```

Вот и всё! Вы успешно построили сверточную нейронную сеть для квалификации изображений. Осторожно, это может занять более 10 минут. Если запустить этот код на ЦП, могут потребоваться уже часы! Можете ли вы представить, каково это — обнаружить ошибку в коде после дня ожидания? Вот почему исследователи глубокого обучения используют мощные компьютеры и графические процессоры для ускорения вычислений.

9.5. Советы и трюки по повышению эффективности

Разработанная нами в этой главе сверточная нейронная сеть является простым методом решения задач классификации изображений, но для повышения эффективности нейронных сетей существуют методы значительно более сложные, чем созданный нами работающий прототип:

- *Дополнение данных*: из одного простого изображения можно легко создать новые обучающие изображения. Для начала наклоним изображение горизонтально или вертикально, в результате чего можно будет в четыре раза увеличить размер данных. Можно также настроить яркость изображения или цветовой оттенок, чтобы нейронная сеть распространила это и на другие колебания. Можно даже добавить случайный шум в изображение и добиться устойчивости метода классификации к небольшим осцилляциям. Полезным может оказаться уменьшение и увеличение размера изображения; использование элементов в обучающем наборе одного размера практически гарантированно приведет к переобучению!
- *Обучение с блокированием*: отслеживайте ошибки обучения и проверки в процессе обучения нейронной сети. Сначала обе ошибки должны медленно убывать, потому что сеть обучается. Но иногда при проверке ошибки снова начинают расти. Это сигнал о том, что нейронная сеть начала переобучаться и не способна обобщать ранее не встречавшиеся данные. Процесс обучения следует прервать в тот момент, когда появились признаки этого явления.

- *Регуляризация весов*: еще один способ противодействовать переобучению состоит в добавлении в функцию стоимости регуляризующего члена. Мы уже встречались с регуляризацией в предыдущих главах, и такие же принципы используются здесь.
- *Прореживание, или дропаут:* TensorFlow комплектуется удобной функцией `tf.nn.dropout`, которую можно применять к любому слою сети, чтобы уменьшить переобучение. Это отключает случайным образом выбранное число нейронов в этом слое во время обучения, чтобы сеть была избыточной и устойчивой к выдаваемому заключению.
- *Более глубокая архитектура*: глубокая архитектура получается в результате добавления в нейронную сеть скрытых слоев. Если имеется достаточно обучающих данных, то добавление скрытых слоев повышает эффективность сети.

УПРАЖНЕНИЕ 9.3

После первой итерации в этой архитектуре CNN попробуйте применить пару советов и трюков, упомянутых в этой главе.

ОТВЕТ

К сожалению, тонкая настройка является частью процесса. Следует начать с настройки гиперпараметров и переобучить алгоритм, пока не будут найдены настройки, обеспечивающие наилучшую эффективность.

9.6. Применение сверточных нейронных сетей

Сверточные нейронные сети достаточно успешны, если входными данными являются звуковые данные или изображения. Для промышленности особый интерес представляют изображения. Например, при регистрации в соцсети вы обычно загружаете фотографию профиля, а не запись своего приветствия. Это связано с тем, что людям в большей степени нравятся фотографии. Поэтому давайте посмотрим, как сверточные нейронные сети могут применяться для распознавания лиц.

По желанию общая архитектура CNN может быть как простой, так и сложной. Вы должны начать с простой и постепенно настраивать свою модель до удовлетворительного состояния. Нет абсолютно правильного пути, потому что распознавание лица не является полностью решенной задачей. Исследователи продолжают публиковать свои результаты, каждый из которых является шагом вперед по сравнению с предыдущими решениями.

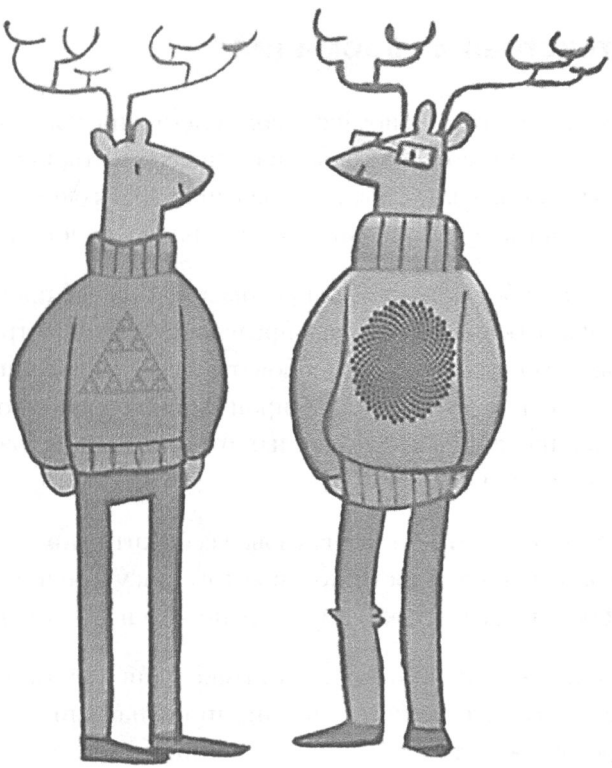
Прежде всего необходимо получить набор изображений. Одним из крупнейших наборов данных произвольных изображений является ImageNet (http://image-net.org/). Здесь можно найти отрицательные примеры для бинарного классификатора. Положительные примеры лиц можно найти во многих базах данных на сайтах, которые специализируются на человеческих лицах:

- Набор данных лиц VGG: www.robots.ox.ac.uk/~vgg/data/vgg_face/
- FDDB: набор данных по распознаванию лиц и опорные метки: http://vis-www.cs.umass.edu/fddb/
- База данных для распознавания лиц и идентификации положений головы: http://mng.bz/25N6
- База данных лиц на YouTube: www.cs.tau.ac.il/~wolf/ytfaces/

9.7. Краткие итоги

- Сверточные нейронные сети предполагают, что захвата локальных шаблонов сигнала достаточно, чтобы характеризовать его и таким образом уменьшать число параметров нейронной сети.
- Очистка данных необходима для работы большинства моделей машинного обучения. Один час, который вы потратите, чтобы написать код для очистки данных, ничто по сравнению с тем временем, которое потребуется нейронной сети, чтобы самой обучиться очистке данных.

Рекуррентные нейронные сети



Эта глава охватывает следующие темы:

- ✓ Знакомство с компонентами рекуррентных нейронных сетей
- ✓ Проектирование прогнозирующей модели данных временного ряда
- ✓ Использование алгоритма прогноза временных рядов для реальных данных

10.1. Контекстная информация

Вспоминая школьные годы, я помню вздох облегчения, когда один из моих промежуточных экзаменов состоял только из вопросов, в ответ на которые требовалось выбрать «истинно» или «ложно». Вряд ли только я один мог предположить, что половина ответов будут «истинно», а остальная половина — «ложно».

Я нашел ответы на большинство вопросов, а остальные проставил случайно. Но это угадывание основывалось на определенной умной стратегии, которую, возможно, вам тоже захочется попробовать. Подсчитав свои правильные ответы, я понял, что для устранения диспропорции не хватает ответов «ложно». Поэтому большинством моих угаданных ответов было «ложно», чтобы восстановить пропорциональность.

И это сработало. В тот момент я чувствовал себя хитрецом. Чем на самом деле является это ощущение хитрости, которое дает нам уверенность в наших решениях, и как нам передать эту же силу уверенности нейронным сетям?

Один из вариантов — использовать контекст для ответа на вопросы. Контекстуальные стимулы являются важными сигналами, которые способны повысить эффективность алгоритмов машинного обучения. Например, необходимо проверить предложение на английском языке и отметить часть речи каждого слова.

Простейший метод состоит в том, чтобы отдельно классифицировать каждое слово как имя существительное, имя прилагательное и так далее, не принимая к сведению соседние слова. Рассмотрим опробование этого метода на словах *этого* предложения. Слово *опробование* используется как глагол, но в зависимости от контекста его можно также использовать как прилагательное, делая отметку частей речи *изнурительной* проблемой.

Более эффективный метод учитывает контекст. Для того чтобы придать нейронным сетям контекстуальный смысл, рассмотрим архитектуры с названием *рекуррентная нейронная сеть* (recurrent neural network). Вместо естественного языка для описания данных мы будем иметь дело с непрерывными временными рядами данных, такими как биржевые курсы, рассмотренные нами в предыдущих главах. В конце этой главы мы сможем моделировать паттерны во временных рядах данных для прогнозирования следующих значений.

10.2. Введение в рекуррентные нейронные сети

Чтобы понять, как работают рекуррентные нейронные сети, рассмотрим сначала простую архитектуру, приведенную на рис. 10.1. В определенный момент времени (t) она берет входной вектор $X(t)$ и генерирует выходной вектор $Y(t)$. Окружность в центре представляет скрытый слой сети.

Рис. 10.1. Нейронная сеть с входным и выходным слоями, помеченными как $X(t)$ и $Y(t)$ соответственно

При достаточном количестве примеров в TensorFlow можно с помощью обучения получить параметры модели. Например, обозначим входные веса через матрицу $W_{вх}$, а выходные веса через матрицу $W_{вых}$. Пусть в сети есть один скрытый слой, который мы обозначим через вектор $Z(t)$.

Как показано на рис. 10.2, первая половина нейронной сети характеризуется функцией $Z(t) = X(t) \times W_{вх}$, а вторая половина нейронной сети принимает форму $Y(t) = Z(t) \times W_{вых}$. Это эквивалентно тому, что вся нейронная сеть представляется функцией $Y(t) = (X(t) \times W_{вх}) \times W_{вых}$.

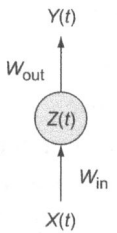


Рис. 10.2. Скрытый слой нейронной сети можно представить как скрытое представление данных, которые кодируются входными весовыми коэффициентами и декодируются выходными весовыми коэффициентами

После настройки сети можно попробовать ее запустить в реальном сценарии, используя обученную модель. Обычно это предполагает многократный вызов модели, может быть, даже повторный, как показано на рис. 10.3.

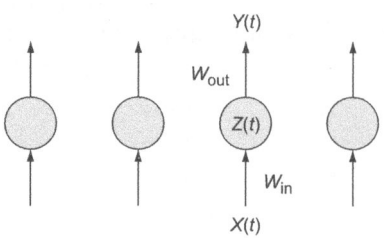


Рис. 10.3. Приходится часто завершать выполнение одной и той же сети много раз, не используя сведения о скрытых состояниях предыдущих прогонов

В каждый момент времени t, когда происходит обращение к обученной модели, эта архитектура не учитывает сведения о предыдущих прогонах. Это напоминает прогнозирование тенденции курса акций, при котором учитываются только данные текущего дня. В этом случае было бы лучше использовать комплексные шаблоны данных за неделю, а еще лучше — за месяц.

Рекуррентная нейронная сеть (RNN) отличается от традиционной нейронной сети тем, что она при передаче информации в течение продолжительного времени вводит вес перехода *W*. На рис. 10.4 показаны три матрицы весов, которые необходимо получить в процессе обучения в RNN. Введение веса перехода означает, что следующее состояние теперь зависит от предыдущей модели, а также от предыдущего состояния. Это значит, что наша модель теперь «помнит» то, что она сделала!

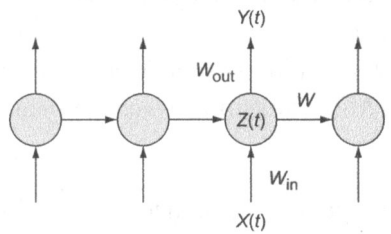


Рис. 10.4. Архитектура нейронной сети может использовать состояния этой сети с выгодой для себя

Составлять схемы — это здорово, но теперь придется засучить рукава. Давайте постараемся сделать все правильно! В следующем разделе описывается, как использовать встроенные в библиотеку TensorFlow модели рекуррентных нейронных сетей. Затем для прогнозирования будущих результатов мы воспользуемся сетью RNN, используя реальные временные ряды данных!

10.3. Использование рекуррентной нейронной сети

По мере реализации сети RNN мы убедимся, что TensorFlow выполняет самую тяжелую часть работы. Нам не нужно будет создавать сеть вручную, как было показано на рис. 10.4, потому что TensorFlow уже поддерживает некоторые надежные модели рекуррентных нейронных сетей.

ПРИМЕЧАНИЕ Информацию по поддержке рекуррентной нейронной сети в TensorFlow можно найти на сайте www.tensorflow.org/tutorials/recurrent.

Один из типов модели сети RNN носит название *долгой краткосрочной памяти* (LSTM — Long Short-Term Memory). Я согласен с тем, что это название очень странное. Оно означает в точности то, как оно звучит: краткосрочные шаблоны не забываются в долгосрочной перспективе.

Точные детали реализации LSTM выходят за рамки этой книги. Поверьте, тщательное знакомство с моделью LSTM уведет нас в сторону от цели этой главы, потому что для этих моделей пока что нет определенного стандарта. Именно здесь TensorFlow придет нам на помощь. Она учитывает то, как была определена модель, чтобы ею можно было воспользоваться в готовом виде. Это также означает, что по мере обновления TensorFlow можно будет пользоваться преимуществами внесенных в модель LSTM улучшений, не делая никаких изменений в коде.

> **СОВЕТ** Чтобы понять, как реализовать LSTM с нуля, я предлагаю следующее объяснение: https://apaszke.github.io/lstm-explained.html. В статье, доступной на сайте http://arxiv.org/abs/1409.2329, описана реализация регуляризации, использованной в следующих листингах.

Начнем писать код в новом файле simple_regression.py. Импортируем необходимые библиотеки, как показано в листинге 10.1.

Листинг 10.1. Импортирование необходимых библиотек

```
import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
```

Теперь определим класс `SeriesPredictor`. Конструктор, показанный в листинге 10.2, задает гиперпараметры модели, веса и функцию стоимости.

Листинг 10.2. Определение класса и его конструктора

```
class SeriesPredictor:
    def  init (self, input_dim, seq_size, hidden_dim=10):

    self.input_dim = input_dim
    self.seq_size = seq_size
    self.hidden_dim = hidden_dim
```

Гиперпараметры

```
        self.W_out = tf.Variable(tf.random_normal([hidden_dim, 1]),
         name='W_out')
        self.b_out = tf.Variable(tf.random_normal([1]), name='b_out')
        self.x = tf.placeholder(tf.float32, [None, seq_size, input_dim])
        self.y = tf.placeholder(tf.float32, [None, seq_size])

        self.cost = tf.reduce_mean(tf.square(self.model() - self.y))
        self.train_op = tf.train.AdamOptimizer().minimize(self.cost)

        self.saver = tf.train.Saver()
```

Переменные весов и входные переменные-заполнители

Оптимизация затрат

Дополнительные операторы

Давайте теперь используем встроенную в TensorFlow модель рекуррентной нейронной сети `BasicLSTMCell`. Скрытая размерность ячейки, переданной объекту модели `BasicLSTMCell`, является размерностью скрытого состояния, которое со временем проходит. Для восстановления результатов можно запустить эту ячейку с данными, используя функцию `rnn.dynamic_rnn`. В следующем листинге показаны детали использования TensorFlow для реализации прогностической модели с помощью LSTM.

Листинг 10.3. Определение модели RNN

```
    def model(self):
        """
        :param x: inputs of size [T, batch_size, input_size]
        :param W: matrix of fully-connected output layer weights
        :param b: vector of fully-connected output layer biases
        """
        cell = rnn.BasicLSTMCell(self.hidden_dim)
        outputs, states = tf.nn.dynamic_rnn(cell, self.x, dtype=tf.float32)
        num_examples = tf.shape(self.x)[0]
        W_repeated = tf.tile(tf.expand_dims(self.W_out, 0),
                     [num_examples, 1, 1])
        out = tf.matmul(outputs, W_repeated) + self.b_out
        out = tf.squeeze(out)
        return out
```

Выполняется ячейка на входе, чтобы получить тензор для выходов и состояний

Создает ячейку LSTM

Вычисляет выходной слой как полносвязную линейную функцию

С выбранной моделью и функцией стоимости теперь можно использовать обучающую функцию, с помощью которой в результате обучения определяются веса LSTM с использованием имеющейся пары входов/выходов. Как показывает листинг 10.4, можно открыть сеанс и повторно запустить процедуру оптимизации на обучающих данных.

ПРИМЕЧАНИЕ Для того чтобы узнать, сколько потребуется итераций для обучения модели, можно воспользоваться перекрестной проверкой. В этом случае предполагается фиксированное число эпох. Некоторые интересные анализы и ответы можно найти на сайте ResearchGate: http://mng.bz/lB92.

После обучения сохраните модель в файл, чтобы позже ее можно было загрузить.

Листинг 10.4. Обучение модели по набору данных

```
def train(self, train_x, train_y):
    with tf.Session() as sess:
        tf.get_variable_scope().reuse_variables()
        sess.run(tf.global_variables_initializer())
        for i in range(1000):  ←—— 1000 раз выполняет оператор обучения
            _, mse = sess.run([self.train_op, self.cost],
 feed_dict={self.x: train_x, self.y: train_y})
            if i % 100 == 0:
                print(i, mse)
        save_path = self.saver.save(sess, 'model.ckpt')
        print('Model saved to {}'.format(save_path))
```

Наша модель успешно прошла курс обучения, в результате которого были определены оптимальные параметры. Оценим теперь прогностическую модель на других данных. В следующем листинге показаны загрузка сохраненной модели и ее работа во время сеанса на проверочных данных. Если обученная модель плохо работает с проверочными данными, можно попытаться подобрать оптимальное число ячеек LSTM со скрытыми размерами.

Листинг 10.5. Проверка обученной модели

```
def test(self, test_x):
    with tf.Session() as sess:
        tf.get_variable_scope().reuse_variables()
        self.saver.restore(sess, './model.ckpt')
        output = sess.run(self.model(), feed_dict={self.x: test_x})
        print(output)
```

Готово! Но для того, чтобы убедиться, что она работает как следует, создадим некоторый набор данных и попытаемся обучить прогностическую модель. В следующем листинге показано создание входных последовательностей, `train_x`, и соответствующих выходных последовательностей, `train_y`.

Листинг 10.6. Обучение и проверка на фиктивных данных

```
if __name__ == '__main__':
    predictor = SeriesPredictor(input_dim=1, seq_size=4, hidden_dim=10)
    train_x = [[[1], [2], [5], [6]],
               [[5], [7], [7], [8]],
               [[3], [4], [5], [7]]]
    train_y = [[1, 3, 7, 11],
               [5, 12, 14, 15],
               [3, 7, 9, 12]]
    predictor.train(train_x, train_y)

    test_x = [[[1], [2], [3], [4]],  ←— Прогнозируемые результаты должны быть 1, 3, 5, 7
              [[4], [5], [6], [7]]]  ←— Прогнозируемые результаты должны быть 4, 9, 11, 13
    predictor.test(test_x)
```

Можно рассматривать прогностическую модель как черный ящик и обучить ее прогнозировать, используя временные ряды данных реального мира. В следующем разделе мы получим необходимые для работы данные.

10.4. Прогностическая модель данных временного ряда

Различные данные временных рядов доступны в интернете. Для нашего примера воспользуемся данными по международным пассажирским перевозкам за определенный период времени. Их можно получить на сайте http://mng.bz/5UWL. Перейдя по ссылке, мы получим превосходный набор данных временных рядов (рис 10.5).

Щелкнув на вкладке Export, а затем выбрав в этой группе CSV (,), можно загрузить необходимые данные. Теперь отредактируем вручную файл CSV, убрав строку заголовка и строку нижнего колонтитула.

Рис. 10.5. Исходные данные международных пассажирских перевозок по годам

В файл data_loader.py добавьте следующий код.

Листинг 10.7. Загрузка данных

```
import csv
import numpy as np
import matplotlib.pyplot as plt

def load_series(filename, series_idx=1):
    try:
        with open(filename) as csvfile:
            csvreader = csv.reader(csvfile)

            data = [float(row[series_idx]) for row in csvreader
                                            if len(row) > 0]
            normalized_data = (data - np.mean(data)) / np.std(data)
        return normalized_data
    except IOError:
        return None

def split_data(data, percent_train=0.80):
    num_rows = len(data) * percent_train
    return data[:num_rows], data[num_rows:]
```

Циклически проходит все строки файла и преобразует данные в формат с плавающей запятой

Вычисляет выборки обучающих данных

Делит набор данных на обучающий и тренировочный набор

Обработка данных центрированием среднего и делением на среднеквадратическое отклонение

Здесь мы определили две функции, `load_series` и `split_data`. Первая загружает файл временного ряда и нормализует его, а вторая делит набор данных на две части: для обучения и проверки.

Поскольку мы уже много раз оценивали модель для прогнозирования будущих значений, давайте изменим функцию `test` из `SeriesPredictor`. Теперь она берет в качестве аргумента сеанс, а не инициализирует сеанс при каждом вызове. Посмотрим в следующем листинге эту настройку конфигурации.

Листинг 10.8. Изменение функции `test` для передачи в сеанс

```
def test(self, sess, test_x):
    tf.get_variable_scope().reuse_variables()
    self.saver.restore(sess, './model.ckpt')
    output = sess.run(self.model(), feed_dict={self.x: test_x})
    return output
```

Теперь можно обучить алгоритм прогноза, загрузив данные в приемлемом формате. В листинге 10.9 показано, как обучать сеть и затем использовать обученную модель для прогноза будущих значений.

Мы создадим набор обучающих данных (`train_x` и `train_y`), чтобы он выглядел так, как показано в листинге 10.6.

Листинг 10.9. Генерация обучающих данных

Загружает данные

Перемещает окно по данным временного ряда для получения набора обучающих данных

```
if __name__ == '__main__':
    seq_size = 5
    predictor = SeriesPredictor(
        input_dim=1,
        seq_size=seq_size,
        hidden_dim=100)

    data = data_loader.load_series('international-airline-passengers.csv')
    train_data, actual_vals = data_loader.split_data(data)

    train_x, train_y = [], []
    for i in range(len(train_data) - seq_size - 1):
        train_x.append(np.expand_dims(train_data[i:i+seq_size],
     axis=1).tolist())
```

Каждый элемент последовательности является скаляром (одномерным массивом)

Длина каждой последовательности

Размер скрытой размерности сети RNN

```
    train_y.append(train_data[i+1:i+seq_size+1])

test_x, test_y = [], []
for i in range(len(actual_vals) - seq_size - 1):
    test_x.append(np.expand_dims(actual_vals[i:i+seq_size],
 axis=1).tolist())
    test_y.append(actual_vals[i+1:i+seq_size+1])

predictor.train(train_x, train_y, test_x, test_y)

with tf.Session() as sess:
    predicted_vals = predictor.test(sess, test_x)[:,0]
    print('predicted_vals', np.shape(predicted_vals))
    plot_results(train_data, predicted_vals, actual_vals,
 'predictions.png')

    prev_seq = train_x[-1]
    predicted_vals = []
    for i in range(20):
        next_seq = predictor.test(sess, [prev_seq])
        predicted_vals.append(next_seq[-1])
        prev_seq = np.vstack((prev_seq[1:], next_seq[-1]))
    plot_results(train_data, predicted_vals, actual_vals,
 'hallucinations.png')
```

Использует ту же стратегию перемещения окна для получения набора тренировочных данных

Визуализирует работу модели

Обучает модель на обучающих данных

Предиктор генерирует два графика. Первый — это результаты предсказания модели с учетом контрольных значений, как показано на рис. 10.6.

Другой график показывает результаты прогноза, если доступны только обучающие данные (синяя линия) и больше ничего (рис. 10.7). У этой процедуры меньше доступной информации, но она все еще неплохо справляется с аппроксимацией временных тенденций данных.

Мы можем пользоваться алгоритмами прогноза временных рядов для воспроизведения реалистических колебаний данных. Представьте, что вы прогнозируете рыночные циклы бума и спада с помощью уже известных вам инструментов. Чего же мы ждем? Возьмите данные торгов и обучите на них свою собственную прогностическую модель!

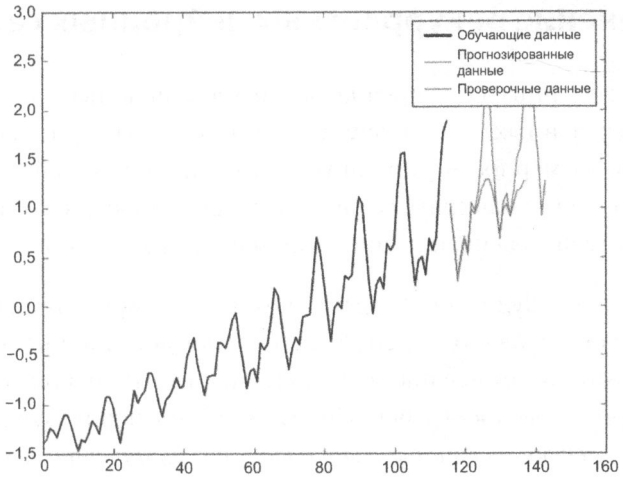


Рис. 10.6. Прогнозирование достаточно хорошо соответствует временным тенденциям при проверке на контрольных данных

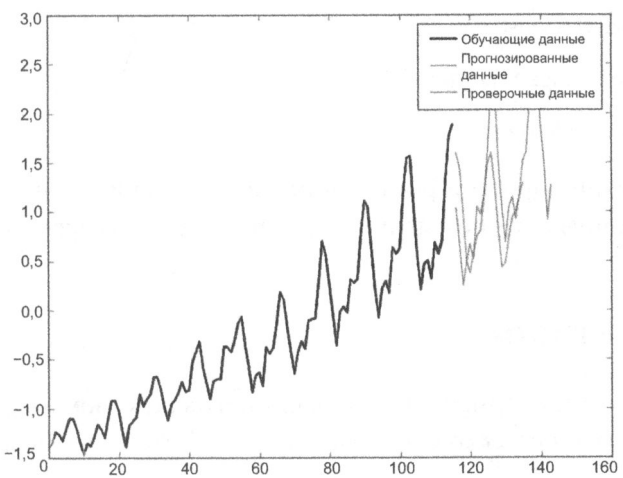


Рис. 10.7. Если алгоритм использует ранее предсказанные результаты для дальнейших прогнозов, то хорошо согласуется общий тренд, а не конкретные горбы на графике

10.5. Применение рекуррентных нейронных сетей

Рекуррентные нейронные сети предназначены для использования с набором последовательных данных. Поскольку звуковые сигналы имеют меньшую, чем у видеосигналов, размерность (линейный сигнал и двумерный массив пикселов), значительно легче начинать со звуковых временных рядов. Посмотрите, насколько стало лучше распознавание речи: теперь это легко решаемая задача!

Так же как при анализе аудиогистограммы, которую мы проводили в главе 5 при кластеризации данных звукозаписи, большинство методов предварительной обработки содержат представление звуковой записи в виде некой хромограммы. В частности, широкое распространение получил метод, использующий признак под названием *мел-спектральных кепстральных коэффициентов* (MFCC — mel-frequency cepstral coefficients). Познакомиться с этим методом можно на сайте: http://mng.bz/411F.

Кроме того, для обучения модели потребуется база. К наиболее популярным относятся следующие базы данных:

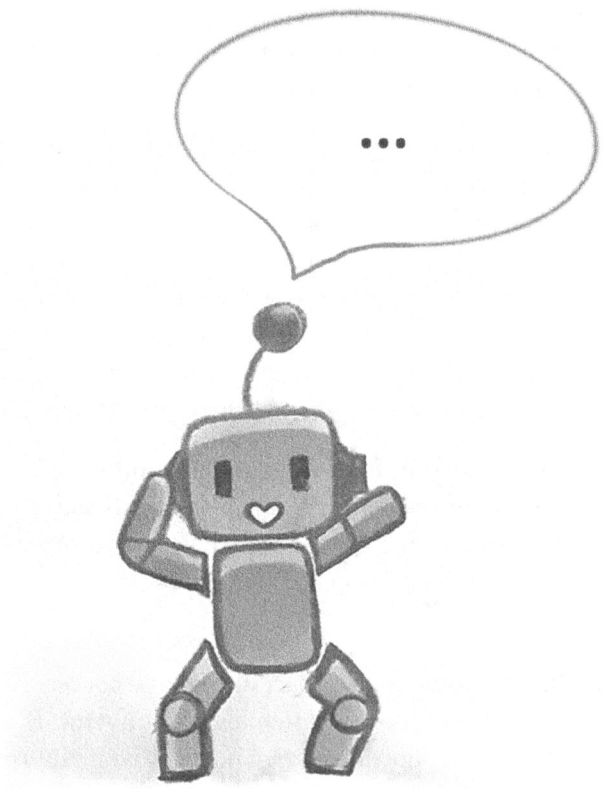
- LibriSpeech: www.openslr.org/12
- TED-LIUM: www.openslr.org/7
- VoxForge: www.voxforge.org

Подробное описание простой реализации распознавания речи в TensorFlow с этими базами данных можно найти на сайте: https://svds.com/tensorflow-rnn-tutorial.

10.6. Краткие итоги

- RNN использует информацию из прошлого. Посредством этого она может делать прогнозы, используя данные с высокой степенью зависимости от времени.
- TensorFlow включает реализацию RNN.
- Прогнозирование временных рядов является одной из областей применения RNN благодаря временной зависимости в данных.

Модели sequence-to-sequence для чат-бота



Эта глава охватывает следующие темы:

- ✓ Проверка архитектуры сопоставления последовательностей
- ✓ Векторы вложений слов
- ✓ Реализация чат-бота с помощью данных реального мира

Общение с клиентами по телефону — это бремя как для клиента, так и для компании. Поставщики услуг платят хорошие деньги, чтобы нанять сотрудников для обслуживания клиентов, но что, если бы можно было автоматизировать большую часть этих усилий? Можем ли мы разработать программное обеспечение для взаимодействия с клиентами, используя естественный язык?

Идея не такая надуманная, как вам представляется. Чат-боты вызывают много шумихи из-за беспрецедентных изменений в обработке естественного языка с использованием методов глубокого обучения. Возможно, что при наличии достаточного объема обучающих данных чат-бот мог бы научиться находить подходящий ответ на большинство наиболее часто возникающих у клиентов вопросов с помощью обработки естественного языка. Если чат-бот был бы действительно эффективным, это могло бы не только экономить деньги компании, устраняя необходимость нанимать людей, но даже ускорять поиск ответа на запрос клиента.

В этой главе вы создадите чат-бот, предоставив нейронной сети тысячи примеров входных и выходных предложений. Ваш обучающий набор данных — это пара выражений на английском языке: например, если вы спросите: «Как дела?», чат-бот должен ответить: «Спасибо, хорошо».

ПРИМЕЧАНИЕ В этой главе мы рассматриваем *последовательности* и *предложения* как взаимозаменяемые понятия. В нашем методе предложение будет последовательностью букв. Другой распространенный метод состоит в представлении предложения в виде последовательности слов.

Этот алгоритм старается дать разумный ответ на естественном языке на каждый запрос, также заданный на естественном языке. Мы будем использовать нейронную сеть, которая применяет две основные концепции, с которыми вы познакомились в предыдущих главах: многоклассовую классификацию и рекуррентные нейронные сети (RNN).

11.1. Построения на основе классификации и RNN

Помните, *классификация* является методом машинного обучения для прогноза категории элемента входных данных. Кроме того, многоклассовая классификация позволяет рассматривать более чем два класса. В главе 4 было показано, как применять этот алгоритм в TensorFlow. В частности, функция стоимости между модельными прогнозами (последовательность чисел) и контрольными данными (унитарный вектор) пытается найти расстояние между двумя последовательностями, используя функцию кросс-энтропийных потерь.

ПРИМЕЧАНИЕ Унитарный вектор состоит практически только из нулей, кроме одного элемента, равного 1.

В этом случае, применяя чат-бот, мы будем использовать функцию кросс-энтропийных потерь для измерения различия между двумя последовательностями: откликом модели (который является последовательностью) и контрольными данными (которые также являются последовательностью).

УПРАЖНЕНИЕ 11.1

В TensorFlow можно использовать функцию кросс-энтропийных потерь для измерения сходства между унитарным вектором, например (1, 0, 0), и выходом нейронной сети, таким как (2.34, 0.1, 0.3). С другой стороны, предложения на английском языке не являются числовыми векторами. Как можно использовать кросс-энтропийные потери для измерения сходства между двумя предложениями на английском языке?

ОТВЕТ

Грубый подход заключается в представлении каждого предложения в виде вектора с подсчетом частоты появления каждого слова внутри предложения. Затем необходимо сравнить векторы, чтобы оценить степень их различия.

Вы можете вспомнить, что RNN представляют собой нейронную сеть, предназначенную не только для входных данных текущего шага по времени, но и для информации о состоянии из предыдущих входных данных. Они подробно описаны в главе 10 и будут снова использованы в этой главе. RNN представляют входные и выходные данные в виде временных рядов, что полностью соответствует тому, как нам нужно представить последовательность.

Простейшая идея состоит в использовании готовой RNN из TensorFlow для реализации чат-бота. Давайте разберемся, почему этот метод не годится. На входе и на выходе RNN используются предложения на естественном языке, поэтому входные ($x_t, x_{t-1}, x_{t-2}, ...$) и выходные ($y_t, y_{t-1}, y_{t-2}, ...$) данные могут быть представлены как последовательности слов. Проблема в использовании рекуррентных нейронных сетей при моделировании разговора состоит в том, что RNN мгновенно выдает результат. Если на входе используется последовательность слов (*как, вы, поживаете*), то первое слово на выходе будет зависеть только от первого слова на входе. Элемент выходной последовательности y_t рекуррентной нейронной сети не может видеть следующих частей входного предложения, чтобы принять соответствующее решение; он будет ограничен знанием только предыдущих входных предложений ($x_t, x_{t-1}, x_{t-2}, ...$). Простейшая модель RNN пытается перейти к ответу на запрос до окончания формулирования этого запроса, что может привести к неправильным результатам.

Вместо этого остановимся на использовании двух RNN: одной для входных предложений, а другой для выходных. После окончания обработки входной последовательности первой рекуррентной сетью она отправит скрытое состояние во вторую сеть для обработки выходного предложения. Одна RNN имеет метку «кодировщик», а вторая — «декодировщик» (рис. 11.1).

Мы используем концепции многоклассовой квалификации и RNN из предыдущих глав для проектирования нейронной сети, которая обучается отображать входную последовательность в выходную. RNN позвляет кодировать входное

предложение, передавая суммированный вектор состояния в декодер, а затем декодирует его в ответное предложение. Для измерения потерь между откликом модели и контрольными данными мы обратились к функции, используемой в многоклассовой классификации, а именно к функции потерь кросс-энтропии.

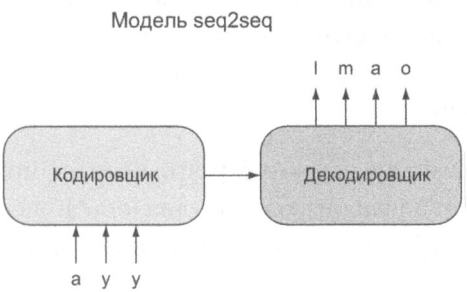


Рис. 11.1. Здесь приводится низкоуровневая модель нейронной сети. Входные данные *a y y* передаются для кодирования в сеть, и ожидается, что декодировщик выдаст *l m a o*. Это простейшие примеры чат-бота, но вы могли бы представить более сложные пары предложений для ввода и вывода

Эта архитектура носит название *sequence-to-sequence* (seq2seq). В качестве обучающих данных мы будем использовать тысячи пар предложений, которые взяты из киносценариев. Алгоритм будет наблюдать за примерами диалогов и в результате научится формировать ответы на произвольные вопросы, которые мы захотим ему задать.

УПРАЖНЕНИЕ 11.2

Какие другие отрасли могут извлечь выгоду из чат-бота?

ОТВЕТ

Одним из вариантов может быть обучающий студентов чат-бот. Он мог бы служить учебным инструментом для обучения таким предметам, как английский язык, математика и даже информатика.

В конце этой главы у нас будет собственный чат-бот, который сможет в какой-то степени осмысленно отвечать на поставленные вопросы. Его нельзя будет

назвать идеальным, потому что модель всегда отвечает одинаково на один и тот же вопрос.

Предположим, что вы путешествуете в другой стране, не зная языка. Умный продавец вручил вам книгу, заверив, что там есть все необходимое, чтобы отвечать фразами на иностранном языке. Вы используете эту книгу как словарь. Когда кто-то что-то сказал на иностранном языке, вы можете найти эту фразу в своем словаре, и он подскажет ответ «Если кто-то говорит: *Привет*, скажите в ответ: *И тебе привет!*».

Это может быть полезно для коротких разговоров, но может ли такой словарь дать правильный ответ для произвольного диалога? Конечно, нет! Как ответить на вопрос «Вы голодны?», если ответ на него пропечатан в книге и никогда не изменится?

В словаре отсутствует информация о состоянии, которая является ключевым компонентом в диалоге. В модели seq2seq вы столкнетесь с похожей проблемой, но это хорошее начало! Верите вы или нет, но до 2017 года представление иерархических состояний для интеллектуального диалога пока еще не стало стандартом; многие чат-боты основываются на моделях seq2seq.

11.2. Архитектура seq2seq

Модель seq2seq пытается обучить нейронную сеть, которая прогнозирует выходную последовательность по входной последовательности. Последовательности несколько отличаются от традиционных векторов, потому что последовательность предполагает упорядочивание событий.

Время является интуитивным способом упорядочивания событий: мы обычно заканчиваем ссылку на что-нибудь словами, связанными со временем, такими как *временный*, *временной ряд*, *прошлый* и *будущий*. Например, мы могли бы сказать, что RNN распространяют информацию *будущие временные шаги*. Или улавливают *временные зависимости*.

ПРИМЕЧАНИЕ RNN подробно описаны в главе 10.

Модель seq2seq используется с несколькими RNN. На рис. 11.2 изображена отдельная ячейка RNN; она служит в качестве строительного блока для остальной архитектуры модели seq2seq.

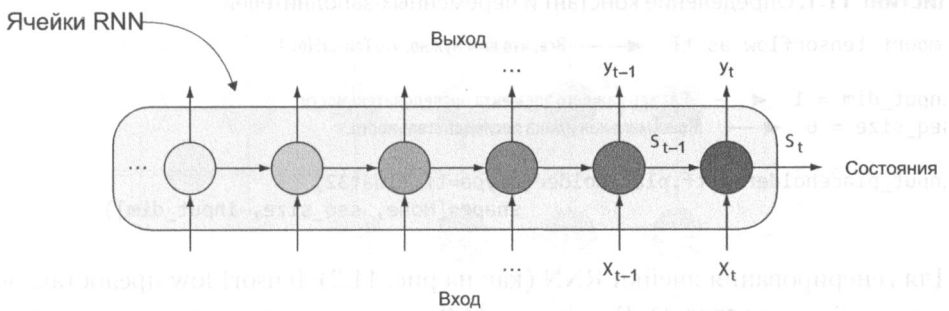


Рис. 11.2. Вход, выход и состояния рекуррентной нейронной сети. Можно опустить подробности того, как в точности реализуется RNN. Имеет значение только представление входа и выхода

Прежде всего, вы узнаете, как накладывать рекуррентные сети друг на друга, чтобы повысить сложность модели. Затем вы узнаете, как передавать скрытое состояние одной RNN другой, чтобы получить «кодировщика» и «декодировщика». Вы увидите, что начать пользоваться RNN достаточно просто.

После этого мы пройдем вводный курс преобразования предложений естественного языка в последовательности векторов. Поскольку RNN понимают только числовые данные, то этот процесс преобразования абсолютно необходим. Так как *последовательность* является другим обозначением «списка тензоров», то эти данные нужно преобразовывать соответствующим образом. Например, предложение является последовательностью слов, но слова не являются тензорами. Процесс преобразования слов в тензоры или в векторы носит название *векторного представления*, или *встраивания* (embeddings).

И наконец, собрав все эти понятия вместе, мы реализуем модель `seq2seq` на данных реального мира. Эти данные тысячи разговоров взяты из киносценариев.

Можно взяться за дело без промедления, используя код из следующего листинга. Откройте новый файл Python и скопируйте в него листинг 11.1, чтобы задать константы и переменные-заполнители. Переменную-заполнитель необходимо

задать в виде `[None, seq_size, input_dim]`, где `None` означает динамичный размер, ведь размер пакета может меняться; `seq_size` — длина последовательности, `input_dim` — размер каждого элемента последовательности.

Листинг 11.1. Определение констант и переменных-заполнителей

```
import tensorflow as tf  ◄── Все, что вам нужно, это TensorFlow!

input_dim = 1  ◄── Размер каждого элемента последовательности
seq_size = 6  ◄── Максимальная длина последовательности

input_placeholder = tf.placeholder(dtype=tf.float32,
                                   shape=[None, seq_size, input_dim])
```

Для генерирования ячейки RNN (как на рис. 11.2) TensorFlow предоставляет полезный класс `LSTMCell`. В листинге 11.2 показано, как его использовать и извлекать из ячейки выходные данные и состояния. Для удобства в листинге задается вспомогательная функция `make_cell`, позволяющая задать ячейку LSTM RNN. Помните, просто задать ячейку недостаточно, необходимо также вызвать `tf.nn.dynamic_rnn` и настроить сеть.

Листинг 11.2. Создание простой ячейки сети RNN

```
def make_cell(state_dim):
    return tf.contrib.rnn.LSTMCell(state_dim)  ◄── Проверьте в документации tf.contrib.rnn другие типы ячеек, такие как GRU

with tf.variable_scope("first_cell") as scope:
    cell = make_cell(state_dim=10)
    outputs, states = tf.nn.dynamic_rnn(cell,  ◄── Будет два результата: выходные данные и состояния
                                        input_placeholder,  ◄── Это входная последовательность для сети RNN
                                        dtype=tf.float32)
```

Помните, что в предыдущих главах было описано, как можно увеличить сложность нейронной сети, если добавлять в нее все больше и больше скрытых слоев? Большее число слоев означает большее число параметров, позволяющих модели представлять больше функций, то есть быть более универсальной.

И знаете, что я подумал? Можно укладывать ячейки друг на друга. Ничто нас не остановит. Эта процедура делает модель более сложной, и поэтому двухслойная

модель RNN будет работать лучше, ведь она обладает большей выразительной способностью. На рис. 11.3 показаны две ячейки, уложенные друг на друга.

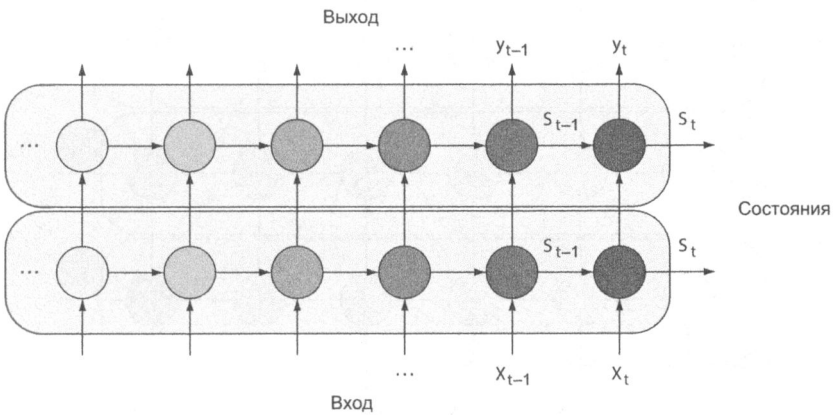


Рис. 11.3. Для формирования более сложной архитектуры можно укладывать ячейки RNN друг на друга

ПРЕДУПРЕЖДЕНИЕ Чем универсальней модель, тем выше вероятность ее переобучения.

В TensorFlow на интуитивном уровне можно реализовать двуслойную RNN. Сначала создается новая переменная область для второй ячейки. Уложив друг на друга две RNN, можно передавать выходные данные первой ячейки на вход другой ячейки. В следующем листинге показано, как это можно реализовать.

Листинг 11.3. Укладка двух ячеек RNN

```
with tf.variable_scope("second_cell") as scope:
    cell2 = make_cell(state_dim=10)
    outputs2, states2 = tf.nn.dynamic_rnn(cell2,
                                          outputs,
                                          dtype=tf.float32)
```

Входом в эту ячейку служит выход другой

Определение области переменных помогает избежать ошибок выполнения из-за повторного использования переменной

А если мы хотим использовать четыре слоя нейронной сети? Или десять? На рис. 11.4 показаны четыре ячейки RNN, уложенные друг на друга.

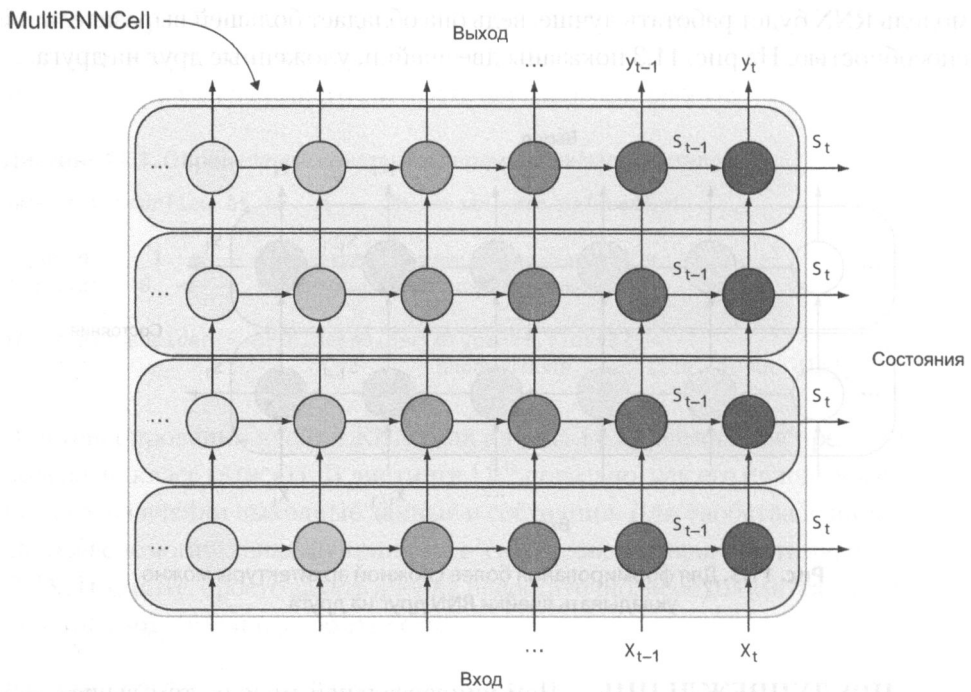


Рис. 11.4. TensorFlow позволяет укладывать друг на друга столько ячеек, сколько потребуется

Полезное ухищрение для укладывания ячеек в TensorFlow носит название `MultiRNNCell`. В следующем листинге показано, как использовать эту функцию для укладывания произвольного числа ячеек нейронной сети.

Листинг 11.4. Использование MultiRNNCell для укладки нескольких ячеек

```
def make_multi_cell(state_dim, num_layers):
    cells = [make_cell(state_dim) for _ in range(num_layers)]
    return tf.contrib.rnn.MultiRNNCell(cells)

multi_cell = make_multi_cell(state_dim=10, num_layers=4)
outputs4, states4 = tf.nn.dynamic_rnn(multi_cell,
                                      input_placeholder,
                                      dtype=tf.float32)
```

Синтаксис for-loop является предпочтительным способом построения списка ячеек сети RNN

До сих пор мы наращивали RNN вертикально, передавая выходные данные одной ячейки на вход другой. В модели seq2seq одна ячейка такой сети

С

предназначена для обработки входного предложения, а другая — для обработки выходного предложения. Для осуществления связи между двумя ячейками сети можно также подключать ячейки горизонтально, присоединяя состояния из одной ячейки к состоянию другой, как показано на рис. 11.5.

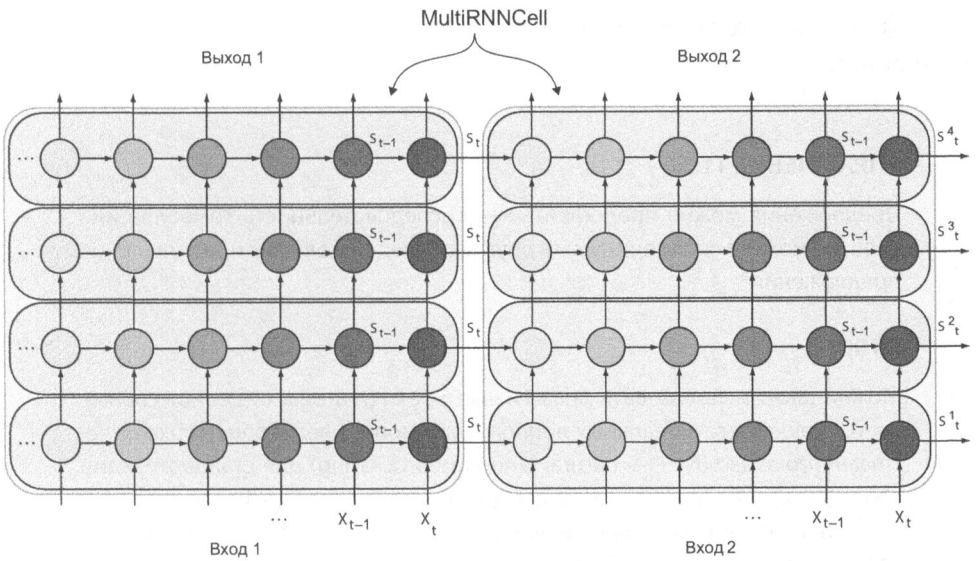


Рис. 11.5. Можно использовать последнее состояние первой ячейки как начальное состояние следующей ячейки. Модель можно обучить отображать входную последовательность в выходную последовательность. Эта модель получила название seq2seq

Мы уложили ячейки сети RNN вертикально и соединили их горизонтально, значительно увеличив число параметров сети. Но это же кощунство какое-то! Да. Мы построили монолитную архитектуру, составляя RNN по всем направлениям. Но для этого безумия существует метод, потому что такая сумасшедшая архитектура нейронной сети является основой модели `seq2seq`.

Как можно видеть на рис. 11.5, модель `seq2seq` имеет две входа и два выхода. Но использоваться будет только первый вход для входного предложения, и только второй выход будет использоваться для выходного предложения.

Вам может быть интересно, что делать с двумя другими последовательностями. Как ни странно, первая выходная последовательность совсем не используется

моделью `seq2seq`. И как мы увидим, вторая входная последовательность с помощью контура обратной связи использует некоторые данные второй последовательности.

В качестве обучающих данных для создания чат-бота используем пары входных и выходных предложений, и нам необходимо лучше понять, как представлять слова в тензоре. Этот раздел покажет, как это сделать с помощью TensorFlow.

УПРАЖНЕНИЕ 11.3

Предложения можно представить как последовательность символов или слов, но можете ли вы придумать другие последовательные представления предложений?

ОТВЕТ

Можно также использовать фразы и части речи (глаголы, существительные и т. п.). Чаще всего в реальных приложениях применяется *обработка естественного языка* (NLP — natural language processing) для стандартизации словоформ, орфографии и смысла. Одним из примеров библиотеки, которая выполняет это преобразование, является *fastText* из Facebook (https://github.com/facebookresearch/fastText).

11.3. Векторное представление символов

Слова и буквы являются символами, а преобразование символов в числовые значения можно осуществить в TensorFlow. Например, пусть в нашем словаре четыре слова: слово$_0$: *the*; слово$_1$: *fight*; слово$_2$: *wind*; и слово$_3$: *like*.

Допустим, мы решили найти векторное представление для предложения «Fight the wind». Символ *fight* находится под индексом 1 в поисковой таблице, *the* — под индексом 0, а *wind* — под индексом 2. Если необходимо найти векторное представление слова *fight* по его индексу, который равен 1, в поисковой таблице нужно определить значение соответствующего этому индексу вложения. В нашем первом примере каждое слово связано с номером, как показано на рис. 11.6.

Слово	Число
the	17
fight	22
wind	35
like	51

Рис. 11.6. Отображение символов в скаляры

В следующем листинге показывается, как задать такое отображение символов в числовые значения и наоборот, используя код TensorFlow.

Листинг 11.5. Определение поисковой таблицы скаляров

```
embeddings_0d = tf.constant([17, 22, 35, 51])
```

Слова также могут быть ассоциированы с векторами, как показано на рис. 11.7. Часто именно этот метод представления слов является предпочтительным. Руководство по векторному представлению слов можно найти в официальной документации TensorFlow: http://mng.bz/35M8.

Слово	Вектор
the	[1, 0, 0, 0]
fight	[0, 1, 0, 0]
wind	[0, 0, 1, 0]
like	[0, 0, 0, 1]

Рис. 11.7. Отображение символов в векторы

Мы можем реализовать отображение слов в векторы и наоборот так, как показано в следующем листинге.

Листинг 11.6. Определение поисковой таблицы четырехмерных векторов

```
embeddings_4d = tf.constant([[1, 0, 0, 0],
                             [0, 1, 0, 0],
                             [0, 0, 1, 0],
                             [0, 0, 0, 1]])
```

Это выглядит преувеличением, но вы можете представить символ как тензор любого ранга, а не только как число (нулевой ранг) или вектор (первый ранг). На рис. 11.8 представлено отображение символов в тензоры второго ранга.

Слово	Тензор
the	[[1, 0], [0, 0]]
fight	[[0, 1], [0, 0]]
wind	[[0, 0], [1, 0]]
like	[[0, 0], [0, 1]]

Рис. 11.8. Отображение символов в тензоры

Следующий листинг показывает, как реализовать отображение слов в тензоры с помощью TensorFlow.

Листинг 11.7. Определение таблицы преобразования тензора

```
embeddings_2x2d = tf.constant([[[1, 0], [0, 0]],
                               [[0, 1], [0, 0]],
                               [[0, 0], [1, 0]],
                               [[0, 0], [0, 1]]])
```

Функция `embedding_lookup` представляет собой оптимальный способ доступа к вложениям по индексам, как показано в следующем листинге.

Листинг 11.8. Поиск вложений

```
ids = tf.constant([1, 0, 2])  ◄—— Поиск вложений, соответствующих словам fight, the и wind
lookup_0d = sess.run(tf.nn.embedding_lookup(embeddings_0d, ids))
print(lookup_0d)

lookup_4d = sess.run(tf.nn.embedding_lookup(embeddings_4d, ids))
print(lookup_4d)

lookup_2x2d = sess.run(tf.nn.embedding_lookup(embeddings_2x2d, ids))
print(lookup_2x2d)
```

Вообще матрица встраивания (embedding matrix) не является чем-то таким, что должно быть жестко задано в коде. Эти листинги приведены только для того, чтобы описать входы и выходы функции `embedding_lookup` в библиотеке

TensorFlow, ведь скоро мы будем ею активно пользоваться. Таблица поиска встраиваний будет автоматически формироваться в ходе обучения нейронной сети. Вы начинаете с определения случайной, нормально распределенной таблицы поиска. Затем оптимизатор TensorFlow будет корректировать значения матрицы, чтобы минимизировать стоимость.

УПРАЖНЕНИЕ 11.4

Ознакомьтесь с руководством word2vec в TensorFlow, чтобы лучше изучить встраивания: www.tensorflow.org/tutorials/word2vec.

ОТВЕТ

Это руководство научит вас визуализировать встраивания с помощью TensorBoard.

11.4. Собирая все вместе

В качестве первого шага по использованию в нейронной сети в качестве входа естественного языка необходимо выбрать отображение символов в целочисленные индексы и обратно. Двумя распространенными способами числового представления предложений является последовательность *букв* и последовательность *слов*. Так как мы занимаемся с последовательностями букв, то нам потребуется построить отображение символов в целочисленные индексы и наоборот.

> **ПРИМЕЧАНИЕ** Официальный репозиторий можно найти на сайте книги (www.manning.com/books/machine-learning-with-tensorflow) и на GitHub (http://mng.bz/EB5A). Оттуда вы можете запустить код без копирования и вставки.

В следующем листинге показано, как построить отображение целых чисел в символы и наоборот. Если на вход этой функции подать набор строк, она создаст два словаря, представляющие отображение.

Листинг 11.9. Извлечение символа из словаря

```
def extract_character_vocab(data):
    special_symbols = ['<PAD>', '<UNK>', '<GO>', '<EOS>']
    set_symbols = set([character for line in data for character in line])
    all_symbols = special_symbols + list(set_symbols)
    int_to_symbol = {word_i: word
                     for word_i, word in enumerate(all_symbols)}
    symbol_to_int = {word: word_i
                     for word_i, word in int_to_symbol.items()}
    return int_to_symbol, symbol_to_int

input_sentences = ['hello stranger', 'bye bye']
output_sentences = ['hiya', 'later alligator']

input_int_to_symbol, input_symbol_to_int =
     extract_character_vocab(input_sentences)

output_int_to_symbol, output_symbol_to_int =
     extract_character_vocab(output_sentences
```

Набор входных предложений для обучения

Набор соответствующих выходных предложений для обучения

Определим теперь все наши гиперпараметры и константы в листинге 11.10. Эти значения обычно можно настроить вручную методом проб и ошибок. Как правило, большее число размерностей или слоев приводит к более сложной модели, что оправданно, если у нас большой объем данных, мощный процессор и много времени.

Листинг 11.10. Определение гиперпараметров

```
NUM_EPOCS = 300
RNN_STATE_DIM = 512
RNN_NUM_LAYERS = 2
ENCODER_EMBEDDING_DIM = DECODER_EMBEDDING_DIM = 64

BATCH_SIZE = int(32)
LEARNING_RATE = 0.0003

INPUT_NUM_VOCAB = len(input_symbol_to_int)
OUTPUT_NUM_VOCAB = len(output_symbol_to_int)
```

Число эпох

Размерность скрытого слоя сети RNN

Число уложенных друг на друга ячеек сети RNN

Размер вложения последовательности элементов для кодировщика и декодировщика

Размер пакета

Допускается использование различных словарей между кодировщиком и декодировщиком

Затем перечислим все наши переменные-заполнители. Как видно из листинга 11.11, переменные-заполнители превосходно организуют входные

и выходные последовательности, которые необходимы для обучения сети. Требуется проследить обе последовательности и их длину. Для декодировочной части необходимо также рассчитать максимальную длину последовательности. Значение `None` в форме этих переменных означает, что тензор может принимать произвольный размер. Например, размер пакета может варьироваться в каждом прогоне. Но для простоты мы будем все время сохранять размер пакета.

Листинг 11.11. Перечень переменных-заполнителей

```
# Encoder placeholders
encoder_input_seq = tf.placeholder(
    tf.int32,
     [None, None],
    name='encoder_input_seq'
)

encoder_seq_len = tf.placeholder(
    tf.int32,
     (None,),
    name='encoder_seq_len'
)
# Decoder placeholders
decoder_output_seq = tf.placeholder
    tf.int32,
     [None, None],
    name='decoder_output_seq'
)

decoder_seq_len = tf.placeholder(
    tf.int32,
     (None,),
    name='decoder_seq_len'
)

max_decoder_seq_len = tf.reduce_max(
    decoder_seq_len,
    name='max_decoder_seq_len'
)
```

Последовательность целых чисел для входа кодировщика

Форма равна размеру пакета, умноженному на длину последовательности

Длина последовательности в пакете

Форма является динамической, потому что длина последовательности может меняться

Последовательность целых чисел для выхода кодировщика

Форма равна размеру пакета, умноженному на длину последовательности

Длина последовательности в пакете

Форма является динамической, потому что длина последовательности может меняться

Максимальная длина последовательности декодера в пакете

Давайте зададим вспомогательную функцию для построения ячеек RNN. Эта функция, приведенная в следующем листинге, должна быть вам знакома из предыдущего раздела.

Листинг 11.12. Вспомогательная функция для построения ячеек сети RNN

```
def make_cell(state_dim):
    lstm_initializer = tf.random_uniform_initializer(-0.1, 0.1)
    return tf.contrib.rnn.LSTMCell(state_dim, initializer=lstm_initializer)

def make_multi_cell(state_dim, num_layers):
    cells = [make_cell(state_dim) for _ in range(num_layers)]
    return tf.contrib.rnn.MultiRNNCell(cells)
```

Мы создадим кодировщик и декодировщик ячеек RNN, используя только что заданные вспомогательные функции. Напомним, что мы скопировали модель seq2seq, представленную на рис. 11.9, для визуализации кодировщика и декодировщика RNN.

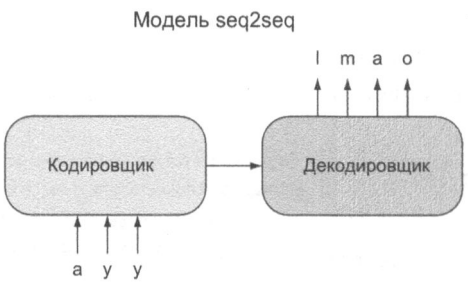


Рис. 11.9. Модель seq2seq обучается преобразованию входной последовательности в выходную последовательность с помощью кодировщика и декодировщика RNN

Сначала поговорим о ячейке кодировщика, потому что в листинге 11.13 вы создадите именно такую ячейку. Созданные состояния кодировщика RNN будут сохранены в переменной `encoder_state`. RNN также создают выходную последовательность, но нам не потребуется доступ к этой последовательности в стандартной модели seq2seq, поэтому ее можно игнорировать или удалить.

Это характерно для преобразования букв или слов в векторное представление, которое часто называют *встраиванием*. TensorFlow предоставляет удобную функцию `embed_sequence`, которая может помочь преобразовывать символы в целочисленное представление. На рис. 11.10 показано, как вход кодировщика принимает числовые значения из поисковой таблицы. Вы можете увидеть ее в действии в начале листинга 11.13.

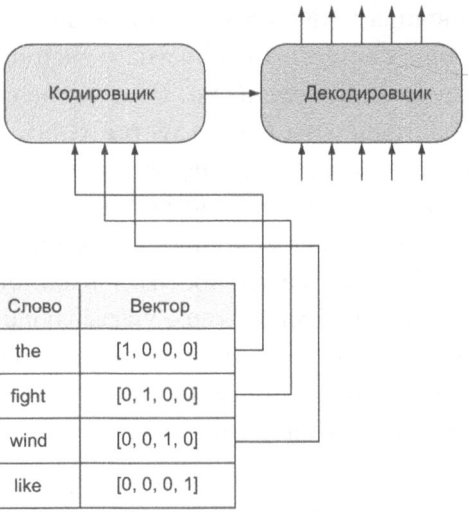


Рис. 11.10. RNN принимают в качестве входных или выходных только числовые значения, поэтому символы необходимо преобразовать в векторы. В нашем случае символами являются слова *the*, *fight*, *wind* и *like*. Их соответствующие векторы ассоциированы в матрицу встраивания

Листинг 11.13. Кодировщик встраивания и ячейки

```
# Encoder embedding

encoder_input_embedded = tf.contrib.layers.embed_sequence(
    encoder_input_seq,      ← Входная последовательность чисел (строка индексов)
    INPUT_NUM_VOCAB,        ← Строки матрицы встраивания
    ENCODER_EMBEDDING_DIM   ← Столбцы матрицы вложения
)

# Encoder output

encoder_multi_cell = make_multi_cell(RNN_STATE_DIM, RNN_NUM_LAYERS)

encoder_output, encoder_state = tf.nn.dynamic_rnn(
    encoder_multi_cell,
    encoder_input_embedded,
    sequence_length=encoder_seq_len,
    dtype=tf.float32
)

del(encoder_output)  ← Не нужно придерживаться этого значения
```

Выходом декодировщика RNN является последовательность числовых значений, представляющих естественный язык, и специальный символ, указывающий на конец последовательности. В качестве специального символа окончания последовательности используется `<EOS>`. На рис. 11.11 показан этот процесс. Входная последовательность декодера RNN будет выглядеть аналогично выходной последовательности декодера, за исключением того, что вместо специального символа конца каждого предложения, `<EOS>` (End Of Sequence — конец последовательности), у входной последовательности используется специальный символ `<GO>`, указывающий на начало последовательности. Таким образом, после того как декодировщик считает слева направо входные данные, он начинает работу без дополнительной информации об ответе, что делает его надежной моделью.

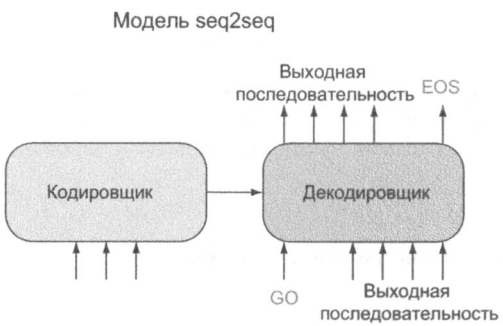


Рис. 11.11. В начале входной последовательности декодировщика используется специальный символ <GO>, а в конце выходной последовательности — <EOS>

В листинге 11.14 показано, как правильно выполнять операции разделения и конкатенации. Новую, только что составленную последовательность для входа декодировщика будем называть `decoder_input_seq`. Для объединения двух матриц используем операцию `tf.concat` из библиотеки TensorFlow.

В этом листинге задается матрица `go_prefixes`, которая будет вектором-столбцом, содержащим один только символ `<GO>`.

Листинг 11.14. Подготовка входной последовательности для декодера

```
decoder_raw_seq = decoder_output_seq[:, :-1]
go_prefixes = tf.fill([BATCH_SIZE, 1], output_symbol_to_int['<GO>'])
decoder_input_seq = tf.concat([go_prefixes, decoder_raw_seq], 1)
```

Обрезает матрицу, игнорируя самый последний столбец

Объединяет вектор с символом <GO> с началом подрезанной матрицы

Создает вектор-столбец из символов <GO>

Теперь построим ячейку декодера. Как показано в листинге 11.15, сначала делается встраивание последовательности декодировщика из целых чисел в последовательность векторов `decoder_input_ embedded`.

Встроенная версия входной последовательности подается в декодировщик RNN и создает ячейку декодировщика. И еще: нам потребуется слой, чтобы отображать выходные последовательности декодировщика в унитарное представление словаря, которое мы называем `output_layer`. Процесс настройки декодировщика начинается так же, как процесс настройки кодировщика.

Листинг 11.15. Декодировщик вложения и ячейки

```
decoder_embedding = tf.Variable(tf.random_uniform([OUTPUT_NUM_VOCAB,
                                                   DECODER_EMBEDDING_DIM]))
decoder_input_embedded = tf.nn.embedding_lookup(decoder_embedding,
                                                decoder_input_seq)

decoder_multi_cell = make_multi_cell(RNN_STATE_DIM, RNN_NUM_LAYERS)

output_layer_kernel_initializer =
    tf.truncated_normal_initializer(mean=0.0, stddev=0.1)
output_layer = Dense(
    OUTPUT_NUM_VOCAB,
    kernel_initializer = output_layer_kernel_initializer
)
```

А теперь начинаются очень странные дела. Есть два способа получить выходную последовательность декодировщика: во время обучения и во время вывода. Декодировщик обучения будет использоваться только во время обучения, тогда как декодировщик вывода будет использоваться для проверки на ранее никогда не встречавшихся данных.

Использование двух способов получения выходной последовательности позволяет во время обучения применить контрольные данные и ускорить процесс обучения. Но во время выхода у нас нет контрольных выходных меток, поэтому необходимо прибегнуть к формированию вывода, используя только входную последовательность.

В следующем листинге применяется декодер обучения. Пустим `decoder_input_seq` на вход декодировщика с помощью `TrainingHelper`. Этот вспомогательный оператор управляет входной последовательностью декодировщика RNN.

Листинг 11.16. Выходная последовательность декодировщика (обучение)

```
with tf.variable_scope("decode"):

    training_helper = tf.contrib.seq2seq.TrainingHelper(
        inputs=decoder_input_embedded,
        sequence_length=decoder_seq_len,
        time_major=False
    )

    training_decoder = tf.contrib.seq2seq.BasicDecoder(
        decoder_multi_cell,
        training_helper,
        encoder_state,
        output_layer
    )

    training_decoder_output_seq, _, _ = tf.contrib.seq2seq.dynamic_decode(
        training_decoder,
        impute_finished=True,
        maximum_iterations=max_decoder_seq_len
    )
```

Если вы хотите получить выходные данные модели seq2seq на проверочных данных, то доступа к `decoder_input_seq` больше не будет. Почему? Дело в том, что входная последовательность декодировщика получается из выходной последовательности декодировщика, которая доступна только с обучающей базой данных.

В следующем листинге показано применение оператора выхода декодера для случая вывода. И снова мы воспользуемся вспомогательным оператором для подачи входной последовательности декодировщика.

Листинг 11.17. Выходная последовательность декодировщика (вывод)

```
with tf.variable_scope("decode", reuse=True):
    start_tokens = tf.tile(
        tf.constant([output_symbol_to_int['<GO>']],
                    dtype=tf.int32),
         [BATCH_SIZE], name='start_tokens')

    inference_helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(
        embedding=decoder_embedding,
        start_tokens=start_tokens,
        end_token=output_symbol_to_int['<EOS>']
    )

    inference_decoder = tf.contrib.seq2seq.BasicDecoder(
        decoder_multi_cell,
        inference_helper,
        encoder_state,
        output_layer
    )

    inference_decoder_output_seq, _, _ = tf.contrib.seq2seq.dynamic_decode(
        inference_decoder,
        impute_finished=True,
        maximum_iterations=max_decoder_seq_len
    )
```

Вспомогательная функция для процесса вывода

Базовый декодировщик

Выполняет динамическое декодирование с помощью декодировщика

Рассчитаем стоимость с помощью метода `sequence_loss` TensorFlow. Вам потребуется доступ к выходной последовательности декодера при выводе и к выходной последовательности при работе с контрольными данными. В следующем листинге определяется функция стоимости.

Листинг 11.18. Функция стоимости

```
training_logits =
    tf.identity(training_decoder_output_seq.rnn_output, name='logits')
inference_logits =
    tf.identity(inference_decoder_output_seq.sample_id, name='predictions')

masks = tf.sequence_mask(
    decoder_seq_len,
    max_decoder_seq_len,
    dtype=tf.float32,
    name='masks'
)
```

Для удобства переименовывает тензоры

Создает весовые коэффициенты для sequence_loss

```
cost = tf.contrib.seq2seq.sequence_loss(
    training_logits,
    decoder_output_seq,
    masks
)
```

Использует функцию потерь из библиотеки TensorFlow

И наконец, вызовем оптимизатор минимизации стоимости. Проделаем еще один трюк, который вы раньше, скорее всего, не видели. В глубоких нейронных сетях нужно ограничивать сильные изменения градиента; воспользуемся для этого методом *градиентной обрезки* (gradient clipping). В листинге 11.19 показано, как это сделать.

УПРАЖНЕНИЕ 11.5

Попробуйте запустить модель seq2seq без градиентной обрезки, чтобы почувствовать разницу.

ОТВЕТ

Без градиентной обрезки сеть иногда настраивает градиент на слишком большое значение, что приводит к численной неустойчивости.

Листинг 11.19. Вызов оптимизатора

```
optimizer = tf.train.AdamOptimizer(LEARNING_RATE)

gradients = optimizer.compute_gradients(cost)
capped_gradients = [(tf.clip_by_value(grad, -5., 5.), var)  ◄── Градиентная обрезка
                        for grad, var in gradients if grad is not None]

train_op = optimizer.apply_gradients(capped_gradients)
```

Этот код завершает реализацию модели `seq2seq`. В целом модель готова к обучению после того, как вы настроили оптимизатор (предыдущий листинг). Можно создать сеанс и запустить оператор `train_op` с пакетами обучающих данных, чтобы определить оптимальные параметры модели.

Кстати, вам же нужны обучающие данные! Как мы можем получить тысячи пар входных и выходных предложений? Не переживайте — в следующем разделе мы это выясним!

11.5. Сбор данных диалога

Cornell Movie Dialogues Corpus (http://mng.bz/W28O) является базой данных, состоящей из более чем 220 000 диалогов из более чем 600 кинофильмов. Вы можете скачать zip-файл с официальной страницы.

> **ПРЕДОСТЕРЕЖЕНИЕ** Поскольку объем имеющихся там данных большой, то следует ожидать, что процесс обучения займет длительное время. Если TensorFlow настроена на работу только с использованием центрального процессора, то обучение может занять целый день. При использовании графического процессора обучение может занять от 30 минут до часа.

Ниже приводится пример небольшого фрагмента диалога двух людей (*A* и *B*):

A: Они этого не делают!

B: Они тоже!

A: Превосходно.

Поскольку цель чат-бота заключается в осмысленных ответах на каждое возможное высказывание, то мы структурируем наши обучающие данные на основе взаимозависимых пар разговора. В нашем примере диалог создает следующие пары входящих и выходящих предложений:

- «Они этого не делают!» → «Они тоже!»
- «Они тоже» → «Превосходно».

Для удобства мы уже обработали данные и сделали их доступными в интернете. Вы можете их найти на сайте www.manning.com/books/machine-learning-with-tensorflow или http://mng.bz/wWo0. Завершив загрузку, выполните код из следующего листинга, где используется вспомогательная функция `load_sentences` из архива GitHub в `Concept03_seq2seq.ipynb` Jupyter Notebook.

Листинг 11.20. Обучение модели

```
input_sentences = load_sentences('data/words_input.txt')
output_sentences = load_sentences('data/words_output.txt')

input_seq = [
     [input_symbol_to_int.get(symbol, input_symbol_to_int['<UNK>'])
        for symbol in line]
    for line in input_sentences
]

output_seq = [
     [output_symbol_to_int.get(symbol, output_symbol_to_int['<UNK>'])
        for symbol in line] + [output_symbol_to_int['<EOS>']]
    for line in output_sentences
]

sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
saver = tf.train.Saver()

for epoch in range(NUM_EPOCS + 1):

    for batch_idx in range(len(input_sentences) // BATCH_SIZE):

        input_data, output_data = get_batches(input_sentences,
                                              output_sentences,
                                              batch_idx)

        input_batch, input_lenghts = input_data[batch_idx]
        output_batch, output_lengths = output_data[batch_idx]

        _, cost_val = sess.run(
            [train_op, cost],
            feed_dict={
                encoder_input_seq: input_batch,
                encoder_seq_len: input_lengths,
                decoder_output_seq: output_batch,
                decoder_seq_len: output_lengths
            }
        )

saver.save(sess, 'model.ckpt')
sess.close()
```

Загружает входные предложения в виде списка строк

Загружает соответствующие выходные предложения

Циклически проходит по буквам

Циклически проходит по строкам текста

Циклически проходит по строкам текста

Добавляет символ EOS в конец выходных данных

Желательно сохранить полученные при обучении параметры

Циклически проходит по всем итерациям

Выполняет цикл по номерам пакетов

Получает входные и выходные пары

Выполняет оптимизацию текущего пакета

Поскольку вы сохранили параметры модели в файл, теперь можно загрузить их в любую программу и запросить у сети ответ на новые входные данные. Запустим оператор `inference_logits`, чтобы получить ответ от чат-бота.

11.6. Краткие итоги

В этой главе вы построили реальный образец модели seq2seq, где реализуются все возможности TensorFlow, с которыми вы познакомились в предыдущих главах:

- вы научились встраивать в TensorFlow естественный язык;
- использовали RNN в качестве строительного блока для более интересной модели;
- после обучения модели на примерах диалога из киносценариев вы смогли обучить алгоритм чат-бота и получить ответы на вопросы, используя естественной язык.

Ландшафт полезности

Эта глава охватывает следующие темы:

- ✓ Реализация нейронной сети для ранжирования
- ✓ Встраивание изображения с использованием VGG16
- ✓ Визуализация полезности

Домашнему роботу-пылесосу, например, Roomba, необходимы датчики, чтобы «видеть» мир. Способность робота обрабатывать получаемые сенсором данные позволяет роботу настраивать модель окружающего его мира. Мебель в комнате могут двигать каждый день, поэтому робот должен обладать способностью адаптироваться к хаотичной среде.

Допустим, у вас есть фантастический робот-домработница, который обладает несколькими основными навыками, но, помимо этого, способен обучиться новым навыкам, которые показывает человек. Например, вам хотелось бы обучить робота складывать одежду.

Обучение робота выполнению нового задания — достаточно сложная задача. В голову сразу приходит несколько вопросов:

- Должен ли робот просто подражать действиям человека? Такой процесс относится к *имитационному обучению* (imitation learning).
- Как могут руки и сочленения робота соответствовать позам человека? Эту дилемму часто называют *проблемой соответствия* (correspondence problem).

УПРАЖНЕНИЕ 12.1

Целью имитационного обучения робота является научить его воспроизводить последовательность действий, которые показывает человек. На бумаге это выглядит красиво, но какие существуют ограничения у этого метода?

ОТВЕТ

Имитация человеческих действий на основе человеческого поведения является примитивным подходом к обучению. Вместо этого агент должен идентифицировать скрытую за демонстрацией действий цель. Например, цель складывания одежды состоит в придании ей плоской формы с последующим сжатием, что не зависит от движений рук человека. Понимая, почему человек выполняет определенную последовательность действий, агент способен лучше обобщать навык, которому его обучают.

В этой главе вы будете моделировать задачу демонстрации человеком определенных действий, избегая имитационного обучения и проблемы соответствия. Вам повезло! Вы сможете добиться этого, изучив способ ранжирования состояний мира с помощью *функции полезности* (utility function), которая берет состояние и возвращает ему действительное значение, представляющее привлекательность этого состояния. Вы не только сможете избежать подражания как меры успеха, но и также обойдете сложность отображения набора действий робота в соответствующие действия человека (проблема соответствия).

В следующем разделе вы научитесь применять функцию полезности к состояниям мира, полученным с помощью видеороликов, где человек демонстрирует выполнение поставленной задачи. Обученная функция полезности является моделью предпочтений.

Мы используем задачу обучения робота складывать предметы одежды. Мятые предметы одежды почти наверняка находятся в конфигурации, которую мы никогда раньше не видели. Как показано на рис. 12.1, фреймворк полезности не имеет ограничений на размер пространства состояний. Модель предпочтения обучается специальным образом на видеороликах, где люди различным образом складывают футболки.

Функция полезности обобщает все состояния (мятая футболка в новой конфигурации по сравнению со сложенной футболкой в знакомой конфигурации) и повторно использует знания по всем типам одежды (складывание футболки по сравнению со складыванием брюк).

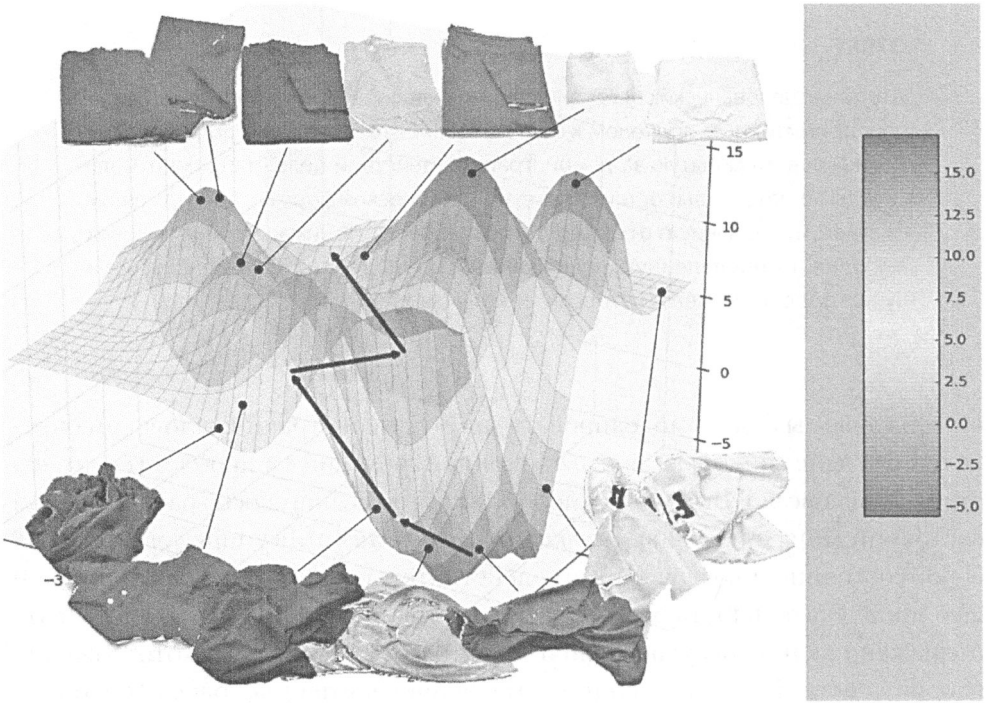


Рис. 12.1. Мятая одежда представляет собой изображение менее благоприятного состояния, чем хорошо сложенная одежда. Эта диаграмма показывает, как можно оценивать каждое состояние материи; более высокая оценка представляет более предпочтительное состояние

Мы можем дальше проиллюстрировать практические применения функции полезности следующим аргументом: в реальном мире не все визуальные наблюдения оптимизированы в отношении задачи обучения. Учитель, демонстрирующий тот или иной навык, может выполнять неуместные, неполные или даже неправильные действия, но, невзирая на это, люди способны игнорировать ошибки.

Когда робот наблюдает за демонстрируемыми человеком действиями, вам нужно, чтобы он понял причинно-следственные связи, которые входят в достижение задачи. Ваша работа позволяет фазе обучения быть интерактивной, когда робот скептически относится к поведению человека, чтобы уточнить данные обучения.

Для этого вы сначала изучите функцию полезности на небольшом количестве видеороликов для ранжирования предпочтений различных состояний. Затем, когда роботу будет показан новый пример навыка с помощью демонстрации, он обращается к функции полезности, чтобы убедиться, что ожидаемая полезность увеличивается с течением времени. Наконец, робот прерывает демонстрацию, чтобы спросить, важно ли это действие для изучения навыка.

12.1. Модель предпочтения

Мы полагаем, что человеческие предпочтения основаны на *утилитарных* соображениях, означающих число, которое определяет ранг элемента. Например, вы опросили людей, чтобы оценить «вкусность» различных продуктов (стейк, хот-дог, коктейль из креветок и бургер).

На рис. 12.2 приводится пара возможных вариантов попарного ранжирования еды. Можно ожидать, что стейк имеет ранг выше, чем хот-дог, а коктейль из креветок имеет ранг выше, чем бургер, по шкале вкусности.

Рис. 12.2. Это возможный набор попарного ранжирования объектов. В частности, у вас есть четыре типа еды, и вы хотите ранжировать их по вкусу, поэтому используете два решения попарного ранжирования: стейк более вкусная еда, чем хот-дог, а коктейль из креветок более вкусная еда, чем бургер

К счастью для опрошенных людей, не каждая пара еды должна быть оценена. Например, не так очевидно, что является более вкусным — хот-дог и гамбургер или стейк и коктейль из креветок. Есть повод для разногласий.

Если состояние s_1 имеет более высокую полезность, чем состояние s_2, тогда соответствующее ранжирование обозначается как $s_1 > s_2$, подразумевая, что полезность состояния s_1 выше, чем полезность состояния s_2. Каждый ролик с демонстрацией содержит последовательность n состояний $s_0, s_1, .., s_n$, которая дает $n(n - 1)/2$ возможных упорядоченных пар ограничений ранжирования. Давайте воспользуемся нашей собственной нейронной сетью, способной выполнить ранжирование. Откроем новый исходный файл и воспользуемся следующим листингом для импорта соответствующих библиотек. Мы подходим к тому, чтобы создать нейронную сеть для обучения функции полезности на основе пар предпочтений.

Листинг 12.1. Импортирование библиотек

```
import tensorflow as tf
import numpy as np
import random

%matplotlib inline
import matplotlib.pyplot as plt
```

Для обучения нейронной сети ранжированию состояний на основе оценок полезности нам потребуются обучающие данные. Давайте создадим фиктивные данные, чтобы с чего-то начать. Мы заменим их впоследствии реалистичными данными. Воспроизведем двумерные данные на рис. 12.3, используя листинг 12.2.

Листинг 12.2. Генерирование фиктивных обучающих данных

Мы создадим двумерные данные, такие, чтобы их можно было легко визуализировать

Это набор точек, которые должны дать более высокую полезность

Это набор менее предпочтительных точек

```
n_features = 2

def get_data():
    data_a = np.random.rand(10, n_features) + 1
    data_b = np.random.rand(10, n_features)

    plt.scatter(data_a[:, 0], data_a[:, 1], c='r', marker='x')
    plt.scatter(data_b[:, 0], data_b[:, 1], c='g', marker='o')
    plt.show()

    return data_a, data_b

data_b data_a, data_b = get_data()
```

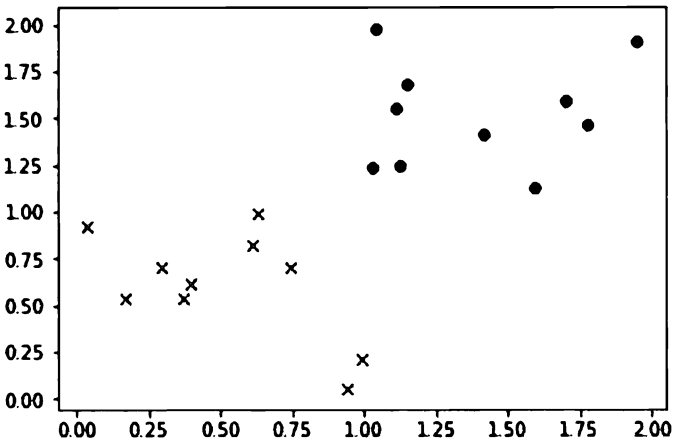


Рис. 12.3. Пример точек, с которыми мы будем работать. Кружки представляют более предпочтительные состояния, а крестики — менее предпочтительные состояния. У нас есть равное число кружков и крестиков, потому что данные приходят парами; каждая пара ранжируется как на рис. 12.2

Затем нам необходимо задать гиперпараметры. Сохраним эту модель простой, используя неглубокую архитектуру. Мы создадим сеть только с одним скрытым слоем. Соответствующий гиперпараметр, который задает число нейронов в скрытом слое, следующий:

```
n_hidden = 10
```

Ранжирующая нейронная сеть будет получать попарно данные, поэтому нам потребуются две отдельные переменные-заполнители, по одной на каждую часть пары. Кроме того, мы создадим переменную-заполнитель для удержания значения параметра `dropout`. Добавим свой скрипт.

Листинг 12.3. Переменные-заполнители

```
with tf.name_scope("input"):
    x1 = tf.placeholder(tf.float32, [None, n_features], name="x1")
    x2 = tf.placeholder(tf.float32, [None, n_features], name="x2")
    dropout_keep_prob = tf.placeholder(tf.float32, name='dropout_prob')
```

Входная переменная-заполнитель для предпочтительных точек

Входная переменная-заполнитель для непредпочтительных точек

Сеть будет содержать только один скрытый слой. В следующем листинге задаются веса и смещения, а затем они используются повторно на каждой из двух входных переменных-заполнителей.

Листинг 12.4. Скрытый слой

```
with tf.name_scope("hidden_layer"):
    with tf.name_scope("weights"):
        w1 = tf.Variable(tf.random_normal([n_features, n_hidden]), name="w1")
        tf.summary.histogram("w1", w1)
        b1 = tf.Variable(tf.random_normal([n_hidden]), name="b1")
        tf.summary.histogram("b1", b1)

    with tf.name_scope("output"):
        h1 = tf.nn.dropout(tf.nn.relu(tf.matmul(x1,w1) + b1),
     keep_prob=dropout_keep_prob)
        tf.summary.histogram("h1", h1)
        h2 = tf.nn.dropout(tf.nn.relu(tf.matmul(x2, w1) + b1),
     keep_prob=dropout_keep_prob)
        tf.summary.histogram("h2", h2)
```

Цель нейронной сети — вычислить оценку для двух предоставленных входов. В следующем листинге задаются веса, значения смещений и полносвязная архитектура выходного слоя сети. Останутся два выходных вектора, `s1` и `s2`, представляющие оценки для попарного входа.

Листинг 12.5. Выходной слой

```
with tf.name_scope("output_layer"):
    with tf.name_scope("weights"):
        w2 = tf.Variable(tf.random_normal([n_hidden, 1]), name="w2")
        tf.summary.histogram("w2", w2)
        b2 = tf.Variable(tf.random_normal([1]), name="b2")
        tf.summary.histogram("b2", b2)

    with tf.name_scope("output"):
        s1 = tf.matmul(h1, w2) + b2   <---  Оценка полезности входа x1
        s2 = tf.matmul(h2, w2) + b2   <---  Оценка полезности входа x2
```

Предположим, что при обучении сети `x1` должен содержать менее благоприятные элементы. Это значит, что `s1` должен быть оценен ниже, чем `s2`, а разница между `s1` и `s2` должна быть отрицательной. В следующем листинге показано,

что функция потерь пытается гарантировать отрицательную разницу, используя многоклассовую кросс-энтропийную функцию потерь. Определим оператор `train_op` для минимизации функции потерь.

Листинг 12.6. Потери и их оптимизация

```
with tf.name_scope("loss"):
    s12 = s1 - s2
    s12_flat = tf.reshape(s12, [-1])

    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
                        labels=tf.zeros_like(s12_flat),
                        logits=s12_flat + 1)

    loss = tf.reduce_mean(cross_entropy)
    tf.summary.scalar("loss", loss)

with tf.name_scope("train_op"):
    train_op = tf.train.AdamOptimizer(0.001).minimize(loss)
```

Воспользуемся листингом 12.7 для настройки сеанса TensorFlow. Для этого инициализируйте все переменные и подготовьте отладку TensorBoard, используя процедуру записи сводки.

> **ПРИМЕЧАНИЕ** Мы уже пользовались этой процедурой раньше, в конце главы 2, при первом знакомстве с TensorBoard.

Листинг 12.7. Подготовка сеанса

```
sess = tf.InteractiveSession()
summary_op = tf.summary.merge_all()
writer = tf.summary.FileWriter("tb_files", sess.graph)
init = tf.global_variables_initializer()
sess.run(init)
```

Теперь для обучения сети все готово! Запустите оператор `train_op` на фиктивных данных, которые были созданы для получения в ходе обучения модели ее оптимальных параметров.

Листинг 12.8. Обучение сети

```
for epoch in range(0, 10000):
    loss_val, _ = sess.run([loss, train_op], feed_dict={x1:data_a, x2:data_b,
     dropout_keep_prob:0.5})
    if epoch % 100 == 0 :
        summary_result = sess.run(summary_op,
                                  feed_dict={x1:data_a,
                                             x2:data_b,
                                             dropout_keep_prob:1})
        writer.add_summary(summary_result, epoch)
```

В результате обучения дропаут keep_prob равен 0,5

Предпочтительные точки

Непредпочтительные точки

При проверке дропаута keep_prob всегда должен быть единицей

И наконец, визуализируйте обученную функцию оценки. Добавьте двумерные точки в список.

Листинг 12.9. Подготовка проверочных данных

```
grid_size = 10 data_test = []
for y in np.linspace(0., 1., num=grid_size):
    for x in np.linspace(0., 1., num=grid_size):
        data_test.append([x, y])
```

Циклически перебирает все строки

Циклически перебирает все столбцы

Запустите оператор `s1`, используя проверочные данные, чтобы получить значения полезности каждого состояния, и визуализируйте его, как показано на рис. 12.4. Для создания визуализации воспользуйтесь кодом из листинга 12.10.

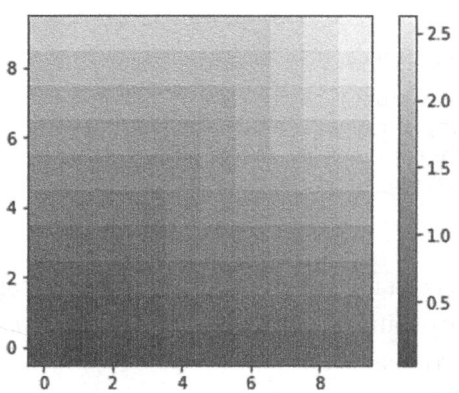


Рис. 12.4. Ландшафт оценок, полученный в результате обучения ранжирующей нейронной сети

Листинг 12.10. Визулизация результатов

```
def visualize_results(data_test):
    plt.figure()
    scores_test = sess.run(s1, feed_dict={x1:data_test, dropout_keep_prob:1})
    scores_img = np.reshape(scores_test, [grid_size, grid_size])
    plt.imshow(scores_img, origin='lower')
    plt.colorbar()

visualize_results(data_test)
```

Вычисляет полезность всех точек

Преобразует полезность в матрицу, чтобы можно было получить изображение, используя Matplotlib

12.2. Встраивание изображения

В главе 11 вы решились подать на вход нейронной сети несколько предложений на естественном языке. Вы сделали это, преобразовав слова или буквы предложений в числовые формы, такие как векторы. Например, каждый символ (будь то слово или буква) был встроен в вектор с помощью поисковой таблицы.

УПРАЖНЕНИЕ 12.2

Почему поисковую таблицу, которая преобразует символы в векторное представление, называют матрицей встраиваний?

ОТВЕТ

Символы встраиваются в векторное пространство.

К счастью, изображения уже в числовой форме. Они представлены в виде матрицы пикселов. Если изображение в оттенках серого, пикселы, вероятно, принимают скалярные значения, указывающие на яркость. Для цветных изображений каждый пиксел представляет интенсивность цвета (обычно три: красный, зеленый и синий). В любом случае изображение легко может быть представлено числовыми структурами данных, такими как тензор, в TensorFlow.

УПРАЖНЕНИЕ 12.3

Возьмите фото домашнего объекта, например стула. Уменьшайте изображение до тех пор, пока возможность идентифицировать объект не исчезнет. На каком коэффициенте уменьшения изображения вы остановились? Каково соотношение количества пикселов в исходном изображении к числу пикселов на уменьшенном изображении? Это соотношение является грубой мерой избыточности данных.

ОТВЕТ

Типичная 5-мегапикселная камера делает фото с разрешением 2560 × 1920, но это изображение еще может быть распознаваемым, если вы уменьшите его в 40 раз (разрешение 64 × 48).

Подача в нейронную сеть большого изображения, скажем, размером 1280 × 720 (почти 1 миллион пикселов) увеличивает количество параметров и, как следствие, увеличивает риск переобучения модели. Пикселы изображения избыточны, поэтому можно попытаться каким-то образом уловить суть изображения в более сжатом представлении. На рис. 12.5 показаны кластеры, образованные в двумерном вложении изображений сложенной одежды.

В главе 7 вы узнали, как использовать автокодировщики для уменьшения размерности изображений. Другим распространенным способом выполнить низкоразмерное встраивание изображений является использование предпоследнего слоя глубокого классификатора сверточной нейронной сети. Давайте рассмотрим это подробнее.

Поскольку проектирование, внедрение и обучение глубокого классификатора изображений не является основным направлением этой главы (см. главу 9 про CNN), то вы будете использовать нестандартную модель. Распространенный классификатор, который упоминается во многих документах по исследованию машинного зрения, называется VGG16.

Для TensorFlow существует множество онлайн-реализаций VGG16. Мы рекомендуем использовать реализацию Дэви Фроссара (Davi Frossard) (www.cs.toronto.edu/~frossard/post/vgg16/). Вы можете скачать файл vgg16.py с кодом TensorFlow, а также предварительно обученные параметры модели `vgg16_weights.npz` с его

сайта, или с сайта книги (www.manning.com/books/machine-learning-with-tensorflow), или в GitHub (https://github.com/ BinRoot/TensorFlow-Book).

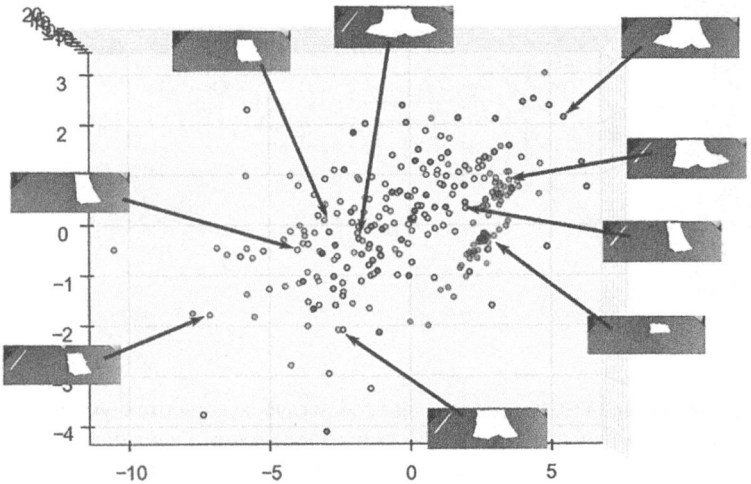


Рис. 12.5. Ландшафт полезности. Изображения могут быть встроены в гораздо более малые размеры, такие как 2D, как показано здесь. Обратите внимание, что точки, представляющие схожие состояния футболки, встречаются в соседних кластерах. Встраивание изображений позволяет использовать ранжирующую нейронную сеть, чтобы узнать предпочтение между состояниями ткани

На рис. 12.6 изображена нейронная сеть VGG16 со страницы Фроссара. Как мы можем видеть, это глубокая нейронная сеть с множеством сверточных слоев. Несколько последних слоев обычно являются полносвязными слоями, а выходной слой представляет собой 1000-мерный вектор, указывающий на вероятности многоклассовой классификации.

Ориентироваться в чужом коде — важный навык. Прежде всего, убедитесь, что вы загрузили vgg16.py и vgg16_weights.npz, и проверьте, что можете запускать код с помощью `python vgg16.py my_image.png`.

ПРИМЕЧАНИЕ Возможно, вам понадобится установить SciPy и Pillow, чтобы получить демокод VGG16 и запускать его без проблем. Вы можете скачать их через `pip`.

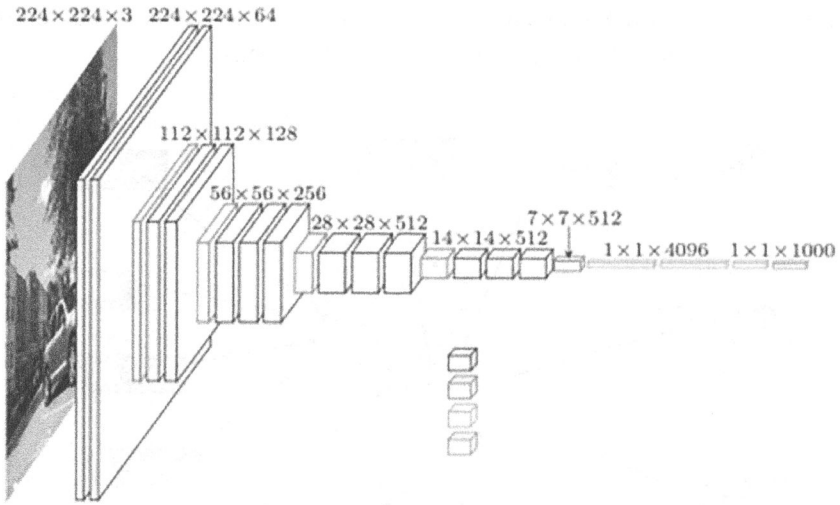


Рис. 12.6. Архитектура VGG16 представляет собой глубокую сверточную нейронную сеть, которая используется для классификации изображений. Схема позаимствована с сайта www.cs.toronto.edu/~frossard/post/vgg16/

Начнем с добавления интеграции TensorBoard для происходящего в коде. В основную функцию после создания переменной сеанса `sess` вставьте следующую строку:

```
my_writer = tf.summary.FileWriter('tb_files', sess.graph)
```

Теперь снова запустим классификатор (`python vgg16.py my_image.png`), в результате чего будет создан каталог tb_files, который будет использоваться TensorBoard. Вы можете запустить TensorBoard, чтобы визуализировать вычисления нейронной сети по графу. Следующая команда запускает TensorBoard:

```
$ tensorboard --logdir=./logs
```

Откройте TensorBoard в браузере и перейдите на вкладку Graphs, чтобы увидеть граф вычислений, как показано на рис. 12.7. Обратите внимание, что при первом же ознакомлении вы сможете сразу получить представление о типах слоев, участвующих в сети: последние три слоя являются полностью полносвязными слоями, отмеченными как fc1, fc2 и fc3.

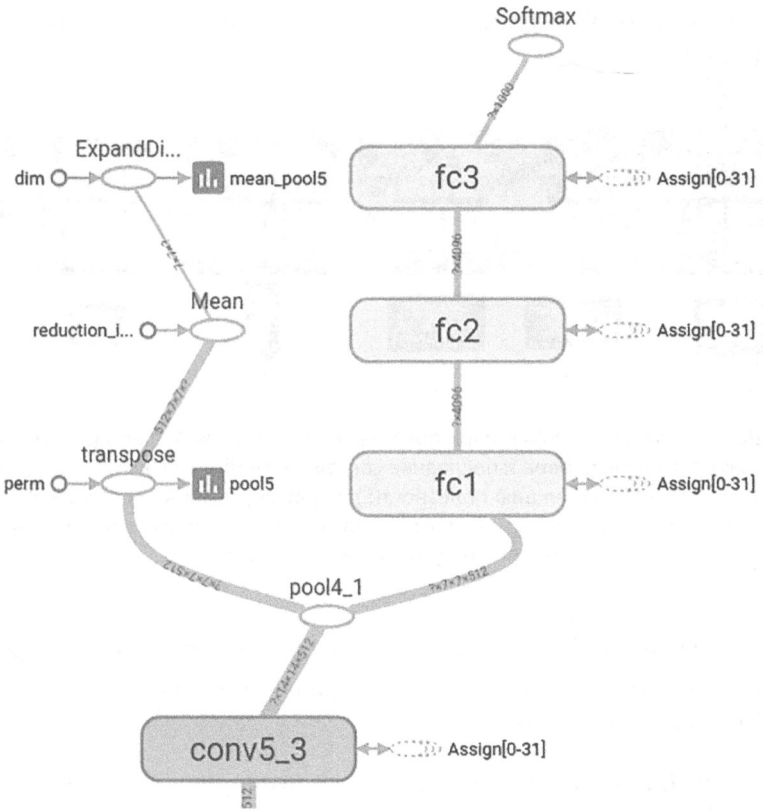


Рис. 12.7. Небольшой сегмент графа вычислений в TensorBoard для нейронной сети VGG16. Самый верхний узел является многоклассовым оператором, применяемым для классификации. Три полносвязных слоя отмечены как fc1, fc2 и fc3

12.3. Ранжирование изображений

Вы будете использовать код VGG16 из предыдущего раздела, чтобы получить векторное представление изображения. Таким образом, вы эффективно можете ранжировать два изображения в нейронной сети, разработанной в разделе 12.1.

Рассмотрим видео складывания футболки, как показано на рис. 12.8. Вы будете обрабатывать видео по кадрам для ранжирования состояний изображений.

Таким образом, в новой ситуации алгоритм может понять, достигнута ли цель складывания.

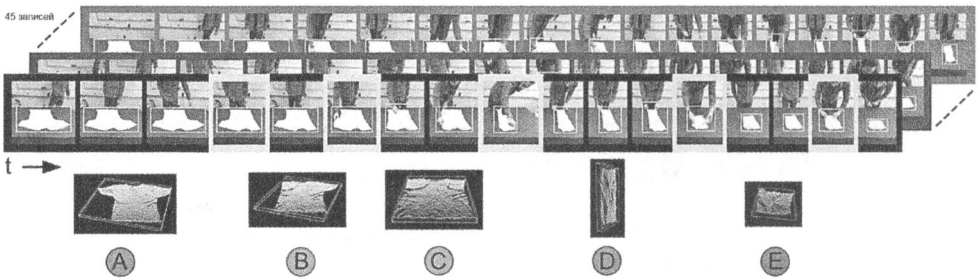


Рис. 12.8. Видео складывания футболки показывает, как изменяется во времени положение ткани. Вы можете извлечь первое и последнее состояния футболки в качестве данных для обучения, чтобы определить функцию полезности для ранжирования состояний. Конечные состояния футболки в каждом видео должны оцениваться с более высокой полезностью, чем состояние футболки в начале видео

Для начала скачайте набор данных по складыванию одежды (сайт http://mng.bz/eZsc). Разархивируйте набор. Запомните, куда вы его извлекаете; назовите его DATASET_DIR, как в листингах.

Откройте новый исходный файл и начните с импорта соответствующих библиотек в Python.

Листинг 12.11. Импортирование библиотек

```
import tensorflow as tf
import numpy as np
from vgg16 import vgg16
import glob, os
from scipy.misc import imread, imresize
```

Для каждого видео запомните первый и последний кадр. Таким образом вы сможете обучить алгоритм ранжирования, исходя из того, что последний кадр имеет более высокое предпочтение, чем первый. Другими словами, последнее состояние складывания ткани приводит вас к более высокому состоянию, чем первое состояние. В следующем листинге показан пример загрузки данных в память.

Листинг 12.12. Подготовка данных для обучения

```
DATASET_DIR = os.path.join(os.path.expanduser('~'), 'res',
     'cloth_folding_rgb_vids')  ◄—— Директория загруженных файлов
NUM_VIDS = 45  ◄—— Число видео для загрузки

def get_img_pair(video_id):  ◄—— Берет начальный и конечный кадры видеозаписи
    img_files = sorted(glob.glob(os.path.join(DATASET_DIR, video_id,
     '*.png')))
    start_img = img_files[0]
    end_img = img_files[-1]
    pair = []

    for image_file in [start_img, end_img]:
        img_original = imread(image_file)
        img_resized = imresize(img_original, (224, 224))
        pair.append(img_resized)
    return tuple(pair)

start_imgs = []
end_imgs= []
for vid_id in range(1, NUM_VIDS + 1):
    start_img, end_img = get_img_pair(str(vid_id))
    start_imgs.append(start_img)
    end_imgs.append(end_img)
print('Images of starting state {}'.format(np.shape(start_imgs)))
print('Images of ending state {}'.format(np.shape(end_imgs)))
```

Запуск листинга 12.12 дает следующий результат:

```
Images of starting state (45, 224, 224, 3)
Images of ending state (45, 224, 224, 3)
```

Воспользуемся следующим листингом, чтобы создать входную переменную-заполнитель для изображения, которое затем будет встроено.

Листинг 12.13. Переменная-заполнитель

```
imgs_plc = tf.placeholder(tf.float32, [None, 224, 224, 3])
```

Скопируем код ранжирующей сети из листингов 12.3–12.7; мы снова воспользуемся ею для ранжирования изображений. Затем подготовим сеанс, как показано в следующем листинге.

Листинг 12.14. Подготовка сеанса

```
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
```

Затем инициализируем модель VGG16, вызвав конструктор. При этом с диска в память загружаются все параметры модели.

Листинг 12.15. Загрузка модели VGG16

```
print('Loading model...')
vgg = vgg16(imgs_plc, 'vgg16_weights.npz', sess)
print('Done loading!')
```

Теперь давайте подготовим обучающие и проверочные данные. Как показано в листинге 12.16, вы скомбинируете модель VGG16 с изображениями, а затем получите доступ к слою рядом с выходом (в данном случае — fc1), чтобы получить встраивание изображения.

В результате у нас будет 4096-мерное встраивание исходных изображений. Из всех 45 видеозаписей одну часть используем для обучения, а другую — для проверки:

- Обучение
  - Размер начального кадра: (33, 4096)
  - Размер конечного кадра: (33, 4096)
- Проверка
  - Размер начального кадра: (12, 4096)
  - Размер конечного кадра: (12, 4096)

Листинг 12.16. Подготовка данных к ранжированию

```
start_imgs_embedded = sess.run(vgg.fc1, feed_dict={vgg.imgs: start_imgs})
end_imgs_embedded = sess.run(vgg.fc1, feed_dict={vgg.imgs: end_imgs})

idxs = np.random.choice(NUM_VIDS, NUM_VIDS, replace=False)
```

```
train_idxs = idxs[0:int(NUM_VIDS * 0.75)]
test_idxs = idxs[int(NUM_VIDS * 0.75):]

train_start_imgs = start_imgs_embedded[train_idxs]
train_end_imgs = end_imgs_embedded[train_idxs]
test_start_imgs = start_imgs_embedded[test_idxs]
test_end_imgs = end_imgs_embedded[test_idxs]

print('Train start imgs {}'.format(np.shape(train_start_imgs)))
print('Train end imgs {}'.format(np.shape(train_end_imgs)))
print('Test start imgs {}'.format(np.shape(test_start_imgs)))
print('Test end imgs {}'.format(np.shape(test_end_imgs)))
```

С подготовленными обучающими данными выполним оператор `train_op` такое количество раз, которое равно `epoch`. После обучения сети запустите модель на проверочных данных для оценки результата.

Листинг 12.17. Обучение ранжирующей сети

```
train_y1 = np.expand_dims(np.zeros(np.shape(train_start_imgs)[0]), axis=1)
train_y2 = np.expand_dims(np.ones(np.shape(train_end_imgs)[0]), axis=1)
for epoch in range(100):
    for i in range(np.shape(train_start_imgs)[0]):
        _, cost_val = sess.run([train_op, loss],
                               feed_dict={x1: train_start_imgs[i:i+1,:],
                               x2: train_end_imgs[i:i+1,:],
                               dropout_keep_prob: 0.5})
    print('{}. {}'.format(epoch, cost_val))
    s1_val, s2_val = sess.run([s1, s2], feed_dict={x1: test_start_imgs,
                                                   x2: test_end_imgs,
                                                   dropout_keep_prob: 1})
print('Accuracy: {}%'.format(100 * np.mean(s1_val < s2_val)))
```

Обратите внимание, что точность со временем приближается к 100 %. Наша ранжирующая модель научилась понимать, что кадры в конце видео более предпочтительны, чем кадры в начале.

Просто из любопытства посмотрим изменение полезности на протяжении одного видео, покадрово, как показано на рис. 12.9. Код для воспроизведения рис. 12.9 нуждается в загрузке всех изображений в видео, как показано в листинге 12.18.

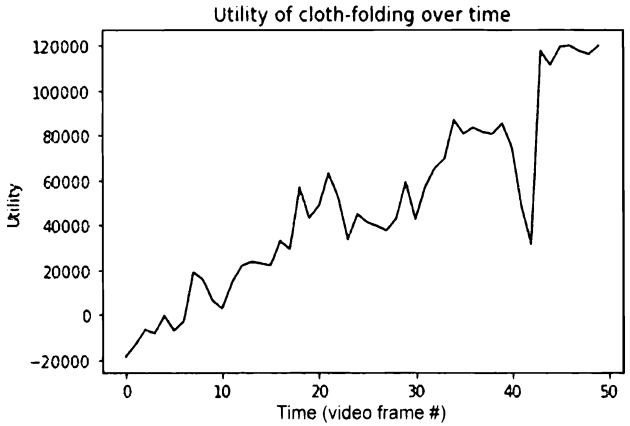


Рис. 12.9. Полезность со временем увеличивается, указывая на то, что цель была достигнута. Полезность состояния одежды в начале видео близка к нулю, но значительно возрастает до 120 000 единиц — к концу видео

Листинг 12.18. Подготовка последовательности изображений из видеозаписи

```
def get_img_seq(video_id):
    img_files = sorted(glob.glob(os.path.join(DATASET_DIR, video_id,
     '*.png')))
    imgs = []
    for image_file in img_files:
        img_original = imread(image_file)
        img_resized = imresize(img_original, (224, 224))
        imgs.append(img_resized)
    return imgs

imgs = get_img_seq('1')
```

Для встраивания изображений вы можете использовать свою модель VGG16, а затем запустить рейтинговую сеть для вычисления оценок, как показано в следующем листинге.

Листинг 12.19. Вычисление полезности изображений

```
imgs_embedded = sess.run(vgg.fc1, feed_dict={vgg.imgs: imgs})
scores = sess.run([s1], feed_dict={x1: imgs_embedded,
                                   dropout_keep_prob: 1})
```

Визуализируйте полученные результаты, чтобы воспроизвести рис. 12.9.

C - !

Листинг 12.20. Визуализация оценок полезности

```
from matplotlib import pyplot as plt
plt.figure()
plt.title('Utility of cloth-folding over time')
plt.xlabel('time (video frame #)')
plt.ylabel('Utility')
plt.plot(scores[-1])
```

12.4. Краткие итоги

- Вы можете ранжировать состояния, представляя объекты как векторы и обучая функцию полезности на этих векторах.
- Поскольку изображения содержат избыточные данные, то вы использовали нейронную сеть VGG16, чтобы уменьшить размерность данных и использовать ранжирующую сеть с изображениями из реального мира.
- Вы научились визуализировать полезность изображений видео во времени, чтобы убедиться, что демонстрация увеличивает полезность.

Вы закончили путешествие по TensorFlow! На протяжении 12 глав этой книги мы рассматривали машинное обучение под разными углами; но вместе они научили вас концепциям, необходимым для овладения следующими навыками:

- формулировка произвольной задачи реального мира в рамках машинного обучения;
- понимание основ многих проблем машинного обучения;
- использование TensorFlow для решения этих проблем;
- визуализация алгоритма машинного обучения и обучение разговору на определенном языке.

12.5. Что дальше?

Концепции, с которыми знакомит эта книга, выходят за рамки одного только машинного обучения; это же относится и к коду в приведенных здесь листингах. Мы активно поддерживаем репозиторий GitHub, чтобы вы всегда могли обратиться к самому актуальному синтаксису и библиотекам https://github.com/

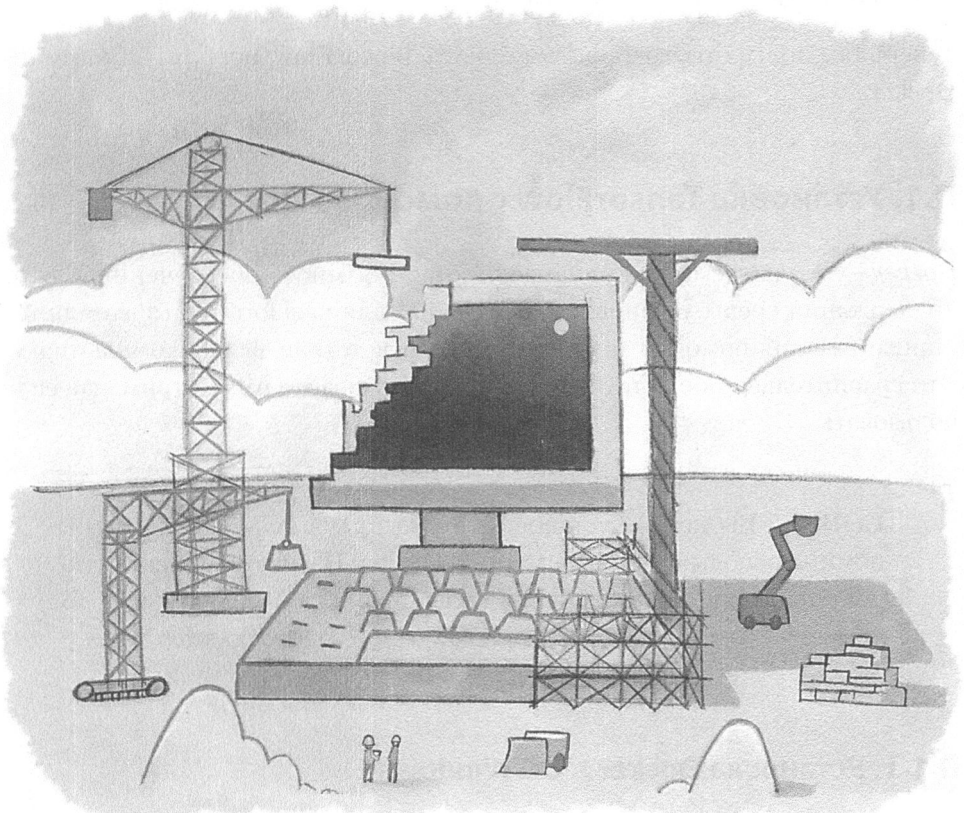
BinRoot/TensorFlow-Book. Не стесняйтесь присоединиться к сообществу, чтобы отслеживать баги в коде или отправлять запросы на включение.

> **СОВЕТ** TensorFlow находится в состоянии быстрого развития, поэтому все время появляются какие-то новые функциональные возможности!

Если вы жаждете других учебных материалов по TensorFlow, мы точно знаем, что вас может заинтересовать:

- *Обучение с подкреплением* (RL) — серия постов в блоге Артура Джулиани (Arthur Juliani) по использованию RL в TensorFlow: http://mng.bz/C17q.
- *Обработка естественного языка* (NLP) — актуальное руководство по TensorFlow для современных архитектур нейронных сетей, используемых в NLP Thushan Ganegedara: http://mng.bz/2Kh7.
- *Генеративные состязательные сети* (GAN) — вводное исследование генеративных и дискриминантных моделей машинного обучения (с использованием TensorFlow) Джона Гловера (John Glover) в AYLIEN: http://mng.bz/o2gc.
- *Веб-инструмент* — тинкер с простой нейронной сетью для визуализации потока данных: http://playground.tensorflow.org.
- *Видеолекции* — вводный курс и практические демонстрации TensorFlow в Google Cloud Big Data и блог по машинному обучению: http://mng.bz/vb7U.
- *Проекты с открытым исходным кодом* — ищите последние обновленные проекты TensorFlow на GitHub: http://mng.bz/wVZ4.

Приложение
Установка



Вы можете установить TensorFlow несколькими способами. В этой книге предполагается, что вы будете использовать Python 3 для каждой главы, если не указано иное. Листинги соответствуют версии Tensor-Flow v1.0, но соответствующий исходный код на GitHub всегда будет последней версии (https://github.com/BinRoot/TensorFlow-Book/). Это приложение охватывает один из методов установки, которые работают на всех платформах, включая Windows. Если вы знакомы с UNIX-системами (например, Linux или macOS), вы можете свободно использовать один из подходов к установке в официальной документации: www.tensorflow.org/ get_started/os_setup.html.

Теперь без долгих разговоров установим TensorFlow, используя контейнер Docker.

П.1. Установка TensorFlow с помощью Docker

Docker — это система для «упаковки» программного обеспечения с целью поддержания среды установки, идентичной для каждого пользователя. Эта стандартизация помогает ограничить несоответствия между компьютерами. Это сравнительно новая технология, поэтому давайте рассмотрим, как ее использовать.

> **СОВЕТ** Вы можете установить TensorFlow разными способами, помимо использования контейнера Docker. Изучите официальную документацию для получения более подробной информации об установке TensorFlow: www.tensorflow.org/get_started/os_setup.html.

П.1.1. Установка Docker в ОС Windows

Docker работает только с 64-разрядной версией Windows 7 и выше с поддержкой виртуализации. К счастью, большинство ноутбуков и ПК соответствуют этому требованию. Чтобы проверить, поддерживает ли ваш компьютер Docker, откройте панель управления, щелкните на System and Security, а затем на System. Здесь можно увидеть все подробности об ОС машины, включая тип процессора и системы. Если система 64-разрядная, то она почти наверняка подходит.

Теперь можно проверить, поддерживает ли ваш процессор виртуализацию. В Windows 8 откройте Task Manager (Ctrl-Shift-Esc) и щелкните на вкладке Performance. Если указана поддержка виртуализации, то у вас все готово для работы (рис. П.1). Если у вас Windows 7, воспользуйтесь инструментом Microsoft Hardware-Assisted Virtualization Detection Tool (http://mng.bz/cBlu).

Теперь, когда вы знаете, поддерживает ли ваш компьютер Docker, давайте установим панель инструментов Docker, расположенную по адресу www.docker.com/products/docker-toolbox. Запустите загруженный исполняемый файл установки и примите все значения по умолчанию, нажимая Next. После установки инструментария запустите Docker Quickstart Terminal.

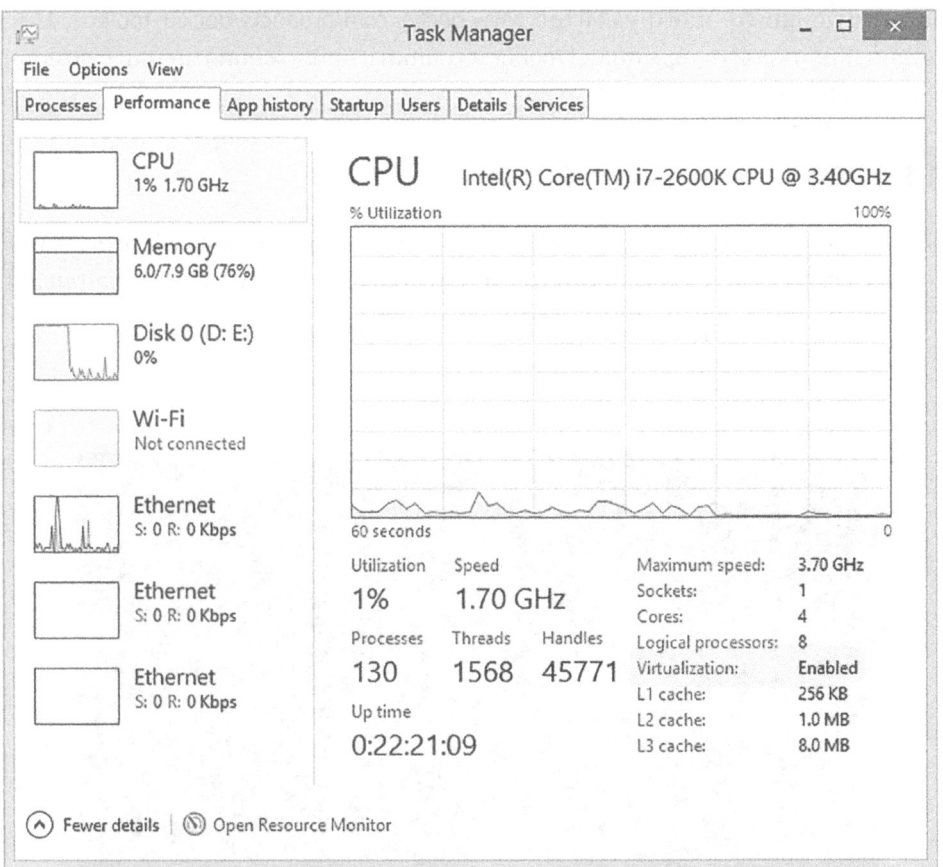


Рис. П.1. Убедитесь, что ваш 64-разрядный компьютер поддерживает виртуализацию

П.1.2. Установка Docker в ОС Linux

Docker официально поддерживается на нескольких дистрибутивах Linux. Официальная документация Docker, которую можно найти на сайте (https://docs.docker.com/engine/installation/linux/), содержит руководство для Arch Linux, CentOS, CRUX Linux, Debian, Fedora, Frugalware, Gentoo, Oracle Linux, Red Hat Enterprise Linux, openSUSE и Ubuntu. Docker является родственным для Linux, поэтому никаких проблем при его установке обычно не возникает.

П.1.3. Установка Docker в macOS

Docker работает в ОС macOS 10.8 Mountain Lion или в более поздних версиях. Загрузите панель инструментов www.docker.com/products/docker-toolbox. После установки откройте терминал Docker из папки с приложениями или с помощью панели запуска.

П.1.4. Как использовать Docker

Запустите терминал Docker. Затем запустите бинарный образ TensorFlow, используя следующую команду в терминале Docker, как показано на рис. П.2:

```
$ docker run -p 8888:8888 -p 6006:6006 b.gcr.io/tensorflow/tensorflow
```

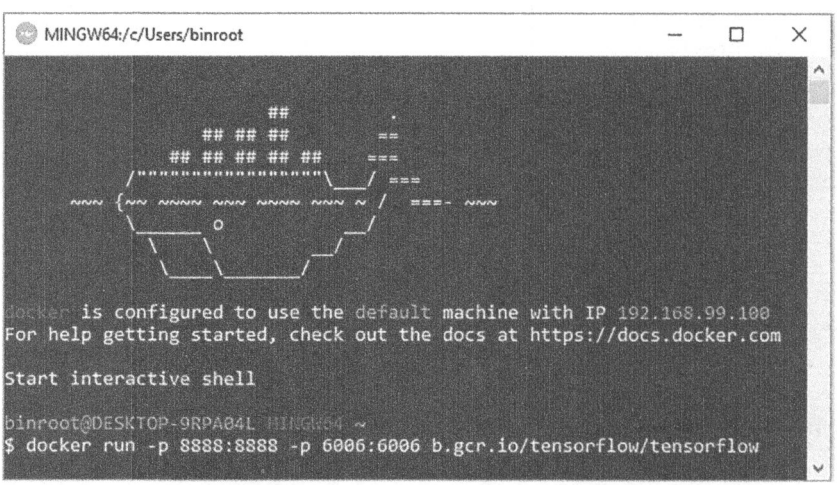


Рис. П.2. Запуск официального контейнера библиотеки TensorFlow

Теперь TensorFlow будет доступна из Jupyter Notebook через локальный IP-адрес. IP-адрес можно найти с помощью команды `docker-machine ip`, как показано на рис. П.3.

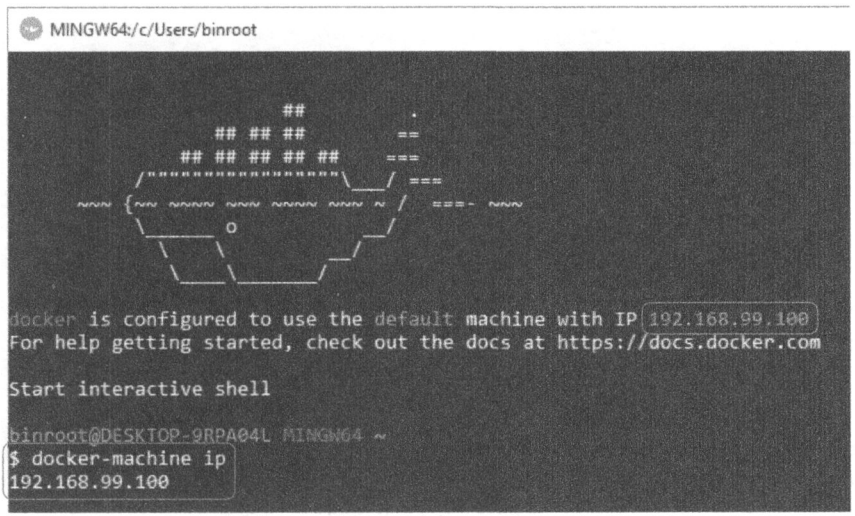


Рис. П.3. IP-адрес Docker можно найти с помощью команды `docker-machine ip` или в тексте под ASCII-китом

Откройте браузер и перейдите на http://<YOUR_IP_ADDRESS>:8888, чтобы начать пользоваться библиотекой TensorFlow. В нашем случае это сайт http://192.168.99.100:8888. На рис. П.4 показан Jupyter Notebook, на который можно перейти из браузера.

Вы можете нажать Ctrl-C или закрыть окно терминала, чтобы остановить работу Jupyter Notebook. Для повторного запуска повторите шаги, описанные в этом разделе.

Если вы столкнулись с сообщением об ошибке, показанным на рис. П.5, значит, Docker уже использует приложение в этом порте.

Чтобы решить эту проблему, вы можете либо переключить порт, либо выйти из вмещающих контейнеров Docker. На рис. П.6 показано, как перечислить все контейнеры с помощью `docker ps`, а затем уничтожить контейнер, используя `docker kill`.

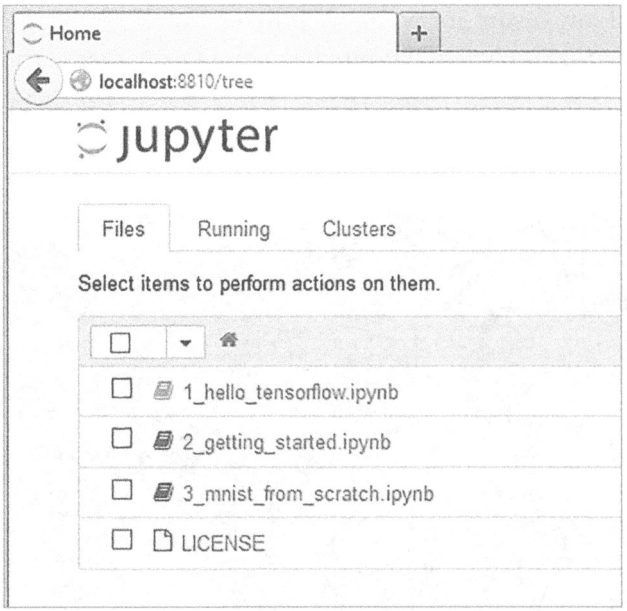


Рис. П.4. С TensorFlow можно взаимодействовать с помощью интерфейса языка Python, названного Jupyter.

```
binroot@DESKTOP-9RPA04L MINGW64 ~
$ docker run -p 8888:8888 -p 6006:6006 b.gcr.io/tensorflow/tensorflow
C:\Program Files\Docker Toolbox\docker.exe: Error response from daemon:
driver failed programming external connectivity on endpoint tender_allen
 (ab6dcf2455a5704f8f2911ac53ea946deb3ed939864c30e8fe867c2f5c88a63d): Bin
d for 0.0.0.0:8888 failed: port is already allocated.
```

Рис. П.5. Возможное сообщение об ошибке из выполняемого контейнера TensorFlow

```
binroot@DESKTOP-9RPA04L MINGW64 ~
$ docker ps
CONTAINER ID        IMAGE
62904e0a4489        b.gcr.io/tensorflow/tensorflow

binroot@DESKTOP-9RPA04L MINGW64 ~
$ docker kill 62904e0a4489
62904e0a4489
```

Рис. П.6. Включение в список и уничтожение контейнера Docker для избавления от сообщения об ошибке на рис. П.5

П.2. Установка Matplotlib

Matplotlib — это кросс-платформенная библиотека Python для построения 2D-визуализации данных. Если ваш компьютер может успешно запустить TensorFlow, то с установкой Matplotlib проблем не возникнет. Установите его, следуя официальной документации по адресу http://matplotlib.org/users/installing.html.

Нишант Шакла

Машинное обучение и TensorFlow

Перевел на русский А. Демьяников

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*К. Тульцева*
Научный редактор	*П. Ковалев*
Литературный редактор	*Е. Самородских*
Художественный редактор	*С. Заматевская*
Корректоры	*Н. Викторова, И. Тимофеева*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 12.2018. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 05.12.18. Формат 70×100/16. Бумага офсетная. Усл. п. л. 27,090. Тираж 1000. Заказ 16018.

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

КНИГА-ПОЧТОЙ

КНИГА · ПОЧТОЙ ПИТЕР

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Qiwi-кошелек.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы,(адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

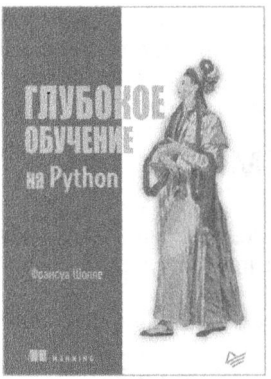
БЕСПЛАТНАЯ ДОСТАВКА:

- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
- почтой России при предварительной оплате заказа на сумму **от 2000 руб.**

СЕРИЯ «БИБЛИОТЕКА ПРОГРАММИСТА»

Франсуа Шолле

ГЛУБОКОЕ ОБУЧЕНИЕ НА PYTHON



Глубокое обучение – Deep learning – это набор алгоритмов машинного обучения, которые моделируют высокоуровневые абстракции в данных, используя архитектуры, состоящие из множества нелинейных преобразований. Согласитесь, эта фраза звучит угрожающе. Но всё не так страшно, если о глубоком обучении рассказывает Франсуа Шолле, который создал Keras – самую мощную библиотеку для работы с нейронными сетями. Познакомьтесь с глубоким обучением на практических примерах из самых разнообразных областей. Книга делится на две части: в первой даны теоретические основы, вторая посвящена решению конкретных задач. Это позволит вам не только разобраться в основах DL, но и научиться использовать новые возможности на практике.

«Обучение – это путешествие длиной в жизнь, особенно в области искусственного интеллекта, где неизвестностей гораздо больше, чем определенности».

Франсуа Шолле

КРАТКОЕ СОДЕРЖАНИЕ

ПИТЕР®

САЛД

САНКТ-ПЕТЕРБУРГСКАЯ АНТИВИРУСНАЯ ЛАБОРАТОРИЯ ДАНИЛОВА

www.SALD.ru
8 (812) 336-3739

Антивирусные программные продукты

Знакомство с машинным обучением и библиотекой TensorFlow похоже на первые уроки в автошколе, когда вы мучаетесь с параллельной парковкой, пытаетесь переключить передачу в нужный момент и не перепутать зеркала, лихорадочно вспоминая последовательность действий, в то время как ваша нога нервно подрагивает на педали газа. Это сложное, но необходимое упражнение. Так и в машинном обучении: прежде чем использовать современные системы распознавания лиц или алгоритмы прогнозирования на фондовом рынке, вам придется разобраться с соответствующим инструментарием и набором инструкций, чтобы затем без проблем создавать собственные системы.

Новички в машинном обучении оценят прикладную направленность этой книги, ведь ее цель — познакомить с основами, чтобы затем быстро приступить к решению реальных задач. От обзора концепций машинного обучения и принципов работы с TensorFlow вы перейдете к базовым алгоритмам, изучите нейронные сети и сможете самостоятельно решать задачи классификации, кластеризации, регрессии и прогнозирования.

ISBN 978-5-4461-0826-8
9 785446 108268

ПИТЕР®

Заказ книг:
тел.: (812) 703-73-74
books@piter.com

WWW.PITER.COM
каталог книг и интернет-магазин

instagram.com/**piterbooks**
youtube.com/**ThePiterBooks**
vk.com/**piterbooks**
facebook.com/**piterbooks**