巍巍交大　百年书香
www.jiaodapress.com.cn
bookinfo@sjtu.edu.cn

丛书策划　张荣昌
责任编辑　王　清　孟海江
封面设计

大数据、云计算、人工智能、信息安全人才培养丛书
“互联网+”新形态一体化教材

大数据
- 大数据基础
- 数据可视化
- 数据清洗与治理
- Hadoop应用与开发
- 数据挖掘基础
- SEO搜索引擎优化
- MySQL数据库
- R语言程序设计
- Go语言程序设计

云计算
- 云计算基础
- 虚拟化容器
- 云安全运维
- 网络工程与组网技术
- 现代通信技术
- 路由交换技术
- 无线网络技术
- 现代网络SDN技术
- 数据网组建与维护
- 局域网组建与维护
- 云数据中心架构与SDN技术

人工智能
- 人工智能基础
- 物联网基础
- ● 深度学习
- 机器学习

信息安全
- 信息安全基础
- Linux服务器安全高级运维
- Web安全与防御
- 防火墙技术与应用
- 计算机病毒与防范
- 数据存储与恢复
- 密码学基础
- 计算机网络安全运维
- 网络设备配置与综合实战
- 无线网络安全技术
- VPN虚拟专用网安全
- 终端数据存储与恢复
- 工控安全
- 渗透测试
- 恶意代码分析
- 网络空间安全态势感知
- 数据库安全技术
- 网络安全协议
- 企业级数据安全与灾备管理
- 信息安全法律法规
- 终端数据安全及防泄密

专业基础
- Linux操作系统基础
- 计算机网络基础
- Ubuntu服务器管理
- Windows Server 2019配置与管理
- 数据结构
- Python程序设计
- Java程序设计
- C语言程序设计
- C#程序设计
- Android程序设计
- XML基础教程
- JavaScript基础教程
- Web前端开发
- OpenStack应用与开发
- Spark应用与开发
- 静态网页设计与制作
- HTML5与JavaScript程序设计
- 数据库设计与应用
- 数据库应用基础
- UML建模与设计模式
- ERP原理与应用
- 综合布线

扫描二维码
关注上海交通大学出版社
官方微信

ISBN 978-7-313-25428-3
9 787313 254283 >
定价：52.00元

『互联网+』新形态一体化教材

深度学习

主编◎刘礼想　田　钧　张振国

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

大数据、云计算、人工智能、信息安全人才培养丛书
“互联网 +”新形态一体化教材

深度学习

SHENDU XUEXI

主编◎刘礼想　田　钧　张振国

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

本书从初学者角度出发，采用理论与实践相结合的方式阐述人工智能深度学习相关概念的背景知识，内容包括深度学习概述、神经网络基础、神经网络、机器学习、感知机、分层概念、卷积学习、单车预测器、生命科学和芯片发展。

本书可作为高等院校计算机应用、人工智能等相关专业的教材，也可作为人工智能入门者的自学用书。

图书在版编目（CIP）数据

深度学习 / 刘礼想，田钧，张振国主编 . — 上海：上海交通大学出版社，2021.9（2025.1 重印）

ISBN 978-7-313-25428-3

Ⅰ. ①深… Ⅱ. ①刘… ②田… ③张… Ⅲ. ①机器学习 Ⅳ. ① TP181

中国版本图书馆 CIP 数据核字（2021）第 186597 号

深度学习
SHENDU XUEXI

主　　编：刘礼想　田钧　张振国
出版发行：上海交通大学出版社
邮政编码：200030
印　　制：北京荣玉印刷有限公司
开　　本：889 mm × 1194 mm　1/16
字　　数：291 千字
版　　次：2021 年 9 月第 1 版
书　　号：ISBN 978-7-313-25428-3
定　　价：52.00 元

地　　址：上海市番禺路 951 号
电　　话：6407 1208
经　　销：全国新华书店
印　　张：13
印　　次：2025 年 1 月第 2 次印刷

前言

深度学习应用在许多软件领域，包括计算机视觉、语音和音频处理、自然语言处理、机器人技术、生物信息学和化学、电子游戏、搜索引擎、网络广告和金融，其最大的成就是在强化学习（reinforcement learning）领域的扩展。在强化学习中，一个自主的智能体必须在没有人类操作者指导的情况下，通过试错来学习执行任务。

深度学习也为其他学科做出了贡献。用于对象识别的现代卷积网络为神经科学家提供了可以研究的视觉处理模型，为处理海量数据以及在科学领域做出有效预测的科学家提供了非常有用的工具，用于预测分子如何相互作用自动解析，以及用于构建人脑三维图的显微镜图像等多个场合。在过去几十年的发展中，深度学习借鉴了大量关于人脑、统计学和应用数学的知识。近年来，得益于更强大的计算机、更大的数据集和能够训练更深网络的技术，深度学习的普及性和实用性都有了极大的发展。未来几年，深度学习将应用到更多的新领域。

本书从初学者角度出发，采用理论与实践相结合的方式阐述深度学习的相关概念和知识。全书共10章，内容包括深度学习概述、神经网络基础、神经网络、机器学习、感知机、分层概念、卷积学习、单车预测器、生命科学和芯片发展。本书所有知识点都结合具体实例和程序讲解，便于读者理解和掌握，内容由浅入深，适合于不同层次的学生使用。

本书在编写上具有如下特点：

（1）通过章前的“学习目标”“知识导图”“本章导读”明确每章要学习的内容，使学生做好学习的准备；通过文中的“说明”“提示”“注意”等模块丰富知识内容，提高学生的学习兴趣。

（2）在精练语言的基础上，充分利用图、表等形象地描述知识点，帮助学生更好地理解所学内容。本书内容由浅入深、循序渐进，符合学生的认知规律。

（3）计算机技术发展很快，本书着重讲解当前的知识和主流技术，使学生学到的知识和技术都与行业密切相关，做到学以致用。

此外，本书作者还为广大一线教师提供了服务于本书的教学资源库，有需要者可致电13810412048或发邮件至2393867076@qq.com。

本书可作为高等院校计算机相关专业的教学用书，也可作为相关技术人员的参考用书。由于编写时间仓促，书中难免存在不足和疏漏之处，敬请广大读者批评指正，在此表示衷心的感谢！

编者

2021年5月

目录

第 5 章 感知机 / 93

第 6 章 分层概念 / 105

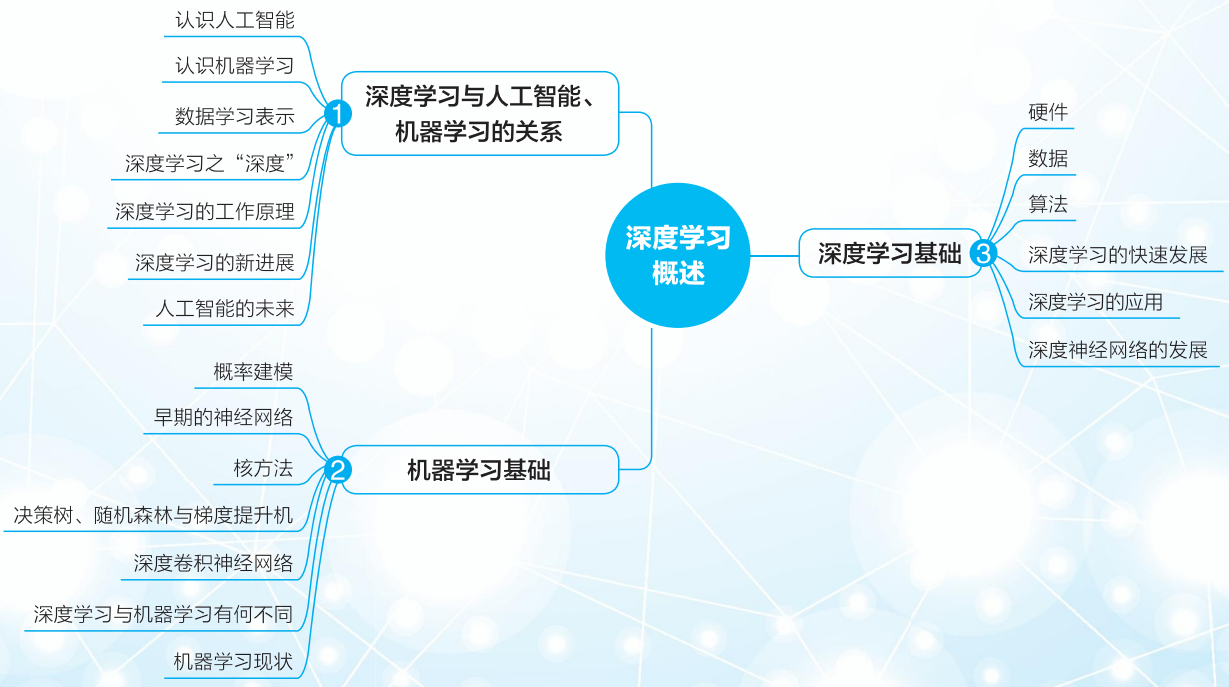
第 7 章 卷积学习 / 131

第 1 章 深度学习概述

学习目标

1. 掌握深度学习与人工智能、机器学习的关系。
2. 了解机器学习的基础知识。
3. 掌握深度学习日益流行的关键因素及其未来潜力。

知识导图



笔记

本章导读

关于未来的描绘少不了智能聊天机器人、自动驾驶汽车和虚拟助手等场景，人类从事的工作将大为减少，大部分经济活动都由机器人或人工智能（artificial intelligence，AI）体来完成。说到人工智能，就离不开深度学习（deep learning，DL）。深度学习是机器学习（machine learning，ML）领域中一个新的研究方向，它被引入机器学习使其更接近于最初的目标——人工智能。深度学习是学习样本数据的内在规律和表示层次，这些学习过程中获得的信息对诸如文字、图像和声音等数据的解释有很大的帮助。它的最终目标是让机器能够像人一样具有分析学习能力，能够识别文字、图像和声音等数据。深度学习是一个复杂的机器学习算法，在语音和图像识别方面取得的效果，远远超过先前的相关技术。

1.1 深度学习与人工智能、机器学习的关系

要了解深度学习，首先需要明确什么是人工智能、机器学习与深度学习。它们之间的关系如图 1-1 所示。

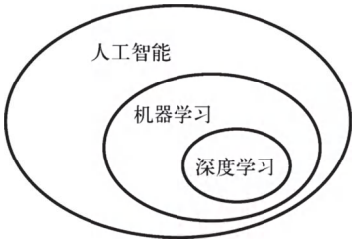


图 1-1　人工智能、机器学习、深度学习之间的关系

1.1.1 认识人工智能

人工智能诞生于 20 世纪 50 年代，当时计算机科学这一新兴领域的少数先驱者开始提出疑问：计算机是否能够“思考”？我们今天仍在探索这一问题的答案。人工智能的简洁定义如下：努力将通常由人类完成的智力任务自动化。因此，人工智能是一个综合性的领域，不仅包括机器学习与深度学习，还包括更多不涉及学习的方法。例如，早期的国际象棋程序仅包含程序员精心编写的硬编码规则，并不属于机器学习。在相当长的时间内，许多专家相信，只要程序员精心编写足够多的明确规则来处理知识，就可以实现与人类水平相当的人工智能。这一方法被称为符号主义人工智能（symbolic AI），从 20 世纪 50 年代到 80 年代末是人工智能的主流范式。在 20 世纪 80 年代的专家系统（expert system）热潮中，这一方法的热度达到了顶峰。虽然符号主义人工智能适合用来解决定义明确的逻辑问题，例如下国际象棋，但它难以给出明确的规则来解决更加复杂、模糊的问题，例如图像分类、语音识别和语言翻译。于是出现了一种新的方法来替代符号主义人工智能，这就是机器学习。

1.1.2 认识机器学习

在维多利亚时代的英格兰，埃达·洛夫莱斯伯爵夫人是查尔斯·巴贝奇的好友兼合作者，后者发明了分析机（analytical engine），即第一台通用的机械式计算机。虽然分析机这

一想法富有远见，并且相当超前，但它在 19 世纪 30 年代至 40 年代被设计出来时并没有打算用作通用计算机，因为当时还没有“通用计算”这一概念。它的用途仅仅是利用机械操作将数学分析领域的某些计算自动化，因此得名“分析机”。1843 年，埃达·洛夫莱斯伯爵夫人对这项发明评论道：“分析机谈不上能创造什么东西。它只能完成我们命令它做的任何事情……它的职责是帮助我们去实现我们已知的事情。”

笔记

随后，人工智能先驱阿兰·图灵在其 1950 年发表的具有里程碑意义的论文“计算机器和智能”中，引用了上述评论并将其称为“洛夫莱斯伯爵夫人的异议”。图灵在这篇论文中介绍了图灵测试以及日后人工智能所包含的重要概念。在引述埃达·洛夫莱斯伯爵夫人的同时，图灵还思考了这样一个问题：通用计算机是否能够学习与创新？他得出的结论是“能”。

机器学习的概念就来自图灵的这个问题：对于计算机而言，除了“我们命令它做的任何事情”之外，它能否自我学习执行特定任务的方法？计算机能否让我们大吃一惊？如果没有程序员精心编写的数据处理规则，计算机能否通过观察数据自动学会这些规则？

图灵的这个问题引出了一种新的编程范式。在经典的程序设计（符号主义人工智能的范式）中，人们输入的是规则（程序）和需要根据这些规则进行处理的数据，系统输出的是答案。利用机器学习，人们输入的是数据和从这些数据中预期得到的答案，系统输出的是规则。这些规则随后可应用于新的数据，并使计算机自主生成答案。机器学习——一种新的编程范式如图 1-2 所示。

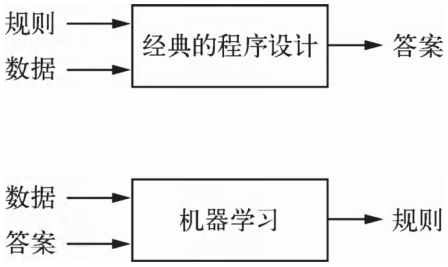


图 1-2　机器学习——一种新的编程范式

机器学习系统是训练出来的，而不是明确地用程序编写出来的。将与某个任务相关的许多示例输入机器学习系统，它会在这些示例中找到统计结构，从而最终找到规则将任务自动化。举个例子，你想为度假照片添加标签，并且希望将这项任务自动化，那么你可以将许多人工打好标签的照片输入机器学习系统，系统将学会将照片与特定标签联系在一起的统计规则。

虽然机器学习在 20 世纪 90 年代才开始蓬勃发展，但它迅速成为人工智能最受欢迎且最为成功的分支领域。这一发展的驱动力来自速度更快的硬件与更大的数据集。机器学习与数理统计密切相关，但两者在几个重要方面有所不同。机器学习不同于统计学，它经常用于处理复杂的大型数据集（如包含数百万张图像的数据集，每张图像又包含数万个像素），用经典的统计分析（如贝叶斯分析）来处理这种数据集是不切实际的。因此，机器学习（尤其是深度学习）呈现出相对较少的数学理论（可能太少了），并且是以工程为导向的。这是一门需要上手实践的学科，想法更多地是靠实践来证明，而不是靠理论推导。

笔记

1.1.3 数据学习表示

为了给出深度学习的定义并搞清楚深度学习与其他机器学习方法的区别，我们首先需要知道机器学习算法在做什么。前面说过，给定包含预期结果的示例，机器学习将会发现执行一项数据处理任务的规则。因此，我们需要以下三个要素来进行机器学习。

（1）输入数据点。例如，你的任务是语音识别，那么这些数据点可能是记录人们说话的声音文件。如果你的任务是为图像添加标签，那么这些数据点可能是图像。

（2）预期输出的示例。对于语音识别任务来说，这些示例可能是人们根据声音文件整理生成的文本。对于图像标记任务来说，预期输出可能是“狗”“猫”之类的标签。

（3）衡量算法效果好坏的方法。这一衡量方法是为了计算算法的当前输出与预期输出的差距。衡量结果是一种反馈信号，用于调节算法的工作方式。这个调节步骤就是我们所说的学习。

机器学习模型将输入数据变换为有意义的输出，这是一个从已知的输入和输出示例中进行“学习”的过程。因此，机器学习和深度学习的核心问题在于有意义地变换数据，换句话说，在于学习输入数据的有用表示（representation）——这种表示可以让数据更接近预期输出。在进一步讨论之前，我们需要先回答一个问题：什么是表示？这一概念的核心在于以一种不同的方式来查看数据（表征数据或将数据编码）。例如，彩色图像可以编码为 RGB（红 – 绿 – 蓝）格式或 HSV（色相 – 饱和度 – 明度）格式，这是对相同数据的两种不同表示。在处理某些任务时，使用某种表示可能会很困难，但换用另一种表示就会变得很简单。举个例子，对于“选择图像中所有红色像素”这个任务，使用 RGB 格式会更简单，而对于“降低图像饱和度”这个任务，使用 HSV 格式则更简单。机器学习模型就是为输入数据寻找合适的表示——对数据进行变换，使其更适合任务（如分类任务）。

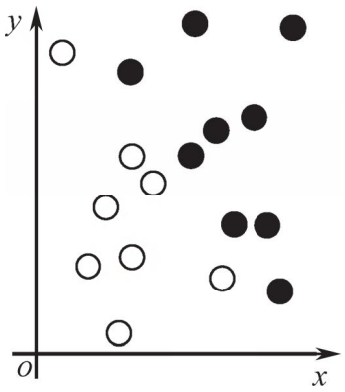


图 1-3　一些样本数据

我们来具体说明这一点。考虑 x 轴、y 轴和在 xOy 坐标系中由坐标表示的一些点，如图 1-3 所示。

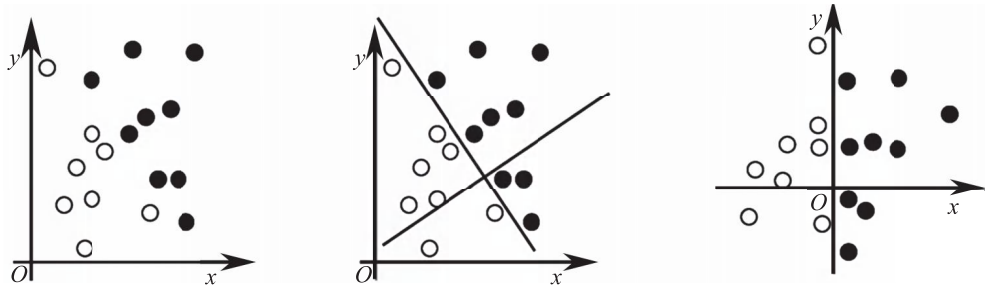
可以看到，图中有一些白点和一些黑点。假设想要开发一个算法，输入一个点的坐标（x，y），就能够判断这个点是黑色还是白色。在这个例子中：

输入是点的坐标；

预期输出是点的颜色；

衡量算法效果好坏的一种方法是，正确分类的点所占的百分比。

此时需要的是一种新的数据表示，可以明确区分白点与黑点。可用的方法有很多，这里用的是坐标变换，如图 1-4 所示。



（a）　（b）　（c）

图 1-4　坐标变换

（a）原始数据；（b）坐标变换；（c）更好的数据表示

在这个新的坐标系中，点的坐标可以看作数据的一种新的表示。利用这种新的表示，用一条简单的规则就可以描述黑点 / 白点分类问题：“x>0 的是黑点”或“x<0 的是白点”。这种新的表示基本上解决了该分类问题。

在这个例子中，我们人为定义了坐标变换。但是，如果尝试系统性地搜索各种可能的坐标变换，并用正确分类的点所占百分比作为反馈信号，那么我们做的就是机器学习。机器学习中的学习指的是寻找更好数据表示的自动搜索过程。

所有机器学习算法都包括自动寻找这样一种变换：这种变换可以根据任务将数据转化为更加有用的表示。这些操作可能是前面提到的坐标变换，也可能是线性投影（可能会破坏信息）、平移、非线性操作（如“选择所有 x>0 的点”）等。机器学习算法在寻找这些变换时通常没有什么创造性，而仅仅是遍历一组预先定义好的操作，这组操作叫作假设空间（hypothesis space）。

这就是机器学习的技术定义：在预先定义好的可能性空间中，利用反馈信号的指引来寻找输入数据的有用表示。这个简单的想法可以解决相当多的智能任务，从语音识别到自动驾驶都能解决。

1.1.4 深度学习之“深度”

深度学习是机器学习的一个分支领域：它是从数据中学习表示的一种新方法，强调从连续的层（Layer）中进行学习，这些层对应于越来越有意义的表示。“深度学习”中的“深度”指的并不是利用这种方法所获取的更深层次的理解，而是指一系列连续的表示层。数据模型中包含的层数，被称为模型的深度（depth）。这一领域的其他名称包括分层表示学习（layered representations learning）和层级表示学习（hierarchical representations learning）。现代深度学习通常包含数十个甚至上百个连续的表示层，这些表示层全都是从训练数据中自动学习的。与此相反，其他机器学习方法的重点往往是仅仅学习一两层的数据表示，因此有时也被称为浅层学习（shallow learning）。

在深度学习中，这些分层表示几乎总是通过叫作神经网络（neural network）的模型来学习得到的。神经网络的结构是逐层堆叠。神经网络这一术语来自神经生物学，然而，虽然深度学习的一些核心概念是从人们对大脑的理解中汲取部分灵感而形成的，但深度学习

笔记

模型不是大脑模型。没有证据表明大脑的学习机制与现代深度学习模型所使用的学习机制相同。你可能会读到一些流行科学的文章，宣称深度学习的工作原理与大脑相似或者是根据大脑的工作原理进行建模的，但事实并非如此。对于这一领域的新人来说，如果认为深度学习与神经生物学存在任何关系，那将使人困惑，只会起到反作用。你无须那种“就像我们的头脑一样”的神秘包装，最好也忘掉读过的深度学习与生物学之间的假想联系。就我们的目的而言，深度学习是从数据中学习表示的一种数学框架。

深度学习算法学到的表示是什么样的？我们来看一个多层网络（见图 1-5）如何对数字图像进行变换，以便识别图像中所包含的数字。

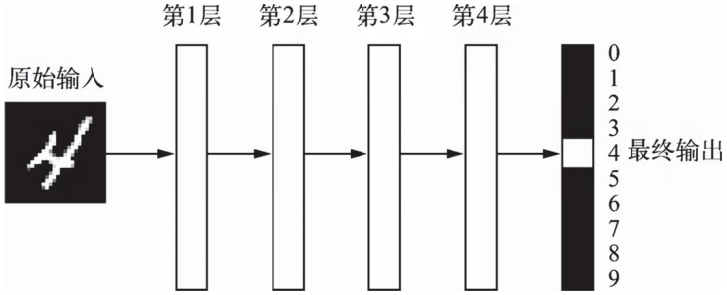


图 1-5　用于数字分类的深度神经网络

数字图像分类模型学到的深度表示如图 1-6 所示，这个网络将数字图像转换成与原始图像差别越来越大的表示，而其中关于最终结果的信息却越来越丰富。你可以将深度网络看作多级信息蒸馏操作：信息穿过连续的过滤器，其纯度越来越高（对任务的帮助越来越大）。

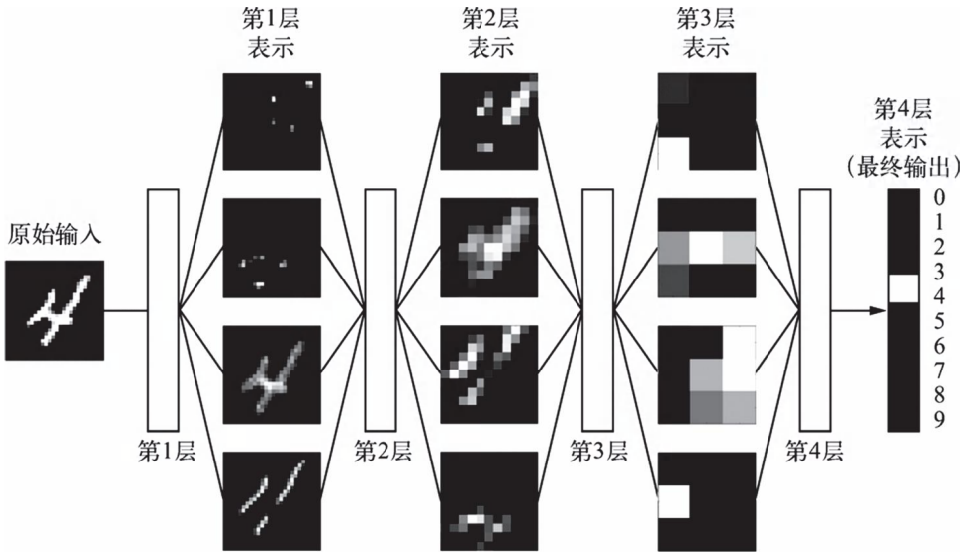


图 1-6　数字图像分类模型学到的深度表示

这就是深度学习的技术定义：学习数据表示的多级方法。这个想法很简单，但事实证明，非常简单的机制如果具有足够大的规模，将会产生魔法般的效果。

1.1.5 深度学习的工作原理

笔记

机器学习是将输入（如图像）映射到目标（如标签“猫”），这一过程是通过观察许多输入和目标的示例来完成的。深度神经网络通过一系列简单的数据变换（层）来实现这种输入到目标的映射，而这些数据变换都是通过观察示例学习到的。下面来具体看一下这种学习过程是如何发生的。

神经网络中每层对输入数据所做的具体操作会保存在该层的权重（weight）中，其本质是一串数字。用术语来说，每层实现的变换由其权重来参数化（parameterize），如图 1-7 所示。权重有时也被称为该层的参数（parameter）。在这种语境下，学习的意思是为神经网络的所有层找到一组权重值，使得该网络能够将每个示例输入与其目标正确地一一对应。但重点来了：一个深度神经网络可能包含数千万个参数。找到所有参数的正确取值是一项非常艰巨的任务，特别是考虑到修改某个参数值将会影响其他所有参数的行为。

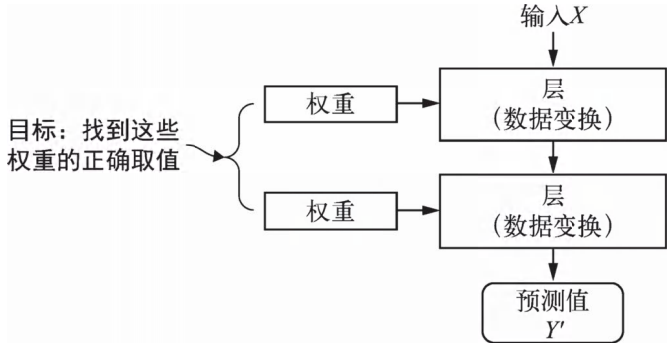


图 1-7　神经网络是由其权重来参数化

想要控制一件事物，首先需要能够观察它。想要控制神经网络的输出，就需要能够衡量该输出与预期值之间的距离。这是神经网络损失函数（loss function）的任务，该函数也称为目标函数（objective function）。损失函数的输入是网络预测值与真实目标值（希望网络输出的结果），然后计算一个距离值，衡量该网络在这个示例上的效果好坏，如图 1-8 所示。

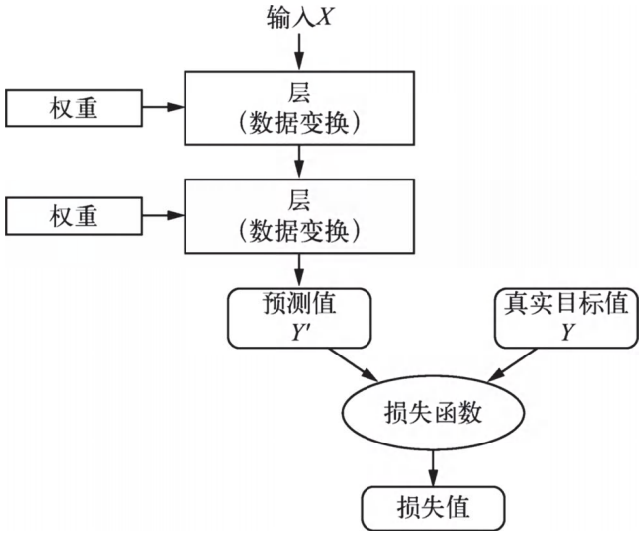


图 1-8　损失函数用来衡量网络输出结果的质量

笔记

深度学习的基本技巧是利用这个距离值作为反馈信号来对权重值进行微调，以降低当前示例对应的损失值，如图 1-9 所示。这种调节由优化器（optimizer）来完成，它实现了所谓的反向传播（back propagation）算法，这是深度学习的核心算法。下一章中会详细地解释反向传播的工作原理。

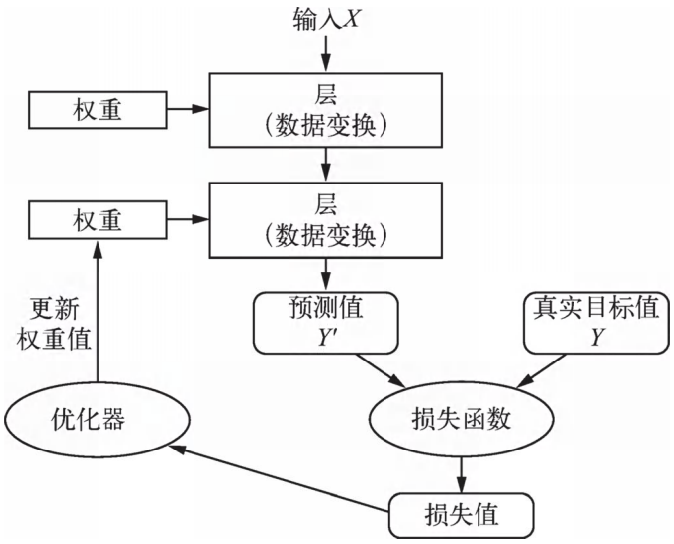


图 1-9　将损失值作为反馈信号来调节权重

一开始对神经网络的权重随机赋值，因此网络只是实现了一系列随机变换。其输出结果自然也与理想值相去甚远，相应地，损失值也很高。但随着网络处理的示例越来越多，权重值也在向正确的方向逐步微调，损失值也逐渐降低，这就是训练循环（training loop）。将这种循环重复足够多的次数（通常对数千个示例进行数十次迭代），得到的权重值可以使损失函数最小。具有最小损失的网络，其输出值与目标值尽可能地接近，这就是训练好的网络。再次强调，这是一个简单的机制，一旦具有足够大的规模，将会产生魔法般的效果。

1.1.6 深度学习的新进展

虽然深度学习是机器学习一个相当有年头的分支领域，但在 21 世纪的前十年才开始崛起。在随后的几年里，它在实践中取得了革命性进展，在视觉和听觉等感知问题上取得了令人瞩目的成果，而这些问题所涉及的技术，在人类看来是非常自然、非常直观的，但长期以来却一直是机器难以解决的。

特别要强调的是，深度学习已经取得了以下突破，它们都是机器学习历史上非常困难的领域：

（1）接近人类水平的图像分类。

（2）接近人类水平的语音识别。

（3）接近人类水平的手写文字转录。

（4）更好的机器翻译。

（5）更好的文本到语音的转换。

（6）数字助理，比如谷歌即时（Google Now）和亚马逊 Alexa。

（7）接近人类水平的自动驾驶。

（8）更好的广告定向投放，Google、百度、必应都在使用。

（9）更好的网络搜索结果。

（10）能够回答用自然语言提出的问题。

（11）在围棋上战胜人类。

笔记

我们仍然在探索深度学习能力的边界。我们已经开始将其应用于机器感知和自然语言理解之外的各种问题，比如形式推理。如果能够成功的话，这可能预示着深度学习将能够协助人类进行科学研究、软件开发等活动。

1.1.7 人工智能的未来

人工智能深度学习应用于许多重要的问题，从医疗诊断到数字助手，在这些问题上深度学习都发挥了变革性作用。过去五年里，人工智能研究一直在以惊人的速度发展，这在很大程度上是由于人工智能短短的历史中前所未见的资金投入，但到目前为止，这些进展却很少能够转化为改变世界的产品和流程。深度学习的大多数研究成果尚未得到应用，至少尚未应用到它在各行各业中能够解决的所有问题上。当然，人们可以向智能手机提出简单的问题并得到合理的回答，也可以在购物网站上得到相当有用的产品推荐。与过去相比，这些技术已大不相同，但这些工具仍然只是日常生活的陪衬。人工智能仍需要进一步转变为我们的工作、思考和生活服务离不开深度学习技术的支撑。

我们似乎很难相信人工智能会对世界产生巨大影响，因为它还没有被广泛地部署应用——正如 1995 年，我们也难以相信互联网在未来会产生的影响。当时，大多数人都没有认识到互联网与他们的关系，以及互联网将如何改变他们的生活。今天的深度学习和人工智能也是如此。但不要怀疑，人工智能即将到来。在不远的未来，人工智能将会成为你的助手，甚至成为你的朋友。它会回答你的问题，帮助你教育孩子，并关注你的健康。它还会将生活用品送到你家门口，并开车将你从 A 地送到 B 地。它还会是你与日益复杂的、信息密集的世界之间的接口。更为重要的是，人工智能将会帮助科学家在所有科学领域（从基因学到数学）取得突破性进展，从而帮助各项研究的发展。

在这个过程中，我们可能会经历一些挫折，也可能会遇到新的人工智能冬天，正如互联网行业那样，在 1998—1999 年被过度炒作，进而在 21 世纪初遭遇破产，并导致投资停止。但我们最终会实现上述目标。人工智能最终将应用到人们社会和日常生活的几乎所有方面，正如今天的互联网一样。

不要相信短期的炒作，但一定要相信长期的愿景。人工智能可能需要一段时间才能充分发挥其潜力。这一潜力的范围大到难以想象，但人工智能终将到来，它将以一种奇妙的方式改变我们的世界。

笔记

1.2 机器学习基础

深度学习已经得到了人工智能历史上前所未有的公众关注度和产业投资，但这并不是机器学习的第一次成功。可以这样说，当前工业界所使用的绝大部分机器学习算法都不是深度学习算法。深度学习不一定总是解决问题的正确工具：有时没有足够的数据，深度学习并不适用；有时用其他算法可以更好地解决问题。如果你第一次接触的机器学习就是深度学习，那你可能会发现手中握着一把深度学习“锤子”，而所有机器学习问题看起来都像是“钉子”。为了避免陷入这个误区，唯一的方法就是熟悉其他机器学习方法并在适当的时候进行实践。

关于经典机器学习方法的详细讨论已经超出了本书范围，但我们将简要回顾这些方法，并介绍这些方法的历史背景。这样我们可以将深度学习放入机器学习的大背景中，并更好地理解深度学习的起源以及它为什么如此重要。

1.2.1 概率建模

概率建模（probabilistic modeling）是统计学原理在数据分析中的应用。它是最早的机器学习形式之一，至今仍在广泛使用。其中最有名的算法之一就是朴素贝叶斯算法。

朴素贝叶斯是一类基于应用贝叶斯定理的机器学习分类器，它假设输入数据的特征都是独立的。这是一个很强的假设，或者说“朴素的”假设，其名称正来源于此。这种数据分析方法比计算机出现得还要早，在其第一次被计算机实现（很可能追溯到 20 世纪 50 年代）的几十年前就已经靠人工计算来应用了。贝叶斯定理和统计学基础可以追溯到 18 世纪，你学会了这两点就可以开始使用朴素贝叶斯分类器了。

另一个密切相关的模型是 logistic 回归（logistic regression，Logreg），它有时被认为是现代机器学习的“ hello world”。不要被它的名称所误导——Logreg 是一种分类算法，而不是回归算法。与朴素贝叶斯类似，Logreg 的出现也比计算机早很长时间，但由于它既简单又通用，至今仍然很有用。面对一个数据集，数据科学家通常会首先尝试使用这个算法，以便初步熟悉手头的分类任务。

1.2.2 早期的神经网络

神经网络早期的迭代方法已经完全被本章所介绍的现代方法所取代，但仍有助于我们了解深度学习的起源。尽管早在 20 世纪 50 年代，人们就用简单的方式研究了神经网络的核心思想，但神经网络这种方法经历了数十年才开始兴起。在很长一段时间内，一直没有训练大型神经网络的有效方法。这一点在 20 世纪 80 年代中期发生了变化，当时很多人都独立地重新发现了反向传播算法——一种利用梯度下降优化来训练一系列参数化运算链的方法（本书后面将给出这些概念的具体定义），并开始将其应用于神经网络。

贝尔实验室于 1989 年第一次成功实现了神经网络的实践应用，当时 Yann LeCun 将卷

积神经网络的早期思想与反向传播算法相结合，并将其应用于手写数字分类问题，由此得到名为 LeNet 的网络，在 20 世纪 90 年代被美国邮政署采用，用于自动读取信封上的邮政编码。

笔记

1.2.3 核方法

上节所述神经网络取得了第一次成功，并在 20 世纪 90 年代开始在研究人员中受到一定的重视，但一种新的机器学习方法在这时声名鹊起，很快就使人们将神经网络抛诸脑后。这种方法就是核方法（kernel method）。核方法是一组分类算法，其中最有名的就是支持向量机（support vector machine，SVM）。虽然 Vladimir Vapnik 和 Alexey Chervonenkis 早在 1963 年就发表了较早版本的线性公式 2，但 SVM 的现代公式由 Vladimir Vapnik 和 Corinna Cortes 于 20 世纪 90 年代初在贝尔实验室提出，并发表于 1995 年 3 月。

SVM 的目标是通过在属于两个不同类别的两组数据点之间找到良好决策边界（decision boundary）来解决分类问题，如图 1-10 所示。决策边界可以看作一条直线或一个平面，将训练数据划分为两块空间，分别对应于两个类别。对于新数据点的分类，你只需判断它位于决策边界的哪一侧。

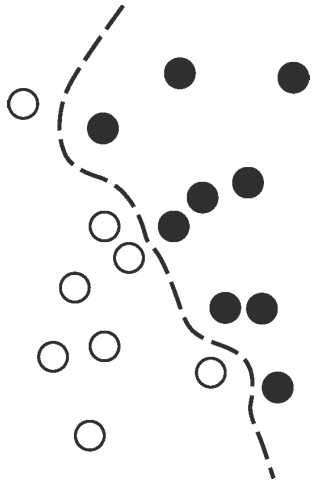


图 1-10　决策边界

SVM 通过两步来寻找决策边界：

（1）将数据映射到一个新的高维表示，这时决策边界可以用一个超平面来表示（如果数据像图 1-10 那样是二维的，那么超平面就是一条直线）。

（2）尽量让超平面与每个类别最近的数据点之间的距离最大化，从而计算出良好决策边界（分割超平面），这一步叫作间隔最大化（maximizing the margin）。这样决策边界可以很好地推广到训练数据集之外的新样本。

将数据映射到高维表示从而使分类问题简化，这一技巧可能听起来很不错，但在实践中通常是难以计算的。这时就需要用到核技巧（kernel trick，核方法正是因这一核心思想而得名）。其基本思想是：要想在新的表示空间中找到良好的决策超平面，并不需要在新空间中直接计算点的坐标，只需要在新空间中计算点对之间的距离，而利用核函数（kernel function）可以高效地完成这种计算。核函数是一个在计算上能够实现的操作，将原始空间中的任意两点映射为这两点在目标表示空间中的距离，完全避免了对新表示进行直接计算。核函数通常是人为选择的，而不是从数据中学到的——对于 SVM 来说，只有分割超平面是通过学习得到的。

SVM 刚刚出现时，在简单的分类问题上表现出了最好的性能。当时只有少数机器学习方法得到大量的理论支持，并且适合用于严肃的数学分析，因而非常易于理解和解释，SVM 就是其中之一。由于 SVM 具有这些有用的性质，很长一段时间里它在实践中非常

笔记

流行。

但是，SVM 很难扩展到大型数据集，并且在图像分类等感知问题上的效果也不好。SVM 是一种比较浅层的方法，因此要想将其应用于感知问题，首先需要手动提取出有用的表示（这叫作特征工程），这一步骤很难，而且不稳定。

1.2.4 决策树、随机森林与梯度提升机

决策树（decision tree）是类似于流程图的结构，可以对输入数据点进行分类或根据给定输入来预测输出值，如图 1-11 所示。决策树的可视化和解释都很简单。在 21 世纪前十年，从数据中学习得到的决策树开始引起研究人员的广泛关注。到了 2010 年，决策树经常比核方法更受欢迎。

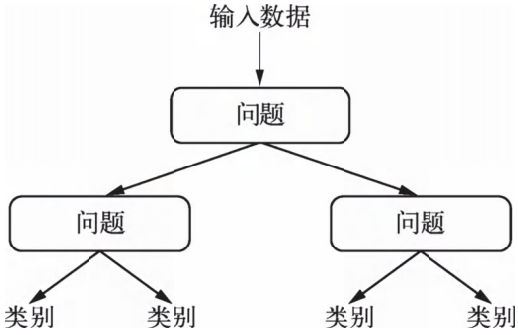


图 1-11　决策树

举个例子，问题可能是：“数据中第 2 个系数是否大于 3.5？”

特别是随机森林（random forest）算法，它引入了一种健壮且实用的决策树学习方法，即首先构建许多决策树，然后将它们的输出集成在一起。随机森林适用于各种各样的问题——对于任何浅层的机器学习任务来说，它几乎总是第二好的算法。广受欢迎的机器学习竞赛网站 Kaggle 在 2010 年上线后，随机森林迅速成为平台上人们的最爱，直到 2014 年才被梯度提升机所取代。梯度提升机（gradient boosting machine）与随机森林类似，它也是将弱预测模型（通常是决策树）集成的机器学习技术。它使用了梯度提升方法，通过迭代地训练新模型来专门解决之前模型的弱点，从而改进任何机器学习模型的效果。将梯度提升技术应用于决策树时，得到的模型与随机森林具有相似的性质，但在绝大多数情况下效果都比随机森林要好。它可能是目前处理非感知数据最好的算法之一。它和深度学习一样，也是 Kaggle 竞赛中最常用的技术之一。

1.2.5 深度卷积神经网络

虽然神经网络几乎被整个科学界完全忽略，但仍有一些人在继续研究神经网络，并在 2010 年左右开始取得重大突破。这些人包括：多伦多大学 Geoffrey Hinton 的小组、蒙特利尔大学的 Yoshua Bengio、纽约大学的 Yann LeCun 和瑞士的 IDSIA。

2011 年，来自 IDSIA 的 Dan Ciresan 开始利用 GPU 训练的深度神经网络赢得学术

性的图像分类竞赛，这是现代深度学习第一次在实践中获得成功。但真正的转折性时刻出现在 2012 年，当年 Hinton 小组参加了每年一次的大规模图像分类挑战赛 ImageNet。ImageNet 挑战赛在当时以困难著称，参赛者需要对 140 万张高分辨率彩色图像进行训练，然后将其划分到 1 000 个不同的类别中。2011 年，获胜的模型是基于经典的计算机视觉方法，其 top-5 精度 4 只有 74.3%。到了 2012 年，由 Alex Krizhevsky 带领并由 Geoffrey Hinton 提供建议的小组，实现了 83.6% 的 top-5 精度——这是一项重大突破。此后，这项竞赛每年都由深度卷积神经网络所主导。到了 2015 年，获胜者的精度达到了 96.4%，此时 ImageNet 的分类任务被认为是一个已经完全解决的问题。

自 2012 年以来，深度卷积神经网络（ConvNet）已成为所有计算机视觉任务的首选算法。更一般地说，它在所有感知任务上都有效。在 2015 年和 2016 年的主要计算机视觉会议上，几乎所有演讲都与 ConvNet 有关。与此同时，深度学习也在许多其他类型的问题上得到应用，比如自然语言处理。它已经在大量应用中完全取代了 SVM 与决策树。举个例子，欧洲核子研究中心（CERN）多年来一直使用基于决策树的方法分析来自大型强子对撞机（LHC）ATLAS 探测器的粒子数据，但 CERN 最终转向基于 Keras 的深度神经网络，因为它的性能更好，而且在大型数据集上易于训练。

笔记

提示

top-5 精度是指给定一张图像，如果模型预测的前 5 个标签中包含正确标签，即为预测正确。

1.2.6 深度学习与机器学习有何不同

深度学习发展得如此迅速，主要原因在于它在许多问题上都表现出很好的性能。但这并不是唯一的原因。深度学习还让解决问题变得更加简单，因为它将特征工程完全自动化，而这曾经是机器学习工作流程中最关键的一步。

先前的机器学习技术（浅层学习）仅包含将输入数据变换到一两个连续的表示空间，通常使用简单的变换，比如高维非线性投影（SVM）或决策树，但这些技术通常无法得到复杂问题所需要的精确表示。因此，人们必须竭尽全力让初始输入数据更适合用这些方法处理，也必须手动为数据设计好的表示层，这叫作特征工程。与此相反，深度学习完全将这个步骤自动化：利用深度学习，你可以一次性学习所有特征，而无须自己手动设计。这极大地简化了机器学习的工作流程，通常将复杂的多阶段流程替换为一个简单的、端到端的深度学习模型。

你可能会问，如果问题的关键在于有多个连续表示层，那么能否重复应用浅层方法，以实现和深度学习类似的效果？在实践中，如果连续应用浅层学习方法，其收益会随着层数增加迅速降低，因为三层模型中最优的第一表示层并不是单层或双层模型中最优的第一表示层。深度学习的变革性在于，模型可以在同一时间共同学习所有表示层，而不是依次连续学习（这被称为贪婪学习）。通过共同的特征学习，一旦模型修改某个内部特征，所有依赖于该特征的其他特征都会相应地自动调节适应，无须人为干预。一切都由单一反馈信号来监督：模型中的每一处变化都是为了最终目标服务。这种方法比单纯地叠加浅层模型更加强大，因为它可以通过将复杂、抽象的表示拆解为很多个中间空间（层）来学习这

笔记

些表示，每个中间空间仅仅是前一个空间的简单变换。

深度学习从数据中进行学习时有两个基本特征：第一，通过渐进的、逐层的方式形成越来越复杂的表示；第二，对中间这些渐进的表示共同进行学习，每一层的变化都需要同时考虑上下两层的需要。总之，这两个特征使得深度学习比先前的机器学习方法更加成功。

1.2.7 机器学习现状

要想了解机器学习算法和工具的现状，一个好方法是看一下 Kaggle 上的机器学习竞赛。Kaggle 上的竞争非常激烈（有些比赛有数千名参赛者，并提供数百万美元的奖金），而且涵盖了各种类型的机器学习问题，因此它提供了一种现实方法来评判哪种方法有效、哪种方法无效。那么哪种算法能够可靠地赢得竞赛呢？顶级参赛者都使用哪些工具？

在 2016 年和 2017 年，Kaggle 上主要有两大方法：梯度提升机和深度学习。具体来说，梯度提升机用于处理结构化数据的问题，而深度学习则用于图像分类等感知问题。使用前一种方法的人几乎都使用 XGBoost 库，它同时支持数据科学最流行的两种语言：Python 和 R。使用深度学习的 Kaggle 参赛者则大多使用 Keras 库，因为它易于使用，非常灵活，并且支持 Python。

要想在如今的应用机器学习中取得成功，应该熟悉这两种技术：梯度提升机，用于浅层学习问题；深度学习，用于感知问题。用术语来说，你需要熟悉 XGBoost 和 Keras，它们是目前主宰 Kaggle 竞赛的两个库。

1.3 深度学习基础

深度学习用于计算机视觉的两个关键思想，即卷积神经网络和反向传播，在 1989 年就已经为人们所知。长短期记忆（long short-term memory，LSTM）算法是深度学习处理时间序列的基础，它在 1997 年就被开发出来了，而且此后几乎没有发生变化。

总的来说，有三种技术力量在推动着机器学习的进步：

（1）硬件。

（2）数据集和基准。

（3）算法上的改进。

由于这一领域是靠实验结果而不是理论指导的，因此只有当合适的数据和硬件可用于尝试新想法时（或者将旧想法的规模扩大，事实往往也是如此），才可能出现算法上的改进。机器学习不是数学或物理学，靠一支笔和一张纸就能实现重大进展。它是一门工程科学。

在 20 世纪 90 年代和 21 世纪前十年，真正的瓶颈在于数据和硬件。但在这段时间内发生了下面这些事情：互联网高速发展，并且针对游戏市场的需求开发出了高性能图形芯片。

1.3.1 硬件

笔记

从 1990 年到 2010 年，非定制 CPU 的速度提高了约 5 000 倍。因此，现在可以在笔记本电脑上运行小型深度学习模型，但在 25 年前是无法实现的。

对于计算机视觉或语音识别所使用的典型深度学习模型，所需要的计算能力要比笔记本电脑的计算能力高几个数量级。在 21 世纪前十年里，NVIDIA 和 AMD 等公司投资数十亿美元来开发快速的大规模并行芯片（图形处理器，GPU），以便为越来越逼真的视频游戏提供图形显示支持。这些芯片是廉价的、单一用途的超级计算机，用于在屏幕上实时渲染复杂的 3D 场景。这些投资为科学界带来了好处。2007 年，NVIDIA 推出了 CUDA，作为其 GPU 系列的编程接口。少量 GPU 开始在各种高度并行化的应用中替代大量 CPU 集群，并且最早应用于物理建模。深度神经网络主要由许多小矩阵乘法组成，它也是高度并行化的。2011 年前后，一些研究人员开始编写神经网络的 CUDA 实现，Dan Ciresan 5 和 Alex Krizhevsky 6 属于第一批人。

这样，游戏市场资助了用于下一代人工智能应用的超级计算。今天，NVIDIA TITAN X（一款游戏 GPU，在 2015 年底售价 1 000 美元）可以实现单精度 6.6 TFLOPS 的峰值，即每秒进行 6.6 万亿次 float 32 运算。这比一台现代笔记本电脑的速度要快约 350 倍。使用一块 TITAN X 显卡，只需几天就可以训练出几年前赢得 ILSVRC 竞赛的 ImageNet 模型。与此同时，大公司还在包含数百个 GPU 的集群上训练深度学习模型，这种类型的 GPU 是专门针对深度学习的需求开发的，比如 NVIDIA Tesla K80。如果没有现代 GPU，这种集群的超级计算能力是不可能实现的。

此外，深度学习行业已经开始超越 GPU，开始投资于日益专业化的高效芯片来进行深度学习。2016 年，Google 在其年度 I/O 大会上展示了张量处理器（TPU）项目，它是一种新的芯片设计，其开发目的完全是为了运行深度神经网络。据报道，它的速度比最好的 GPU 还要快 10 倍，而且能效更高。

1.3.2 数据

人工智能有时被称为新的工业革命。如果深度学习是这场革命的蒸汽机，那么数据就是煤炭，即驱动智能机器的原材料，没有煤炭一切皆不可能。就数据而言，除了过去 20 年里存储硬件的指数级增长（遵循摩尔定律），最大的变革来自互联网的兴起，它使得收集与分发用于机器学习的超大型数据集变得可行。如今，大公司使用的图像数据集、视频数据集和自然语言数据集，如果没有互联网的话根本无法收集。例如，Flickr 网站上用户生成的图像标签一直是计算机视觉的数据宝库。维基百科则是自然语言处理的关键数据集。

如果有一个数据集是深度学习兴起的催化剂的话，那么一定是 ImageNet 数据集。它包含 140 万张图像，这些图像已经被人工划分为 1 000 个图像类别（每张图像对应 1 个类

笔记

别）。但 ImageNet 的特殊之处不仅在于其数量之大，还在于与它相关的年度竞赛。

正如 Kaggle 自 2010 年以来所展示的那样，公开竞赛是激励研究人员和工程师挑战极限的极好方法。研究人员通过竞争来挑战共同基准，这极大地促进了近期深度学习的兴起。

1.3.3 算法

除了硬件和数据之外，直到 21 世纪前十年的末期，我们仍没有可靠的方法来训练非常深的神经网络。因此，神经网络仍然很浅，仅使用一两个表示层，无法超越更为精确的浅层方法，比如 SVM 和随机森林。关键问题在于通过多层叠加的梯度传播。随着层数的增加，用于训练神经网络的反馈信号会逐渐消失。

这一情况在 2009—2010 年发生了变化，当时出现了几个很简单但很重要的算法改进，可以实现更好的梯度传播。

更好的神经层激活函数（activation function）。

更好的权重初始化方案（weight-initialization scheme），一开始使用逐层预训练的方法，不过这种方法很快就被放弃了。

更好的优化方案（optimization scheme），比如 RMSProp 和 Adam。

只有这些改进可以训练 10 层以上的模型时，深度学习才开始大放异彩。

最后，在 2014 年、2015 年和 2016 年，人们发现了更先进的有助于梯度传播的方法，例如批标准化、残差连接和深度可分离卷积。今天，我们可以从头开始训练上千层的模型。

1.3.4 深度学习的快速发展

随着深度学习于 2012—2013 年在计算机视觉领域成为新的最优算法，并最终在所有感知任务上都成为最优算法，业界领导者开始注意到它。

2011 年，就在深度学习大放异彩之前，在人工智能方面的风险投资总额大约为 1 900 万美元，几乎全都投给了浅层机器学习方法的实际应用。到了 2014 年，这一数字已经涨到了惊人的 3.94 亿美元。这三年里创办了数十家创业公司，试图从深度学习炒作中获利。与此同时，Google、Facebook、百度、微软等大型科技公司已经在内部研究部门进行投资，其金额很可能已经超过了风险投资的现金流。其中只有少数金额被公之于众：2013 年，Google 收购了深度学习创业公司 DeepMind，报道称收购价格为 5 亿美元，这是历史上对人工智能公司的最高收购价格。2014 年，百度在硅谷启动了深度学习研究中心，为该项目投资 3 亿美元。2016 年，深度学习硬件创业公司 Nervana Systems 被英特尔收购，收购价格逾 4 亿美元。

机器学习，特别是深度学习，已成为这些科技巨头产品战略的核心。2015 年末，

笔记

Google 首席执行官 Sundar Pichai 表示：“机器学习这一具有变革意义的核心技术将促使我们重新思考做所有事情的方式。我们用心将其应用于所有产品，无论是搜索、广告、YouTube 还是 Google Play。我们尚处于早期阶段，但你将会看到我们系统性地将机器学习应用于所有这些领域。”

短短五年间从事深度学习的人数从几千人增加到数万人，研究进展也达到了惊人的速度。目前没有迹象表明这种趋势会在短期内放缓。

1.3.5 深度学习的应用

近年来，有许多新面孔进入深度学习领域，而主要的驱动因素之一是该领域所使用工具集的大众化。在早期，从事深度学习需要精通 C++ 和 CUDA，而它们只有少数人才能掌握。如今，具有基本的 Python 脚本技能，就可以从事高级的深度学习研究。这主要得益于 Theano 及随后的 TensorFlow 的开发，以及 Keras 等用户友好型库的兴起。Theano 和 TensorFlow 是两个符号式的张量运算的 Python 框架，都支持自动求微分，这极大地简化了新模型的实现过程。Keras 等用户友好型库则使深度学习变得像操纵乐高积木一样简单。Keras 在 2015 年初发布，并且很快就成为大量创业公司、研究生和研究人员转向该领域的首选深度学习解决方案。

1.3.6 深度神经网络的发展

深度神经网络成为企业投资和研究人员纷纷选择的正确方法，它究竟有何特别之处。深度学习有几个重要的性质，证明了它确实是人工智能的革命，并且能长盛不衰。20 年后我们可能不再使用神经网络，但我们那时所使用的工具都是直接来自现代深度学习及其核心概念。这些重要的性质可大致分为以下三类。

（1）简单。深度学习不需要特征工程，它将复杂的、不稳定的、工程量很大的流程替换为简单的、端到端的可训练模型，这些模型通常只用到五六种不同的张量运算。

（2）可扩展。深度学习非常适合在 GPU 或 TPU 上并行计算，因此可以充分利用摩尔定律。此外，深度学习模型通过对小批量数据进行迭代来训练，因此可以在任意大小的数据集上进行训练（唯一的瓶颈是可用的并行计算能力，而由于摩尔定律，这一限制会越来越小）。

（3）多功能与可复用。深度学习与之前的许多机器学习方法不同，深度学习模型无须从头开始就可以在附加数据上进行训练，因此可用于连续在线学习，这对于大型生产模型而言是非常重要的特性。此外，训练好的深度学习模型可用于其他用途，因此是可以重复使用的。举个例子，可以将一个对图像分类进行训练的深度学习模型应用于视频处理流程。这样我们可以将以前的工作重新投入到日益复杂和强大的模型中。这也使得深度学习可以适用于较小的数据集。

笔记

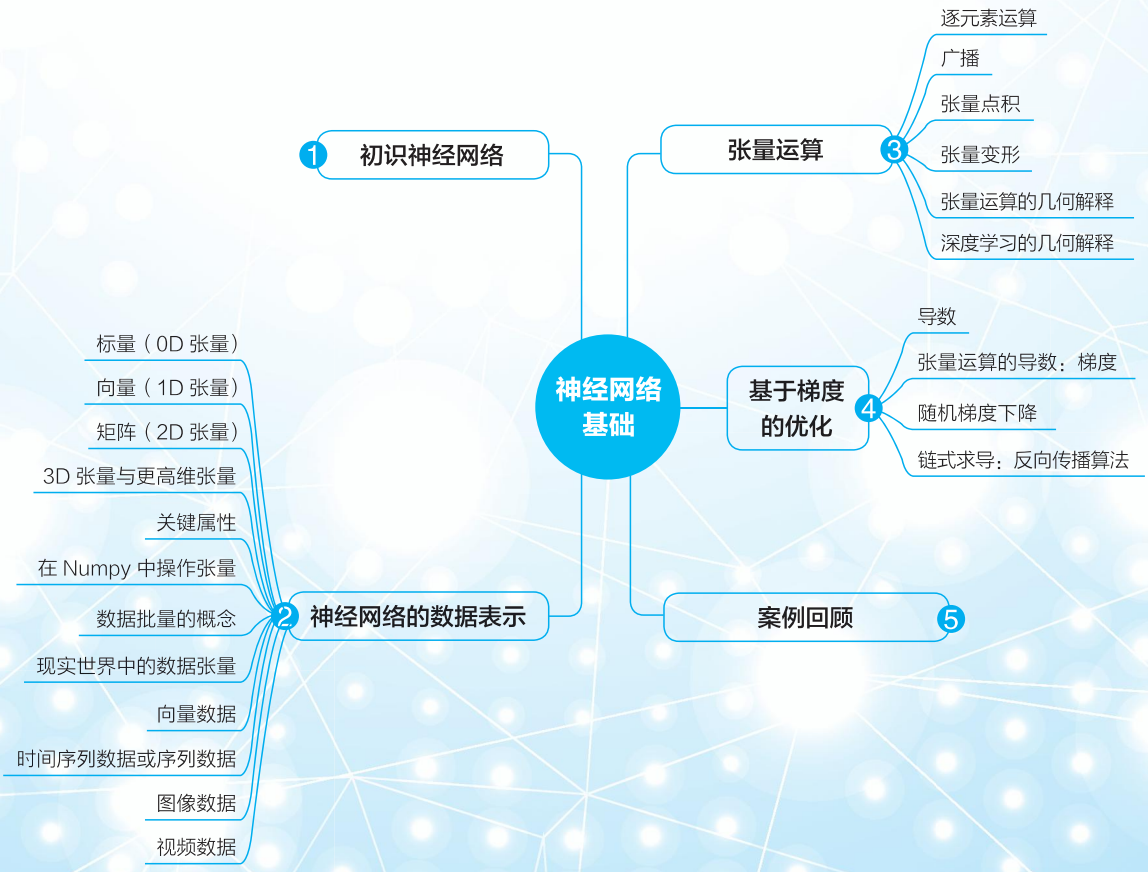
深度学习数年来一直备受关注，目前还没有发现其能力的界限。每过一个月，我们都会学到新的用例和工程改进，从而突破先前的局限。在一次科学革命之后，科学发展的速度通常会遵循一条 S 形曲线：首先是一个快速发展时期，接着随着研究人员受到严重限制而逐渐稳定下来，然后进一步的改进又逐渐增多。深度学习在 2017 年似乎处于这条 S 形曲线的前半部分，在未来几年将会取得更多进展。

第 2 章 神经网络基础

学习目标

1. 理解神经网络的概念。
2. 掌握神经网络的数据表示。
3. 掌握神经网络的张量运算。
4. 了解神经网络如何通过反向传播与梯度下降进行学习。

知识导图



笔记

本章导读

要理解深度学习，需要熟悉很多简单的数学概念：张量、张量运算、微分和梯度下降等。本章目的是用非技术化的文字帮助人们建立对这些概念的直觉。特别地，我们将避免使用数学符号，因为数学符号可能会令没有任何数学背景的人反感，而且其对解释问题也不是绝对必要的。本章将首先给出一个神经网络的示例，引出张量和梯度下降的概念，然后逐个详细介绍。请记住，这些概念对于理解后续章节中的示例至关重要。

2.1 初识神经网络

首先来看一个具体的神经网络示例——使用 Python 的 Keras 库学习手写数字分类。将手写数字的灰度图像（28 像素 ×28 像素）划分到 10 个（0~9）类别中。如果没用过 Keras 或类似的库，可能无法立刻搞懂这个例子中的全部内容。我们将使用 MNIST 数据集，它是机器学习领域的一个经典数据集，其历史几乎与这个领域一样长，而且已被人们深入研究。这个数据集包含 6 000 张训练图像和 10 000 张测试图像，由美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）在 20 世纪 80 年代收集得到。可以将“解决”MNIST 问题看作深度学习的“HelloWorld”，正是用它来验证的算法是否按预期运行。当成为机器学习从业者后，会发现 MNIST 一次又一次地出现在科学论文、博客文章等中。如图 2-1 所示给出了 MNIST 数据集的一些样本。

图 2-1　MNIST 数字图像样本

关于类和标签的说明：

在机器学习中，分类问题中的某个类别叫作类（class）。数据点叫作样本（sample）。某个样本对应的类叫作标签（label）。

如果需要在计算机上运行这个例子，可以尝试安装 Keras。Keras 是由 Python 编写的基于 Tensorflow 或 Theano 的一个高层神经网络 API，具有高度模块化、极简和可扩充等特性，能够实现简易和快速的原型设计，支持 CNN 和 RNN 或者两者的结合，可以无缝切换 CPU 和 GPU。

具体安装步骤如下：

（1）卸载机器上本来安装的 Python，然后安装 Anaconda。Anaconda 中内置 Python，下载地址为 https：//www.anaconda.com/download/#windows。

（2）下载好后进行安装，此处同安装普通软件相同。安装过程中可以选择自动配置环境变量。这样安装结束后直接打开 cmd 输入 python--version 就能够输出安装的 Python 的版本号，如未选择自行配置 Python 环境变量（下面会专门说配置环境变量）。

（3）安装 MinGw，打开 cmd，输入“conda install mingw libpython”，将会自动下载并安装 MinGW。此时 Anaconda 的安装目录下将会有 MinGW 文件夹。

（4）配置环境变量。

① Path 中追加 C：\Anaconda；C：\Anaconda\Scripts。（自己的安装目录）

②新建变量 pythonpath，变量值为 C：\Anaconda\Lib\site-packages\theano。（新建）

③新建 theanorc.txt（C：\Users\kongcong），写入以下代码：

```
[global]
openmp=False
[blas]
ldflags=
[gcc]
cxxflags=-IC:\Anaconda\MinGW
```

（5）安装 Theano。

①通过 https：//github.com/Theano/Theano 下载 Theano 安装包。

②解压到 C：\Anaconda\Lib\site-packages\Theano-master。

③切换到相应目录下执行 pythonsetup.pyinstall。

（6）安装 keras。执行 pipinstallkeras。

（7）修改 C：\Users\kongcong\.keras\keras.json 文件 TensorFlow=>Theano。

（8）修改 D：\ProgramData\Anaconda2\Lib\site-packages\keras\backend__init__.py，line21，_BACKEND='tensorflow'=>_BACKEND='theano'。

MNIST 数据集预先加载在 Keras 库中，其中包括 4 个 Numpy 数组。

加载到 Keras 中的 MNIST 数据集如下：

```
fromkeras.datasetsimportmnist
(train_images,train_labels),(test_images,test_labels)=mnist.load_data()
```

train_images 和 train_labels 组成了训练集（trainingset），模型将从这些数据中进行学习。然后在测试集（testset，即 test_images 和 test_labels）上对模型进行测试。

图像被编码为 Numpy 数组，而标签是数字数组，取值范围为 0~9。图像和标签一一对应。

看一下训练数据：

```
>>>train_images.shape(60000,28,28)
>>>len(train_labels)60000>>>train_labels
array([5,0,4,...,5,6,8],dtype=uint8)
```

下面是测试数据：

```
>>>test_images.shape(10000,28,28)
>>>len(test_labels)10000>>>test_labels
array([7,2,1,...,4,5,6],dtype=uint8)
```

笔记

笔记

接下来的工作流程如下：

首先，将训练数据（train_images 和 train_labels）输入神经网络；其次，通过网络学习将图像和标签关联在一起；最后，通过网络对 test_images 生成预测，而我们将验证这些预测与 test_labels 中的标签是否匹配。

下面来构建网络，不需要理解这个例子的全部内容。

网络架构如下：

```
from keras import models from keras import layers
network=models.Sequential()
network.add(layers.Dense(512,activation='relu',input_shape=(28*28,)))
network.add(layers.Dense(10,activation='softmax'))
```

神经网络的核心组件是层（layer），它是一种数据处理模块，可以将它看成数据过滤器。进去一些数据，出来的数据变得更加有用。具体来说，层从输入数据中提取表示——我们期望这种表示有助于解决手头的问题。大多数深度学习都是将简单的层链接起来，从而实现渐进式的数据蒸馏（datadistillation）。深度学习模型就像是数据处理的筛子，包含一系列越来越精细的数据过滤器（层）。

本例中的网络包含 2 个 Dense 层，它们是密集连接（也叫全连接）的神经层。第二层（也是最后一层）是一个 10 路 Softmax 层，它将返回一个由 10 个概率值（总和为 1）组成的数组。每个概率值表示当前数字图像属于 10 个数字类别中某一个的概率。

要想训练网络，我们还需要选择编译（compile）步骤的三个参数。

损失函数（lossfunction）：网络如何衡量在训练数据上的性能，即网络如何朝着正确的方向前进。

优化器（optimizer）：基于训练数据和损失函数来更新网络的机制。

在训练和测试过程中需要监控的指标（metric）：本例只关心精度，即正确分类的图像所占的比例。

后续章节会详细解释损失函数和优化器的确切用途。

编译步骤如下：

```
network.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

在开始训练之前，我们将对数据进行预处理，将其变换为网络要求的形状，并缩放到所有值都在 [0，1] 区间。例如，之前训练图像保存在一个 uint 8 类型的数组中，其形状为（60000，28，28），取值区间为 [0，255]。我们需要将其变换为一个 float 32 数组，其形状为（60000，28*28），取值范围为 0~1。

准备图像数据：

```
train_images=train_images.reshape((60000,28*28))
train_images=train_images.astype('float32')/255
test_images=test_images.reshape((10000,28*28))
test_images=test_images.astype('float32')/255
```

我们还需要对标签进行分类编码，第 3 章将会对这一步骤进行解释。

准备标签：

```
fromkeras.utilsimportto_categorical
train_labels=to_categorical(train_labels)
test_labels=to_categorical(test_labels)
```

现在我们准备开始训练网络，在 Keras 中这一步是通过调用网络的 fit 方法来完成的——我们在训练数据上拟合（fit）模型。

```
>>>network.fit(train_images,train_labels,epochs=5,batch_size=128)
Epoch1/560000/60000[==============================]-9s-loss:0.2524-acc:0.9273
Epoch2/551328/60000[============================>.....]-ETA:1s-loss:0.1035-acc:0.9692
```

训练过程中显示了两个数字：一个是网络在训练数据上的损失（loss），另一个是网络在训练数据上的精度（acc）。

我们很快就在训练数据上达到了 0.989（98.9%）的精度。现在检查一下模型在测试集上的性能。

```
>>>test_loss,test_acc=network.evaluate(test_images,test_labels)>>>print('test_acc:',test_acc)
test_acc:0.9785
```

测试集精度为 97.8%，比训练集精度低不少。训练精度和测试精度之间的这种差距是过拟合（overfit）造成的。过拟合是指机器学习模型在新数据上的性能往往比在训练数据上要差。

第一个例子到这里就结束了。刚刚看到了如何构建和训练一个神经网络，用不到 20 行的 Python 代码对手写数字进行分类。下一章会详细介绍这个例子中的每一个步骤，并讲解其背后的原理。接下来将要学到张量（输入网络的数据存储对象）、张量运算（层的组成要素）和梯度下降（可以让网络从训练样本中进行学习）。

2.2 神经网络的数据表示

前面例子使用的数据存储在多维 Numpy 数组中，也叫张量（tensor）。一般来说，当前所有机器学习系统都使用张量作为基本数据结构。张量对这个领域非常重要，重要到 Google 的 TensorFlow 都以它来命名。那么什么是张量？

笔记

张量这一概念的核心在于，它是一个数据容器。它包含的数据几乎总是数值数据，因此它是数字的容器。可矩阵是二维张量。张量是矩阵向任意维度的推广［注意，张量的维度（dimension）通常叫作轴（axis）］。

2.2.1 标量（0D 张量）

仅包含一个数字的张量叫作标量（scalar，也叫标量张量、零维张量和 0D 张量）。在 Numpy 中，一个 float 32 或 float 64 的数字就是一个标量张量（或标量数组）。可以用 ndim 属性来查看一个 Numpy 张量的轴的个数。标量张量有 0 个轴（ndim==0）。张量轴的个数也叫作阶（Rank）。下面是一个 Numpy 标量。

```
>>>import numpy as np
>>>x=np.array(12)>>>x
array(12)>>>x.ndim0
```

2.2.2 向量（1D 张量）

数字组成的数组叫作向量（vector）或一维张量（1D 张量）。一维张量只有一个轴。下面是一个 Numpy 向量。

```
>>>x=np.array([12,3,6,14,7])
>>>xarray([12,3,6,14,7])>>>x.ndim1
```

这个向量有 5 个元素，因此被称为 5D 向量。注意不要把 5D 向量和 5D 张量弄混，5D 向量只有一个轴，沿着轴有 5 个维度，而 5D 张量有 5 个轴（沿着每个轴可能有任意个维度）。维度（dimensionality）可以表示沿着某个轴上的元素个数（比如 5D 向量），也可以表示张量中轴的个数（比如 5D 张量）。对于后一种情况，技术上更准确的说法是 5 阶张量（张量的阶数即轴的个数），但 5D 张量这种模糊的写法更常见。

2.2.3 矩阵（2D 张量）

向量组成的数组叫作矩阵（matrix）或二维张量（2D 张量）。矩阵有 2 个轴（通常叫作行和列）。可以将矩阵直观地理解为数字组成的矩形网格。下面是一个 Numpy 矩阵。

```
>>>x=np.array([[5,78,2,34,0],
[6,79,3,35,1],
[7,80,4,36,2]])
>>>x.ndim2
```

第一个轴上的元素叫作行（row），第二个轴上的元素叫作列（column）。在上面的例子中，[5,78,2,34,0] 是 x 的第一行，[5,6,7] 是第一列。

2.2.4 3D 张量与更高维张量

笔记

将多个矩阵组合成一个新的数组，可以得到一个 3D 张量，可以将其直观地理解为数字组成的立方体。下面是一个 Numpy 的 3D 张量。

```
>>>x=np.array([[[5,78,2,34,0],
[6,79,3,35,1],
[7,80,4,36,2]],
[[5,78,2,34,0],
[6,79,3,35,1],
[7,80,4,36,2]],
[[5,78,2,34,0],
[6,79,3,35,1],
[7,80,4,36,2]]])
>>>x.ndim3
```

将多个 3D 张量组合成一个数组，可以创建一个 4D 张量，以此类推。深度学习处理的一般是 0D 到 4D 的张量，但处理视频数据时可能会遇到 5D 张量。

2.2.5 关键属性

张量是由以下三个关键属性来定义的：

（1）轴的个数（阶）。例如，3D 张量有 3 个轴，矩阵有 2 个轴。这在 Numpy 等 Python 库中也叫张量的 ndim。

（2）形状。这是一个整数元组，表示张量沿每个轴的维度大小（元素个数）。例如，前面矩阵示例的形状为（3，5），3D 张量示例的形状为（3，3，5）。向量的形状只包含一个元素，比如（5），而标量的形状为空，即 ( )。

（3）数据类型（在 Python 库中通常叫作 Dtype）。这是张量中所包含数据的类型，例如，张量的类型可以是 float 32、uint 8 和 float 64 等。在极少数情况下，可能会遇到字符（char）张量。注意，Numpy（以及大多数其他库）中不存在字符串张量，因为张量存储在预先分配的连续内存段中，而字符串的长度是可变的，无法用这种方式存储。

为了具体说明 MNIST 例子中处理的数据，首先加载 MNIST 数据集。

```
fromkeras.datasets import mnist
(train_images,train_labels),(test_images,test_labels)=mnist.load_data()
```

接下来，给出张量 train_images 的轴的个数，即 ndim 属性。

```
>>>print(train_images.ndim)3
```

下面是它的形状。

笔记

```
>>>print(train_images.shape)(60000,28,28)
```

下面是它的数据类型，即 dtype 属性。

```
>>>print(train_images.dtype)
uint8
```

因此，这里 train_images 是一个由 8 位整数组成的 3D 张量。更确切地说，它是由 60 000 个矩阵组成的数组，每个矩阵由 28 × 28 个整数组成。每个这样的矩阵都是一张灰度图像，元素取值范围为 0~255。

用 Matplotlib 库（Python 标准科学套件的一部分）来显示这个 3D 张量中的第 4 个样本，如图 2-2 所示。

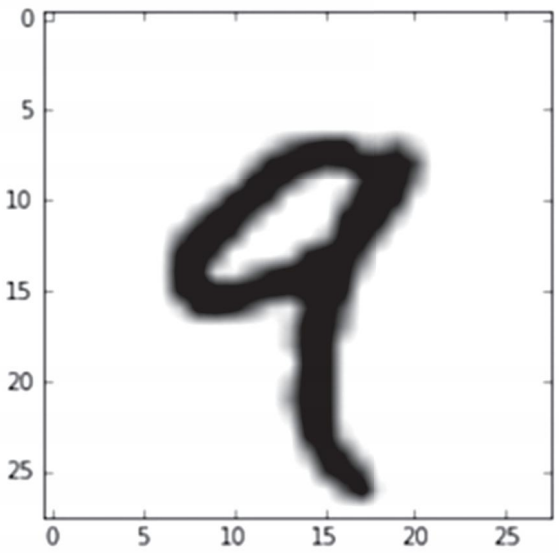


图 2-2　数据集中的第 4 个样本

显示第 4 个样本：

```
digit=train_images[4]
import matplotlib.pyplotasplt
plt.imshow(digit,cmap=plt.cm.binary)
plt.show()
```

2.2.6 在 Numpy 中操作张量

在前面的例子中，我们使用语法 train_images[i] 来选择沿着第一个轴的特定数字。选择张量的特定元素叫作张量切片（tensor slicing）。接下来看一下 Numpy 数组上的张量切片运算。

在下面这个例子中，选择第 10~100 个数字（不包括第 100 个），并将其放在形状为（90，28，28）的数组中。

```
>>>my_slice=train_images[10:100]
>>>print(my_slice.shape)(90,28,28)
```

笔记

它等同于下面这个更复杂的写法，给出了切片沿着每个张量轴的起始索引和结束索引。注意，它等同于选择整个轴。

```
>>>my_slice=train_images[10:100,:,:] ← ------ 等同于前面的例子
>>>my_slice.shape(90,28,28)
>>>my_slice=train_images[10:100,0:28,0:28] ← ------ 也等同于前面的例子
>>>my_slice.shape(90,28,28)
```

一般来说，可以沿着每个张量轴在任意两个索引之间进行选择。例如，可以在所有图像的右下角选出 14 像素 ×14 像素的区域：

```
my_slice=train_images[:,14:,14:]
```

也可以使用负数索引。它与 Python 列表中的负数索引类似，表示与当前轴终点的相对位置。可以在图像中心裁剪出 14 像素 ×14 像素的区域：

```
my_slice=train_images[:,7:-7,7:-7]
```

2.2.7 数据批量的概念

通常来说，深度学习中所有数据张量的第一个轴（0 轴，因为索引从 0 开始）都是样本轴（Samplesaxis，有时也叫样本维度）。在 MNIST 的例子中，样本就是数字图像。

此外，深度学习模型不会同时处理整个数据集，而是将数据拆分成小批量。具体来看，下面是 MNIST 数据集的一个批量，批量大小为 128。

```
batch=train_images[:128]
```

然后是下一个批量：

```
batch=train_images[128:256]
```

然后是第 *n* 个批量：

```
batch=train_images[128*n:128*(n+1)]
```

对于这种批量张量，第一个轴（0 轴）叫作批量轴（batchaxis）或批量维度（batch dimension）。在使用 Keras 和其他深度学习库时，会经常遇到这个术语。

2.2.8 现实世界中的数据张量

我们用几个未来会遇到的示例来具体介绍数据张量。需要处理的数据几乎总是以下类别之一。

笔记

（1）向量数据：2D 张量，形状为（samples，features）。

（2）时间序列数据或序列数据：3D 张量，形状为（samples，timesteps，features）。

（3）图像数据：4D 张量，形状为（samples，height，width，channels）或（samples，channels，height，width）。

（4）视频数据：5D 张量，形状为（samples，frames，height，width，channels）或（samples，frames，channels，height，width）。

2.2.9 向量数据

向量数据是最常见的数据。对于这种数据集，每个数据点都被编码为一个向量，因此一个数据批量就被编码为 2D 张量（向量组成的数组），其中第一个轴是样本轴，第二个轴是特征轴。

我们来看两个例子：

（1）人口统计数据集，其中包括每个人的年龄、邮编和收入。每个人可以表示为包含 3 个值的向量，而整个数据集包含 100 000 个人，因此可以存储在形状为（100000，3）的 2D 张量中。

（2）文本文档数据集，我们将每个文档表示为每个单词在其中出现的次数（字典中包含 20 000 个常见单词）。每个文档可以被编码为包含 20 000 个值的向量（每个值对应于字典中每个单词的出现次数），整个数据集包含 500 个文档，因此可以存储在形状为（500，20000）的张量中。

2.2.10 时间序列数据或序列数据

当时间（或序列顺序）对于数据很重要时，应该将数据存储在带有时间轴的 3D 张量中。每个样本可以被编码为一个向量序列（2D 张量），因此一个数据批量就被编码为一个 3D 张量（图 2-3）。

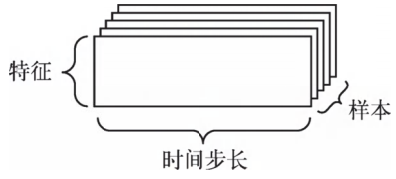


图 2-3　时间序列数据组成的 3D 张量

根据惯例，时间轴始终是第 2 个轴（索引为 1 的轴）。我们来看几个例子：

（1）股票价格数据集。每一分钟，我们将股票的当前价格、前一分钟的最高价格和前一分钟的最低价格保存下来。因此每分钟被编码为一个 3D 向量，整个交易日被编码为一个形状为（390，3）的 2D 张量（一个交易日有 390 分钟），而 250 天的数据则可以保存在一个形状为（250，390，3）的 3D 张量中。这里每个样本是一天的股票数据。

（2）推文数据集。我们将每条推文编码为 280 个字符组成的序列，而每个字符又来自 128 个字符组成的字母表。在这种情况下，每个字符可以被编码为大小为 128 的二进制向量（只有在该字符对应的索引位置取值为 1，其他元素都为 0）。那么每条推文可以被编码

为一个形状为（280，128）的 2D 张量，而包含 100 万条推文的数据集则可以存储在一个形状为（1000000，280，128）的张量中。

笔记

2.2.11 图像数据

图像通常具有三个维度：高度、宽度和颜色深度。虽然灰度图像（比如 MNIST 数字图像）只有一个颜色通道，因此可以保存在 2D 张量中，但按照惯例，图像张量始终都是 3D 张量，灰度图像的彩色通道只有一维。因此，如果图像大小为 256 × 256，那么 128 张灰度图像组成的批量可以保存在一个形状为（128，256，256，1）的张量中，而 128 张彩色图像组成的批量则可以保存在一个形状为（128，256，256，3）的张量中（见图 2-4）。

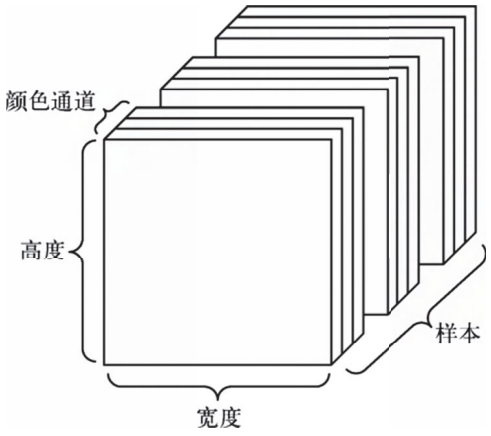


图 2-4　图像数据组成的 4D 张量（通道在前的约定）

图像张量的形状有两种约定：通道在后（channels-last）的约定（在 TensorFlow 中使用）和通道在前（channels-first）的约定（在 Theano 中使用）。Google 的 TensorFlow 机器学习框架将颜色深度轴放在最后：（samples，height，width，color_depth）。与此相反，Theano 将图像深度轴放在批量轴之后：（samples，color_depth，height，width）。如果采用 Theano 约定，前面的两个例子将变成（128，1，256，256）和（128，3，256，256）。Keras 框架同时支持这两种格式。

2.2.12 视频数据

视频数据是现实生活中需要用到 5D 张量的少数数据类型之一。视频可以看作一系列帧，每一帧都是一张彩色图像。由于每一帧都可以保存在一个形状为（height，width，color_depth）的 3D 张量中，因此一系列帧可以保存在一个形状为（frames，height，width，color_depth）的 4D 张量中，而不同视频组成的批量则可以保存在一个 5D 张量中，其形状为（samples，frames，height，width，color_depth）。

举个例子，一个以每秒 4 帧采样的 60 秒 YouTube 视频片段，视频尺寸为 144 × 256，这个视频共有 240 帧。4 个这样的视频片段组成的批量将保存在形状为（4，240，144，256，3）的张量中，总共有 106 168 320 个值。如果张量的数据类型（Dtype）是 float 32，每个值都是 32 位，那么这个张量共有 405 MB。在现实生活中遇到的视频要比它小得多，因为它们不以 float 32 格式存储，而且通常被大大压缩，比如 MPEG 格式。

笔记

2.3 张量运算

所有计算机程序最终都可以简化为二进制输入上的一些二进制运算（AND、OR 和 NOR 等），与此类似，深度神经网络学到的所有变换也都可以简化为数值数据张量上的一些张量运算（tensor operation），例如加上张量、乘以张量等。

在最开始的例子中，我们通过叠加 Dense 层来构建网络。Keras 层的实例如下。

```
keras.layers.Dense(512,activation='relu')
```

这个层可以理解为一个函数，输入一个 2D 张量，返回另一个 2D 张量，即输入张量的新表示。具体而言，这个函数如下（其中 $\boldsymbol{W}$ 是一个 2D 张量，$\boldsymbol{b}$ 是一个向量，两者都是该层的属性）。

```
output=relu(dot(W,input)+b)
```

我们将上式拆开来看。这里有三个张量运算：输入张量和张量 W 之间的点积运算（dot），得到的 2D 张量与向量 b 之间的加法运算（+），最后的 relu 运算。relu（x）是 max（x，0）。

需要注意的是，虽然本节的内容都是关于线性代数表达式，但却找不到任何数学符号。对于没有数学背景的程序员来说，如果用简短的 Python 代码而不是数学方程来表达数学概念，他们将更容易掌握。

2.3.1 逐元素运算

relu 运算和加法都是逐元素（element-wise）的运算，即该运算独立地应用于张量中的每个元素，也就是说，这些运算非常适合大规模并行实现（向量化实现，这一术语来自 1970—1990 年向量处理器超级计算机架构）。如果想对逐元素运算编写简单的 Python 实现，那么可以用 for 循环。下列代码是对逐元素 relu 运算的简单实现。

```
def naive_relu(x):
assertlen(x.shape)==2 ← ------x 是一个 Numpy 的 2D 张量
x=x.copy() ← ------ 避免覆盖输入张量
for i in range(x.shape[0]):
for j in range(x.shape[1]):
x[i,j]=max(x[i,j],0)
return x
```

对于加法采用同样的实现方法。

```
def naive_add(x,y):
assertlen(x.shape)==2 ← ------x 和 y 是 Numpy 的 2D 张量
assertx.shape==y.shape
```

```
    x=x.copy() ← ------ 避免覆盖输入张量
    for i in range(x.shape[0]):
    for j in range(x.shape[1]):
    x[i,j]+=y[i,j]
    return x
```

笔记

根据同样的方法，可以实现逐元素的乘法、减法等。

在实践中处理 Numpy 数组时，这些运算都是优化好的 Numpy 内置函数，这些函数将大量运算交给安装好的基础线性代数子程序（basic linear algebra subprograms，BLAS）实现。BLAS 是低层次的、高度并行的、高效的张量操作程序，通常用 Fortran 或 C 语言来实现。

因此，在 Numpy 中可以直接进行下列逐元素运算，速度非常快。

```
    import numpy as np
    z=x+y ← ------ 逐元素的相加
    z=np.maximum(z,0) ← ------ 逐元素的 relu
```

2.3.2 广播

上一节 naive_add 的简单实现仅支持两个形状相同的 2D 张量相加。但在前面介绍的 Dense 层中，我们将一个 2D 张量与一个向量相加。如果将两个形状不同的张量相加，会发生什么？

如果没有歧义的话，较小的张量会被广播（broadcast），以匹配较大张量的形状。广播包含以下两步。

（1）向较小的张量添加轴（叫作广播轴），使其 ndim 与较大的张量相同。

（2）将较小的张量沿着新轴重复，使其形状与较大的张量相同。

先看一个具体的例子。假设 x 的形状是（32，10），y 的形状是（10，）。首先，给 y 添加空的第一个轴，这样 y 的形状变为（1，10）。然后，将 y 沿着新轴重复 32 次，这样得到的张量 x 的形状为（32，10），并且 y[i，：]==y for i in range（0，32）。现在，可以将 x 和 y 相加，因为它们的形状相同。

在实际的实现过程中并不会创建新的 2D 张量，因为那样做非常低效。重复的操作完全是虚拟的，它只出现在算法中，而没有发生在内存中。但想象将向量沿着新轴重复 10 次，是一种很有用的思维模型。下面是一种简单的实现。

```
    def naive_add_matrix_and_vector(x,y):
    assertlen(x.shape)==2 ← ------x 是一个 Numpy 的 2D 张量
    assertlen(y.shape)==1 ← ------y 是一个 Numpy 向量
    assertx.shape[1]==y.shape[0]
    x=x.copy() ← ------ 避免覆盖输入张量
```

笔记

```
for i in range(x.shape[0]):
for j in range(x.shape[1]):
x[i,j]+=y[j]
returnx
```

如果一个张量的形状是（a，b，…，n，n+1，…，m），另一个张量的形状是（n，n+1，…，m），那么通常可以利用广播对它们做两个张量之间的逐元素运算。广播操作会自动应用于从 a 到 n–1 的轴。

下面这个例子利用广播将逐元素的 Maximum 运算应用于两个形状不同的张量。

```
import numpy as np
x=np.random.random((64,3,32,10)) ← ------x 是形状为 (64,3,32,10) 的随机张量
y=np.random.random((32,10)) ← ------y 是形状为 (32,10) 的随机张量
z=np.maximum(x,y) ← ------ 输出 z 的形状是 (64,3,32,10)，与 x 相同
```

2.3.3 张量点积

点积运算，也叫张量积（tensorproduct，不要与逐元素的乘积弄混），它是最常见也最有用的张量运算。点积运算与逐元素的运算不同，它将输入张量的元素合并在一起。

在 Numpy、Keras、Theano 和 TensorFlow 中，都是用 * 实现逐元素乘积。TensorFlow 中的点积使用了不同的语法，但在 Numpy 和 Keras 中，都是用标准的 dot 运算符来实现点积。

```
import numpy as np
z=np.dot(x,y)
```

数学符号中的点（ · ）表示点积运算。

```
z=x · y
```

从数学的角度来看，点积运算做了什么？我们首先看一下两个向量 x 和 y 的点积。其计算过程如下。

```
def naive_vector_dot(x,y):
assertlen(x.shape)==1|x 和 y 都是 Numpy 向量
assertlen(y.shape)==1|
assertx.shape[0]==y.shape[0]
z=0.
for i in range(x.shape[0]):
z+=x[i]*y[i]
return z
```

注意，两个向量之间的点积是一个标量，而且只有元素个数相同的向量之间才能做点

积。还可以对一个矩阵 ***x*** 和一个向量 ***y*** 做点积，返回值是一个向量，其中每个元素分别对应的是 ***y*** 和 ***x*** 的每一行之间的点积。其实现过程如下。

笔记

```
import numpy as np
def naive_matrix_vector_dot(x,y):
assertlen(x.shape)==2 ← ------x 是一个 Numpy 矩阵
assertlen(y.shape)==1 ← ------y 是一个 Numpy 向量
assertx.shape[1]==y.shape[0] ← ------x 的第 1 维和 y 的第 0 维大小必须相同
z=np.zeros(x.shape[0]) ← ------ 这个运算返回一个全是 0 的向量，其形状与 x.shape[0] 相同
for i in range(x.shape[0]):
for j in range(x.shape[1]):
z[i]+=x[i,j]*y[j]
return z
```

还可以复用前面写过的代码，从中可以看出矩阵 - 向量点积与向量点积之间的关系。

```
def naive_matrix_vector_dot(x,y):
z=np.zeros(x.shape[0])
for i in range(x.shape[0]):
z[i]=naive_vector_dot(x[i,:],y)
return z
```

注意，如果两个张量中有一个的 ndim 大于 1，那么 dot 运算就不再是对称的，也就是说，dot（***x***，***y***）不等于 dot（***y***，***x***）。

当然，点积可以推广到具有任意个轴的张量。最常见的应用可能就是两个矩阵之间的点积。对于两个矩阵 ***x*** 和 ***y***，当且仅当 *x*.shape[1]==*y*.shape[0] 时，才可以对它们做点积 [dot（***x***，***y***）]。得到的结果是一个形状为（*x*.shape[0]，*y*.shape[1]）的矩阵，其元素为 ***x*** 的行与 ***y*** 的列之间的点积。其简单实现如下。

```
def naive_matrix_dot(x,y):
assertlen(x.shape)==2|x 和 y 都是 Numpy 矩阵
assertlen(y.shape)==2|
assertx.shape[1]==y.shape[0] ← ------x 的第 1 维和 y 的第 0 维大小必须相同

z=np.zeros((x.shape[0],y.shape[1])) ← ------ 这个运算返回特定形状的零矩阵
for i in range(x.shape[0]): ← ------ 遍历 x 的所有行……
for j in range(y.shape[1]): ← ------……然后遍历 y 的所有列
row_x=x[i,:]
column_y=y[:,j]
z[i,j]=naive_vector_dot(row_x,column_y)
return z
```

笔记

为了便于理解点积的形状匹配，可以将输入张量和输出张量像图 2-5 中那样排列，利用可视化来帮助理解。

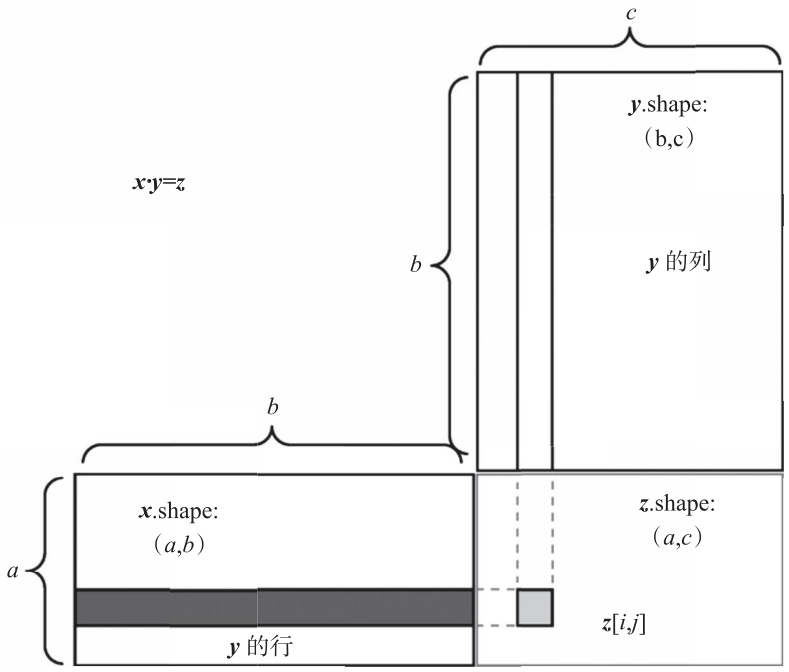


图 2-5　图解矩阵点积

在图 2-5 中，***x***、***y*** 和 ***z*** 都用矩形表示（元素按矩形排列）。***x*** 的行数和 ***y*** 的列数必须大小相同，因此 ***x*** 的宽度一定等于 ***y*** 的高度。如果打算开发新的机器学习算法，可能经常要画这种图。

可以对更高维的张量做点积，只要其形状匹配遵循与前面 2D 张量相同的原则：

```
(a,b,c,d).(d,)->(a,b,c)
(a,b,c,d).(d,e)->(a,b,c,e)
```

以此类推。

2.3.4 张量变形

第三个重要的张量运算是张量变形（Tensor Reshaping）。虽然前面神经网络第一个例子的 Dense 层中没有用到它，但在将图像数据输入神经网络之前，我们在预处理时用到了这个运算。

```
train_images=train_images.reshape((60000,28*28))
```

张量变形是指改变张量的行和列，以得到想要的形状。变形后的张量的元素总个数与初始张量相同。以下这个简单的例子可以帮助我们理解张量变形。

```
>>>x=np.array([[0.,1.],
[2.,3.],
[4.,5.]])
>>>print(x.shape)(3,2)
>>>x=x.reshape((6,1))
>>>x
array([[0.],
[1.],
[2.],
[3.],
[4.],
[5.]])
>>>x=x.reshape((2,3))>>>x
array([[0.,1.,2.],
[3.,4.,5.]])
```

笔记

经常遇到的一种特殊的张量变形是转置（transposition）。对矩阵做转置是指将行和列互换，使 *x*[i，：] 变为 *x*[：，i]。

```
>>>x=np.zeros((300,20)) ← ------ 创建一个形状为 (300,20) 的零矩阵
>>>x=np.transpose(x)
>>>print(x.shape)(20,300)
```

2.3.5 张量运算的几何解释

对于张量运算所操作的张量，其元素可以被解释为某种几何空间内点的坐标，因此所有的张量运算都有几何解释。举个例子，我们来看加法。首先有这样一个向量：

```
A=[0.5,1]
```

它是二维空间中的一个点（图 2-6）。常见的做法是将向量描绘成原点到这个点的箭头，如图 2-7 所示。

笔记

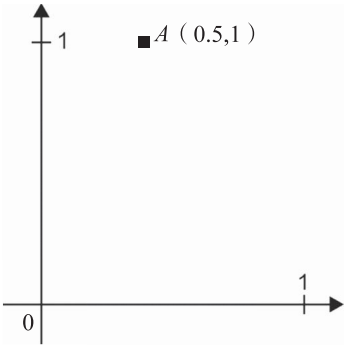


图 2-6　二维空间中的一个点

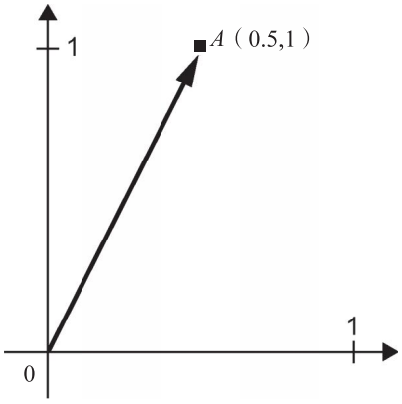


图 2-7　将二维空间中的一个点描绘成一个箭头

假设又有一个点 B=[1，0.25]，将它与前面的 A 相加。从几何上来看，这相当于将两个向量箭头连在一起，得到的位置表示两个向量之和对应的向量（见图 2-8）。

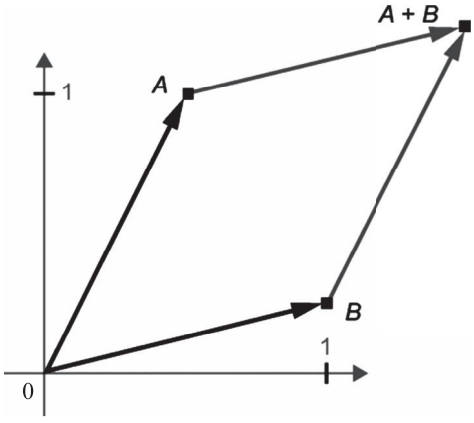


图 2-8　两个向量之和的几何解释

通常来说，仿射变换、旋转和缩放等基本的几何操作都可以表示为张量运算。要将一个二维向量旋转 θ，可以通过与一个 2×2 矩阵做点积来实现，这个矩阵为 $\boldsymbol{R}$=[$\boldsymbol{u}$，$\boldsymbol{v}$]，其中 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 都是平面向量：$\boldsymbol{u}$=（$\cos\theta$，$\sin\theta$），$\boldsymbol{v}$=（$-\sin\theta$，$\cos\theta$）。

2.3.6 深度学习的几何解释

笔记

前面讲过，神经网络完全由一系列张量运算组成，而这些张量运算都只是输入数据的几何变换。因此，可以将神经网络解释为高维空间中非常复杂的几何变换，这种变换可以通过许多简单的步骤来实现。

对于三维的情况，下面这个思维图像是很有用的。想象有两张彩纸：一张红色，一张蓝色。将其中一张纸放在另一张上。现在将两张纸一起揉成小球。这个皱巴巴的纸球就是输入数据，每张纸对应于分类问题中的一个类别。神经网络（或者任何机器学习模型）要做的就是找到可以让纸球恢复平整的变换，从而能够再次让两个类别明确可分。通过深度学习，这一过程可以用三维空间中一系列简单的变换来实现，比如用手指对纸球做变换，每次做一个动作，如图 2-9 所示。

图 2-9　解开复杂的数据流形

让纸球恢复平整就是机器学习的内容。它为复杂的、高度折叠的数据流形找到简洁的表示。现在应该能够很好地理解，为什么深度学习特别擅长这一点：它将复杂的几何变换逐步分解为一长串基本的几何变换，这与人类展开纸球所采取的策略大致相同。深度网络的每一层都通过变换使数据解开一点点，许多层堆叠在一起，就可以实现非常复杂的解开过程。

2.4 基于梯度的优化

前面介绍过，我们的第一个神经网络示例中，每个神经层都用下述方法对输入数据进行变换。

```
output=relu(dot(W,input)+b)
```

在这个表达式中，W 和 b 都是张量，均为该层的属性。它们被称为该层的权重（weight）或可训练参数（trainable parameter），分别对应 Kernel 和 Bias 属性。这些权重包含网络从观察训练数据中学到的信息。

一开始，这些权重矩阵取较小的随机值，这一步叫作随机初始化（random initialization）。当然，W 和 b 都是随机的，relu（dot（W，input）+b）肯定不会得到任何有用的表示。虽然得到的表示是没有意义的，但这是一个起点。下一步则是根据反馈信号逐渐调节这些权重。这个逐渐调节的过程叫作训练，也就是机器学习中的学习。

笔记

上述过程发生在一个训练循环（training loop）内，其具体过程如下（必要时一直重复这些步骤）：

（1）抽取训练样本 x 和对应目标 y 组成的数据批量。

（2）在 x 上运行网络［这一步叫作前向传播（forward pass）］，得到预测值 y_pred。

（3）计算网络在这批数据上的损失，用于衡量 y_pred 和 y 之间的距离。

（4）更新网络的所有权重，使网络在这批数据上的损失略微下降。

最终得到的网络在训练数据上的损失非常小，即预测值 y_pred 和预期目标 y 之间的距离非常小。网络“学会”将输入映射到正确的目标。乍一看比较复杂，但如果将其简化为基本步骤，那么会变得非常简单。

第一步看起来非常简单，只是输入 / 输出（I/O）的代码。第二步和第三步仅仅是一些张量运算的应用，因此完全可以利用上一节学到的知识来实现这两步。难点在于第四步：更新网络的所有权重。考虑网络中某个权重系数，怎么知道这个系数应该增大还是减小，以及变化多少？

一种简单的解决方案是，保持网络中其他权重不变，只考虑某个标量系数，让其尝试不同的取值。假设这个系数的初始值为 0.3。对一批数据做完前向传播后，网络在这批数据上的损失是 0.5。如果将这个系数的值改为 0.35 并重新运行前向传播，损失会增大到 0.6。但如果将这个系数减小到 0.25，损失会减小到 0.4。在这个例子中，将这个系数减小 0.05 似乎有助于使损失最小化。对于网络中的所有系数都要重复这一过程。

但这种方法是非常低效的，因为对每个系数（系数很多，通常有上千个，有时甚至多达上百万个）都需要计算两次前向传播（计算代价很大）。一种更好的方法是利用网络中所有运算都是可微的（differentiable）这一事实，计算损失相对于网络系数的梯度（gradient），然后向梯度的反方向改变系数，从而使损失降低。

如果已经了解可微和梯度这两个概念，可以直接跳到 2.4.3 节。如果不了解，下面两小节将有助于理解这些概念。

2.4.1 导数

假设有一个连续的光滑函数 $f(x)=y$，将实数 x 映射为另一个实数 y。由于函数是连续的，x 的微小变化只能导致 y 的微小变化——这就是函数连续性的直观解释。假设 x 增大了一个很小的因子 epsilon_x，这导致 y 也发生了很小的变化，即 epsilon_y：

```
f(x+epsilon_x)=y+epsilon_y
```

此外，由于函数是光滑的（函数曲线没有突变的角度），在某个点 p 附近，如果 epsilon_x 足够小，就可以将 $f(x)$ 近似为斜率为 a 的线性函数，这样 epsilon_y 就变成了 a*epsilon_x：

```
f(x+epsilon_x)=y+a*epsilon_x
```

笔记

显然，只有在 x 足够接近 p 时，这个线性近似才有效。

斜率 a 被称为 $f(x)$ 在 p 点的导数（derivative）。如果 a 是负的，说明 x 在 p 点附近的微小变化将导致 $f(x)$ 减小（见图 2-10）；如果 a 是正的，那么 x 的微小变化将导致 $f(x)$ 增大。此外，a 的绝对值（导数大小）表示增大或减小的速度快慢。

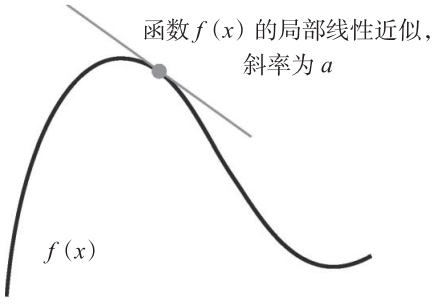


图 2-10　$f(x)$ 在 p 点的导数

对于每个可微函数 $f(x)$（可微的意思是“可以被求导”。例如，光滑的连续函数可以被求导），都存在一个导数函数 $f'(x)$，将 x 的值映射为 $f(x)$ 在该点的局部线性近似的斜率。例如，cos$x$ 的导数是 –sin$x$，$f(x)=a*x$ 的导数是 $f'(x)=a$，等等。

如果想要将 x 改变一个小因子 epsilon_x，目的是将 $f(x)$ 最小化，并且知道 $f(x)$ 的导数，那么问题解决了：导数完全描述了改变 x 后 $f(x)$ 会如何变化。如果希望减小 $f(x)$ 的值，只需将 x 沿着导数的反方向移动一小步。

2.4.2 张量运算的导数：梯度

梯度（gradient）是张量运算的导数。它是导数这一概念向多元函数导数的推广。多元函数是以张量作为输入的函数。

假设有一个输入向量 $\boldsymbol{x}$、一个矩阵 $\boldsymbol{W}$、一个目标 y 和一个损失函数 Loss。可以用 $\boldsymbol{W}$ 来计算预测值 y_pred，然后计算损失，或者说预测值 y_pred 和目标 y 之间的距离。

```
y_pred=dot(W,x)
loss_value=loss(y_pred,y)
```

如果输入数据 x 和 y 保持不变，那么这可以看作将 $\boldsymbol{W}$ 映射到损失值的函数。

```
loss_value=f(W)
```

假设 $\boldsymbol{W}$ 的当前值为 W0。f 在 W0 点的导数是一个张量 gradient（f（W0））, 其形状与 $\boldsymbol{W}$ 相同，每个系数 gradient（f（W0[i，j]））表示改变 W0[i，j] 时 loss_value 变化的方向和大小。张量 gradient（f（W0））是函数 $f(\boldsymbol{W})$ =loss_value 在 W0 的导数。

单变量函数 $f(x)$ 的导数可以看作函数 $f(x)$ 曲线的斜率。同样，gradient（f（W0））也可以看作表示 $f(\boldsymbol{W})$ 在 W0 附近曲率（curvature）的张量。

笔记

对于一个函数 $f(x)$，可以通过将 x 向导数的反方向移动一小步来减小 $f(x)$ 的值。同样，对于张量的函数 $f(\boldsymbol{W})$，也可以通过将 $\boldsymbol{W}$ 向梯度的反方向移动来减小 $f(\boldsymbol{W})$，比如 W1=W0-step*gradient（f（W0））, 其中 step 是一个很小的比例因子。也就是说，沿着曲率的反方向移动，直观上来看在曲线上的位置会更低。注意，比例因子 step 是必需的，因为 gradient（f（W0））只是 W0 附近曲率的近似值，不能离 W0 太远。

2.4.3 随机梯度下降

给定一个可微函数，理论上可以用解析法找到它的最小值：函数的最小值是导数为 0 的点，因此只需找到所有导数为 0 的点，然后计算函数在其中哪个点具有最小值。将这一方法应用于神经网络，就是用解析法求出最小损失函数对应的所有权重值。可以通过对方程 gradient（$f(\boldsymbol{W})$）=0 求解 $\boldsymbol{W}$ 来实现这一方法。这是包含 N 个变量的多项式方程，其中 N 是网络中系数的个数。N=2 或 N=3 时可以对这样的方程求解，但对于实际的神经网络是无法求解的，因为参数的个数不会少于几千个，而且经常有上千万个。

相反，可以使用 2.4 节开头总结的四步算法：基于当前在随机数据批量上的损失，一点一点地对参数进行调节。由于处理的是一个可微函数，因此可以计算出它的梯度，从而有效地实现第四步。沿着梯度的反方向更新权重，损失每次都会变小一点。

（1）抽取训练样本 x 和对应目标 y 组成的数据批量。

（2）在 x 上运行网络，得到预测值 y_pred。

（3）计算网络在这批数据上的损失，用于衡量 y_pred 和 y 之间的距离。

（4）计算损失相对于网络参数的梯度［一次反向传播（backwardpass）］。

（5）将参数沿着梯度的反方向移动一点，比如 W=-step*gradient，从而使这批数据上的损失减小一点。

这种描述的方法叫作小批量随机梯度下降（mini-batch stochastic gradient descent，小批量 SGD）。随机（stochastic）是指每批数据都是随机抽取的（stochastic 是 random 在科学上的同义词，中文意思都是“随机的”）。如图 2-11 所示给出了一维的情况，网络只有一个参数，并且只有一个训练样本。

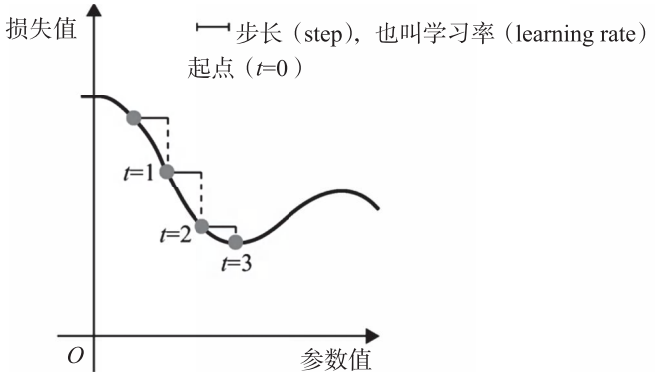


图 2-11　沿着一维损失函数曲线的随机梯度下降（一个需要学习的参数）

笔记

直观上来看，为 step 因子选取合适的值是很重要的。如果取值太小，则沿着曲线的下降需要很多次迭代，而且可能会陷入局部极小点。如果取值太大，则更新权重值之后可能会出现在曲线上完全随机的位置。

注意，小批量 SGD 算法的一个变体是每次迭代时只抽取一个样本和目标，而不是抽取一批数据，这叫作真 SGD（有别于小批量 SGD）。还有另一种极端，每一次迭代都在所有数据上运行，这叫作批量 SGD。这样做的话，每次更新都更加准确，但计算代价也高得多。这两个极端之间的有效折中则是选择合理的批量大小。

图 2-11 描述的是一维参数空间中的梯度下降，但在实践中需要在高维空间中使用梯度下降。神经网络的每一个权重参数都是空间中的一个自由维度，网络中可能包含数万个甚至上百万个参数维度。为了让对损失曲面有更直观的认识，还可以将梯度下降沿着二维损失曲面可视化，如图 2-12 所示。但不可能将神经网络的实际训练过程可视化，因为无法用人类可以理解的方式来可视化 1 000 000 维空间。因此，在这些低维表示中形成的直觉在实践中不一定总是准确的。这一直是深度学习研究的问题来源。

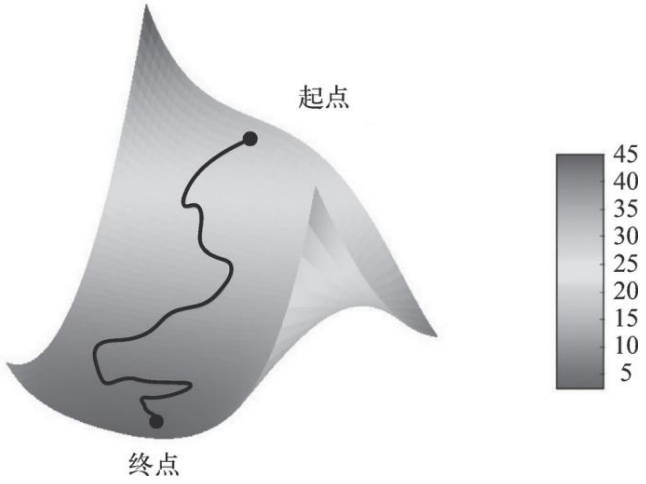


图 2-12　沿着二维损失曲面的梯度下降（两个需要学习的参数）

此外，SGD 还有多种变体，其区别在于计算下一次权重更新时还要考虑上一次权重更新，而不是仅仅考虑当前梯度值，比如带动量的 SGD、Adagrad 和 RMSProp 等变体。这些变体被称为优化方法（optimization method）或优化器（optimizer）。其中动量的概念尤其值得关注，它在许多变体中都有应用。动量解决了 SGD 的两个问题：收敛速度和局部极小点。如图 2-13 所示给出了损失作为网络参数的函数的曲线。

笔记

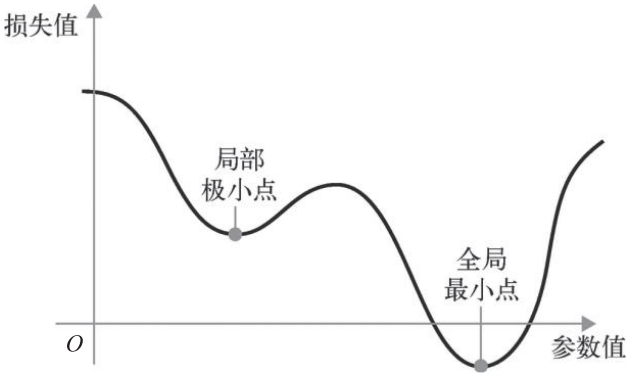


图 2-13　损失作为网络参数的函数的曲线

如图 2-13 所示，在某个参数值附近，有一个局部极小点（local minimum）：在这个点附近，向左移动和向右移动都会导致损失值增大。如果使用小学习率的 SGD 进行优化，那么优化过程可能会陷入局部极小点，导致无法找到全局最小点。

使用动量方法可以避免这样的问题，这一方法的灵感来源于物理学。有一种有用的思维图像，就是将优化过程想象成一个小球从损失函数曲线上滚下来。如果小球的动量足够大，那么它不会卡在峡谷里，最终会到达全局最小点。动量方法的实现过程是每一步都移动小球，不仅要考虑当前的斜率值（当前的加速度），还要考虑当前的速度（来自之前的加速度）。这在实践中指的是，更新参数 w 不仅要考虑当前的梯度值，还要考虑上一次的参数更新，其简单实现如下所示：

```
past_velocity=0.
momentum=0.1 ← ------ 不变的动量因子 whileloss>0.01: ← ------ 优化循环
w,loss,gradient=get_current_parameters()
velocity=past_velocity*momentum-learning_rate*gradient
w=w+momentum*velocity-learning_rate*gradient
past_velocity=velocity
update_parameter(w)
```

2.4.4 链式求导：反向传播算法

在前面的算法中，我们假设函数是可微的，因此可以明确计算其导数。在实践中，神经网络函数包含许多连接在一起的张量运算，每个运算都有简单的、已知的导数。例如，下面这个网络 f 包含 3 个张量运算 a、b 和 c，还有 3 个权重矩阵 $\boldsymbol{W}1$、$\boldsymbol{W}2$ 和 $\boldsymbol{W}3$。

```
f(W1,W2,W3)=a(W1,b(W2,c(W3)))
```

根据微积分的知识，这种函数链可以利用下面这个恒等式进行求导，它称为链式法则（chain rule）：[$(f(g(x)))'=f'(g(x))*g'(x)$]。将链式法则应用于神经网络梯度值的计算，得到的算法叫作反向传播（back propagation，有时也叫反式微分，reverse-mode differentiation）。反向传播从最终损失值开始，从最顶层反向作用至最底层，利用链式法

则计算每个参数对损失值的贡献大小。

现在以及未来数年，人们将使用能够进行符号微分（symbolic differentiation）的现代框架来实现神经网络，比如 TensorFlow。也就是说，给定一个运算链，并且已知每个运算的导数，这些框架就可以利用链式法则来计算这个运算链的梯度函数，将网络参数值映射为梯度值。对于这样的函数，反向传播就简化为调用这个梯度函数。由于符号微分的出现，无须手动实现反向传播算法。因此，我们不会在本节推导反向传播的具体公式。只需充分理解基于梯度的优化方法的工作原理。

笔记

2.5 案例回顾

现在应该对神经网络背后的原理有了大致的了解。看一下第一个例子，并根据前面三节学到的内容来重新阅读这个例子中的每一段代码。

1. 输入数据

```
(train_images,train_labels),(test_images,test_labels)=mnist.load_data()
train_images=train_images.reshape((60000,28*28))
train_images=train_images.astype('float32')/255
test_images=test_images.reshape((10000,28*28))
test_images=test_images.astype('float32')/255
```

输入图像保存在 float 32 格式的 Numpy 张量中，形状分别为（60000，784）（训练数据）和（10000，784）（测试数据）。

2. 构建网络

```
network=models.Sequential()
network.add(layers.Dense(512,activation='relu',input_shape=(28*28,)))
network.add(layers.Dense(10,activation='softmax'))
```

这个网络包含两个 Dense 层，每层都对输入数据进行一些简单的张量运算，这些运算都包含权重张量。权重张量是该层的属性，里面保存了网络所学到的知识（knowledge）。

3. 网络的编译

```
network.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

categorical_crossentropy 是损失函数，它是用于学习权重张量的反馈信号，在训练阶段应使它最小化。减小损失是通过小批量随机梯度下降来实现的。梯度下降的具体方法由第一个参数给定，即 Rmsprop 优化器。

4. 训练循环

```
network.fit(train_images,train_labels,epochs=5,batch_size=128)
```

笔记

在调用 fit 时，网络开始在训练数据上进行迭代（每个小批量包含 128 个样本），共迭代 5 次［在所有训练数据上迭代一次叫作一个轮次（epoch）］。在每次迭代过程中，网络会计算批量损失相对于权重的梯度，并相应地更新权重。5 轮之后，网络进行了 2 345 次梯度更新（每轮 469 次），网络损失值将变得足够小，使得网络能够以很高的精度对手写数字进行分类。