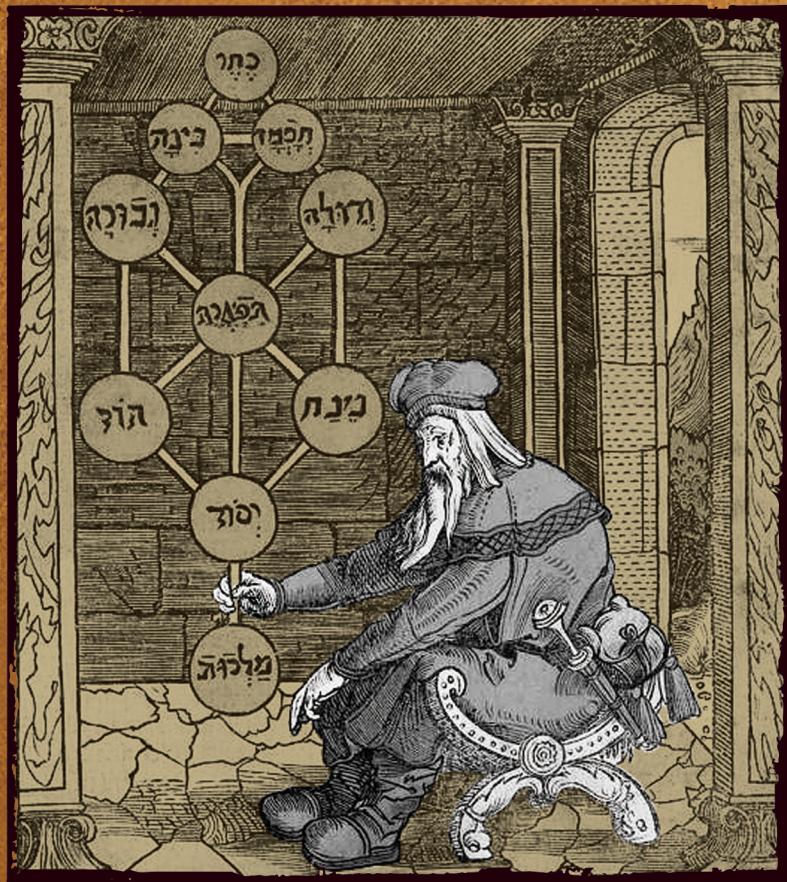


深入理解神经网络

从逻辑回归到CNN

张觉非 ◎著



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS



— 张觉非 —

本科毕业于复旦大学计算机系，
于中国科学院古脊椎动物与
古人类研究所取得古生物学硕士学位，
目前在互联网行业
从事机器学习算法相关工作。

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

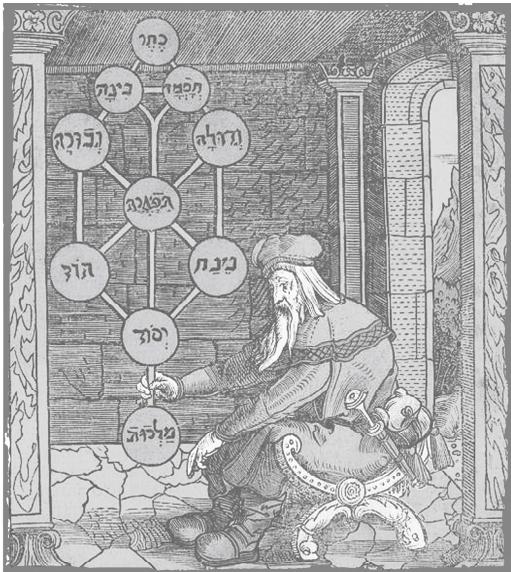
我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

深入理解神经网络

从逻辑回归到CNN

张觉非 ◎著



人民邮电出版社
北京

图书在版编目（C I P）数据

深入理解神经网络：从逻辑回归到CNN / 张觉非著.
-- 北京 : 人民邮电出版社, 2019.9
(图灵原创)
ISBN 978-7-115-51723-4

I. ①深… II. ①张… III. ①人工神经网络—研究
IV. ①TP183

中国版本图书馆CIP数据核字(2019)第155659号

内 容 提 要

本书以神经网络为线索，沿着从线性模型到深度学习的路线讲解神经网络的原理和实现。本书将数学基础知识与机器学习和神经网络紧密结合，包含线性模型的结构与局限、损失函数、基于一阶和二阶信息的优化算法、模型自由度与正则化、神经网络的表达能力、反向传播与计算图自动求导、卷积神经网络等主题，帮助读者建立基于数学原理的较深刻的洞见和认知。本书还提供了逻辑回归、多层全连接神经网络和多种训练算法的 Python 实现，以及运用 TensorFlow 搭建和训练多种卷积神经网络的代码实例。

本书适合渴望加深对神经网络和深度学习原理理解的高年级本科生与研究生、广大程序员与工程师，以及对机器学习的原理和编程实现感兴趣的所有读者阅读。

-
- ◆ 著 张觉非
 - 责任编辑 陈兴璐
 - 责任印制 周昇亮
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
 - 邮编 100164 电子邮件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 北京 印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 20.25
 - 字数: 479千字 2019年9月第1版
 - 印数: 1~3 000册 2019年9月北京第1次印刷
-

定价: 89.00元

读者服务热线: (010)51095183转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京东工商广登字 20170147 号

献给周怡萍女士，我生命中的奇异吸引子。

前　　言

“理论是灰色的，我亲爱的朋友，而生命的金树长青。”

——歌德，《浮士德》

近年来，深度学习的浪潮席卷了学术界和产业界，它强大的能力给人们留下了深刻的印象。在产业界，一些已经成熟应用机器学习技术的领域，如计算广告、推荐系统、机器视觉、自然语言处理等，纷纷尝试深度学习并取得了良好的效果。各个尚未应用机器学习的领域也都在积极进行智能化的尝试。另外，开源社区贡献的众多框架、工具和平台，也为这股浪潮推波助澜。可以说，机器学习，尤其是神经网络与深度学习，是当今最被寄予厚望的技术。

不论这股浪潮将止于何处，也不论它最终能否达到人们的期许，机器学习的思想都已经深深植入了人们的心中。在工程和业务实践中，将数据和建模纳入解决方案已是一种常规做法。机器学习已成为工程师武器库中一个重要的工具，越来越多工程师开始尝试了解和应用机器学习技术。

但是人们会发现，机器学习有着艰深的理论背景，不能透彻理解原理，就难以在实践中自由地运用。当人们试图进入机器学习领域时，数学原理是横亘在面前的一座大山。笔者通过自身经历以及与他人的交流，深切体会到，广大学生和工程师在基础理论知识的理解和运用上普遍有所欠缺，这给他们理解机器学习的原理制造了障碍。

举个例子，对于梯度和链式法则，许多人只能在一元函数情况下理解，大量文章和书籍也都只在一元情况下做浅尝辄止的讲解。而当人们阅读代码时，会发现其中满是矩阵和向量。原理与实现之间仿佛弥漫着一团迷雾，不穿透这团迷雾，对这些概念的理解就只能是浮光掠影、隔靴搔痒。笔者曾读到有文章将梯度下降的矩阵实现称为“向量化”。其实何须向量化，多元函数的梯度与求导原本就是用矩阵和向量语言来描述的。

再比如，神经网络与深度学习的火热虽然给机器学习这门学科带来进步和活力，但是也给人们造成一个错误的印象，即它是全新的、颠覆性的、与传统割裂的。其实神经网络与深度学习深

深扎根于传统机器学习之中。读者可以看到，本书的大量篇幅都在讲解线性模型，这是因为神经网络与深度学习的结构和训练方法都扎根于线性模型之中。

目标读者

笔者遇到过不少理工科相关专业的工程师，他们学习过微积分、线性代数和概率论等基础课程，但是对这些知识的理解尚没有融会贯通，不知如何运用，而且搁置多年，遗忘严重。还有一些高年级本科生和研究生感觉各门基础课程仿佛是分散的、孤立的，不理解它们的联系和用途。本书的目标就是帮助这些读者回忆并夯实基础知识，深入理解机器学习，特别是神经网络和深度学习的原理。

本书以神经网络为线索，沿着从线性模型到深度学习的路线，串起核心知识点。数学内容在必要和恰当的时机引入，从基础讲起，不留黑盒。具有理工科背景的读者能够容易地回忆起相关知识，而不需要再回头求诸于大部头教科书。本书内容取舍有当，紧紧围绕主题，使读者能将数学与它在机器学习中的应用紧密结合起来。

本书试图帮助读者打通关节，提高视角，加深洞见，做到知其然并知其所以然。在当今人工智能时代，新方法不断涌现，新工具层出不穷，但是万变不离其宗，新的方法和工具都深深扎根于原理之中。把众多方法比作岛屿，在水面之下看，大大小小的岛屿就不再是孤立的、个别的，它们都建筑于基岩之上，基岩就是机器学习的基础理论。掌握了基础理论，就可以对各种方法形成统一而深刻的洞见，在工程实践中拥有坚实有效的理论依据，并能够快速理解和运用业界新涌现的众多方法和工具。

具体来说，本书适合以下几类读者：

- 渴望进入这一领域的高年级本科生和研究生，本书帮助这类读者将数学基础知识与机器学习和神经网络结合起来，深入理解原理；
- 希望了解神经网络与机器学习的广大程序员与工程师，这类读者需要花更多一些力气回忆数学知识，本书包含了他们需要的全部知识点；
- 对模型的编程实现感兴趣的读者，本书包含逻辑回归、多层全连接神经网络以及各种训练算法的 Python 实现，可供这类读者学习和参考；
- 业界的机器学习、数据挖掘工程师，本书关于模型原理的高级主题，特别是关于模型自由度与偏置-方差权衡方面的内容，可为他们提供一些有趣的洞见。

明确本书不包含的内容以及不适合的读者同样重要，以下是本书不涉及的内容：

- 本书围绕着神经网络这个主题，并涵盖了机器学习学科的相当一部分重点内容，但并不涵盖机器学习的全部领域；
- 本书包括逻辑回归、多层全连接神经网络以及多种训练算法的 Python 实现，但书中代码实现的目的是为了理解原理，本书并不是一本实现各类机器学习模型的指南；
- 本书包含使用 TensorFlow 搭建并训练模型的实例，但本书远远不是一本 TensorFlow 的实用教材；
- 本书从原理上讲解了模型超参数、自由度、过拟合与欠拟合等概念，但本书并非关于调参和优化模型的实践指南，也不包含特征预处理、特征工程等方面的内容；
- 本书以卷积神经网络为例讲解深度学习，涵盖了训练深度神经网络的方法、问题以及技术，但本书并不包含深度学习众多五花八门的新应用领域和网络结构。

内容概览

本书沿着从线性模型到神经网络，再到深度学习的路径，层层递进，逐渐深入。这是一条从根至叶、自简入繁的天然路径。各章节的安排如下。

- 第 1 章以逻辑回归为例讲解线性模型。这一章在引入逻辑回归的定义后回顾必要的向量几何，使读者对逻辑回归的原理、特性和局限形成清晰的认识。
- 第 2 章介绍模型训练和评价的基本概念，并引入损失函数。损失函数将模型训练问题转化为函数优化问题。这一章以交叉熵为例讲解损失函数，并从信息论、贝叶斯和几何特性三个角度探究交叉熵的原理。
- 第 3 章介绍基于函数局部一阶信息的优化算法。这一章首先回顾多元函数微分的相关知识并引入梯度概念，接着讲解梯度下降法及其变体，最后介绍如何用梯度下降法训练逻辑回归模型。
- 第 4 章介绍基于函数局部二阶信息的优化算法。这一章首先回顾矩阵的相关知识，接着讲解多元函数的赫森矩阵，以牛顿法和共轭方向法为例介绍二阶优化算法，最后介绍如何用牛顿法训练逻辑回归模型。
- 第 5 章首先回顾概率论相关知识，之后从线性回归入手，介绍模型自由度、正则化和偏置-方差权衡等概念，最后介绍如何在逻辑回归的训练中应用正则化。
- 第 6 章讲解如何将线性模型连接成网络以克服其局限，并介绍多层全连接神经网络。
- 第 7 章讲解训练多层全连接神经网络的反向传播加梯度下降法。这一章是对之前各章节知识的一个综合，最后还谈到了训练深层网络所面临的一些困难。

- 第 8 章介绍计算图和自动求导。深度学习中的各种网络具有比多层全连接神经网络复杂得多的连接方式，而计算图和自动求导是搭建和训练复杂网络结构的有力工具。
- 第 9 章介绍卷积神经网络的原理、结构和训练。深度神经网络五花八门，卷积神经网络是其中最典型、最具代表性的一种。理解了卷积神经网络及其训练，也就能够理解其他各种深度神经网络。
- 第 10 章介绍了 5 种经典的卷积神经网络，介绍它们的结构和性能，最后简单讨论卷积神经网络结构的发展演化趋势。
- 第 11 章展示如何使用 TensorFlow 搭建和训练本书提到的几种主要模型，并将它们用于手写数字识别。
- 在附录中，我们尝试在卷积神经网络和元胞自动机之间建立联系，从动力学角度理解“深度”的含义。元胞自动机是一类计算模型，它们中的一些具备图灵完备性。元胞自动机的计算方式与卷积具有相似性，这一章从元胞自动机的动力学特性的视角看卷积神经网络，希望为读者提供一些有趣的洞见。

阅读方式

本书各章之间存在依赖关系，但对于不同背景和需求的读者，可以选择不同的阅读路径。

- 数学基础过硬的读者可以略过 1.2 节、3.1 节、4.1 节、5.1 节和 7.1 节，但温习一下总是有益的。
- 熟悉传统机器学习，想进一步了解神经网络和深度学习的读者，可以略过第一部分。
- 对于第 4 章，读者可以略过 4.1 节之后的内容，因为后续章节没有用到二阶优化算法。但 4.1 节回顾的矩阵知识在后续章节中还将用到，且非常关键。

本书各章节组成一个有机的整体，而并非孤立存在。本书路线图中的知识点前后呼应，能为读者提供一些有趣而深刻的洞见。所以，笔者强烈建议读者按顺序阅读全部内容。

代码与网络资源

读者可在 gitee.com/neural_network/neural_network_code 找到本书的样例代码，后续我们将继续维护和扩展这个代码库。笔者在知乎开设了专栏“计算主义” (zhuanlan.zhihu.com/pillgrim)，以后将继续在专栏中讨论和分享本书相关主题。鉴于笔者水平有限，错误难免，欢迎读者来信指正，邮箱是 zhangjuefei83@163.com。

致谢

感谢 360 智能工程部“Prophet 机器学习平台”团队的同事：组长陈震、黄斌、林灯博士、孙晓冬、马宗超、吕仁杰、刘韦菠、蔡春蒙、缪路文。孙晓冬为本书第 11 章编写了样例代码，感谢他杰出的工作。

感谢北京图灵文化发展有限公司的陈兴璐女士，她的邀约和鼓励促使本书得以面世，她出色的编辑工作使本书避免了很多问题和缺陷。

感谢中国科学院古脊椎动物与古人类研究所的潘雷博士和北京理工大学生物医学工程系的陈端端教授，她们在参考文献方面给予极大的帮助。

感谢中国科学院古脊椎动物与古人类研究所副研究员朱幼安博士和上海社会科学院哲学研究所钱立卿博士，感谢他们多年来与我在生命演化、混沌系统和计算理论方面的共同兴趣和探讨。

感谢北京自然博物馆标本部副研究员刘迪，蒙他惠允于本书中使用鸟类骨骼与生态类群数据集。

最后，感谢我的家人。

目 录

第一部分 线性模型

第 1 章 逻辑回归	2
1.1 作为一个神经元的逻辑回归	2
1.2 基础向量几何	4
1.2.1 向量	4
1.2.2 向量的和、数乘与零向量	6
1.2.3 向量的内积、模与投影	8
1.2.4 线性空间、基与线性函数	11
1.2.5 直线、超平面与仿射函数	14
1.3 从几何角度理解逻辑回归的能力 和局限	17
1.4 实例：根据鸟类骨骼判断生态类群	20
1.5 小结	24
第 2 章 模型评价与损失函数	25
2.1 训练集与测试集	25
2.2 分类模型的评价	26
2.2.1 混淆矩阵	26
2.2.2 正确率	27
2.2.3 查准率	27
2.2.4 查全率	27
2.2.5 ROC 曲线	28
2.3 损失函数	29
2.3.1 K-L 散度与交叉熵	29
2.3.2 最大似然估计	31
2.3.3 从几何角度理解交叉熵损失	33
2.4 小结	35

第 3 章 梯度下降法	36
3.1 多元函数的微分	36
3.1.1 梯度	37
3.1.2 方向导数	40
3.1.3 偏导数	43
3.1.4 驻点	43
3.1.5 局部极小点	44
3.2 梯度下降法	46
3.2.1 反梯度场	47
3.2.2 梯度下降法	49
3.2.3 梯度下降法的问题	50
3.3 梯度下降法的改进	52
3.3.1 学习率调度	52
3.3.2 冲量法	54
3.3.3 AdaGrad	55
3.3.4 RMSProp	56
3.3.5 Adam	57
3.4 运用梯度下降法训练逻辑回归	59
3.5 梯度下降法训练逻辑回归的 Python 实现	61
3.6 小结	67
第 4 章 超越梯度下降	68
4.1 矩阵	68
4.1.1 矩阵基础	68
4.1.2 矩阵的逆	71
4.1.3 特征值与特征向量	73
4.1.4 对称矩阵的谱分解	74

4.1.5 奇异值分解	76
4.1.6 二次型	77
4.2 多元函数的局部二阶特性	79
4.2.1 赫森矩阵	79
4.2.2 二阶泰勒展开	79
4.2.3 驻点的类型	82
4.2.4 赫森矩阵的条件数	84
4.3 基于二阶特性的优化	87
4.3.1 牛顿法	87
4.3.2 共轭方向法	92
4.4 运用牛顿法训练逻辑回归	95
4.5 牛顿法训练逻辑回归的 Python 实现	98
4.6 小结	100
第 5 章 正则化	102
5.1 概率论回顾	102
5.1.1 随机变量	102
5.1.2 多元随机变量	105
5.1.3 多元随机变量的期望和协方差 矩阵	106
5.1.4 样本均值和样本协方差矩阵	106
5.1.5 主成分	108
5.1.6 正态分布	111
5.2 模型自由度与偏置-方差权衡	115
5.2.1 最小二乘线性回归	116
5.2.2 模型自由度	118
5.2.3 偏置-方差权衡	119
5.3 正则化	122
5.3.1 岭回归与 L_2 正则化	122
5.3.2 L_2 正则化的贝叶斯视角	125
5.3.3 L_1 正则化	126
5.4 过拟合与欠拟合	127
5.5 运用 L_2 正则化训练逻辑回归	130
5.6 运用 L_2 正则化训练逻辑回归的 Python 实现	132
5.7 小结	135

第二部分 神经网络

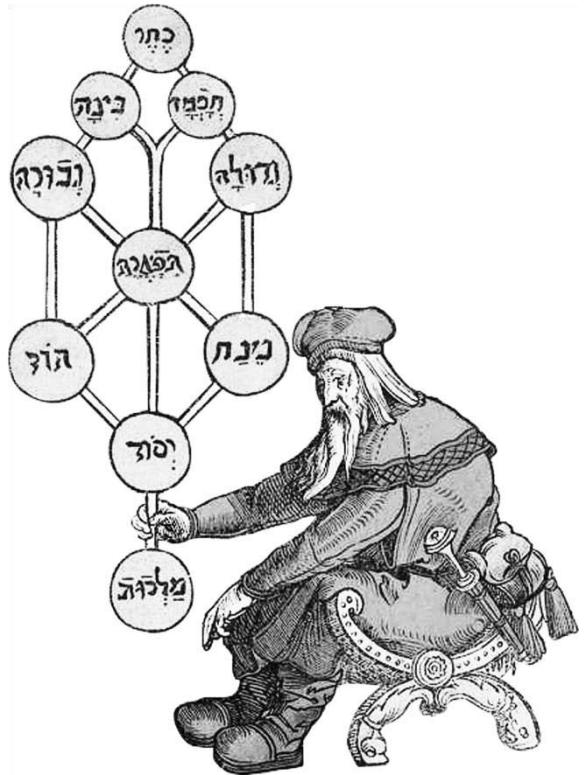
第 6 章 神经网络	138
6.1 合作的神经元	138
6.2 多层全连接神经网络	142
6.3 激活函数	145
6.3.1 Linear	145
6.3.2 Logistic	146
6.3.3 Tanh	148
6.3.4 ReLU	150
6.3.5 Leaky ReLU 以及 PReLU	151
6.3.6 SoftPlus	153
6.4 多分类与 SoftMax	154
6.5 小结	157
第 7 章 反向传播	158
7.1 映射	158
7.1.1 仿射映射	158
7.1.2 雅可比矩阵	159
7.1.3 链式法则	160
7.2 反向传播	162
7.2.1 网络的符号表示	162
7.2.2 原理	163
7.2.3 实现	166
7.3 相关问题	169
7.3.1 计算量	169
7.3.2 梯度消失	170
7.3.3 正则化	170
7.3.4 权值初始化	170
7.3.5 提前停止	171
7.4 多层全连接神经网络的 Python 实现	173
7.5 小结	181
第 8 章 计算图	183
8.1 计算图模型	183
8.1.1 简介	183
8.1.2 多层全连接神经网络的 计算图	187

8.1.3 其他神经网络结构的计算图	188	9.3.5 数据增强	239
8.2 自动求导	190	9.4 小结	239
8.3 自动求导的实现	192	第 10 章 经典 CNN	241
8.4 计算图的 Python 实现	195	10.1 LeNet-5	241
8.5 小结	214	10.2 AlexNet	245
第 9 章 卷积神经网络	215	10.3 VGGNet	248
9.1 卷积	215	10.4 GoogLeNet	251
9.1.1 一元函数的卷积	215	10.5 ResNet	255
9.1.2 多元函数的卷积	219	10.6 小结	257
9.1.3 滤波器	223	第 11 章 TensorFlow 实例	258
9.2 卷积神经网络的组件	228	11.1 多分类逻辑回归	258
9.2.1 卷积层	228	11.2 多层全连接神经网络	266
9.2.2 激活层	230	11.3 LeNet-5	269
9.2.3 池化层	231	11.4 AlexNet	273
9.2.4 全连接层	233	11.5 VGG16	277
9.2.5 跳跃连接	234	11.6 小结	280
9.3 深度学习的正则化方法	236	附录 A CNN 与元胞自动机	281
9.3.1 权值衰减	236	参考文献	311
9.3.2 Dropout	237		
9.3.3 权值初始化	237		
9.3.4 批标准化	238		

第一部分

线性模型

- 第 1 章 逻辑回归
- 第 2 章 模型评价与损失函数
- 第 3 章 梯度下降法
- 第 4 章 超越梯度下降
- 第 5 章 正则化



逻辑回归



统计学习领域的泰斗 Trevor Hastie 曾经说过：“每一个机器学习研究者都应该像熟悉自家后院那样熟悉线性模型。”线性模型是许多非线性模型的基础，它良好的数学结构能够为理解非线性模型提供深刻的洞见。人工神经网络的基本组成单元——神经元，正是线性模型。本章将介绍一种常用的线性模型——逻辑回归：首先引入逻辑回归的定义，之后回顾必要的线性代数知识，并从几何角度阐述逻辑回归及所有线性模型的能力局限。

1.1 作为一个神经元的逻辑回归

我们以逻辑回归（logistic regression）为例讨论线性模型。假如样本由 n 个数值型特征 x_1, x_2, \dots, x_n 组成，则逻辑回归的计算式是：

$$f(x_1, x_2, \dots, x_n) = \frac{1}{1+e^{-(b+w_1x_1+w_2x_2+\dots+w_nx_n)}} = \frac{1}{1+e^{-(b+\sum_{i=1}^n w_i x_i)}} \quad (1.1)$$

其中， e 是自然对数的底。式 (1.1) 将各个特征 x_i 乘以对应的权重系数 w_i 后相加，之后再加上 b ， b 称作偏置（bias）。这种计算称为关于 x_1, x_2, \dots, x_n 的仿射函数（affine function），仿射函数的值 a 如下：

$$a = b + \sum_{i=1}^n w_i x_i \quad (1.2)$$

接下来，逻辑回归对仿射函数的值 a 施加 Logistic 函数：

$$\text{Logistic}(a) = \frac{1}{1+e^{-a}} \quad (1.3)$$

Logistic 函数的图像是 S 形曲线（也称 sigmoid 曲线）。当 a 趋向于负无穷时，函数值趋向于 0；当 a 趋向于正无穷时，函数值趋向于 1。Logistic 的函数值在 $(0, 1)$ 内，当 $a = 0$ 时，函数值是 0.5，如图 1-1 所示。

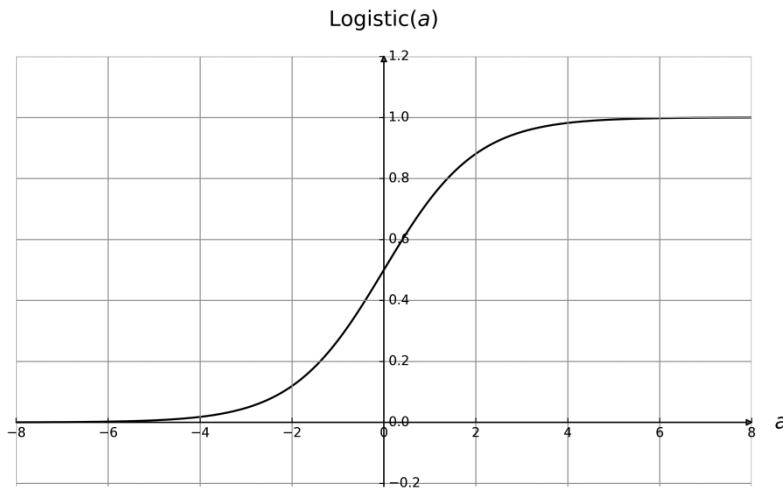


图 1-1 Logistic 函数的图像

在二分类问题中，用 n 个数值型特征表示一个样本，例如用体重、身高和年龄表示一个人。这些样本属于两个互斥的类别——正类 (positive) 或负类 (negative)。逻辑回归的输出值可视作样本属于正类的概率 p_p ：

$$p_p = \frac{1}{1+e^{-a}} = \frac{1}{1+e^{-(b+\sum_{i=1}^n w_i x_i)}} \quad (1.4)$$

因为样本必属于正类或负类之一，而且不能既属于正类又属于负类，所以样本属于正类的概率与属于负类的概率之和为 1，于是样本属于负类的概率 p_n 就是：

$$p_n = 1 - p_p = 1 - \frac{1}{1+e^{-a}} = \frac{e^{-a}}{1+e^{-a}} \quad (1.5)$$

p_p 与 p_n 之比的对数（除非特殊说明，本书中对数都以 e 为底）是：

$$\log \frac{p_p}{p_n} = \log \frac{1}{1+e^{-a}} \cdot \frac{1+e^{-a}}{e^{-a}} = \log e^a = a = b + \sum_{i=1}^n w_i x_i \quad (1.6)$$

可见，逻辑回归模型中两类别的对数概率比是关于特征的仿射函数。我们可以根据概率判定样本类别，如果规定当 $p_p \geq \frac{1}{2}$ ，即 $\frac{p_p}{p_n} \geq 1$ 时，将样本判定为正类，那么根据式 (1.6) 有：

$$a = \log \frac{p_p}{p_n} \geq \log 1 = 0 \quad (1.7)$$

也就是说，以 $\frac{1}{2}$ 为概率阈值时，预测类别取决于 a ： a 大于等于 0 时，预测为正类，否则预测为负类。 a 大于 0 有什么几何意义？逻辑回归为什么属于线性模型？答案将在下文揭晓。在那之前，我们先看看逻辑回归的一种图示，如图 1-2 所示。

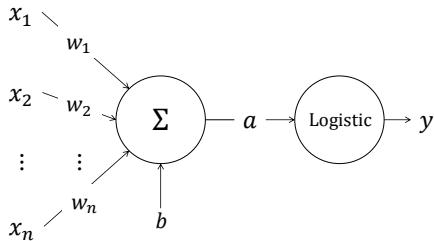


图 1-2 逻辑回归的神经元表示

图 1-2 表达的就是逻辑回归：将输入值加权求和再加偏置之后施加 Logistic 函数。在人工神经网络（本书后面会省略“人工”二字）语境下，这个结构就是一个神经元。这里的 Logistic 函数称为激活函数（activation function）。除了 Logistic 函数外，还有许多其他种类的激活函数，例如阶跃函数：

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (1.8)$$

以阶跃函数为激活函数的神经元是最早的神经元模型——感知机（perceptron）。感知机这个名称一直流传下来，以至于今天多层全连接神经网络还被称为多层感知机（multilayer perceptron, MLP）。关于激活函数的类型和性质，第 6 章还有阐述。无论取哪种激活函数，神经元的基本结构都是仿射函数加激活函数。

1.2 基础向量几何

神经元是神经网络乃至深度学习的基石，在深入考察神经元的能力和局限之前，有必要回顾一下基础的向量几何。本节概述向量知识。像这样介绍数学基础的小节会在书中多次出现，它们穿插在章节之中，紧密与主题结合，可以帮助读者回忆相关知识点，加深理解。虽是概览，但本书对数学内容的介绍会尽量做到穷根究底，不留黑盒，每一个结论都给出说明。

1.2.1 向量

一个包含 n 个数值型特征 x_1, x_2, \dots, x_n 的样本可用 n 维向量表示：

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = (x_1, x_2, \dots, x_n)^T \quad (1.9)$$

本书用黑斜体小写字母表示向量，例如 \mathbf{x} 和 \mathbf{y} 。斜体小写字母表示标量（实数），例如 a 、 b 或

x 。用脚标表示向量的分量，例如 x_2 表示向量 x 的第 2 个分量。用上标表示一组对象中的某一个，例如 x^3 表示一组向量中的第 3 个。样本有多少特征， x 就有多少分量。 n 是向量的分量个数，称为向量的维数。本书中向量专指列向量（其分量竖着排列）。有时为了节省空间，把列向量写成行向量的转置，如式 (1.9)。转置就是将行向量变成列向量，或将列向量变成行向量。

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}^T = (x_1, x_2, \dots, x_n), \quad (x_1, x_2, \dots, x_n)^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad (1.10)$$

为了方便查看，本节以 2 维或 3 维向量为例来介绍。现实问题中向量的维数会远远超过 3，但是本节的理论结果可以扩展到任意维数。

如果将各个分量看作坐标值，那么向量 x 可以表示坐标系上的一个点（point）。向量 x 也可以看作从原点指向这个点的一个有长度和方向的“箭头”。向量包含方向和长度这两个信息，点和箭头都是向量的几何表现形式，如图 1-3 所示。论述中可根据讨论视角的不同采用不同的表现形式。

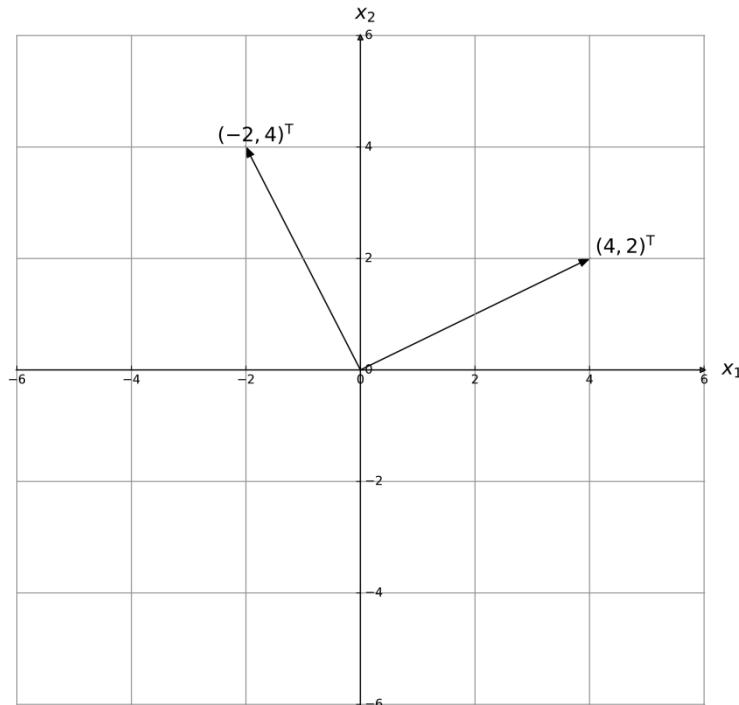


图 1-3 向量的几何表示

1.2.2 向量的和、数乘与零向量

向量可以求和。令 \mathbf{x} 和 \mathbf{y} 是两个维数相同的向量，它们的和 $\mathbf{x} + \mathbf{y}$ 是一个向量：

$$\mathbf{x} + \mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix} \quad (1.11)$$

$\mathbf{x} + \mathbf{y}$ 的各分量是 \mathbf{x} 和 \mathbf{y} 的对应分量之和。如果用“箭头”表示向量，那么 $\mathbf{x} + \mathbf{y}$ 是以 \mathbf{x} 和 \mathbf{y} 为邻边组成的平行四边形的对角线，从原点指向相对的顶点。该顶点也就是向量 $\mathbf{x} + \mathbf{y}$ 表示的点，如图 1-4 所示。

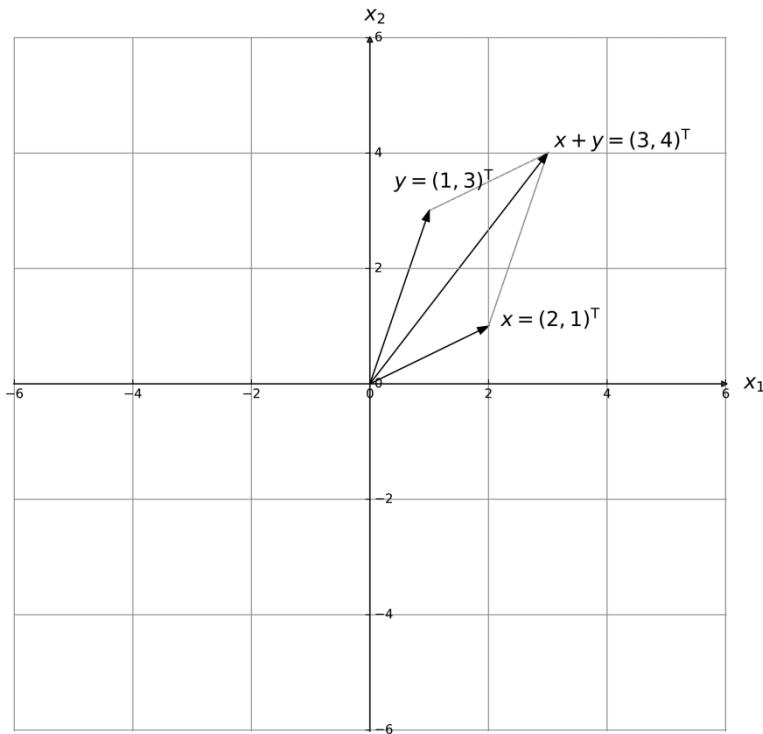


图 1-4 向量和的几何表示

用一个标量（实数） k 乘以一个向量，即向量的数乘。定义 $k\mathbf{x}$ 为：

$$k\mathbf{x} = \begin{pmatrix} kx_1 \\ kx_2 \\ \vdots \\ kx_n \end{pmatrix} \quad (1.12)$$

向量数乘的几何意义是对向量的长度进行缩放。用 2 乘以向量 \mathbf{x} 得到 $2\mathbf{x}$, $2\mathbf{x}$ 与 \mathbf{x} 方向相同, 长度是 \mathbf{x} 的 2 倍。向量的数乘如图 1-5 所示。

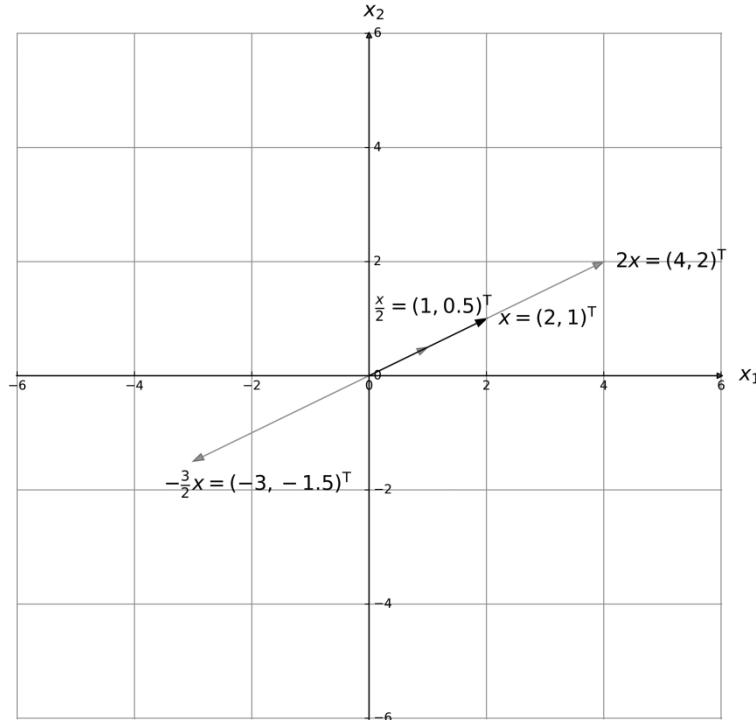


图 1-5 向量的数乘——缩放

用 0 乘以任何向量得到 “零向量”:

$$\mathbf{0} = 0 \times \mathbf{x} = \begin{pmatrix} 0x_1 \\ 0x_2 \\ \vdots \\ 0x_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad (1.13)$$

零向量是坐标系原点。我们很容易看出, 零向量与任何向量相加都得到该向量本身。如果用 -1 乘以向量 \mathbf{x} , 得到:

$$(-1) \times \mathbf{x} = -\mathbf{x} = \begin{pmatrix} -x_1 \\ -x_2 \\ \vdots \\ -x_n \end{pmatrix} \quad (1.14)$$

显然, $-\mathbf{x}$ 与 \mathbf{x} 相加得到零向量。用 $-\mathbf{x}$ 可以定义向量的减法:

$$\mathbf{y} - \mathbf{x} = \mathbf{y} + (-\mathbf{x}) \quad (1.15)$$

将箭头 $\mathbf{y} - \mathbf{x}$ 平移，使其尾部与 \mathbf{x} 重合，头部与 \mathbf{y} 重合，则 $\mathbf{y} - \mathbf{x}$ 构成以 \mathbf{x} 和 \mathbf{y} 为邻边的三角形的第三边，如图 1-6 所示。

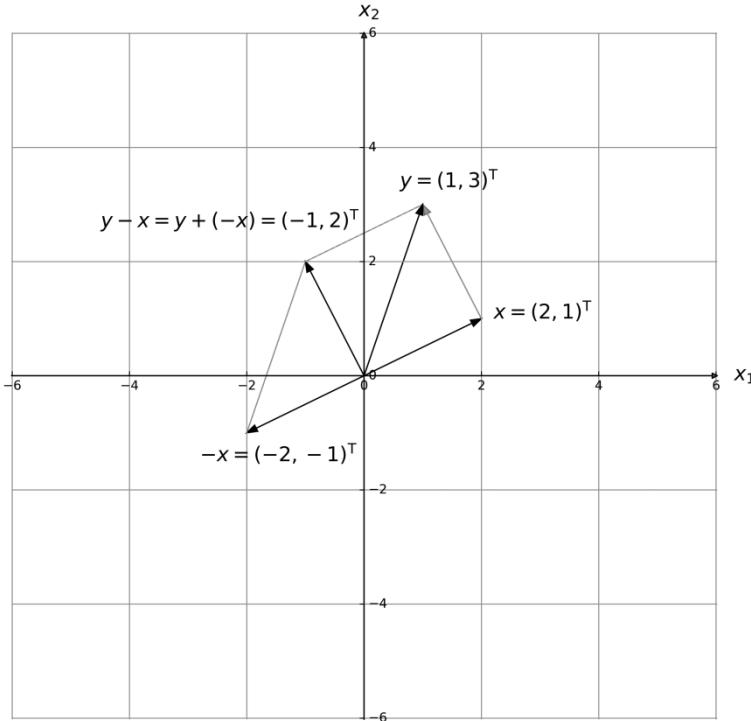


图 1-6 向量的差——三角形的第三边

向量有长度，2维向量的长度是：

$$\text{length}(\mathbf{x}) = \sqrt{x_1^2 + x_2^2} \quad (1.16)$$

这个长度是向量与原点之间的欧式距离。3维乃至更高维向量的长度也是它们与原点之间的欧式距离——各分量平方和的平方根。

1.2.3 向量的内积、模与投影

向量 \mathbf{x} 和 \mathbf{y} 的内积 (inner product) 定义为：

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i \quad (1.17)$$

\mathbf{x} 和 \mathbf{y} 的内积就是将 \mathbf{x} 和 \mathbf{y} 的对应分量相乘再相加。根据式 (1.17), \mathbf{x} 与自身的内积是:

$$\langle \mathbf{x}, \mathbf{x} \rangle = \sum_{i=1}^n x_i^2 \quad (1.18)$$

所以 \mathbf{x} 与自身的内积一定大于等于 0。只有当 \mathbf{x} 的所有分量都是 0, 即 \mathbf{x} 为零向量时, \mathbf{x} 与自身的内积才为 0。向量内积满足交换律:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle \quad (1.19)$$

向量内积对向量加法满足分配律:

$$\langle \mathbf{w}, \mathbf{x} + \mathbf{y} \rangle = \langle \mathbf{w}, \mathbf{x} \rangle + \langle \mathbf{w}, \mathbf{y} \rangle \quad (1.20)$$

向量内积对向量数乘满足:

$$\langle k\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, k\mathbf{x} \rangle = k\langle \mathbf{x}, \mathbf{y} \rangle \quad (1.21)$$

以上三条定律根据向量内积的定义很容易证明, 本书从略。向量 \mathbf{x} 的模 $\|\mathbf{x}\|$ 定义为 \mathbf{x} 与自身的内积的平方根:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \sqrt{\sum_{i=1}^n x_i^2} \quad (1.22)$$

可以看出, 向量的模就是它的长度。根据定义, $k\mathbf{x}$ 的模是:

$$\|k\mathbf{x}\| = \sqrt{\sum_{i=1}^n k^2 x_i^2} = \sqrt{k^2 \sum_{i=1}^n x_i^2} = |k| \|\mathbf{x}\| \quad (1.23)$$

根据式 (1.23), 用标量 $\frac{1}{\|\mathbf{x}\|}$ 乘以 \mathbf{x} , 得到的向量 $\frac{\mathbf{x}}{\|\mathbf{x}\|}$ 的长度是 1, 其方向与 \mathbf{x} 一致。也就是说, 用标量 $\frac{1}{\|\mathbf{x}\|}$ 乘以 \mathbf{x} 的效果是保持 \mathbf{x} 的方向不变, 将它的长度缩放到 1。长度为 1 的向量称为单位向量 (unit vector)。

对于 2 维或 3 维的情况, 上文提到, $\mathbf{y} - \mathbf{x}$ 经过平移, 得到以 \mathbf{x} 和 \mathbf{y} 为邻边的三角形的第三边, $\mathbf{y} - \mathbf{x}$ 的模就是第三边的长度。三角形第三边的长度可用余弦公式求得。假如三角形的边长分别是 a 、 b 和 c , a 和 b 之间的夹角是 θ , 则 c 是:

$$c^2 = a^2 + b^2 - 2ab \cos \theta \quad (1.24)$$

当 $\theta = \frac{\pi}{2}$ 时, $\cos \theta = 0$, 式 (1.24) 就是毕达哥拉斯定理 (勾股定理)。将三角形的边长替换为向量的模, 则有:

$$\|\mathbf{y} - \mathbf{x}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\|\mathbf{x}\|\|\mathbf{y}\| \cos \theta \quad (1.25)$$

根据模的定义将式(1.25)展开,消除等号两边相同项,得到:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \quad (1.26)$$

式(1.26)说明:向量 \mathbf{x} 和 \mathbf{y} 的内积等于它们的长度之积再乘以它们之间夹角的余弦。在2维或3维的情况下,向量之间的夹角有直观的定义,这时运用余弦定理得到式(1.26)。在更高维的情况下,向量的夹角反过来由式(1.26)定义。如果向量 \mathbf{x} 和 \mathbf{y} 的内积为0,则它们之间夹角的余弦是0,即夹角 θ 是 $\frac{\pi}{2}$,这时称 \mathbf{x} 与 \mathbf{y} 正交(orthogonal)。零向量与任意向量正交。若 \mathbf{x} 和 \mathbf{y} 都不是零向量,则它们的箭头互相垂直,如图1-7所示。

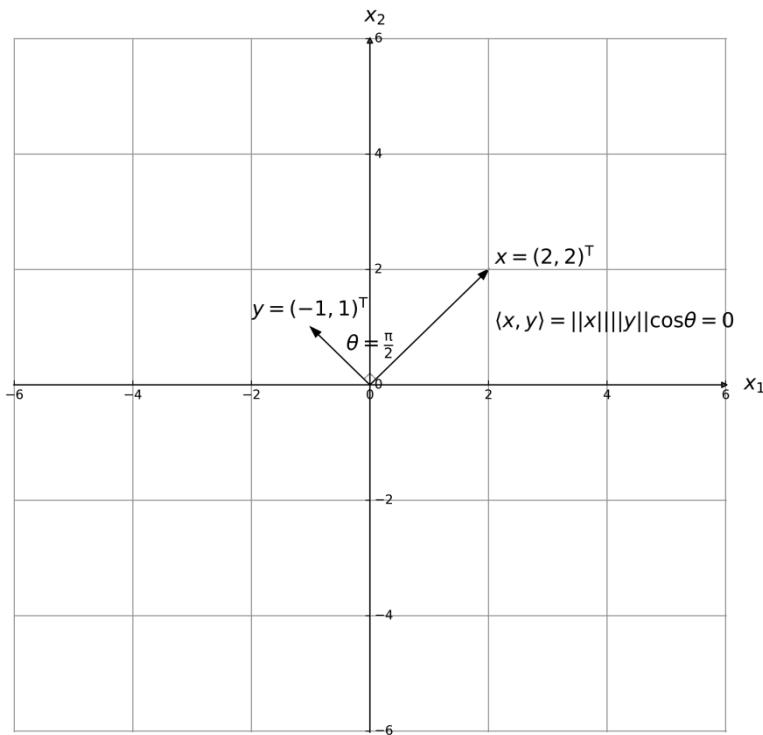


图1-7 向量正交

如果 \mathbf{x} 和 \mathbf{y} 之间的夹角为 θ ,那么 $\|\mathbf{x}\| \cos \theta$ 是 \mathbf{x} 向 \mathbf{y} 的投影的长度,如图1-8所示。投影长度等于 $\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{y}\|}$,如果 \mathbf{y} 是单位向量,则 \mathbf{x} 向 \mathbf{y} 的投影的长度就等于 $\langle \mathbf{x}, \mathbf{y} \rangle$ 。

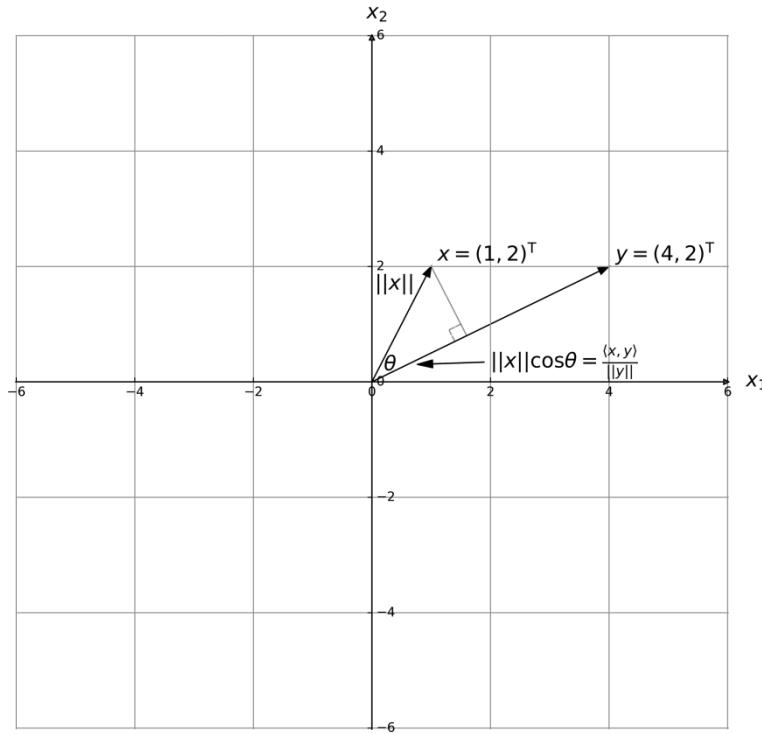


图 1-8 向量投影

如果把列向量看作 $n \times 1$ 矩阵，则向量的内积可以用矩阵乘积表示（关于矩阵知识的详细讲解见 4.1 节）：

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i = (x_1, x_2, \dots, x_n) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \mathbf{x}^T \mathbf{y} \quad (1.27)$$

多数时候，我们用 $\mathbf{x}^T \mathbf{y}$ 表示向量 \mathbf{x} 和 \mathbf{y} 的内积。

1.2.4 线性空间、基与线性函数

我们可以将任何一个 n 维向量 \mathbf{x} 分解：

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = x_1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + x_n \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \sum_{i=1}^n x_i \mathbf{e}^i \quad (1.28)$$

其中， \mathbf{e}^i 是 n 维向量，它的第*i*维分量是1，其余分量都是0。上述公式称 \mathbf{x} 可被向量组 $\mathbf{e}^i (i = 1, \dots, n)$ 线性表出。任意 n 维向量都可以被这个向量组线性表出，但任何一个 \mathbf{e}^i 都无法被该向量组中的其他向量线性表出。如果一组向量中的任何一个都不能被组内其他向量线性表出，则称这组向量是线性独立的（linear independent）。如果一组向量不是线性独立的，则称它们线性相关。向量组 $\mathbf{e}^i (i = 1, \dots, n)$ 是线性独立的。

线性独立有一个等价定义。对于一组向量 $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^n$ ，如果不存在一组不全为0的系数 w^1, w^2, \dots, w^n 满足

$$\sum_{i=1}^n w^i \mathbf{v}^i = \mathbf{0} \quad (1.29)$$

则 $\mathbf{v}^i (i = 1, \dots, n)$ 是线性独立的。我们来证明这个结论。假设 $\mathbf{v}^i (i = 1, \dots, n)$ 是线性独立的，但是存在一组不全为0的系数满足式(1.29)。如果 $w^j \neq 0$ ，则 \mathbf{v}^j 就可以被其他向量线性表出，即 $\mathbf{v}^j = \sum_{i \neq j} \frac{w^i}{w^j} \mathbf{v}^i$ ，这就会产生矛盾。

如果不存在不全为0的系数满足式(1.29)，但是 $\mathbf{v}^i (i = 1, \dots, n)$ 线性相关，那么其中某个向量 \mathbf{v}^j 可以被其他向量线性表出，即 $\mathbf{v}^j = \sum_{i \neq j} w^i \mathbf{v}^i$ 。于是有 $-\mathbf{v}^j + \sum_{i \neq j} w^i \mathbf{v}^i = \mathbf{0}$ ，也就是存在不全为0的系数满足式(1.29)，这也会产生矛盾，所以两种线性独立的定义是等价的。

如果一组向量 $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^s$ 可以被另一组向量 $\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^r$ 线性表出，并且 $r < s$ ，那么向量组 $\mathbf{v}^i (i = 1, \dots, s)$ 是线性相关的。我们来证明这个结论，每一个向量 \mathbf{v}^i 都可以被 $\mathbf{u}^j (j = 1, \dots, r)$ 线性表出，即 $\mathbf{v}^i = \sum_{j=1}^r a^{i,j} \mathbf{u}^j$ 。如果有一组系数 b^1, b^2, \dots, b^s 满足

$$\sum_{i=1}^s b^i \mathbf{v}^i = \sum_{i=1}^s b^i \sum_{j=1}^r a^{i,j} \mathbf{u}^j = \sum_{j=1}^r (\sum_{i=1}^s a^{i,j} b^i) \mathbf{u}^j = \mathbf{0} \quad (1.30)$$

若要式(1.30)成立，只须对所有 $j = 1, \dots, r$ ，有 $\sum_{i=1}^s a^{i,j} b^i = 0$ 即可，这是一个方程组：

$$\begin{cases} a^{1,1}b^1 + a^{2,1}b^2 + \dots + a^{s,1}b^s = 0 \\ a^{1,2}b^1 + a^{2,2}b^2 + \dots + a^{s,2}b^s = 0 \\ \vdots \\ a^{1,r}b^1 + a^{2,r}b^2 + \dots + a^{s,r}b^s = 0 \end{cases} \quad (1.31)$$

这个方程组有 s 个自变量、 r 个方程。因为 $r < s$ ，方程数小于自变量数，所以这个方程组存在非全零解，即 $\mathbf{v}^i (i = 1, \dots, s)$ 线性相关。

如果一个向量集合满足“对于该集合中任意两个向量 \mathbf{x} 和 \mathbf{y} ，以及任意实数 a 和 b ，向量 $a\mathbf{x} + b\mathbf{y}$ 仍属于该集合”，则该集合是一个线性空间。全体 n 维向量的集合是一个线性空间，记为 \mathbb{R}^n 。如果一个线性空间中的所有向量都可以被 k 个线性独立的向量线性表出，则称这组向量为该线性空间的基（basis）。这个线性空间中任何多于 k 个向量的向量组必线性相关，因为它们可以被更少的

向量线性表出。任何少于 k 个向量的向量组必不能线性表出这个线性空间，否则那组 k 个向量的基就可以被更少的向量线性表出，这显然不可能。也就是说，凡是能线性表出该线性空间，且线性独立的向量组都只能有 k 个向量，不多不少。 k 称为该线性空间的维数。 \mathbf{e}^i ($i = 1, \dots, n$) 是 \mathbb{R}^n 的一组基，称为标准基。 \mathbb{R}^n 的维数是 n 。

两两正交的非零向量 $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^k$ 一定是线性独立的，因为假如存在一组系数 w^1, w^2, \dots, w^k 满足 $\sum_{i=1}^k w^i \mathbf{v}^i = \mathbf{0}$ ，那么对于任何一个 j ，有：

$$(\mathbf{v}^j)^T \sum_{i=1}^k w^i \mathbf{v}^i = \sum_{i=1}^k w^i (\mathbf{v}^j)^T \mathbf{v}^i = w^j (\mathbf{v}^j)^T \mathbf{v}^j = 0 \quad (1.32)$$

因为 \mathbf{v}^j 是非零向量，它与自己的内积不为 0，所以必有 $w^j = 0$ 。这对所有 j 都成立，即 w^1, w^2, \dots, w^k 都只能是 0，所以 $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^k$ 是线性独立的。

现在介绍线性函数 (linear function)。如果函数 $f(\mathbf{x})$ 是线性的，则对于任意两个向量 \mathbf{x} 和 \mathbf{y} 与任意两个实数 a 和 b ，公式 (1.33) 成立：

$$f(a\mathbf{x} + b\mathbf{y}) = af(\mathbf{x}) + bf(\mathbf{y}) \quad (1.33)$$

如果 $f(\mathbf{x})$ 是线性函数，则必然存在一个向量 \mathbf{w} 满足

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \quad (1.34)$$

也就是说，线性函数一定等于某个向量 \mathbf{w} 与输入向量 \mathbf{x} 的内积。这样构造向量 \mathbf{w} ：

$$\mathbf{w} = \begin{pmatrix} f(\mathbf{e}^1) \\ f(\mathbf{e}^2) \\ \vdots \\ f(\mathbf{e}^n) \end{pmatrix} \quad (1.35)$$

其中， \mathbf{w} 的第 i 维分量是对 \mathbf{e}^i 施加 $f(\mathbf{x})$ 后得到的值。由于 $f(\mathbf{x})$ 是线性函数，有：

$$f(\mathbf{x}) = f(\sum_{i=1}^n x_i \mathbf{e}^i) = \sum_{i=1}^n x_i f(\mathbf{e}^i) = (f(\mathbf{e}^1), f(\mathbf{e}^2), \dots, f(\mathbf{e}^n)) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \mathbf{w}^T \mathbf{x} \quad (1.36)$$

这就证明了式 (1.34) 的结论。线性函数的图像一定过原点，因为：

$$f(\mathbf{0}) = f(0 \times \mathbf{x}) = 0 \times f(\mathbf{x}) = 0 \quad (1.37)$$

1.2.5 直线、超平面与仿射函数

对于向量 $\mathbf{w} \neq \mathbf{0}$, 令 \mathbf{x} 是任意与 \mathbf{w} 的内积为常数 C 的向量:

$$\mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta = C \quad (1.38)$$

式 (1.38) 表明 $\|\mathbf{x}\| \cos \theta = \frac{C}{\|\mathbf{w}\|}$, 即 \mathbf{x} 向 \mathbf{w} 的投影的长度是常数 $\frac{C}{\|\mathbf{w}\|}$ 。在 2 维的情况下, 将 \mathbf{x} 看作平面上的一个点, 则所有满足式 (1.38) 的点都位于垂直于 \mathbf{w} 的一条直线上, 如图 1-9 所示。

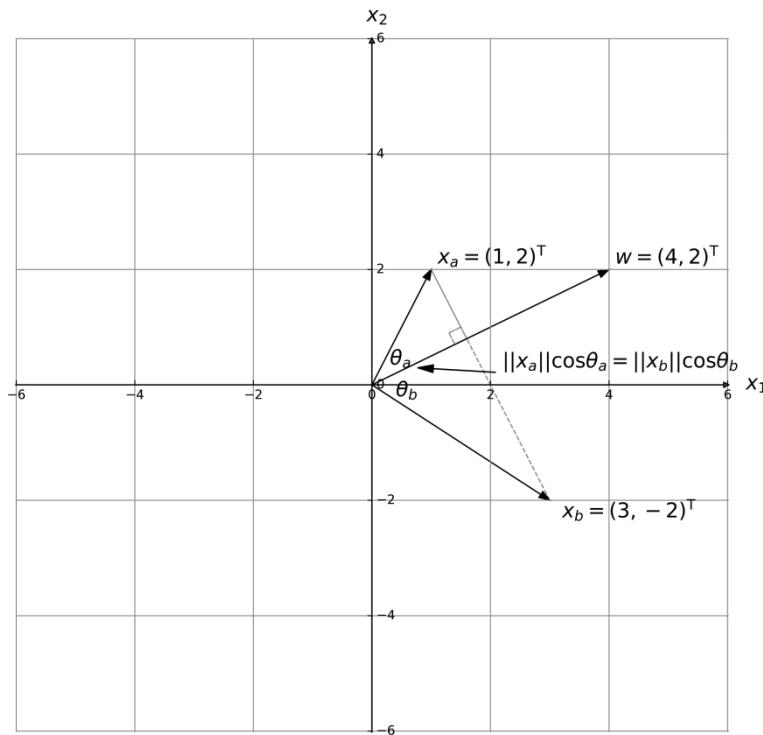


图 1-9 2 维空间中与非零向量内积相同的点构成垂直于该向量的直线

在 3 维空间中, 所有与向量 $\mathbf{w} \neq \mathbf{0}$ 的内积相同的点, 构成垂直于 \mathbf{w} 的平面, 如图 1-10 所示。在更高维空间中, 这样的点构成垂直于 \mathbf{w} 的超平面 (hyperplane)。

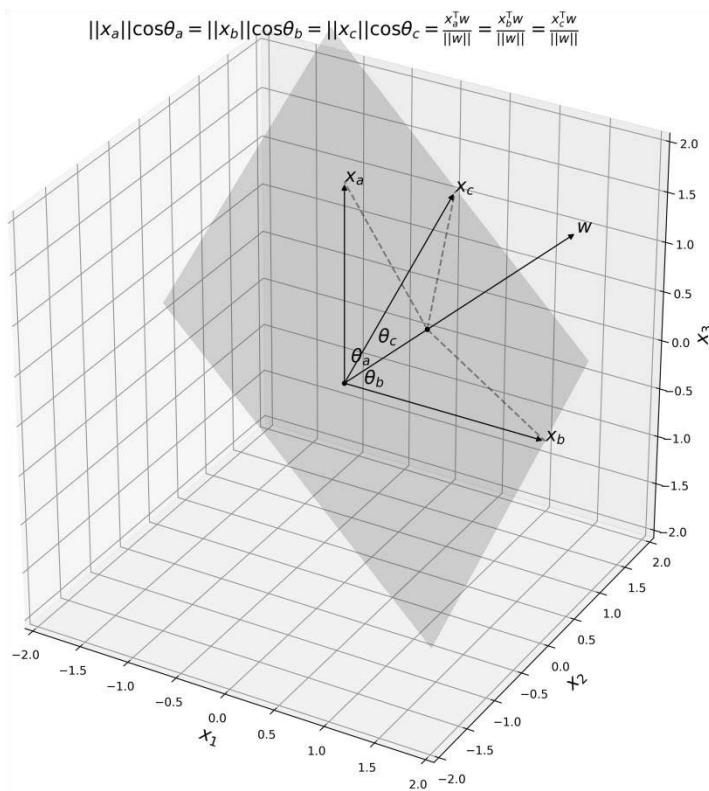


图 1-10 3 维空间中与非零向量内积相同的点构成垂直于该向量的平面

直线是 2 维空间中的超平面，平面是 3 维空间中的超平面。我们将直线、平面和更高维的超平面统称为超平面。 \mathbf{w} 称为超平面的法向量 (norm)。对于任意非零实数 k ，向量 $k\mathbf{w}$ 也是这个超平面的法向量，因为该超平面上所有向量与 $k\mathbf{w}$ 的内积也相同：

$$(k\mathbf{w})^T \mathbf{x} = \langle k\mathbf{w}, \mathbf{x} \rangle = |k| \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta = |k| C \quad (1.39)$$

前文曾提到的仿射函数 y ，重新写在这里：

$$y = b + \sum_{i=1}^n w_i x_i \quad (1.40)$$

如果自变量是 2 维，将 y 移到等号另一侧，可得到：

$$w_1 x_1 + w_2 x_2 - y = (w_1, w_2, -1) \begin{pmatrix} x_1 \\ x_2 \\ y \end{pmatrix} = -b \quad (1.41)$$

根据式 (1.41)，在 $x_1 x_2 y$ 三维空间中，所有满足式 (1.40) 的点与向量 $\mathbf{w} = (w_1, w_2, -1)^T$ 的

内积为常数 $-b$ 。也就是说，仿射函数的图像是3维空间中的一张平面。该平面的法向量是 \mathbf{w} 。当 $x_1 = x_2 = 0$ 时， y 的值是 b ， b 称为该平面的截距。由式(1.40)可以看出，仿射函数是线性函数加上一个常量 b 。

在3维情况下，若 w_1 和 w_2 的绝对值较大，则 \mathbf{w} 更贴近 x_1x_2 平面，以 \mathbf{w} 为法向量的平面接近竖立；若 w_1 和 w_2 的绝对值较小，则 \mathbf{w} 更贴近 y 轴，以 \mathbf{w} 为法向量的平面接近水平。平面的倾斜程度与 w_1 和 w_2 的绝对值大小有关，也就是与向量 $(w_1, w_2)^T$ 的模有关。 $(w_1, w_2, 0)^T$ 是法向量 \mathbf{w} 在 x_1x_2 平面上的投影，它的方向决定了平面的朝向。图1-11集中展示了这几个概念。

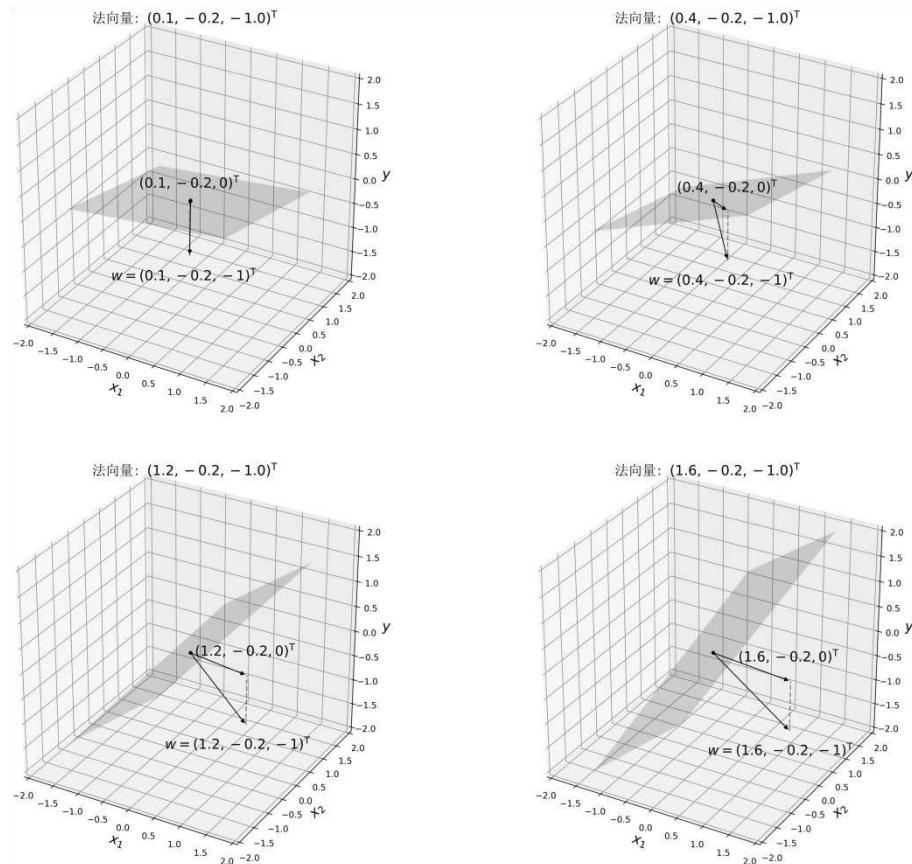


图1-11 法向量决定平面的朝向和倾斜程度

仿射函数是一类最简单的函数，它的图像在自变量为1维的情况下是直线，在自变量为2维的情况下是平面，在更高维情况下是高维超平面。超平面在任意位置的性质都相同，例如2维平面任意位置的朝向和倾斜程度都相同。

1.3 从几何角度理解逻辑回归的能力和局限

1

我们回忆一下逻辑回归的表达式：

$$f(\mathbf{x}) = \frac{1}{1+e^{-(b+\sum_{i=1}^n w_i x_i)}} = \frac{1}{1+e^{-(b+\mathbf{w}^T \mathbf{x})}} \quad (1.42)$$

\mathbf{w} 是逻辑回归的权值向量， b 是偏置。下面还是以2维为例，与权值向量 $\mathbf{w} = (w_1, w_2)$ 垂直的直线上的向量和 \mathbf{w} 的内积都相同，所以对它们施加仿射函数 $b + \mathbf{w}^T \mathbf{x}$ 的结果都相同，再施加Logistic函数的结果也都相同。逻辑回归先对输入向量施加仿射函数，这样就丢失了垂直于 \mathbf{w} 的方向上的信息。所以逻辑回归的输出只在 \mathbf{w} 的方向上有差异，在垂直于 \mathbf{w} 的方向上无差异。如果根据逻辑回归的输出是否大于阈值 t 来判断样本的类别：

$$\text{output}(\mathbf{x}) = \begin{cases} 0, & f(\mathbf{x}) < t \\ 1, & f(\mathbf{x}) \geq t \end{cases} \quad (1.43)$$

则只能得到垂直于 \mathbf{w} 的超平面分界面，这就是逻辑回归属于线性模型的原因。仿射函数是“罪魁祸首”，它选择了一个方向，忽略了其他方向上的信息，如图 1-12 所示。垂直于 \mathbf{w} 的那条虚线上的向量与 \mathbf{w} 的内积都相同，于是它们有相同的逻辑回归值。

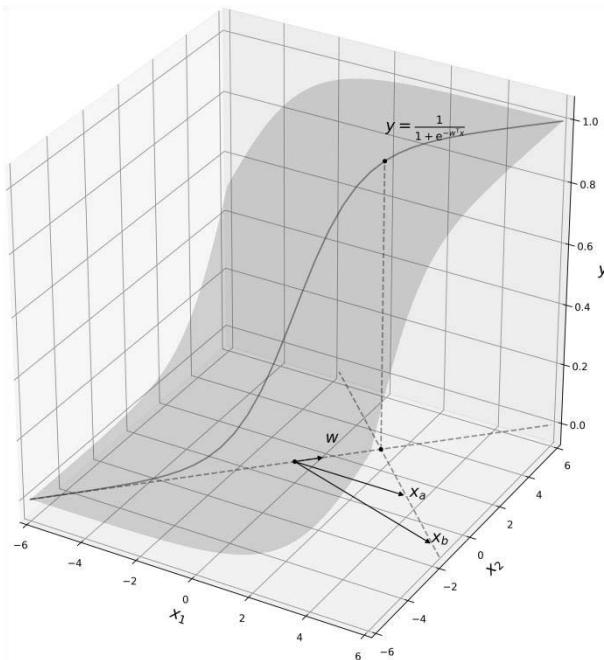


图 1-12 逻辑回归只能形成超平面分界面

神经元可以选用其他激活函数，但只要激活函数是单调的，就只能形成超平面分界面。采用单调激活函数的神经元无法处理异或(XOR)问题，因为异或问题是线性不可分的——无法用直线将正负样本点分隔开，如图 1-13 所示。

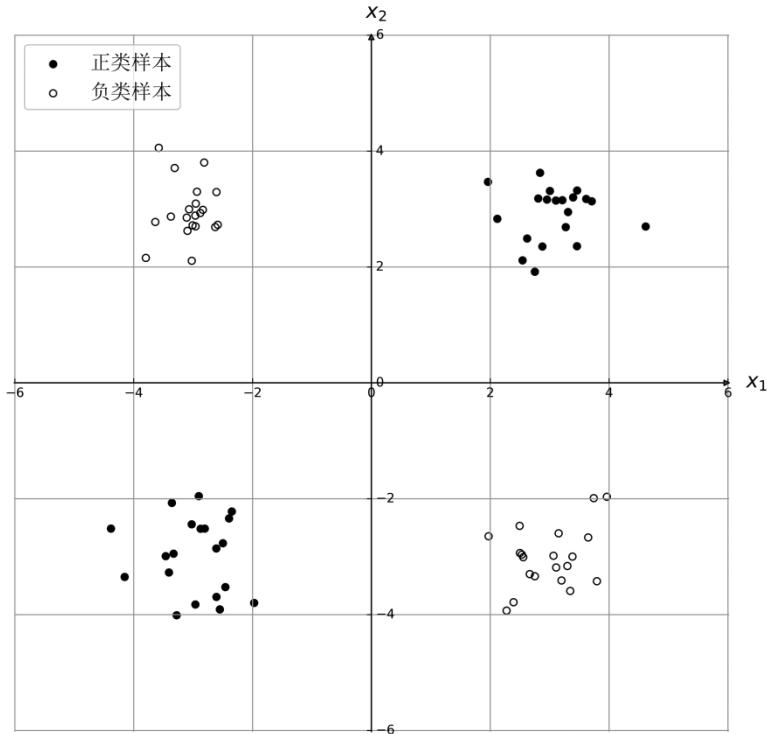


图 1-13 异或问题

解决异或问题，可以采用非单调激活函数，比如平方函数：

$$f(\mathbf{x}) = 0.02 \times (b + \mathbf{w}^T \mathbf{x})^2 \quad (1.44)$$

$f(\mathbf{x})$ 沿 \mathbf{w} 的方向两头翘起，中间凹下。若 \mathbf{w} 取合适的方向并选择恰当的阈值，平方激活函数就可以解决异或问题，如图 1-14 所示。

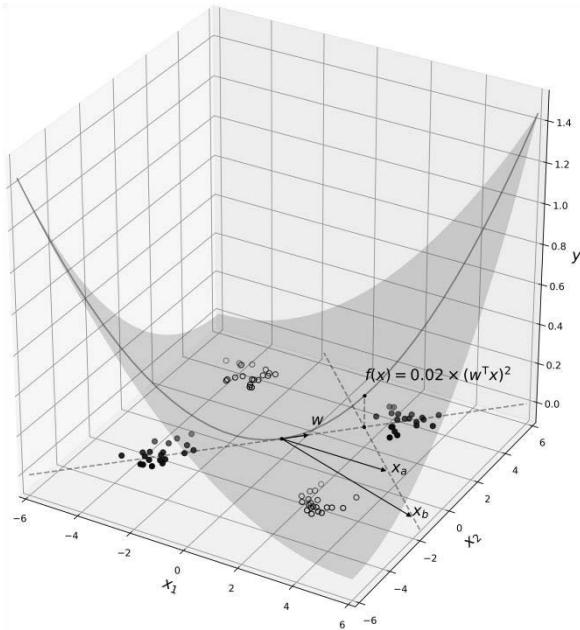


图 1-14 使用平方激活函数解决异或问题（偏置为 0）

但是式 (1.44) 的能力也仅限于用两条平行直线划分平面，就算使用正弦 \sin 函数作激活函数，也只能形成无数条平行分界线。总之，垂直于权值向量方向的信息已经丢失了，神经元对于同心圆状分布的数据是无能为力的，如图 1-15 所示。

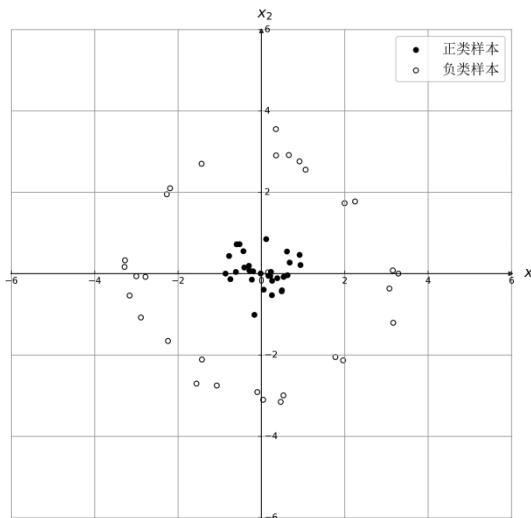


图 1-15 同心圆数据

同心圆数据的正类样本分布在中央，负类样本围绕在外围。沿着任意方向，都无法将两类样本分开。所以无论采用哪种激活函数，都无法分类同心圆数据。要想获得超越线性的分界能力，必须将多个神经元连接成网络，允许它们合作形成复杂分界面，这就是神经网络的思想。

1.4 实例：根据鸟类骨骼判断生态类群

本节中，我们将逻辑回归运用于一个实际问题：根据鸟类的前后肢骨骼测量指标来判断它所属的生态类群。全世界现存的鸟类有 9000 余种，可分为 8 个生态类群——游禽、涉禽、陆禽、猛禽、攀禽、鸣禽、走禽以及海洋性鸟类，不同生态类群的鸟类具有不同的生活习性。自然选择使鸟类的身体形态适应其生活环境。例如，涉禽有长长的脖颈和腿，这利于它们涉水捕鱼；猛禽有强壮的翅膀和有力的爪，这利于它们捕猎。

我们测量了 400 余个鸟类个体的前后肢骨骼，测量指标分别是：肱骨长度（huml）、肱骨宽度（humw）、尺骨长度（ulnal）、尺骨宽度（ulnaw）、股骨长度（feml）、股骨宽度（femw）、胫骨长度（tibl）、胫骨宽度（tibw）、跗跖骨长度（tarl）以及跗跖骨宽度（tarw）。这 10 个指标都是以毫米为单位的实数，精确到小数点后两位。括号中的内容是指标的名称，也就是特征名。最后一个字母 l 代表长度，w 代表宽度，之前是该骨骼拉丁文名称的前 3~4 个字母，具体见图 1-16。这 10 个特征能够反映鸟类翅膀和腿的绝对/相对长度和强壮程度。

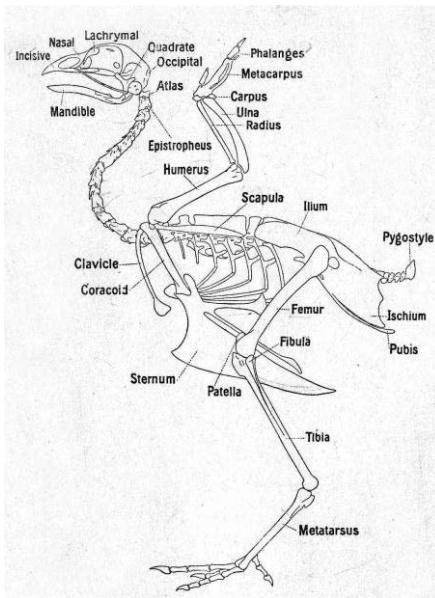


图 1-16 鸟类骨骼示意图

数据集的样本覆盖 8 个生态类群中的 6 个，它们分别是游禽 (SW)、涉禽 (W)、陆禽 (T)、猛禽 (R)、攀禽 (P) 以及鸣禽 (SO)，其中括号中的内容是我们为生态类群取的代号。数据集包含分属于 6 个类别的 413 个样本，每个样本包含 10 个数值特征和一个类别标签。随机选取一些例子，如表 1-1 所示。

表 1-1 数据集样例

huml	humw	ulnal	ulnaw	feml	femw	tibl	tibw	tarl	tarw	type
20.19	1.85	26.41	1.53	15.71	1.29	29.66	1.15	21.9	1.05	SO
85	5.07	93.17	4.21	38.33	2.57	67.32	2.73	40	2.53	W
107.41	5.69	110.5	5.32	40.62	3.15	81.86	3.71	44.62	2.95	SW
22.26	1.77	24.58	1.67	21.28	1.68	36.06	1.43	24.14	1.41	SO
48.19	3.44	43.71	3.01	50.46	3.6	94.86	3.45	55	4.57	W
100.38	5.52	113.24	5.04	44.72	4.07	82.87	3.69	54.8	3.25	W
23.27	2.15	27.62	2.16	21.86	1.64	35.55	1.51	23.23	1.32	SO
310	14.4	315	9.51	88.77	8.1	180	9.45	96.13	7.69	SW
29.26	2.66	27.63	2.23	27.87	2.03	38.9	1.81	25.22	1.62	P
186	9.83	152	8.76	56.02	7.02	185	8.07	90.8	4.59	SW

这 6 个类别的样本数量不均衡，具体如图 1-17 所示。

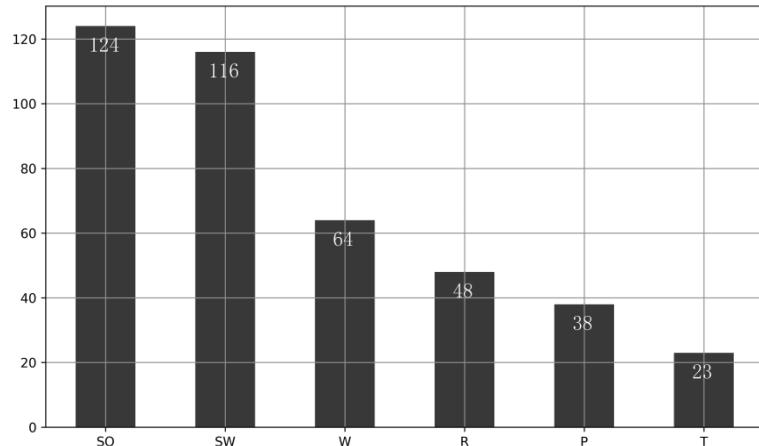


图 1-17 6 个类别（生态类群）的样本数量

第 6 章会介绍多分类逻辑回归，但现在我们介绍的逻辑回归只处理二分类问题，所以我们将 6 个类别归并为 2 个类别。首先，我们对数据做一些简单的分析。我们将 10 个特征每两个作为一个对，画出二维散点图矩阵，如图 1-18 所示。

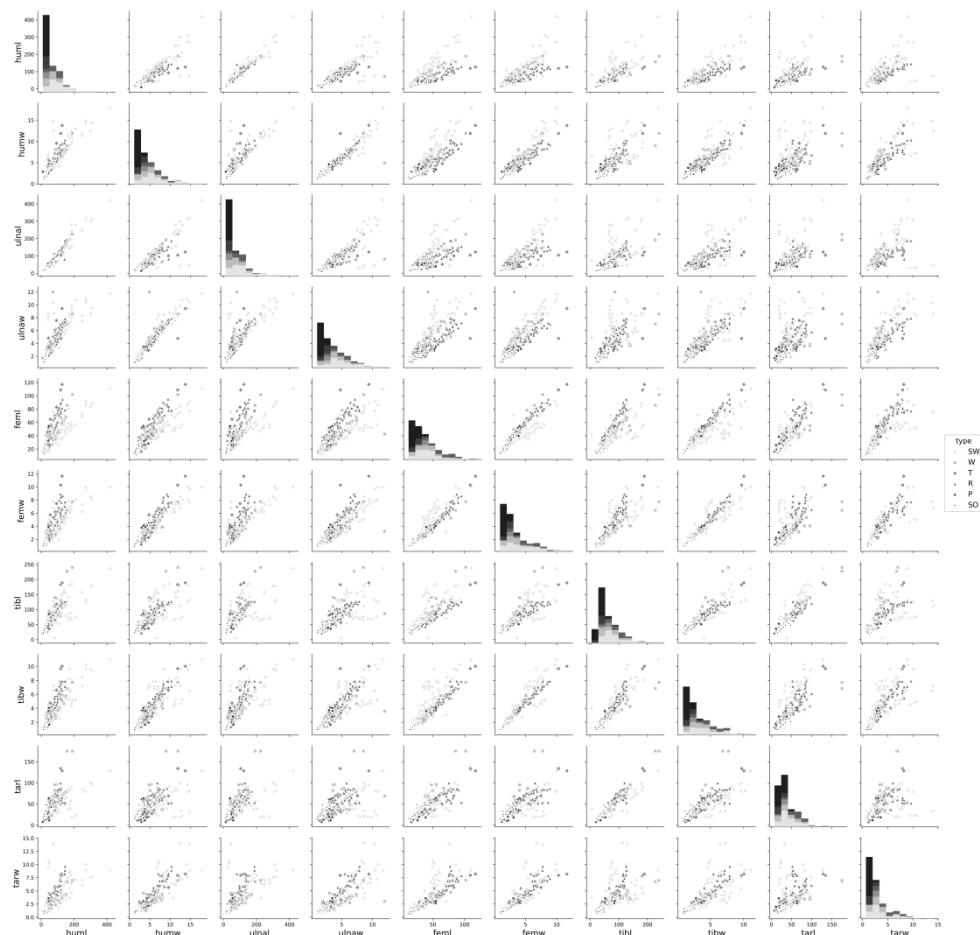


图 1-18 特征成对散点图

图 1-18 是一个 10×10 的阵列，每一个子图是行和列对应的两个特征的散点图，不同类别的样本用不同的标记画出。在对角线上，行和列对应同一个特征，散点图没有意义，所以对角线上的子图是该特征的分布直方图。各个类别有各自的直方图，以不同灰度显示。可以看出，特征与特征之间有较强的相关性，因为无论鸟的具体形态如何，大体型的鸟的所有骨骼都较长，小体型的鸟的所有骨骼都较短。

我们再用箱状图展示每个特征在每个类别上的分布，如图 1-19 所示。对于每个类别的每个特征，箱状图的“箱”中的三条横线分别对应特征的 25%、50% 和 75% 分位数，记为 Q_1 、 Q_2 和 Q_3 。令 $\Delta Q = Q_3 - Q_1$ ，向上向下延伸的“须”分别达到 $Q_3 + \Delta Q$ 和 $Q_1 - \Delta Q$ 。在上下“须”之外的点，可以认为是异常点。箱状图大致勾画了特征的分布。

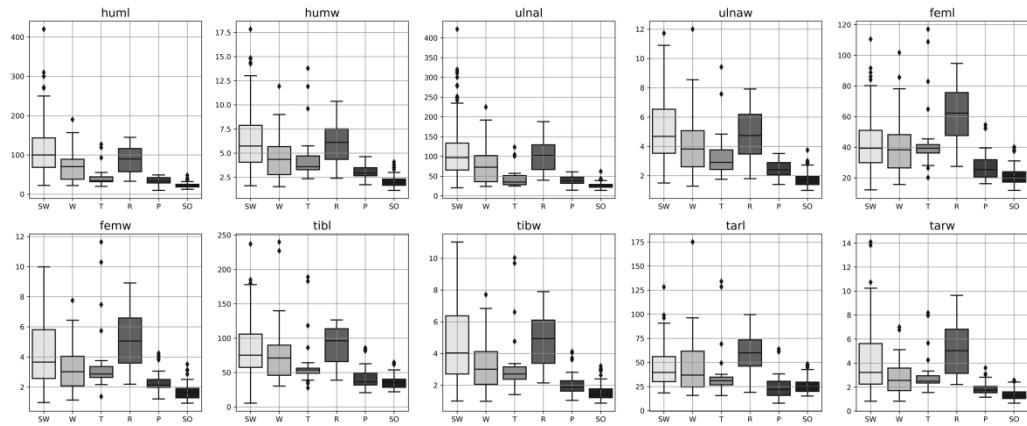


图 1-19 特征箱状图

从图 1-19 中可以看出，游禽（SW）、涉禽（W）和猛禽（R）这三个类别的各个测量值都较大，可以认为这三个类别是大型鸟，例如醍醐、火烈鸟和秃鹫。而陆禽（T）、攀禽（P）和鸣禽（SO）这三个类别是小型鸟，例如鸽子、鹦鹉和家燕。我们将 6 个类别归并为大型鸟和小型鸟这两个类别，以大型鸟为正类，这就构造了一个二分类问题。以不同标记表示大型鸟和小型鸟两个类别，5 个长度特征（名称以 1 为后缀的特征）的成对散点图如图 1-20 所示。

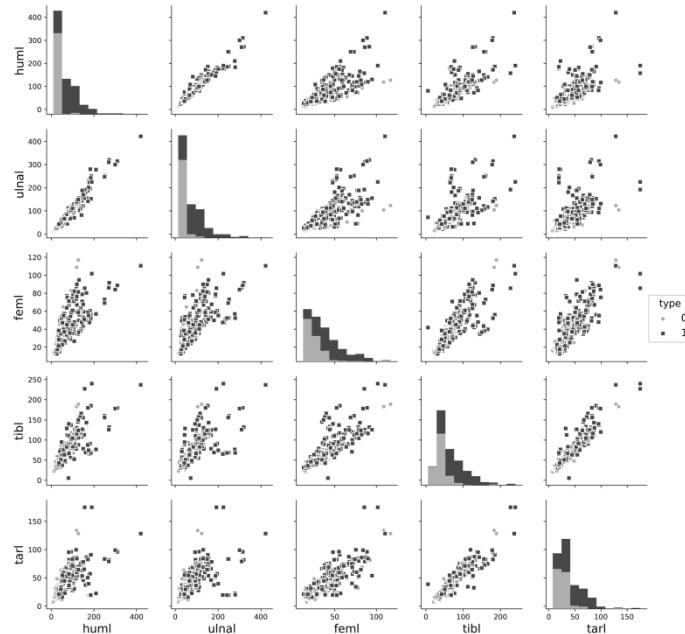


图 1-20 5 个长度特征区分两个类别的成对散点图

至此，我们构造了一个 413 个样本、10 个数值特征的二分类问题。对于这个问题，逻辑回归需要 10 个权值和 1 个偏置。在后续的章节中，我们将用原生 Python 实现逻辑回归和神经网络来对这个鸟类骨骼与生态类群问题进行建模。

1.5 小结

逻辑回归模型是一种线性模型，在神经网络的语境下，它是一个神经元。逻辑回归模型的分类能力的源泉在于它的权值向量。权值向量在空间中指示了一个方向，仿射函数值提取出数据沿该方向的差异。递增的激活函数随仿射函数值而变化，根据权值向量方向上的差异对不同样本产生不同输出，它可解释为概率。确定了概率阈值后，逻辑回归模型的分界面就是一个以权值向量为法向量的超平面。分界面将空间分成两个区域，不同区域中的样本被预测为不同的类别。

逻辑回归乃至线性模型的局限也就在于此：它们只能提取一个方向上的信息，垂直于权值向量方向上的信息丢失了。如果数据在所有方向上都有差异，就像同心圆数据那样，那么线性模型必然无法把握全部信息。后文我们会看到，只有将神经元连接成网络，才能克服这个局限。但在那之前，我们在下一章中先介绍如何根据目标调整逻辑回归模型的权值向量和偏置，即模型的训练问题。

第3章

梯度下降法

3

训练就是寻找使损失函数最小的模型参数，于是模型训练问题就成了函数优化问题。函数优化是应用数学的一个分支，是一个丰富的理论武器库。机器学习与函数优化联系紧密，特别是基于函数局部特性的迭代优化算法应用最为普遍。梯度下降法就是一种基于函数局部一阶特性的优化算法，它在神经网络和深度学习领域占统治地位。

要深入理解函数优化，必须具备多元微分的背景知识。本章首先回顾多元函数的梯度、方向导数、偏导数等概念。梯度包含函数在自变量空间某一点附近的一阶近似信息，或称线性近似。理解了梯度概念后，我们介绍梯度下降法。梯度下降法利用梯度确定函数值下降最快的方向，并沿该方向前进一段距离。算法迭代执行此步骤，使函数值不断下降，期待能够找到函数的全局最小点。

一阶近似是对原函数的最粗糙的近似。由于梯度下降法只利用了线性近似信息，所以它是短视的。本章阐述梯度下降法的一些问题，之后介绍几种对原始梯度下降法的改进。最后本章展示如何运用梯度下降法训练逻辑回归模型。通过本章，读者能够透彻理解梯度下降法的原理和局限。

3.1 多元函数的微分

微分刻画函数的局部线性近似，即用仿射函数近似原函数。仿射函数的图像是超平面，所以线性近似就是在局部用超平面近似原函数的图像。并不是所有函数在任意位置都能被仿射函数近似。若要这种近似能够成立，原函数必须满足一定条件，就是在该位置可微。前文介绍过，超平面的朝向和倾斜程度蕴含在它的法向量中。函数的局部近似超平面的法向量与函数在该位置的梯度有关。

多元函数的自变量在某点可以沿无数方向运动，函数值在每个方向上都有不同的变化率。多元函数沿某一方向的变化率称为方向导数。各个方向的方向导数也都蕴含在梯度之中。若要寻找函数值下降最快的方向，就要用到梯度。本节深入讲解这些概念。

3.1.1 梯度

首先回忆一下一元函数 $f(x)$ 的可导性及其导数 $f'(x)$ 的定义：

$$f'(x) = \frac{df}{dx}(x) = \lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h} \quad (3.1)$$

如果式(3.1)的极限存在，则称 $f(x)$ 在 x 可导，导数是 $f'(x)$ 。 h 是一个变化量。在 $f(x)$ 的图像中用一个线段连接 $(x, f(x))^\top$ 和 $(x+h, f(x+h))^\top$ 两点，这个线段称为割线。式(3.1)极限中的商 $\frac{f(x+h)-f(x)}{h}$ 是割线的斜率。随着 h 趋近于0，割线趋近于 $f(x)$ 在 x 的切线(tangent line)。函数在某一点割线斜率的极限就是函数在该点的导数——切线的斜率，如图3-1所示。

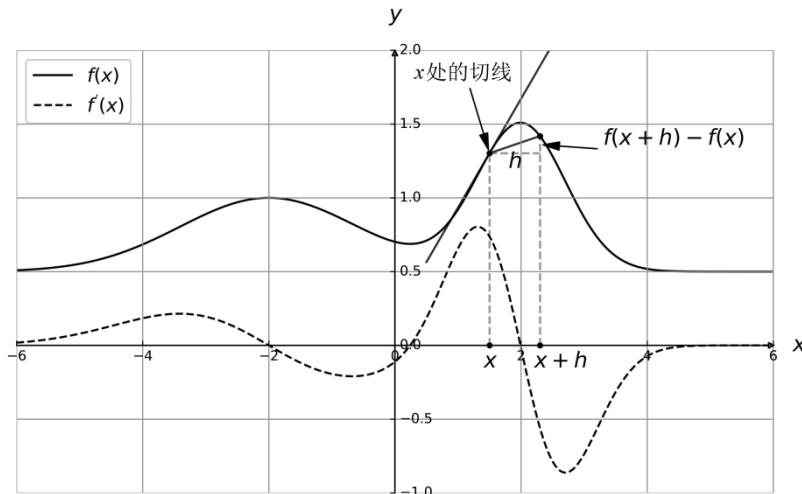


图3-1 一元函数的割线、切线和导数

自变量从 x 变化到 $x+h$ 时，割线斜率 $\frac{f(x+h)-f(x)}{h}$ 也是函数值的平均变化率。导数 $f'(x)$ 是平均变化率的极限——函数值在 x 的瞬时变化率。在一元情况下，自变量只能沿着 x 轴前后运动，这时可以像式(3.1)那样用瞬时变化率定义导数。但多元函数的自变量是向量，可以沿无数方向运动，那么如何定义多元函数的导数呢？

一元函数的可导性还有另一种等价定义：若在 x 附近能用直线近似 $f(x)$ 的图像，则称 $f(x)$ 在 x 可导。后面我们将这种可导定义扩展到多元函数。但首先我们要说明“能用直线近似”是什么意思，并证明在一元情况下这两种可导定义是等价的。假设 $f(x)$ 在 x 可导，构造一个关于 h 的仿射函数：

$$g(h) = f(x) + hf'(x) \quad (3.2)$$

将 $f(x + h)$ 写成一个关于 h 的仿射函数再加余项的形式：

$$f(x + h) = f(x) + hf'(x) + \mathcal{R}(h) \quad (3.3)$$

若函数在某点可导，则它在该点连续。于是 $f(x)$ 在 x 连续，即关于 h 的函数 $f(x + h)$ 在 $h = 0$ 连续。仿射函数 $g(h)$ 也是连续的。余项 $\mathcal{R}(h)$ 是两个连续函数的差，所以它也在 $h = 0$ 连续，即 $\lim_{h \rightarrow 0} \mathcal{R}(h) = \mathcal{R}(0) = 0$ 。另外，根据式(3.1)有：

$$\lim_{h \rightarrow 0} \left| \frac{\mathcal{R}(h)}{h} \right| = \lim_{h \rightarrow 0} \left| \frac{f(x+h)-f(x)}{h} - f'(x) \right| = 0 \quad (3.4)$$

式(3.4)等价于：

$$\lim_{h \rightarrow 0} \frac{\mathcal{R}(h)}{|h|} = 0 \quad (3.5)$$

所以，当 h 趋近于0时， $\mathcal{R}(h)$ 和 $\left| \frac{\mathcal{R}(h)}{h} \right|$ 都趋近于0。这说明随着变化量 h 消失， $\mathcal{R}(h)$ 也消失，而且消失得比 h 还快。这种情况称 $\mathcal{R}(h)$ 是 h 的高阶无穷小。对式(3.3)做一个变量替换，令 $x' = x + h$ ，得到：

$$f(x') = f(x) + f'(x)(x' - x) + \mathcal{R}(h) \approx f(x) + f'(x)(x' - x) \quad (3.6)$$

式(3.6)的含义是：函数在 x 附近可被一个仿射函数近似，近似误差是 x' 与 x 之间距离 $|x' - x|$ 的高阶无穷小。该仿射函数的图像是经过点 $(x, f(x))^\top$ 的直线，即 $f(x)$ 在 x 的切线。切线在 x 点与 $f(x)$ 的图像重合，在其他点与 $f(x)$ 的图像有差距，差距是自变量与 x 之间距离的高阶无穷小。这就是函数的图像在 x 点附近可被直线近似的含义，如图3-2所示。

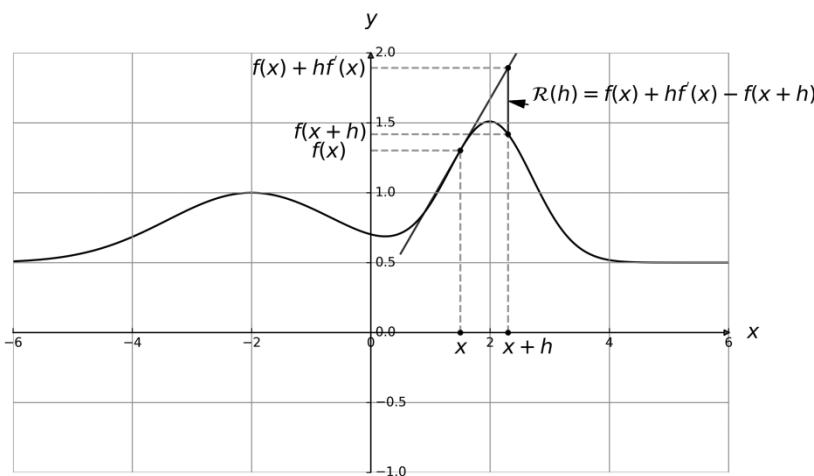


图3-2 可导函数在某点附近可被其切线近似

现在反过来证明：如果 $f(x)$ 的图像在 x 点附近可被直线近似，则 $f(x)$ 在 x 可导且导数是该直线的斜率。如果 $f(x)$ 在 x 附近的值 $f(x+h)$ 可以写成 $f(x+h) = f(x) + ha + \mathcal{R}(h)$ ，其中 $\mathcal{R}(h)$ 是 h 的高阶无穷小，那么有：

$$\lim_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h} = \lim_{h \rightarrow 0} \frac{ha+\mathcal{R}(h)}{h} = a + \lim_{h \rightarrow 0} \frac{\mathcal{R}(h)}{h} = a \quad (3.7)$$

式 (3.7) 说明 $f(x)$ 在 x 可导，导数 $f'(x) = a$ 。这就证明了两种可导定义是等价的。以“可被仿射函数近似”作为可导的定义，就可将可导性扩展到多元函数 $f(\mathbf{x})$ 。如果 $f(\mathbf{x})$ 在 \mathbf{x} 附近的值 $f(\mathbf{x}+\mathbf{h})$ 可以被某个仿射函数近似，即存在记为 $\nabla f(\mathbf{x})$ 的向量，满足：

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{h} + \mathcal{R}(\mathbf{h}) \quad (3.8)$$

其中，余项 $\mathcal{R}(\mathbf{h})$ 是 \mathbf{h} 的长度 $\|\mathbf{h}\|$ 的高阶无穷小：

$$\lim_{\|\mathbf{h}\| \rightarrow 0} \frac{\mathcal{R}(\mathbf{h})}{\|\mathbf{h}\|} = 0 \quad (3.9)$$

则称多元函数 $f(\mathbf{x})$ 在 \mathbf{x} 可导。式 (3.8) 中的向量 $\nabla f(\mathbf{x})$ 称为 $f(\mathbf{x})$ 在 \mathbf{x} 的梯度 (gradient)。对式 (3.8) 做变量替换，令 $\mathbf{x}' = \mathbf{x} + \mathbf{h}$ ，得到：

$$f(\mathbf{x}') = f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{x}' - \mathbf{x}) + \mathcal{R}(\mathbf{x}' - \mathbf{x}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{x}' - \mathbf{x}) \quad (3.10)$$

式 (3.10) 的含义是：函数在 \mathbf{x} 附近可被一个仿射函数近似，近似误差是 \mathbf{x}' 与 \mathbf{x} 之间距离 $\|\mathbf{x}' - \mathbf{x}\|$ 的高阶无穷小。这个仿射函数的图像是经过点 $(\mathbf{x}^T, f(\mathbf{x}))^T$ 的超平面，即函数在 \mathbf{x} 的切平面 (tangent hyperplane)。切平面的法向量是 $(\nabla f(\mathbf{x})^T, -1)^T$ 。于是梯度的方向决定切平面的朝向，梯度的长度 (模) 决定切平面的倾斜程度，如图 3-3 所示。

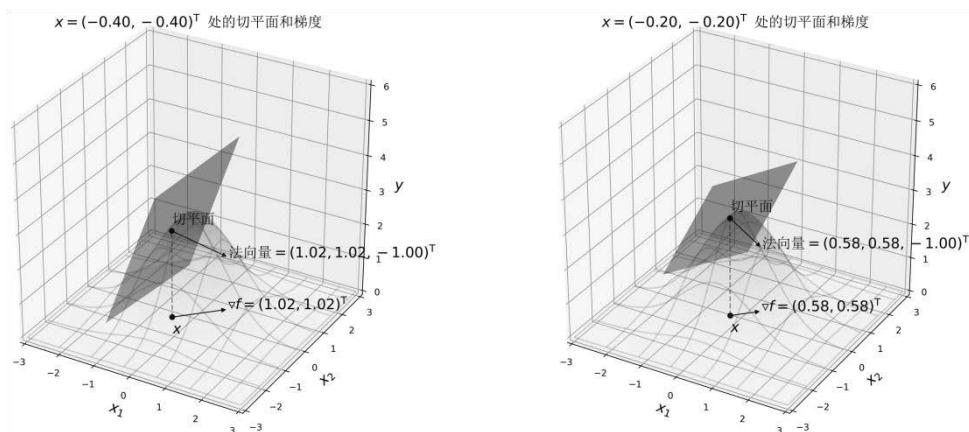


图 3-3 多元函数的切平面和梯度

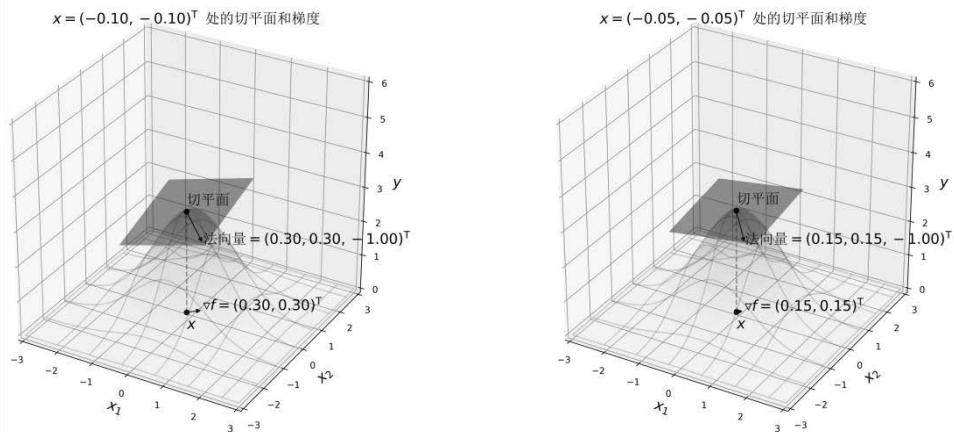


图 3-3 (续)

3.1.2 方向导数

如果 $f(\mathbf{x})$ 在 \mathbf{x} 可导, 如何定义 $f(\mathbf{x})$ 在 \mathbf{x} 沿各个方向的变化率? 可以指定一条经过 \mathbf{x} 的直线, 然后讨论当自变量沿着这条直线运动时函数值的变化率。经过 \mathbf{x} 的直线可定义为:

$$l(t) = \mathbf{x} + t\mathbf{d}, \quad t \in \mathbb{R}, \quad \|\mathbf{d}\| = 1 \quad (3.11)$$

其中, \mathbf{d} 是单位向量, 它决定了直线的走向。 t 决定了 $\mathbf{x} + t\mathbf{d}$ 离 \mathbf{x} 的距离, $l(0) = \mathbf{x}$ 。接下来在直线 $l(t)$ 的基础上定义一个复合函数 $(f \oplus l)(t)$:

$$(f \oplus l)(t) = f(l(t)) = f(\mathbf{x} + t\mathbf{d}) \quad (3.12)$$

$(f \oplus l)(t)$ 是自变量为 t 的一元函数。根据式 (3.1), $(f \oplus l)(t)$ 在 0 的导数是:

$$\frac{d(f \oplus l)}{dt}(0) = \lim_{h \rightarrow 0} \frac{f(l(h)) - f(l(0))}{h} = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{d}) - f(\mathbf{x})}{h} \quad (3.13)$$

式 (3.13) 就是 $f(\mathbf{x})$ 在 \mathbf{x} 沿 \mathbf{d} 的方向导数 (directional derivative), 用 $\nabla_{\mathbf{d}} f(\mathbf{x})$ 表示。 \mathbf{x} 和单位向量 \mathbf{d} 定义了一个坐标轴, \mathbf{x} 是该坐标轴的原点。坐标轴的走向沿着 \mathbf{d} , 以 \mathbf{d} 的方向为正方向。如果限制自变量只能在这条坐标轴上变化, 这就相当于定义了一个一元函数。 $\nabla_{\mathbf{d}} f(\mathbf{x})$ 是这个一元函数在原点的导数, 如图 3-4 所示。

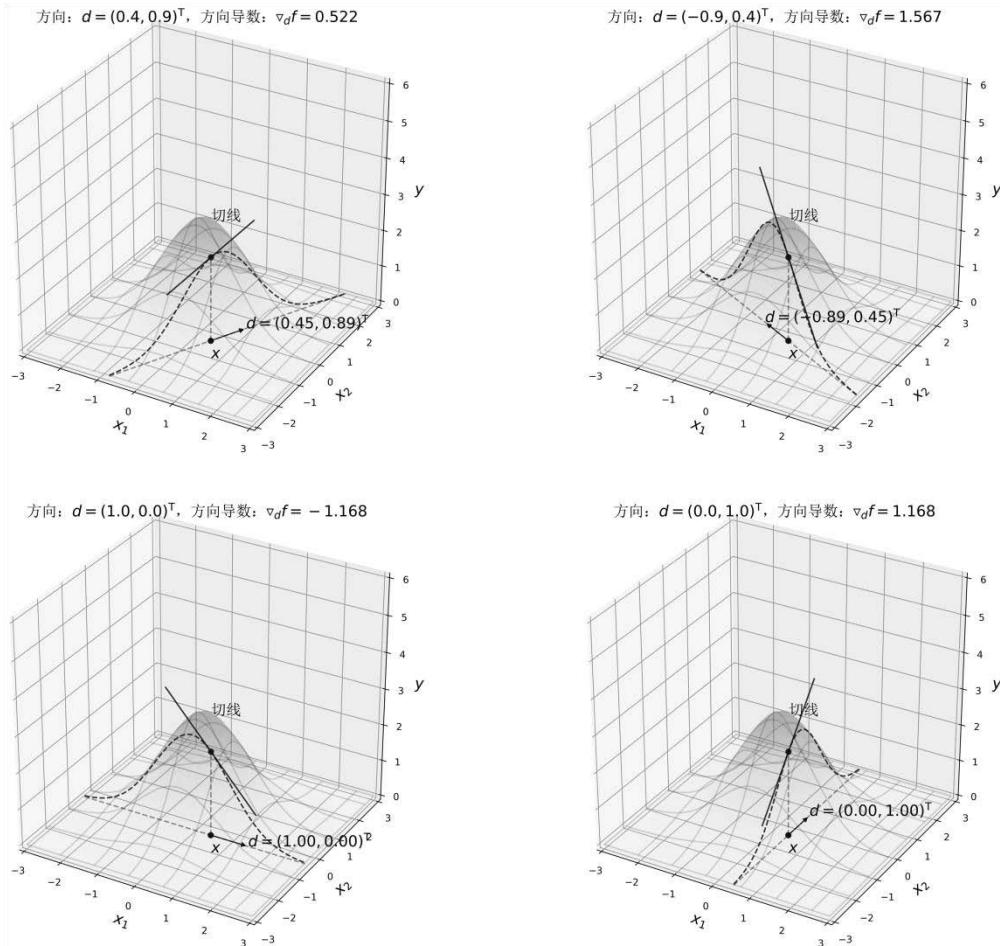


图 3-4 方向导数

d必须是单位向量，否则 $(f \oplus l)(t)$ 的 t 就不再是自变量沿新坐标轴运动的距离。 $-d$ 确定了一个方向相反的坐标轴，函数的变化率相反，即 $\nabla_{-d}f(\mathbf{x}) = -\nabla_d f(\mathbf{x})$ ，这容易验证。 $f(\mathbf{x})$ 在 \mathbf{x} 沿各个方向的方向导数都蕴含在梯度 $\nabla f(\mathbf{x})$ 中。因为 $f(\mathbf{x})$ 在 \mathbf{x} 可导，根据式 (3.8) 有：

$$(f \oplus l)(h) = f(\mathbf{x} + h\mathbf{d}) = f(\mathbf{x}) + h\nabla f(\mathbf{x})^T \mathbf{d} + \mathcal{R}(h\mathbf{d}) \quad (3.14)$$

其中， $\mathcal{R}(h\mathbf{d})$ 是变化量 $h\mathbf{d}$ 的高阶无穷小，即：

$$\lim_{\|h\mathbf{d}\| \rightarrow 0} \frac{\mathcal{R}(h\mathbf{d})}{\|h\mathbf{d}\|} = 0 \quad (3.15)$$

因为 $\|h\mathbf{d}\| = |h|\|\mathbf{d}\| = |h|$ ，所以当 h 趋近于 0 时， $\|h\mathbf{d}\|$ 也趋近于 0。这时有：

$$\lim_{h \rightarrow 0} \frac{\mathcal{R}(h\mathbf{d})}{|h|} = \lim_{h \rightarrow 0} \frac{\mathcal{R}(h\mathbf{d})}{\|h\mathbf{d}\|} = \lim_{\|h\mathbf{d}\| \rightarrow 0} \frac{\mathcal{R}(h\mathbf{d})}{\|h\mathbf{d}\|} = 0 \quad (3.16)$$

所以 $\mathcal{R}(h\mathbf{d})$ 也是 h 的高阶无穷小。另外, 因为 $f(\mathbf{x}) = (f \oplus l)(0)$, 所以式(3.14)表明 $(f \oplus l)(t)$ 在 0 的导数是 $\nabla f(\mathbf{x})^T \mathbf{d}$, 即 $\nabla_{\mathbf{d}} f(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{d}$ 。于是我们有:

$$\nabla_{\mathbf{d}} f(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{d} = \|\nabla f(\mathbf{x})\| \cdot \|\mathbf{d}\| \cos \theta = \|\nabla f(\mathbf{x})\| \cos \theta \quad (3.17)$$

其中, θ 是 $\nabla f(\mathbf{x})$ 与 \mathbf{d} 之间的夹角。由式(3.17)可知, $f(\mathbf{x})$ 在 \mathbf{x} 沿 \mathbf{d} 的方向导数等于 $f(\mathbf{x})$ 在 \mathbf{x} 的梯度 $\nabla f(\mathbf{x})$ 向 \mathbf{d} 的投影长度。当 \mathbf{d} 与 $\nabla f(\mathbf{x})$ 同方向 ($\theta = 0$) 时, $\nabla_{\mathbf{d}} f(\mathbf{x})$ 最大。所以 $\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$ 是 $f(\mathbf{x})$ 变化率最大的方向, 其变化率是 $\|\nabla f(\mathbf{x})\| \geq 0$ 。沿着 $-\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$ 方向 ($\theta = \pi$) 的方向导数最小, 等于 $-\|\nabla f(\mathbf{x})\| \leq 0$ 。所以, 梯度反方向 $-\nabla f(\mathbf{x})$ 是 $f(\mathbf{x})$ 变化率最小的方向, 即函数值下降最快的方向。

在 2 维的情况下, 可以用切平面描述梯度 $\nabla f(\mathbf{x})$ 与方向导数 $\nabla_{\mathbf{d}} f(\mathbf{x})$ 的关系。切平面的法向量是 $(\nabla f(\mathbf{x})^T, -1)^T$, 第 3 维-1 说明法向量指向下方。法向量在 x_1x_2 平面的投影是 $\nabla f(\mathbf{x})$, 它指向切平面的上坡方向, $-\nabla f(\mathbf{x})$ 指向切平面的下坡方向, 如图 3-3 所示。

自变量沿任意方向 \mathbf{d} 的运动可分解为两个分量: 平行于 $\nabla f(\mathbf{x})$ 的分量和垂直于 $\nabla f(\mathbf{x})$ 的分量。垂直于 $\nabla f(\mathbf{x})$ 的分量上, $\nabla_{\mathbf{d}} f(\mathbf{x}) = 0$ 。自变量沿 \mathbf{d} 的运动一部分摊在了垂直分量上, 这部分运动不会导致 $f(\mathbf{x})$ 变化。只有在平行于 $\nabla f(\mathbf{x})$ 的分量上的运动才导致 $f(\mathbf{x})$ 变化。所以 $f(\mathbf{x})$ 沿 \mathbf{d} 的变化率就打了折扣, 折扣系数是平行于 $\nabla f(\mathbf{x})$ 的分量所占的“份额”—— $\cos \theta$, 如图 3-5 所示。

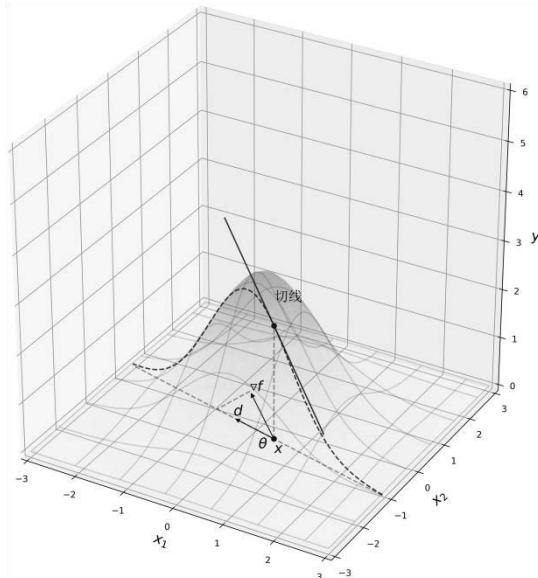


图 3-5 梯度与方向导数

3.1.3 偏导数

$f(\mathbf{x})$ 在 \mathbf{x} 点对其第 i 分量 x_i 的偏导数是把其他分量 x_j ($j \neq i$)，当作常数时 $f(\mathbf{x})$ 对 x_i 的导数。这时候将 $f(\mathbf{x})$ 看作关于 x_i 的一元函数。根据导数的定义：

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}^i) - f(\mathbf{x})}{h} \quad (3.18)$$

其中， $\mathbf{x} + h\mathbf{e}^i$ 保持 x_j ($j \neq i$) 不变，只让 x_i 发生变化，变化量是 h 。 $f(\mathbf{x})$ 有 n 个偏导数： $\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}$ 。

3

根据式 (3.13)，有：

$$\nabla_{\mathbf{e}^i} f(\mathbf{x}) = \frac{\partial f}{\partial x_i}(\mathbf{x}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}^i) - f(\mathbf{x})}{h} \quad (3.19)$$

$f(\mathbf{x})$ 对 x_i 的偏导数就是 $f(\mathbf{x})$ 沿 \mathbf{e}^i 的方向导数。偏导数是方向导数的特例，它们的方向是各个坐标轴的正方向。由于 $\nabla f(\mathbf{x})$ 与 \mathbf{e}^i 的内积是 $\nabla f(\mathbf{x})$ 的第 i 分量 $\nabla f(\mathbf{x})_i$ ，根据式 (3.17) 和式 (3.19)，有：

$$\nabla f(\mathbf{x})_i = \nabla f(\mathbf{x})^T \mathbf{e}^i = \nabla_{\mathbf{e}^i} f(\mathbf{x}) = \frac{\partial f}{\partial x_i}(\mathbf{x}), \quad i = 1, \dots, n \quad (3.20)$$

所以梯度 $\nabla f(\mathbf{x})$ 的第 i 分量是 $f(\mathbf{x})$ 对自变量第 i 分量 x_i 的偏导数，于是梯度就是：

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{x}) \end{pmatrix} \quad (3.21)$$

$f(\mathbf{x})$ 对自变量各个分量的偏导数是唯一的，所以 $\nabla f(\mathbf{x})$ 也是唯一的。

3.1.4 驻点

函数 $f(\mathbf{x})$ 的驻点 (stationary point) 是梯度为零向量的点。驻点也称临界点 (critical point)。既然驻点的梯度是零向量，所以 $f(\mathbf{x})$ 在驻点的切平面的法向量是：

$$\mathbf{w} = \begin{pmatrix} \nabla f(\mathbf{x}) \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix} \quad (3.22)$$

驻点的切平面的法向量 \mathbf{w} 垂直指向下方，所以切平面是水平的，如图 3-6 所示。 $f(\mathbf{x})$ 在驻点沿任意方向 \mathbf{d} 的方向导数是 $\nabla f(\mathbf{x})^T \mathbf{d} = 0$ ，所以 $f(\mathbf{x})$ 在驻点向任意方向的变化率都为 0。

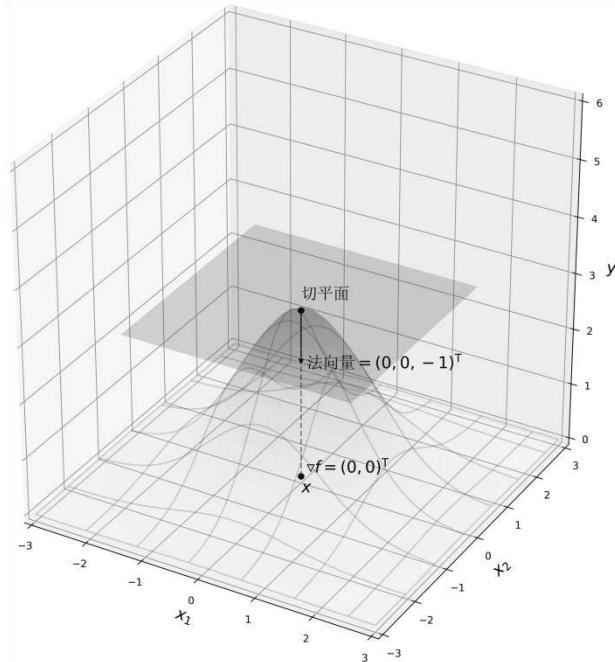


图 3-6 驻点的切平面

3.1.5 局部极小点

如果在点 \mathbf{x}^* 周围存在一个半径为 $\varepsilon > 0$ 的邻域，该邻域内所有点的函数值都不小于 $f(\mathbf{x}^*)$ ，则 \mathbf{x}^* 是 $f(\mathbf{x})$ 的一个局部极小点 (local minima)，用公式表示就是：

$$f(\mathbf{x}) \geq f(\mathbf{x}^*), \quad \|\mathbf{x} - \mathbf{x}^*\| < \varepsilon \quad (3.23)$$

如果对于自变量空间的所有 \mathbf{x} ，都有 $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ ，则 \mathbf{x}^* 是 $f(\mathbf{x})$ 的全局最小点 (global minima)。很显然，全局最小点是局部极小点。但是局部极小点不一定是全局最小点。类似还可以定义局部极大点 (local maxima) 和全局最大点 (global maxima)，如图 3-7 所示。

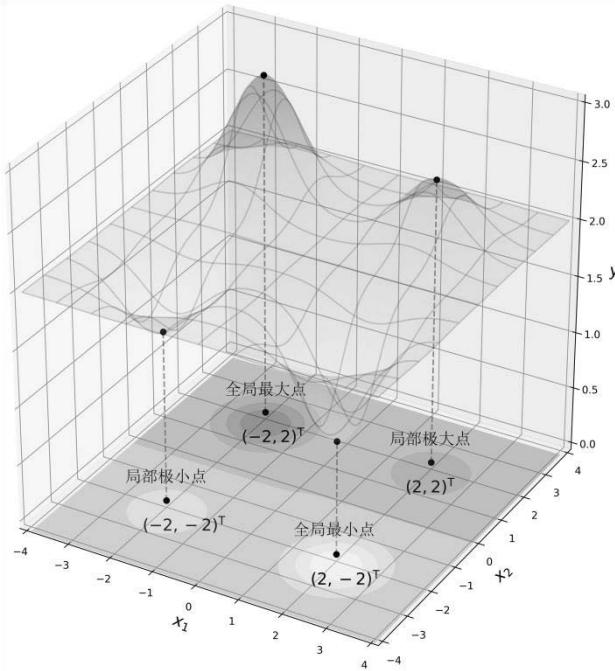


图 3-7 局部极小/大点和全局最小/大点

局部极小点 \mathbf{x}^* 一定是驻点, 即 $\|\nabla f(\mathbf{x}^*)\| = 0$ 。运用反证法, 假设 $\|\nabla f(\mathbf{x}^*)\| \neq 0$, 从 \mathbf{x}^* 点出发沿 $-\nabla f(\mathbf{x}^*)$ 方向做一个位移 $-t\nabla f(\mathbf{x}^*)$, $t > 0$, 因为 $f(\mathbf{x})$ 在 \mathbf{x}^* 可导, 有:

$$f(\mathbf{x}^* - t\nabla f(\mathbf{x}^*)) = f(\mathbf{x}^*) - t\|\nabla f(\mathbf{x}^*)\|^2 + \mathcal{R}(-t\nabla f(\mathbf{x}^*)) \quad (3.24)$$

$\mathcal{R}(-t\nabla f(\mathbf{x}^*))$ 是位移 $-t\nabla f(\mathbf{x}^*)$ 的高阶无穷小, 于是有:

$$\lim_{t \rightarrow 0} \frac{-t\|\nabla f(\mathbf{x}^*)\|^2 + \mathcal{R}(-t\nabla f(\mathbf{x}^*))}{\|-t\nabla f(\mathbf{x}^*)\|} = -\|\nabla f(\mathbf{x}^*)\| + \lim_{t \rightarrow 0} \frac{\mathcal{R}(-t\nabla f(\mathbf{x}^*))}{\|-t\nabla f(\mathbf{x}^*)\|} = -\|\nabla f(\mathbf{x}^*)\| < 0 \quad (3.25)$$

这说明当 t 趋近于 0 时, 式 (3.24) 的后两项的极限是负值。所以对于足够小的 $\varepsilon > 0$, 当 $t < \varepsilon$ 时有:

$$f(\mathbf{x}^* - t\nabla f(\mathbf{x}^*)) - f(\mathbf{x}^*) = -t\|\nabla f(\mathbf{x}^*)\|^2 + \mathcal{R}(-t\nabla f(\mathbf{x}^*)) < 0 \quad (3.26)$$

随着 t 继续靠近 0, $\mathbf{x}^* - t\nabla f(\mathbf{x}^*)$ 在无限靠近 \mathbf{x}^* 的同时保持 $f(\mathbf{x}^* - t\nabla f(\mathbf{x}^*)) < f(\mathbf{x}^*)$ 。这与 \mathbf{x}^* 是 $f(\mathbf{x})$ 的局部极小点矛盾。所以 $\nabla f(\mathbf{x}^*)$ 只能是零向量, 即 \mathbf{x}^* 是驻点。类似可以证明, 局部极大点也一定是驻点。驻点是局部极小点的必要非充分条件。驻点也有可能是局部极大点或者鞍点 (saddle point)。鞍点的梯度也是零向量, 但在任意一个邻域内都同时存在函数值更大的点和更小的点,

如图 3-8 所示。仅靠梯度难以判断驻点的类型，驻点的类型可由赫森矩阵的特征值揭示，这将在下一章讨论。

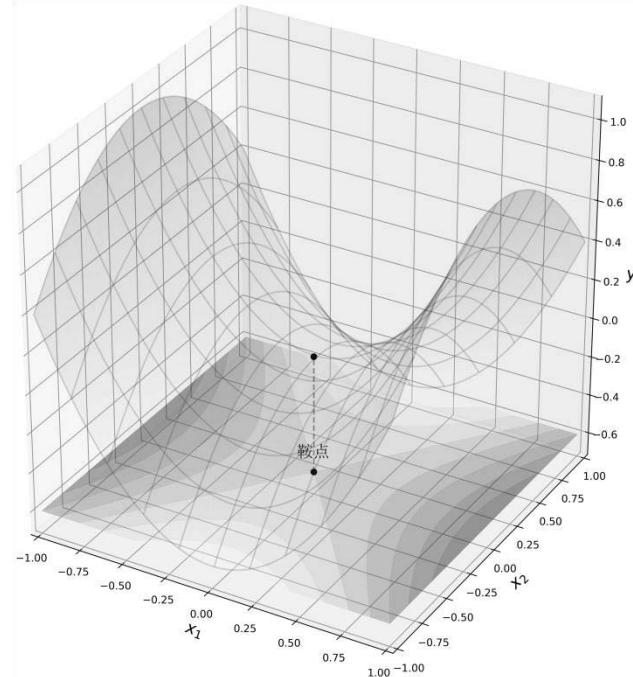


图 3-8 鞍点

3.2 梯度下降法

为了寻找函数的全局最小点，可以先找到满足必要条件的点——驻点。根据定义，可以求函数的梯度，令梯度为零向量而求得驻点，但大多数时候这并不可行。举个简单的例子，有一个二次型（quadratic form）函数：

$$f(x^1, x^2, \dots, x^n) = \frac{1}{2} \sum_{i,j} w^{i,j} x^i x^j \quad (3.27)$$

其中， $w^{i,j} = w^{j,i}$ 。该二次型对每一个自变量 x^i 的偏导数是：

$$\frac{\partial f}{\partial x^i} = \sum_{j=1}^n w^{i,j} x^j \quad (3.28)$$

令梯度是零向量，求 x^1, x^2, \dots, x^n 。这是解 n 元一次方程组：

$$\sum_{j=1}^n w^{i,j} x^j = 0, \quad i = 1, \dots, n \quad (3.29)$$

一般情况下这需要 $O(n^3)$ 的时间复杂度。当 n 非常大时(这在神经网络和深度学习中是必然的),求驻点的解析解是不可接受的。这还仅仅是简单的二次型的情况,当情况更复杂时,驻点的解析解有可能不存在,我们需要迭代的数值解法。

3.2.1 反梯度场

如果 $f(\mathbf{x})$ 是 n 元函数,则 $\nabla f(\mathbf{x})$ 是 n 维向量。可导函数 $f(\mathbf{x})$ 在自变量空间中每一个点都有一个反梯度向量 $-\nabla f(\mathbf{x})$,指向在该点函数值下降最快的方向。这就形成了一个速度(向量)场。可以将 $-\nabla f(\mathbf{x})$ 画成箭头,尾部移到 \mathbf{x} 的位置,以这种方式将反梯度场呈现出来,如图 3-9 所示。

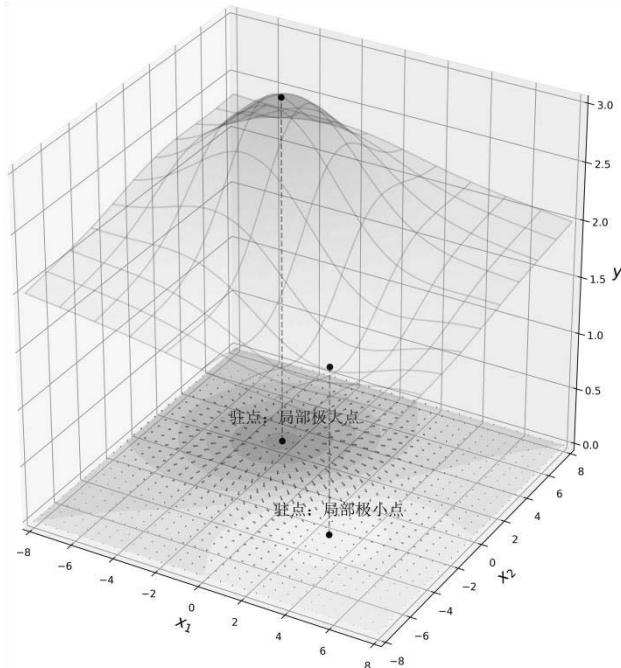


图 3-9 反梯度场

速度场在每一点指定了该位置的速度——方向和速率。在反梯度场的情况下,速度指向函数值下降最快的方向,速率大小是函数值的下降速率。将一个粒子(particle)从任意位置放入速度场中,它就会按照场指定的速度运动。在反梯度场的情况下,粒子就是按照梯度反方向,朝函数值下降最快的方向运动。

这里解释一下反梯度场的物理意义。在 2 维情况下可以将函数图像看作一幅起伏不平的地形,设重力加速度的大小是 g ,有一个质量是 m 的小球被放在任意位置。地面对小球的支持力垂直于

坡面，沿着该位置的切平面的朝上的法向量 $(-\nabla f(\mathbf{x})_1, -\nabla f(\mathbf{x})_2, 1)^T$ 。这个法向量与 x_1x_2 平面的夹角 θ 的正切是 $\tan \theta = \|\nabla f(\mathbf{x})\|^{-1}$ 。支持力的垂直分量抵消重力 mg ，水平分量的大小则是 $mg/\tan \theta = mg\|\nabla f(\mathbf{x})\|$ ，水平分量的方向是法向量向 x_1x_2 平面的投影 $-\nabla f(\mathbf{x})$ 的方向。小球受到的水平作用力正是 $-mg\nabla f(\mathbf{x})$ ，小球的水平加速度就是 $-g\nabla f(\mathbf{x})$ 。所以若将函数图形看作地形，反梯度场是该地形的加速度场，而不是速度场，如图 3-10 所示。

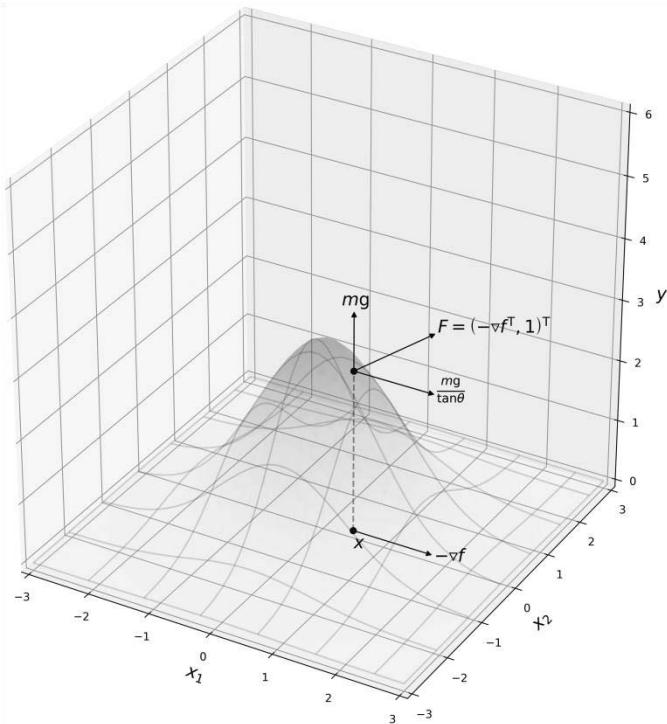
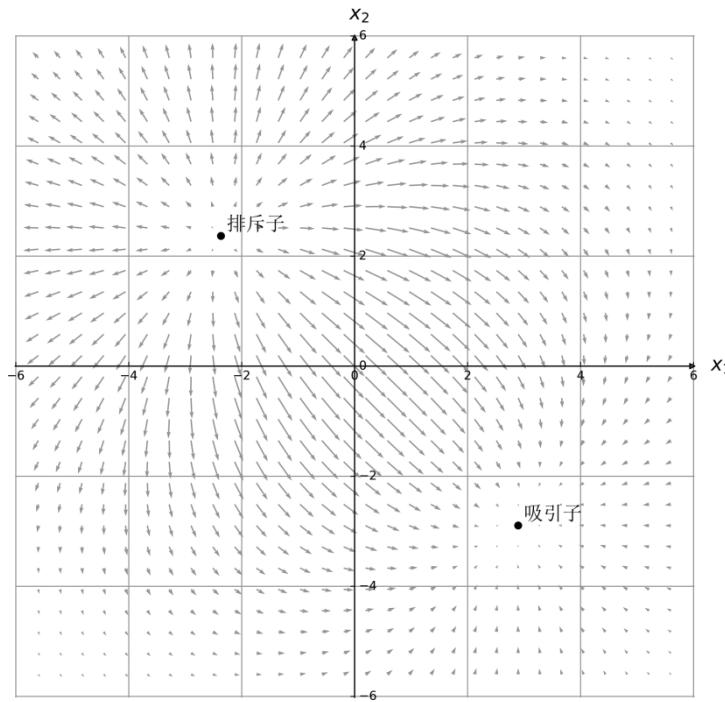


图 3-10 反梯度场的物理意义

局部极小点的梯度是零向量，它们是反梯度场的静止点。如果一个粒子处于静止点上，它将保持静止。同理，局部极大点也是静止点。局部极小点是稳定静止点，或者说吸引子 (attractor)。当粒子偏离局部极小点一个小位移，它将被吸引回局部极小点。局部极大点是不稳定静止点，或者说排斥子 (repeller)。当粒子位于局部极大点时，它是静止的，但是一旦有一个微小的扰动使它发生极小的位移，它将被推得远离局部极大点，如图 3-11 所示。



3

图 3-11 吸引子和排斥子

可以从任意位置开始模拟粒子在反梯度场中的运动。除非粒子的初始位置刚好是静止点，否则粒子将向函数值下降最快的方向，即反梯度方向运动，并最终逼近吸引子——局部极小点。从理论上，反梯度场只能保证粒子运动到局部极小点，不能保证运动到全局最小点，逼近的速度也没有保证。实际算法中无法精确模拟粒子的连续运动，而只能以离散迭代的方式近似，这会带来更多问题，甚至不收敛。

3.2.2 梯度下降法

在计算机中模拟粒子在速度场中的运动，这属于数值积分问题。梯度下降法（gradient descent, GD）是一种简单的数值积分算法，伪代码如下：

```

 $x^0 \leftarrow$  随机初始化
 $t \leftarrow 0$ 
while  $\|\nabla f(x^t)\| \geq \varepsilon:$ 
     $x^{t+1} \leftarrow x^t - \eta \cdot \nabla f(x^t)$ 
     $t \leftarrow t + 1$ 
return  $x^t$ 

```

$\varepsilon > 0$ 是一个预设的阈值，当 $\|\nabla f(\mathbf{x})\| < \varepsilon$ 时，认为 $\nabla f(\mathbf{x})$ 已经足够接近零向量，算法停止。也可以采用其他停止标准，例如循环次数达到预设的最大值，或者函数值的下降幅度小于阈值。 $\eta > 0$ 是另一个预设值，称为学习率 (learning rate, LR) 或步长。每一次迭代中，自变量向 $-\nabla f(\mathbf{x})$ 方向运动，运动的距离是 $\eta \cdot \|\nabla f(\mathbf{x})\|$ 。

η 是梯度下降法的一个超参数 (hyper parameter)。我们可以保证在任意 \mathbf{x} ，能够找到一个合适的步长 η_x ，使 $f(\mathbf{x} - \eta_x \nabla f(\mathbf{x})) < f(\mathbf{x})$ ，也就是说，保证从 \mathbf{x} 出发移动 $-\eta_x \nabla f(\mathbf{x})$ ，可以使函数值下降。这是因为：

$$f(\mathbf{x} - \eta_x \nabla f(\mathbf{x})) = f(\mathbf{x}) - \eta_x \|\nabla f(\mathbf{x})\|^2 + \mathcal{R}(-\eta_x \nabla f(\mathbf{x})) \quad (3.30)$$

其中， $\mathcal{R}(-\eta_x \nabla f(\mathbf{x}))$ 是 $-\eta_x \nabla f(\mathbf{x})$ 的高阶无穷小。式 (3.30) 与式 (3.24) 相似，可以证明存在一个 $\varepsilon > 0$ ，当 $\eta_x < \varepsilon$ 时，满足 $f(\mathbf{x} - \eta_x \nabla f(\mathbf{x})) < f(\mathbf{x})$ 。我们说 $-\nabla f(\mathbf{x})$ 是确保下降的方向，但确保下降只是理论上的，因为不同 \mathbf{x} 的 η_x 不同，也无法计算出 η_x 的值，所以只能使用固定步长 η 。在固定步长下，每一次更新不一定确保函数值下降。

3.2.3 梯度下降法的问题

因为梯度只包含函数的局部线性近似信息，在距离 \mathbf{x} 较远的地方，函数的形状会较大地偏离近似超平面，这是梯度所无法体现的。如果 η 设置得过大，函数值有可能不降反升。较小的 η 更有可能保证函数值下降，但是如果 η 过小，收敛的速度会很慢。函数的图像可能有千奇百怪的“地形”，很多病态情况都会对梯度下降法的效果产生负面影响，这里举几个例子。

“悬崖” (cliff) 如图 3-12 所示。局部极小点位于悬崖脚下，靠近悬崖的位置具有较大梯度，这时梯度下降法会把点弹得远离悬崖，之后需要很多次迭代才能将点拉回悬崖脚下。

“峡谷” (valley) 如图 3-13 所示。局部极小点在谷底，在峡谷壁上，反梯度方向并不指向局部极小点，而是指向峡谷对侧。这种情况下，梯度下降法会发生震荡，轻则延缓收敛速度，重则导致收敛失败。下一章介绍函数的局部二阶特性后，我们会知道峡谷的成因与赫森矩阵各个特征值的相对大小有关。

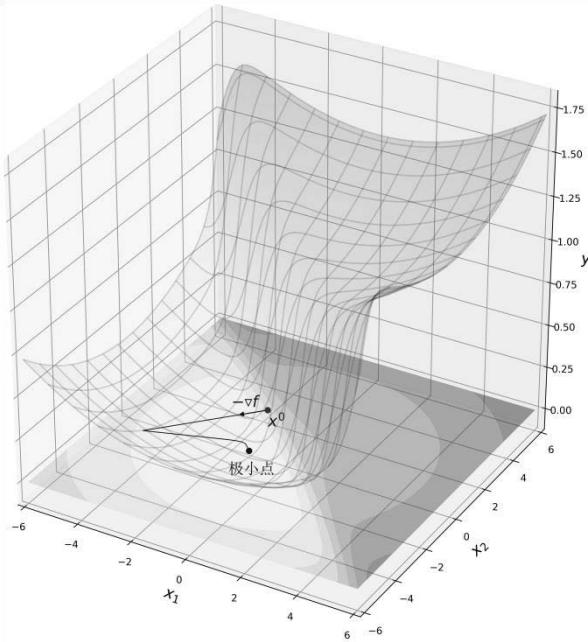


图 3-12 “悬崖”对梯度下降法的影响

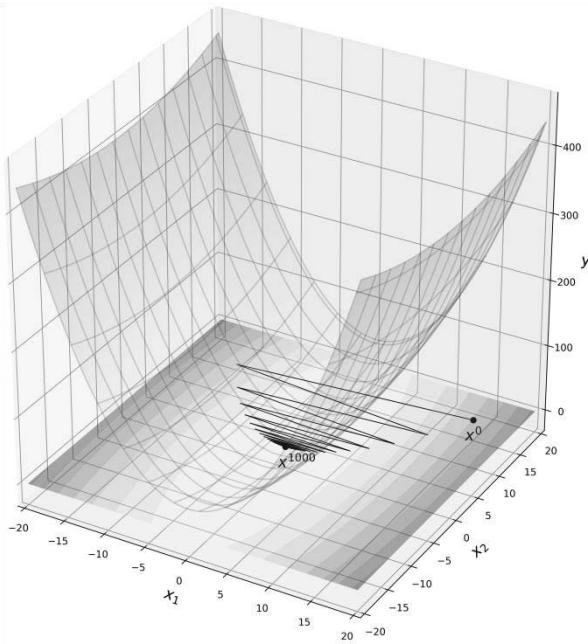


图 3-13 “峡谷”使梯度下降法发生震荡

“平原”(plain)如图3-14所示。在这样的区域里梯度的模非常小，这将导致收敛缓慢。本节仅举了几个简单直观的例子，来说明复杂函数地形下梯度下降法可能遇到的困难。实际的情况还要更复杂，也更难直接观察。

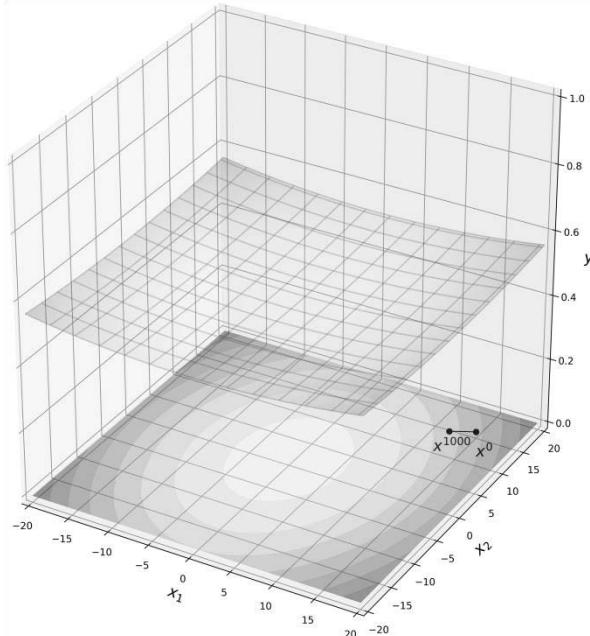


图3-14 “平原”对梯度下降法的影响

3.3 梯度下降法的改进

针对梯度下降法的问题，人们提出了一些改进算法。这些算法有的着眼于调整学习率，有的对搜索方向做某种修正。它们都是原始梯度下降法的变体，终归还是利用局部一阶信息。有些变体会积累历史梯度信息，根据它们调整前进方向。积累梯度的历史也就是观察梯度的变化，尝试从梯度的历史变化中提取信息，模拟函数在当前位置的二阶信息。二阶优化算法用二次函数在局部模拟原函数，是一种更精确的近似。

梯度下降法的变体有很多，本节挑选了在原理上有代表性的，并有前后承接关系的五种进行讲解。它们是学习率调度、冲量法、AdaGrad、RMSProp以及Adam。

3.3.1 学习率调度

学习率是梯度下降法的一个重要超参数，它对算法的结果和效率都有重要的影响。原始梯度

下降法采用固定步长，如果步长设置得过大，则算法有可能不收敛，或者因震荡而延缓收敛。如果步长设置得过小，收敛则会非常缓慢，特别是在梯度较小的“平原”地带。学习率对算法的影响如图 3-15 所示。

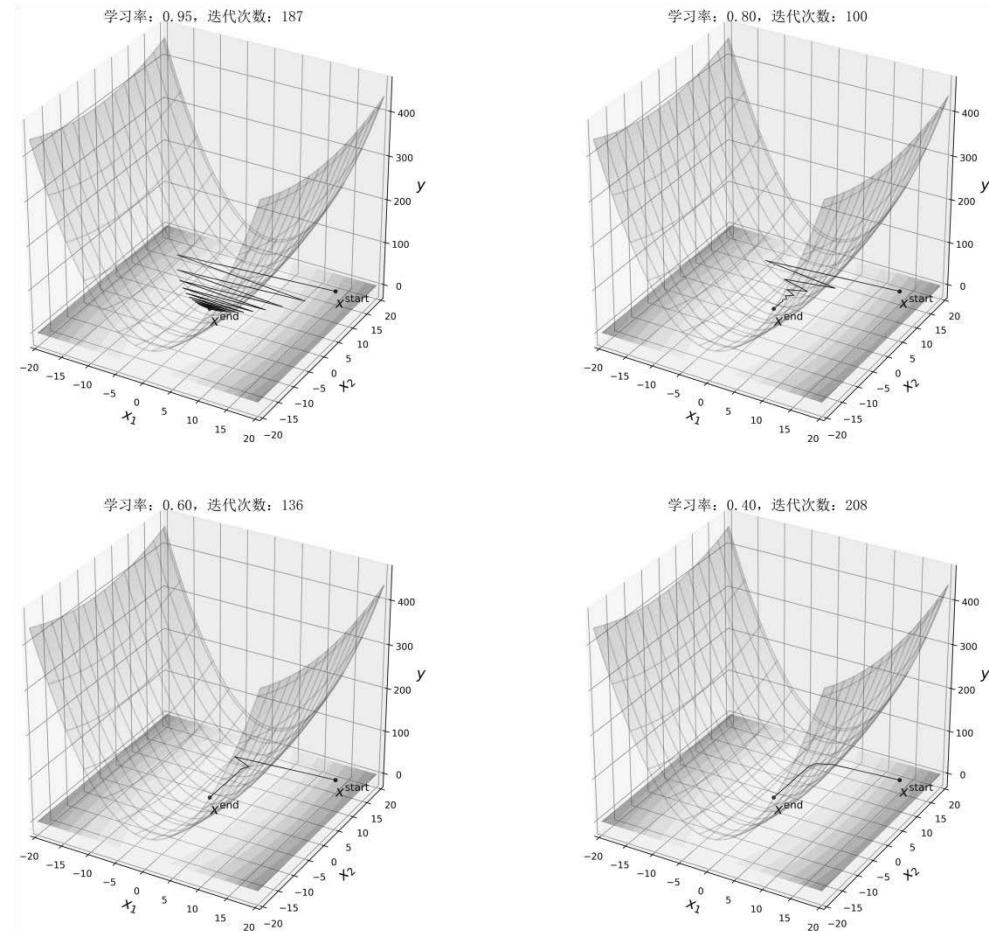


图 3-15 学习率对梯度下降法的影响

一个自然而然的想法就是抛弃固定的学习率，使学习率随着迭代而动态调整。例如，当发现最近若干轮迭代中函数值下降较快，则说明当前区域梯度较大，这时缩小步长；反之如果发现最近若干轮迭代中函数值下降很慢，则说明当前区域梯度较小，这时增加步长。或者当训练开始时采用较大的学习率，随着迭代逐渐缩小学习率。训练开始时可以假设当前距离目标尚远，采用较大的学习率可以加快收敛速度。而随着训练的进行，点逐渐向目标靠近，这时减小学习率以防止震荡。当然距离目标是远还是近在训练时是不知道的，所以这只是一种启发式的策略。这类办法

统称学习率调度 (learning rate scheduling)。

实验表明, 简单的学习率指数衰减策略(exponential decay scheduling)就能取得不错的效果。学习率指数衰减的公式为:

$$\eta^t \leftarrow \eta^{\text{init}} \cdot 10^{-\frac{t}{r}} \quad (3.31)$$

其中, η^{init} 是初始学习率, t 是迭代步数, 超参数 r 控制衰减的速度。 r 越大, 则衰减越慢。动态学习率 η^t 随迭代步数 t 变化的典型曲线如图 3-16 所示。

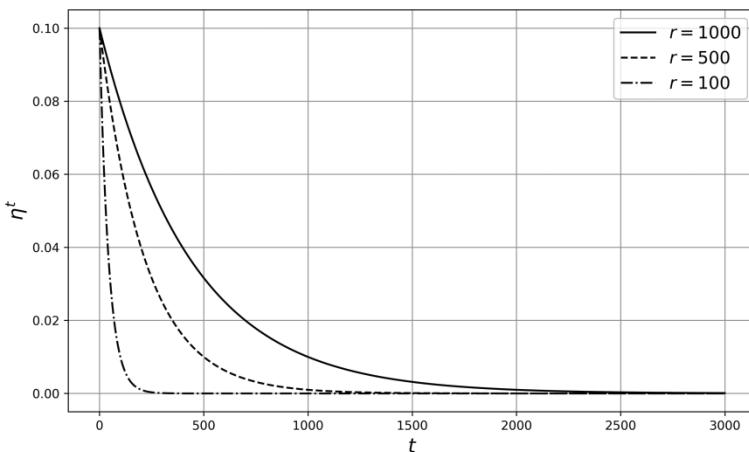


图 3-16 指数衰减的动态学习率随迭代步数变化的曲线

将学习率指数衰减更新式 (3.31) 插入到原始梯度下降算法中, 伪代码如下:

```

 $x^0 \leftarrow$  随机初始化
 $t \leftarrow 0$ 
while  $\|\nabla f(x^t)\| \geq \varepsilon$ :
     $\eta^t \leftarrow \eta^{\text{init}} \cdot 10^{-\frac{t}{r}}$ 
     $x^{t+1} \leftarrow x^t - \eta^t \cdot \nabla f(x^t)$ 
     $t \leftarrow t + 1$ 
return  $x^t$ 

```

3.3.2 冲量法

3.2.1 节解释了反梯度场的物理意义, 如果将函数图像视作重力场中的地形, 那么反梯度场则是水平加速度场, 而不是速度场。原始梯度下降法相当于将加速度场当作速度场, 模拟粒子在其中的运动。冲量 (momentum) 法则恢复了反梯度场的加速度场本质, 以反梯度向量为水平加

速度来模拟粒子的运动，伪代码如下：

```

 $x^0 \leftarrow$  随机初始化
 $v^0 \leftarrow \mathbf{0}$ 
 $t \leftarrow 0$ 
while  $\|\nabla f(x^t)\| \geq \varepsilon$ :
     $v^{t+1} \leftarrow \beta v^t - \eta \cdot \nabla f(x^t)$ 
     $x^{t+1} \leftarrow x^t + v^{t+1}$ 
     $t \leftarrow t + 1$ 
return  $x^t$ 

```

3

从物理角度阐释，学习率 η 就是一个时间单元，反梯度向量 $-\nabla f(x^t)$ 现在是加速度向量，速度向量 v^t 累积了从运动开始以来的速度变化。为了避免速度变得无限大，冲量法引入了摩擦。 v^t 的更新式中的 $0 < \beta < 1$ 相当于摩擦系数（实际上它是物理意义上的摩擦系数的一个变体）。如果梯度不变，是常向量 ∇f ，则有：

$$v^\infty = -(\eta \cdot \sum_{i=0}^{\infty} \beta^i) \cdot \nabla f = -\frac{\eta \cdot \nabla f}{1-\beta} \quad (3.32)$$

邮电

可见如果梯度保持不变，冲量法的极限更新量是原始梯度下降法的 $\frac{1}{1-\beta}$ 。如果 $\beta = 0.9$ ，则冲量法最终将趋近原始梯度下降法 10 倍的更新速度。如果梯度方向有变化，冲量法通过将之前的梯度进行滑动平均，能起到减少震荡的作用。

冲量法有一个变体：Nesterov 法。它与冲量法的区别在于 v^{t+1} 更新式中的梯度不是在 x^t 处计算，而是在 $x^t + \beta v^t$ 处计算。因为冲量法的更新式可以写成 $x^{t+1} \leftarrow x^t + \beta v^t - \eta \nabla f$ ，相当于从 $x^t + \beta v^t$ 出发前进 $-\eta \nabla f$ ，所以使用 $x^t + \beta v^t$ 处的梯度会更合适。

3.3.3 AdaGrad

原始梯度下降法对反梯度向量的每一个分量使用同样的学习率。用标量 η 乘 $-\nabla f(x^t)$ ，是对其长度进行缩放，但不改变其方向。AdaGrad 方法会为反梯度向量的每一个分量适配一个不同的学习率，这样就等于对前进方向做了调整，偏离了反梯度方向。先看伪代码：

```

 $x^0 \leftarrow$  随机初始化
 $s^0 \leftarrow \mathbf{0}$ 
 $t \leftarrow 0$ 
while  $\|\nabla f(x^t)\| \geq \varepsilon$ :
     $s^{t+1} \leftarrow s^t + \nabla f(x^t) \otimes \nabla f(x^t)$ 
     $x^{t+1} \leftarrow x^t - \eta \cdot \nabla f(x^t) \oslash \sqrt{s^{t+1} \oplus \epsilon}$ 
     $t \leftarrow t + 1$ 
return  $x^t$ 

```

\otimes 、 \oslash 和 \oplus 分别表示将两个向量的对应元素相乘、相除和相加。 $\nabla f(\mathbf{x}^t) \otimes \nabla f(\mathbf{x}^t)$ 计算 $f(\mathbf{x})$ 在 \mathbf{x}^t 的梯度的每个分量的平方，将其累加到向量 \mathbf{s}^{t+1} 中。 ϵ 是一个所有分量都是微小值（例如 10^{-18} ）的向量。更新自变量时，将 \mathbf{s}^{t+1} 与 ϵ 的对应分量相加后开方，除梯度的每一分量，得到前进方向。

\mathbf{s}^{t+1} 的每个分量是算法执行至此时梯度每个分量的非中心方差（二阶矩）的累加。如果梯度的某一分量一直较大，则 AdaGrad 会对当前梯度的这个分量做压缩，减小该分量上的更新量，对前进方向进行调整，如图 3-17 所示。

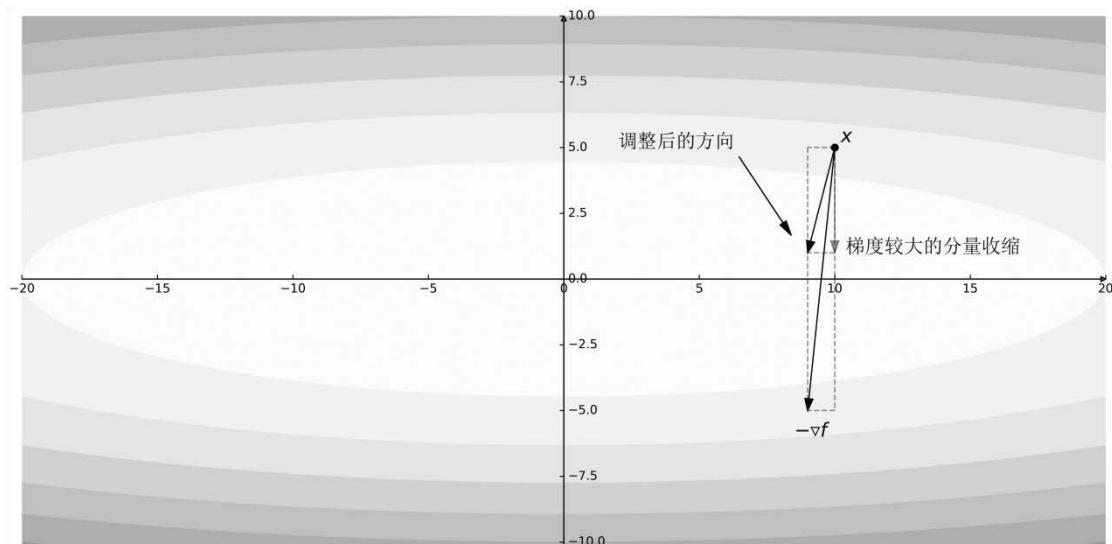


图 3-17 AdaGrad 对梯度一直较大的分量进行惩罚

下一章介绍牛顿法后，会看出 AdaGrad 是试图对函数局部二阶信息进行粗糙的估计。因为 AdaGrad 毕竟没有计算二阶信息，而只是用历史一阶信息模拟二阶信息，所以它还是属于一阶优化算法。因为累加项永远是正值， \mathbf{s}^{t+1} 的各元素只增不减，所以 AdaGrad 还有学习率衰减的效果，但如果加以限制，算法最终将陷入停滞。

3.3.4 RMSProp

AdaGrad 算法对梯度的每一个分量积累历史上所有值的平方和，调整前进方向时，各分量会根据自己的历史值平方和而获得不同的权值，历史值平方和较大的梯度分量会受到较大的惩罚。但使用梯度的全部历史对当前梯度进行修正并不适当，应该使用近期一个时间窗内的梯度历史信息。时间上的局部导致空间上的局部，最近时间窗内的梯度历史反映的是函数的线性近似在局部的变化情况，这是对函数局部二阶信息更合理的模拟。

另外，无衰减地积累梯度分量的历史平方和，会导致权重趋近于 0，这起到类似学习率衰减的作用，但也会使算法最终陷入停滞。针对这些问题，RMSProp 对 AdaGrad 进行了改进，其伪代码如下：

```

 $\mathbf{x}^0 \leftarrow$  随机初始化
 $\mathbf{s}^0 \leftarrow \mathbf{0}$ 
 $t \leftarrow 0$ 
while  $\|\nabla f(\mathbf{x}^t)\| \geq \varepsilon$ :
     $\mathbf{s}^{t+1} \leftarrow \beta \mathbf{s}^t + (1 - \beta) \cdot \nabla f(\mathbf{x}^t) \otimes \nabla f(\mathbf{x}^t)$ 
     $\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t - \eta \cdot \nabla f(\mathbf{x}^t) \oslash \sqrt{\mathbf{s}^{t+1}} \oplus \epsilon$ 
     $t \leftarrow t + 1$ 
return  $\mathbf{x}^t$ 

```

3

RMSProp 引入了一个滑动平均系数 $0 < \beta < 1$ 。每次累加梯度分量平方时，用 β 和 $1 - \beta$ 为历史累加值和当前值加权。RMSProp 引入了一个新的超参数 β ，一般情况取 $\beta = 0.9$ 即可。

3.3.5 Adam

Adam 是 adaptive moment estimation 的缩写。Adam 算法结合了冲量法和 RMSProp 的思想。先来看它的伪代码：

```

 $\mathbf{x}^0 \leftarrow$  随机初始化
 $\mathbf{s}^0 \leftarrow \mathbf{0}$ 
 $\mathbf{v}^0 \leftarrow \mathbf{0}$ 
 $t \leftarrow 0$ 
while  $\|\nabla f(\mathbf{x}^t)\| \geq \varepsilon$ :
     $\mathbf{v}^{t+1} \leftarrow \beta^1 \mathbf{v}^t + (1 - \beta^1) \cdot \nabla f(\mathbf{x}^t)$ 
     $\mathbf{s}^{t+1} \leftarrow \beta^2 \mathbf{s}^t + (1 - \beta^2) \cdot \nabla f(\mathbf{x}^t) \otimes \nabla f(\mathbf{x}^t)$ 
     $\mathbf{v}^{t+1} \leftarrow \frac{\mathbf{v}^{t+1}}{1 - (\beta^1)^{t+1}}$ 
     $\mathbf{s}^{t+1} \leftarrow \frac{\mathbf{s}^{t+1}}{1 - (\beta^2)^{t+1}}$ 
     $\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t - \eta \cdot \mathbf{v}^{t+1} \oslash \sqrt{\mathbf{s}^{t+1}} \oplus \epsilon$ 
     $t \leftarrow t + 1$ 
return  $\mathbf{x}^t$ 

```

在每一轮迭代中，Adam 以系数 $0 < \beta^1 < 1$ 计算历史梯度的滑动平均 \mathbf{v}^{t+1} ，以系数 $0 < \beta^2 < 1$ 计算梯度各分量平方的滑动平均 \mathbf{s}^{t+1} 。因为 \mathbf{v}^0 和 \mathbf{s}^0 用零向量初始化，且 β^1 和 β^2 接近 1， \mathbf{v}^{t+1} 和 \mathbf{s}^{t+1} 在迭代初期会偏向零向量，故在迭代初期采用一个小于 1 的值做分母进行调整，使 \mathbf{v}^{t+1} 和 \mathbf{s}^{t+1} 远离零向量，随着迭代进行，分母趋近于 1，修正效果趋向于消失。Adam 用 \mathbf{v}^{t+1} 和 \mathbf{s}^{t+1} 更新自变量。

Adam 的超参数虽多，但默认值就可取得不错的效果，默认取 $\beta^1 = 0.9$ 和 $\beta^2 = 0.99$ 。由于 \mathbf{s}^t 的作用，迭代时的实际步长会发生衰减，所以 η 的取值不像在原始梯度下降中那样重要，一般

$\eta = 0.001$ 。图 3-18 比较了原始梯度下降法和几种变体的效果。注意在本示例的函数上，各种变体不见得优于原始梯度下降。

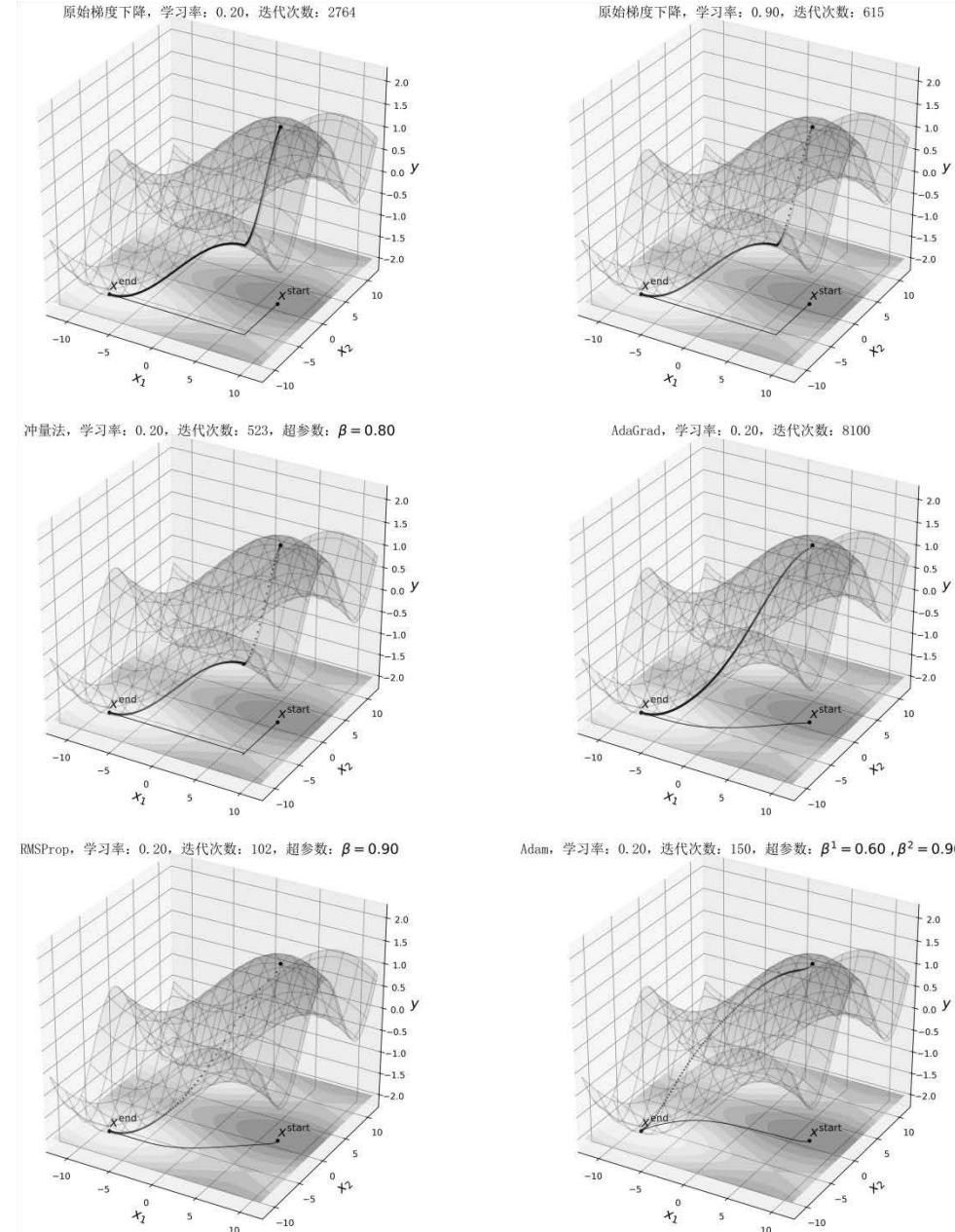


图 3-18 原始梯度下降法与几种变体

3.4 运用梯度下降法训练逻辑回归

运用梯度下降法训练逻辑回归，需要首先计算交叉熵损失对逻辑回归的参数 w_i ($i = 1, \dots, n$)和 b 的梯度。计算梯度需要计算交叉熵损失对各个参数的偏导数。交叉熵损失是每一个训练样本的损失的平均，第*i*个训练样本 $\{\mathbf{x}^i, y^i\}$ 的损失是：

$$\text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i) = -y^i \log \frac{1}{1+e^{-b-\mathbf{w}^T \mathbf{x}^i}} - (1-y^i) \log \frac{1}{1+e^{b+\mathbf{w}^T \mathbf{x}^i}} \quad (3.33)$$

现在计算 $\text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)$ 对 w_j 的偏导数。分两种情况考虑：训练样本属于正类， $y^i = 1$ ；以及训练样本属于负类， $y^i = 0$ 。当 $y^i = 1$ 时：

$$\frac{\partial \text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)}{\partial w_j} = \frac{e^{-b-\mathbf{w}^T \mathbf{x}^i}}{1+e^{-b-\mathbf{w}^T \mathbf{x}^i}} (-x_j^i) = (1-p^i(\mathbf{w}, b))(-x_j^i) = -(y^i - p^i(\mathbf{w}, b))x_j^i \quad (3.34)$$

其中， x_j^i 是 \mathbf{x}^i 的第*j*分量， $p^i(\mathbf{w}, b)$ 是逻辑回归对 \mathbf{x}^i 的预测概率，将其视为关于 \mathbf{w} 和 b 的函数。当 $y^i = 0$ 时：

$$\frac{\partial \text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)}{\partial w_j} = \frac{e^{b+\mathbf{w}^T \mathbf{x}^i}}{1+e^{b+\mathbf{w}^T \mathbf{x}^i}} x_j^i = p^i(\mathbf{w}, b)x_j^i = -(y^i - p^i(\mathbf{w}, b))x_j^i \quad (3.35)$$

喜闻乐见的事情发生了，两种情况统一成一种情况：

$$\frac{\partial \text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)}{\partial w_j} = -(y^i - p^i(\mathbf{w}, b))x_j^i \quad (3.36)$$

在继续之前先观察一下式 (3.36)，括号中的 $y^i - p^i(\mathbf{w}, b)$ 是真实标签 (1 或 0) 与预测概率之差。这个差可以看作模型在样本 $\{\mathbf{x}^i, y^i\}$ 上的误差。更新参数时是加上负梯度，所以 $(y^i - p^i(\mathbf{w}, b))x_j^i$ 会被加到 w_j 上。 $y^i - p^i(\mathbf{w}, b)$ 是真实值与预测概率的差距，这个差距以 x_j^i 为权重分配到 w_j 上。误差分配是看待梯度下降的一个视角，在后文介绍神经网络的反向传播算法时，会看到误差不仅在同一层内部分配，还要在层与层之间分配。

回到梯度计算中来，损失函数 $\text{loss}(\mathbf{w}, b)$ 是对全部训练样本的损失做平均，所以 $\text{loss}(\mathbf{w}, b)$ 对 w_j 的偏导数是每一个 $\text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)$ 对 w_j 的偏导数的平均：

$$\frac{\partial \text{loss}(\mathbf{w}, b)}{\partial w_j} = \frac{1}{M} \sum_{i=1}^M \frac{\partial \text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)}{\partial w_j} = -\frac{1}{M} \sum_{i=1}^M (y^i - p^i(\mathbf{w}, b))x_j^i \quad (3.37)$$

现在考察 $\text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)$ 对 b 的偏导数，类似的计算揭示 $y^i = 1$ 和 $y^i = 0$ 两种情况统一成一个表达式：

$$\frac{\partial \text{loss}(\mathbf{w}, b | \mathbf{x}^i, y^i)}{\partial b} = -(y^i - p^i(\mathbf{w}, b)) \quad (3.38)$$

于是损失函数 $\text{loss}(\mathbf{w}, b)$ 对 b 的偏导数就是：

$$\frac{\partial \text{loss}(\mathbf{w}, b)}{\partial b} = \frac{1}{M} \sum_{i=1}^M \frac{\partial \text{loss}(\mathbf{w}, b | x^i, y^i)}{\partial b} = -\frac{1}{M} \sum_{i=1}^M (y^i - p^i(\mathbf{w}, b)) \quad (3.39)$$

有了各个偏导数就可以计算 $\text{loss}(\mathbf{w}, b)$ 对 w_i ($i = 1, \dots, n$) 和 b 的梯度了：

$$\nabla \text{loss}(\mathbf{w}, b) = -\frac{1}{M} \sum_{i=1}^M (y^i - p^i(\mathbf{w}, b)) \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_n^i \\ 1 \end{pmatrix} \quad (3.40)$$

有了交叉熵损失对模型参数的梯度，就可以应用梯度下降法训练逻辑回归模型了，伪代码如下：

```

 $\mathbf{w}^0 \leftarrow$  随机初始化
 $b^0 \leftarrow$  随机初始化
 $t \leftarrow 0$ 
while  $\|\nabla \text{loss}(\mathbf{w}^t, b^t)\| \geq \varepsilon$ :
     $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \frac{\eta}{M} \sum_{i=1}^M (y^i - p^i(\mathbf{w}^t, b^t)) \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_n^i \\ 1 \end{pmatrix}$ 
     $b^{t+1} \leftarrow b^t + \frac{\eta}{M} \sum_{i=1}^M (y^i - p^i(\mathbf{w}^t, b^t))$ 
     $t \leftarrow t + 1$ 
return  $\mathbf{w}^t, b^t$ 

```

上述训练过程本质上是根据交叉熵损失的梯度更新模型参数，但我们可以抛开损失函数和梯度概念来观察一下 \mathbf{w} 和 b 的更新公式。模型的输出 $p^i(\mathbf{w}, b)$ 是 0 和 1 之间的一个概率值，对于正类训练样本， $y^i = 1$ ，我们希望提高 $p^i(\mathbf{w}, b)$ ，使它更接近 1。更新公式中 $y^i - p^i(\mathbf{w}, b)$ 大于 0，如果 $x_j^i > 0$ ，对 w_j 的更新是加上一个正值；如果 $x_j^i < 0$ ，对 w_j 的更新是加上一个负值。两种情况都是增加 w_j 与 x_j^i 之积，也就是增加 $p^i(\mathbf{w}, b)$ 。对于负类训练样本， $y^i = 0$ ，我们希望降低 $p^i(\mathbf{w}, b)$ ，使它更接近 0。更新公式中 $y^i - p^i(\mathbf{w}, b)$ 小于 0，如果 $x_j^i > 0$ ，对 w_j 的更新是加上一个负值；如果 $x_j^i < 0$ ，对 w_j 的更新是加上一个正值。两种情况都是减小 w_j 与 x_j^i 之积，也就是减小 $p^i(\mathbf{w}, b)$ 。

对 b 的更新不依赖模型输入 \mathbf{x} ，正类训练样本增大 b ，负类训练样本压低 b ，变化的幅度与概率 $p^i(\mathbf{w}, b)$ 和理想值（1 或 0）的差距有关。训练样本上模型输出概率与理想值差距越大，则对 b 的影响越大。

每一个训练样本都将模型参数向对自己来说更理想的方向拉，梯度下降算法取所有训练样本的更新的平均，所有训练样本的“合力”将模型参数拉向能更好地分类训练集的方向。最早的感觉

知机模型的更新规则也是从类似的视角出发，但是每一次更新只取一个训练样本，将模型参数拉向它更理想的方向，之后再取下一个训练样本，如此循环往复。

3.5 梯度下降法训练逻辑回归的 Python 实现

本节，我们用原生 Python 以及 Numpy 库实现逻辑回归模型和梯度下降法及变体，将其用于鸟类生态类群二分类问题。注意，为了容易阅读并能清晰地展现原理，本书代码不做过多的抽象和封装。我们首先实现几个优化器，见如下代码：

```
import numpy as np

class Gradient:
    """
    原始梯度下降
    """
    def __init__(self, learning_rate = 0.001):
        self.learning_rate = learning_rate

    def delta(self, gradient):
        """
        接受当前点的梯度，给出前进向量：学习率乘以梯度反方向
        """
        return -self.learning_rate * gradient

class Decay:
    """
    学习率衰减
    """
    def __init__(self, learning_rate = 0.001, r = 500):
        self.learning_rate = learning_rate
        self.r = r
        self.global_steps = 0

    def delta(self, gradient):
        """
        接受当前点的梯度，给出前进向量。根据学习率衰减公式调整学习率
        """
        eta = self.learning_rate * 10 ** (-self.global_steps / self.r)
        self.global_steps += 1
        return -eta * gradient
```

```
class Momentum:
    """
    冲量梯度下降
    """
    def __init__(self, learning_rate = 0.001, beta = 0.9):
        self.learning_rate = learning_rate
        self.v = None
        self.beta = beta

    def delta(self, gradient):
        """
        接受当前点的梯度，给出前进向量。加入冲量机制
        """
        if self.v is None:
            # 将 v 初始化为与梯度同维数的全零向量
            self.v = np.mat(np.zeros(gradient.shape[0])).T

        self.v = self.beta * self.v - self.learning_rate * gradient
        return self.v


class AdaGrad:
    """
    AdaGrad
    """
    def __init__(self, learning_rate = 0.001):
        self.learning_rate = learning_rate
        self.s = None

    def delta(self, gradient):
        """
        接受当前点的梯度，根据 AdaGrad 算法得到前进向量
        """
        if self.s is None:
            # 将 s 初始化为与梯度同维数的全零向量
            self.s = np.mat(np.zeros(gradient.shape[0])).T

        self.s = self.s + np.power(gradient, 2)
        return -self.learning_rate * gradient / np.sqrt(self.s + 1e-10)


class RMSProp:
    """
    RMSProp
    """
```

```

def __init__(self, learning_rate = 0.001, beta = 0.9):
    self.learning_rate = learning_rate
    self.s = None
    self.beta = beta

def delta(self, gradient):
    """
    接受当前点的梯度，根据 RMSProp 算法得到前进向量
    """
    if self.s is None:
        # 将 s 初始化为与梯度同维数的全零向量
        self.s = np.mat(np.zeros(gradient.shape[0])).T

    self.s = self.beta * self.s + (1 - self.beta) * np.power(gradient, 2)
    return -self.learning_rate * gradient / np.sqrt(self.s + 1e-10)

class Adam:
    """
    Adam
    """
    def __init__(self, learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.99):
        self.learning_rate = learning_rate
        self.s = None
        self.v = None
        self.beta_1 = beta_1
        self.beta_2 = beta_2
        self.global_steps = 0

    def delta(self, gradient):
        """
        接受当前点的梯度，根据 Adam 算法得到前进向量
        """
        if self.s is None or self.v is None:
            # 将 s 初始化为与梯度同维数的全零向量
            self.s = np.mat(np.zeros(gradient.shape[0])).T
            # 将 v 初始化为与梯度同维数的全零向量
            self.v = np.mat(np.zeros(gradient.shape[0])).T

        self.v = self.beta_1 * self.v + (1 - self.beta_1) * gradient
        self.s = self.beta_2 * self.s + (1 - self.beta_2) * np.power(gradient, 2)

        self.v = self.v / (1 - self.beta_1 ** (self.global_steps + 1))
        self.s = self.s / (1 - self.beta_2 ** (self.global_steps + 1))
        self.global_steps += 1

        return -self.learning_rate * self.v / np.sqrt(self.s + 1e-10)

```

我们将原始梯度下降法、学习率衰减、冲量法、AdaGrad、RMSProp 和 Adam 封装成类，构造时传入相关超参数。这些类的 `delta` 方法接受当前梯度向量，返回更新向量。算法需要保存的状态都放在类的成员变量中。接下来我们实现逻辑回归，见如下代码：

```
from optimizer import *
import numpy as np

class LogisticRegression:

    def __init__(self, optimizer, iterations = 100000):
        assert(optimizer is not None)

        self.optimizer = optimizer
        self.iterations = iterations # 迭代次数

    def train(self, x, y):
        """
        x 为矩阵, 形状是 n_samples * n_features , 每一行是一个样本
        y 为矩阵, 形状是 n_samples * 1, 元素为样本的标签, 正类为 1, 负类为 0
        """

        # 在 x 最前面添加一列常数 1, 作为偏置值的输入, 以简化公式
        x = np.mat(np.c_[[1.0] * x.shape[0], x])

        # 根据 x 的列数 (特征数) 随机初始化权值, 此时偏置值纳入了权值向量, 相当于第一个权值
        # 权值向量为 n_features + 1 维向量, 每个分量以 0 均值、0.01 标准差的正态分布初始化
        self.weights = np.mat(np.random.normal(0, 0.01, size=x.shape[1])).T

        for i in range(self.iterations):

            # 计算当前模型对训练集样本的输出
            p = self.predict(x, False)

            gradient = -x.T * (y - p) / x.shape[0] # 交叉熵损失对模型参数的梯度
            self.weights += self.optimizer.delta(gradient) # 更新模型参数

            # 评估当前模型并打印训练信息
            if i % 100 == 0:
                # 交叉熵损失
                cross_entropy = (-y.T * np.log(p) - (1.0 - y).T * np.log(1 - p)) / y.shape[0]

                # 正确率
                accuracy = np.sum(((p > 0.5).astype(np.int) == y).
```

```

        astype(np.int)) / y.shape[0]

    print("迭代: {:d}, 交叉熵: {:.6f}, 正确率: {:.2f}%".format(
        i + 1, cross_entropy[0, 0], accuracy * 100))

def predict(self, x, augment = True):
    """
    预测函数。x 为矩阵, 形状是 n_samples * n_features, 每一行是一个样本
    augment 参数指示是否要在特征矩阵前添加一列常量 1
    """
    # 在 x 最前面添加一列常数 1, 作为偏置值的输入
    if augment:
        x = np.mat(np.c_[[1.0] * x.shape[0], x])

    a = -np.matmul(x, self.weights)
    a[a > 1e2] = 1e2 # 防止数值过大
    p = 1.0 / (1.0 + np.power(np.e, a))

    # 剪裁概率值, 保证其为合法的概率值
    p[p >= 1.0] = 1.0 - 1e-10
    p[p <= 0.0] = 1e-10

    return p

```

我们将逻辑回归算法封装成一个类 `LogisticRegression`, 它的构造函数接受一个 `optimizer` 对象和迭代次数。每次迭代, 计算训练集全体样本的平均交叉熵对模型参数的梯度, 之后调用 `optimizer` 对象的 `delta` 方法, 得到更新向量并更新参数。注意, 我们给特征矩阵添加一列, 列中所有元素都是常量 1。这相当于给每个样本添加一个永远为 1 的“特征”, 将该“特征”作为偏置值的输入。这样做可以将偏置值纳入权值向量, 简化公式。接下来我们将这个逻辑回归模型应用于鸟类生态类群问题, 代码如下:

```

import pandas as pd
import numpy as np
from optimizer import *
from lr import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

# 读入数据
bird = pd.read_csv("bird.csv").dropna().drop("id", axis=1)

# 根据标签是否属于 "SW", "W", "R" 三类, 构造二分类 1/0 标签
bird["type"] = bird.type.apply(lambda t: t in ["SW", "W", "R"]).astype(np.int)

```

```

data = bird.values
# 将样本随机洗牌
np.random.shuffle(data)

# 前 300 个样本作为训练集
train_x = np.mat(data[:300,:-1])
train_y = np.mat(data[:300,-1]).T

# 其余样本作为测试集
test_x = np.mat(data[300:,:-1])
test_y = np.mat(data[300:,-1]).T

# 构造逻辑回归对象，优化器为 Adam，各超参数取默认值
lr = LogisticRegression(Adam())

# 在训练集上训练
lr.train(train_x, train_y)

# 对测试集进行预测
p = lr.predict(test_x) # 模型预测的正类概率
pred = (p > 0.5).astype(np.int) # 以 0.5 为阈值时，模型预测的类别

print("正确率: {:.2f}%, 查准率: {:.2f}%, 查全率: {:.2f}%, ROC 曲线下面积: {:.3f}".format(
    accuracy_score(test_y, pred) * 100,
    precision_score(test_y, pred) * 100,
    recall_score(test_y, pred) * 100,
    roc_auc_score(test_y.A, p)))

```

首先使用 pandas 库读入数据集 bird.csv 文件，去掉含有空值的行，再去掉名为“id”的列。之后，根据 type 列的值是否属于大型鸟而取 1 或 0，并重新赋值 type 列。将特征和标签当作 numpy 数组取出，随机扰乱行的顺序。取前 300 行作为训练集，后 113 行作为测试集。

构造 LogisticRegression 对象，传给构造函数一个 Adam 对象，超参数都取默认值，迭代数量也取默认值。调用 LogisticRegression 对象的 train 方法，参数是训练集的特征矩阵和标签向量。训练过程中，每 100 次迭代，打印当前模型在训练集上的交叉熵和正确率。若运行该代码，可以看到随着训练进行，训练集上的交叉熵在下降而正确率在上升。

训练完成后，用 LogisticRegression 对象对测试集样本做预测，输出预测为正类的概率。以 0.5 为概率阈值，得到模型对测试集预测的类别。之后，调用 scikit-learn 库的 accuracy_score、precision_score、recall_score 以及 roc_auc_score 函数计算模型在测试集上的正确率、查准率、

查全率以及 ROC 曲线下的面积。读者可以自行试验这份代码，替换不同的 `optimizer` 并观察模型的表现。

3.6 小结

梯度包含了多元函数的局部一阶近似信息。根据梯度可以得到函数沿自变量空间中任意方向的变化率。函数在某一点梯度决定了函数在该点的切平面的法向量。当自变量为 2 维时，我们可以直观地从函数图像中看出，梯度指向的方向是该点切平面的上坡方向。函数沿梯度方向具有最大的、正的方向导数。梯度的反方向是该点切平面的下坡方向，函数沿梯度反方向具有最小的、负的方向导数。梯度反方向是函数值下降最快的方向，在每一次迭代中，沿着当前点的梯度反方向运动一个距离，这就是梯度下降法。

仿射函数是一次函数，没有二次或高次项，其图像是“平直”的——直线、平面和超平面。线性近似是函数在局部的最粗糙的近似，当自变量远离当前点，基于当前点梯度的线性近似有可能与原函数大相径庭，所以梯度下降法是一种缺乏远见的贪心算法。如果用更高次的函数——例如二次函数——在局部拟合原函数，则可以得到对原函数的更精确的近似，以及关于函数局部形态的更丰富的信息，这称为二阶近似。下一章我们将介绍基于函数局部二阶信息的优化算法。



微信连接



回复“深度学习”查看相关书单



微博连接

关注[@图灵教育](#) 每日分享IT好书



QQ连接

图灵读者官方群I: 218139230

图灵读者官方群II: 164939616

图灵社区 iTuring.cn

在线出版,电子书,《码农》杂志,图灵访谈

机器学习有着艰深的理论背景，不能透彻理解原理，就难以在实践中自由地运用。当广大工程师和学生试图进入机器学习领域时，数学原理是横亘在面前的一座大山。本书就将数学原理、编程实现与实际应用紧密结合起来，为读者引导一条翻越这座大山的通途。

首先，本书以神经网络为线索，沿着从线性模型到深度学习的路线，串起全部核心知识点。我们在必要和恰当的时机引入相关数学基础内容，具有理工科背景的读者能够容易地回忆起相关知识，而不需要再回头求诸于大部头教科书。

其次，除了对原理的讲解，本书也包含编程实现。我们基于Python和Numpy库实现了多种训练算法、二分类/多分类逻辑回归模型以及多层次全连接神经网络。另外，本书还实现了一个简单的计算图框架，并展示如何使用该框架搭建和训练多种结构各异的神经网络。

如果你是下面的某一类读者，那么本书就是为你准备的：

- **渴望进入机器学习，特别是神经网络/深度学习领域的高年级本科生和研究生**，本书帮助你将数学基础知识与机器学习和神经网络结合起来，透彻理解其原理和实现；
- **希望了解神经网络与机器学习的广大程序员和工程师**，你可能需要花更多力气回忆数学知识，本书包含了所需要的全部知识点；
- **对机器学习模型的编程实现感兴趣的读者**，本书包含逻辑回归、多层次全连接神经网络以及计算图框架的Python实现；
- **对工业界的机器学习、数据科学和数据挖掘工程师**，本书关于模型原理的高级主题，特别是关于模型自由度与偏置-方差权衡方面的内容，可为你提供一些深刻而有趣的洞见。

图灵社区：iTuring.cn

热线：(010)51095183 转 600

分类建议 计算机 / 人工智能 / 神经网络

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-51723-4



9 787115 517234 >

ISBN 978-7-115-51723-4

定价：89.00元

欢迎加入

图灵社区

最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。

ituring.com.cn