第一章

最优化简介

最优化问题 (也称优化问题) 泛指定量决策问题, 主要关心如何对有限资源进行有效分配和控制, 并达到某种意义上的最优. 它通常需要对需求进行定性和定量分析, 建立恰当的数学模型来描述该问题, 设计合适的计算方法来寻找问题的最优解, 探索研究模型和算法的理论性质, 考察算法的计算性能等. 由于很多数学问题难以直接给出显式解, 最优化模型就成为人们最常见的选择, 计算机的高速发展也为最优化方法提供了有力辅助工具. 因此最优化方法被广泛应用于科学与工程计算、金融与经济、管理科学、工业生产、图像与信号处理、数据分析与人工智能、计算物理与化学等众多领域.

本章将介绍最优化问题的一般形式和一些重要的基本概念, 并通过实际应用中的例子让读者更加直观地理解最优化问题.

1.1　最优化问题概括

1.1.1　最优化问题的一般形式

最优化问题一般可以描述为

$$\begin{aligned} \min \quad & f(x), \\ \text{s.t.} \quad & x \in \mathcal{X}, \end{aligned} \tag{1.1.1}$$

其中 $x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$ 是**决策变量**, $f\colon \mathbb{R}^n \to \mathbb{R}$ 是**目标函数**, $\mathcal{X} \subseteq \mathbb{R}^n$ 是**约束集合**或**可行域**, 可行域包含的点称为**可行解**或**可行点**. 记号 s.t. 是 “subject to” 的缩写, 专指约束条件. 当 $\mathcal{X} = \mathbb{R}^n$ 时, 问题 (1.1.1) 称为无约束优化问题. 集合 $\mathcal{X}$ 通常可以由约束函数 $c_i(x)\colon \mathbb{R}^n \to \mathbb{R}, i = 1, 2, \cdots, m + l$ 表达为如下具体形式:

$$\begin{aligned} \mathcal{X} = \{x \in \mathbb{R}^n \mid\ & c_i(x) \leqslant 0, \quad i = 1, 2, \cdots, m, \\ & c_i(x) = 0, \quad i = m + 1, m + 2, \cdots, m + l\}. \end{aligned}$$

在所有满足约束条件的决策变量中, 使目标函数取最小值的变量 x^* 称为优化问题 (1.1.1) 的最优解, 即对任意 $x \in \mathcal{X}$ 都有 $f(x) \geqslant f(x^*)$. 如果我们求解在约束集合 $\mathcal{X}$ 上目标函数 $f(x)$ 的最大值, 则问题 (1.1.1) 的 “min” 应相应地替换为 “max”. 注意到在集合 $\mathcal{X}$ 上, 函数 f 的最小 (最大) 值不一定存在, 但是其下 (上) 确界 “inf f(sup f)” 总是存在的. 因此, 当目标函数的最小 (最大) 值不存在时, 我们便关心其下 (上) 确界, 即将问题 (1.1.1) 中的 “min(max)” 改为 “inf(sup)”. 为了叙述简便, 问题 (1.1.1) 中 x 为 $\mathbb{R}^n$ 空间中的向量. 实际上, 根据具体应用和需求, x 还可以是矩阵、多维数组或张量等, 本书介绍的很多理论和算法可以相应推广.

由于本书涉及较多公式, 请读者根据上下文区分公式中的标量、向量、矩阵. 在不加说明的情况下, 向量一般用小写英文字母或希腊字母表示, 矩阵一般用大写英文字母或希腊字母表示. 公式中的标量可能使用多种记号, 需要根据上下文确定. 读者也可参考书后符号表.

1.1.2 最优化问题的类型与应用背景

最优化问题 (1.1.1) 的具体形式非常丰富, 我们可以按照目标函数、约束函数以及解的性质将其分类. 按照目标函数和约束函数的形式来分: 当目标函数和约束函数均为线性函数时, 问题 (1.1.1) 称为线性规划; 当目标函数和约束函数中至少有一个为非线性函数时, 相应的问题称为非线性规划; 若目标函数是二次函数而约束函数是线性函数则称为二次规划; 包含非光滑函数的问题称为非光滑优化; 不能直接求导数的问题称为无导数优化; 变量只能取整数的问题称为整数规划; 在线性约束下极小化关于半正定矩阵的线性函数的问题称为半定规划, 其广义形式为锥规划. 按照最优解的性质来分: 最优解只有少量非零元素的问题称为稀疏优化; 最优解是低秩矩阵的问题称为低秩矩阵优化. 此外还有几何优化、二次锥规划、张量优化、鲁棒优化、全局优化、组合优化、网络规划、随机优化、动态规划、带微分方程约束优化、微分流形约束优化、分布式优化等. 就具体应用而言, 问题 (1.1.1) 可涵盖统计学习、压缩感知、最优运输、信号处理、图像处理、机器学习、强化学习、模式识别、金融工程、电力系统等领域的优化模型.

需要指出的是, 数学建模很容易给出应用问题不同的模型, 可以对应性质很不相同的问题, 其求解难度和需要的算法也将差别很大. 在投资组合优化中, 人们希望通过寻求最优的投资组合以降低风险、提高收益. 这时决策变量 x_i 表示在第 i 项资产上的投资额, 向量 $x \in \mathbb{R}^n$ 表示整体的投资分配. 约束条件可能为总资金数、每项资产的最大 (最小) 投资额、最低收益等. 目标函数通常是某种风险度量. 若是极小化收益的方差, 则该问题是典型的二次规划; 若极小化风险价值 (value at risk) 函数, 则该问题是混合整数规划; 若极小化条件风险价值 (conditional value at risk) 函数, 则该问题是非光滑优化, 也可以进一步化成线性规划.

在本章后面的三节中, 我们通过一些实际应用中的例子更直观、深入地理解最优化问题. 由于篇幅限制, 我们通常只简要给出它们的一些典型形式, 而且叙述并不严格, 主要是提供这些应用的大致形式, 详细的定义和描述请读者参考本书后面的章节或者相关参考文献. 本书的目标之一是使得读者通过学习本书的理论和算法, 能用算法软件包来求解这些模型, 并了解这些算法有哪些优缺点, 更进一步地, 使得读者能独立设计类似问题的算法.

1.2　实例: 稀疏优化

考虑线性方程组求解问题:

$$Ax = b, \tag{1.2.1}$$

其中向量 $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, 矩阵 $A \in \mathbb{R}^{m\times n}$, 且向量 b 的维数远小于向量 x 的维数, 即 $m \ll n$. 在自然科学和工程中常常遇到已知向量 b 和矩阵 A, 想要重构向量 x 的问题. 例如在信号传输过程中, 希望通过接收到长度为 m 的数字信号精确地重构原始信号. 注意到由于 $m \ll n$, 方程组 (1.2.1) 是欠定的, 因此存在无穷多个解, 重构出原始信号看似很难. 所幸的是, 这些解当中大部分是我们不感兴趣的, 真正有用的解是所谓的 “稀疏解”, 即原始信号中有较多的零元素. 如果加上稀疏性这一先验信息, 且矩阵 A 以及原问题的解 u 满足某些条件, 那么我们可以通过求解稀疏优化问题把 u 与方程组 (1.2.1) 的其他解区别开. 这类技术广泛应用于压缩感知 (compressive sensing), 即通过部分信息恢复全部信息的解决方案.

先来看一个具体的例子. 在 MATLAB 环境里构造 A, u 和 b:

```
m = 128; n = 256;
A = randn(m, n);
u = sprandn(n, 1, 0.1);
b = A * u;
```

在这个例子中, 我们构造了一个 128×256 矩阵 A, 它的每个元素都服从高斯 (Gauss) 随机分布. 精确解 u 只有 10% 的元素非零, 每一个非零元素也服从高斯分布. 这些特征可以在理论上保证 u 是方程组 (1.2.1) 唯一的非零元素最少的解, 即 u 是如下 ℓ_0 范数[①]问题的最优解:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|x\|_0, \\ \text{s.t.} \quad & Ax = b. \end{aligned} \tag{1.2.2}$$

其中 $\|x\|_0$ 是指 x 中非零元素的个数. 由于 $\|x\|_0$ 是不连续的函数, 且取值只可能是整数, 问题 (1.2.2) 实际上是 NP(non-deterministic polynomial) 难的, 求解起来非常困难. 因此当 n 较大时通过直接求解问题 (1.2.2) 来恢复出原始信号 u 是行不通的. 那有没有替代的方法呢? 答案是有的. 若定义 ℓ_1 范数: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$, 并将其替换到问题 (1.2.2) 当中, 我们得到了另一个形式上非常相似的问题 (又称 ℓ_1 范数优化问题, 基追踪问题):

① 实际上, ℓ_0 范数不是一个范数, 这里为了叙述统一而采用了这个术语, 读者应当注意这个区别.

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|x\|_1, \\ \text{s.t.} \quad & Ax = b. \end{aligned} \tag{1.2.3}$$

令人惊讶的是, 可以从理论上证明: 若 A, b 满足一定的条件 (例如使用前面随机产生的 A 和 b), 向量 u 也是 ℓ_1 范数优化问题 (1.2.3) 的唯一最优解. 这一发现的重要之处在于, 虽然问题 (1.2.3) 仍没有显式解, 但与问题 (1.2.2) 相比难度已经大大降低. 前面我们提到 ℓ_0 范数优化问题是 NP 难问题, 但 ℓ_1 范数优化问题的解可以非常容易地通过现有优化算法得到! 从这个例子不难发现, 优化学科的研究能够极大程度上帮助我们攻克现有的困难问题. 既然有如上令人兴奋的结果, 我们是否能使用其他更容易求解的范数替代 ℓ_0 范数呢? 事实并非如此. 如果简单地把 ℓ_1 范数修改为 ℓ_2 范数: $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$, 即求解如下优化问题:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|x\|_2, \\ \text{s.t.} \quad & Ax = b. \end{aligned} \tag{1.2.4}$$

几何学的知识表明, 问题 (1.2.4) 实际上就是原点到仿射集 $Ax = b$ 的投影, 我们可以直接写出它的显式表达式. 但遗憾的是, u 并不是问题 (1.2.4) 的解. 事实上, 图 1.1(a)—(c) 分别给出了一组随机数据下的 u, 以及问题 (1.2.3) 和问题 (1.2.4) 的数值解. 可以看出图 1.1(a) 和 (b) 是完全一样的, 而 (c) 则与 u 相去甚远, 虽然隐约能看出数据点的大致趋势, 但已经不可分辨非零元素的具体位置.

为什么会出现这种情况呢? 这要追溯到 ℓ_0, ℓ_1, ℓ_2 范数的性质. 下面用图示的方式来直观说明为什么 ℓ_1 范数优化问题的解具有稀疏性而 ℓ_2 范数优化问题的解不具有该性质. 为了方便起见, 我们在二维空间上讨论求解欠定方程组 $Ax = b$, 此时 $Ax = b$ 是一条直线. 在几何上, 三种优化问题实际上要找到最小的 C, 使得 "范数球" $\{x|\|x\| \leqslant C\}$($\|\cdot\|$ 表示任何一种范数) 恰好与 $Ax = b$ 相交. 而图 1.2 分别展示了三种范数球的几何直观: 对 ℓ_0 范数, 当 $C = 2$ 时 $\{x|\|x\|_0 \leqslant C\}$ 是全平面, 它自然与 $Ax = b$ 相交, 而当 $C = 1$ 时退化成两条直线 (坐标轴), 此时问题的解是 $Ax = b$ 和这两条直线的交点; 对 ℓ_1 范数, 根据 C 不同 $\{x|\|x\|_1 \leqslant C\}$ 为一系列正方形, 这些正方形的顶点恰好都在坐标轴上, 而最小的 C 对应的正方形和直线 $Ax = b$ 的交点一般都是顶点, 因此 ℓ_1 范数的解有稀疏性; 对 ℓ_2 范数, 当 C 取值不同时 $\{x|\|x\|_2 \leqslant C\}$ 为一系列圆, 而圆有光滑的边界, 它和直线 $Ax = b$ 的切点可以是圆周上的任何一点, 所以 ℓ_2 范数优化问题一般不能保证解的稀疏性.

问题 (1.2.3) 的理论和算法研究在 2006 年左右带来了革命性的影响. 理论上研究的课题包括什么条件下问题 (1.2.3) 的解具有稀疏性、如何改进这些条件、如何推广这些条件到其他应用. 常见的数据矩阵 A 一般由离散余弦变换、小波变换、傅里叶 (Fourier) 变换等生成. 虽然这些矩阵本身并没有稀疏性, 但通常具有很好的分析性质, 保证稀疏

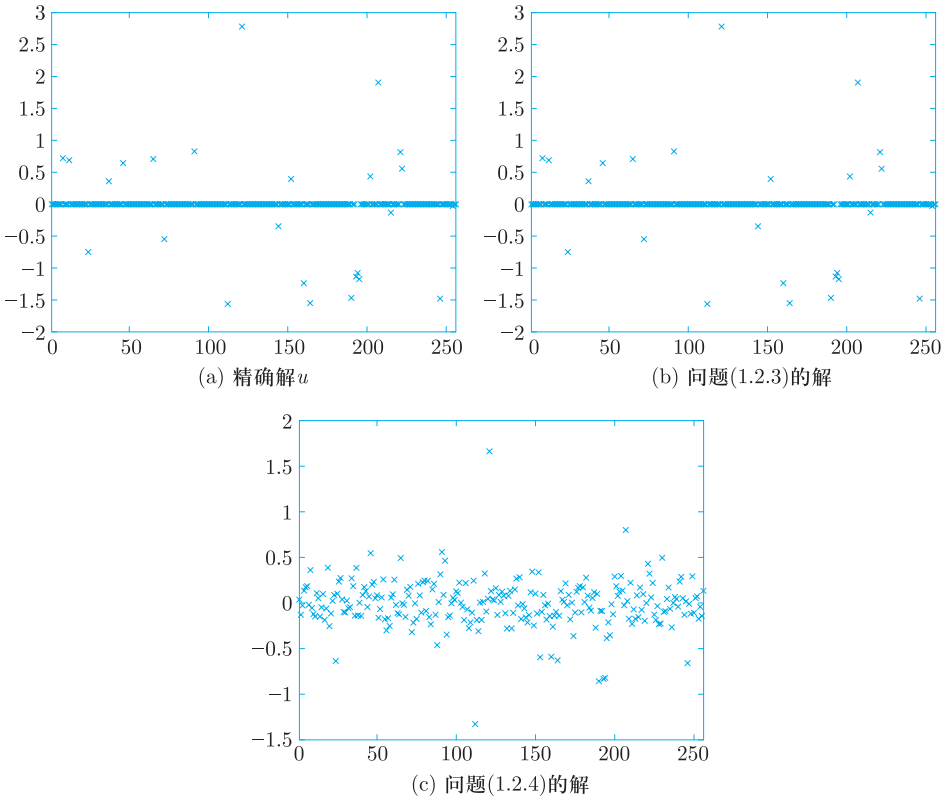


图 1.1　稀疏优化的例子

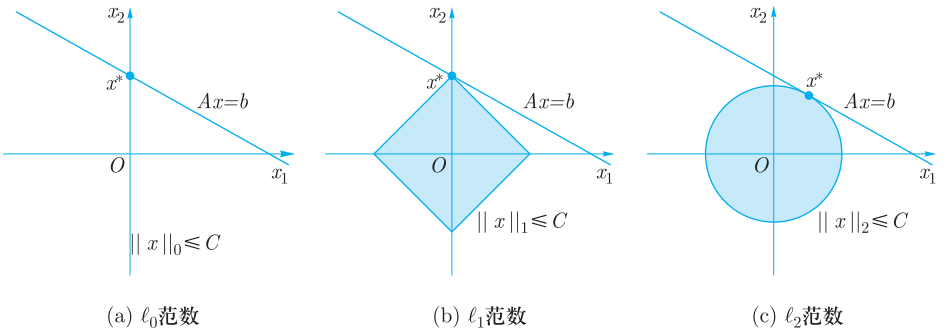


图 1.2　三种范数优化问题求解示意图

解的存在性. 注意到绝对值函数在零点处不可微, 问题 (1.2.3) 是非光滑优化问题. 虽然它可以等价于线性规划问题, 但是数据矩阵 A 通常是稠密矩阵, 甚至 A 的元素未知或者不能直接存储, 只能提供 Ax 或 $A^{\mathrm{T}}y$ 等运算结果. 在这些特殊情况下, 线性规划经典的单纯形法和内点法通常不太适用于求解大规模的问题 (1.2.3). 本书的一个主要目的就是根据这些问题的特点设计合适的算法进行求解. 需要强调的是, 问题 (1.2.3) 主要特点是其最优解是稀疏向量, 它是稀疏优化的一种典型形式.

本书还将考虑带 ℓ_1 范数正则项的优化问题

$$\min_{x\in\mathbb{R}^n} \quad \mu\|x\|_1+\frac{1}{2}\|Ax-b\|_2^2, \tag{1.2.5}$$

其中 $\mu>0$ 是给定的正则化参数. 问题 (1.2.5) 又称为 LASSO(least absolute shrinkage and selection operator), 该问题可以看成是问题 (1.2.3) 的二次罚函数形式. 由于它是无约束优化问题, 形式上看起来比问题 (1.2.3) 简单. 本书大部分数值算法都将针对问题 (1.2.3) 或问题 (1.2.5) 给出具体形式. 因此全面掌握它们的求解方法是掌握基本最优化算法的一个标志.

1.3 实例: 低秩矩阵恢复

某视频网站提供了约 48 万用户对 1.7 万部电影的上亿条评级数据, 希望对用户的电影评级进行预测, 从而改进用户电影推荐系统, 为每个用户更有针对性地推荐影片.

显然每一个用户不可能看过所有的电影, 每一部电影也不可能收集到全部用户的评级. 电影评级由用户打分 1 星到 5 星表示, 记为取值 1~5 的整数. 我们将电影评级放在一个矩阵 M 中, 矩阵 M 的每一行表示不同用户, 每一列表示不同电影. 由于用户只对看过的电影给出自己的评价, 矩阵 M 中很多元素是未知的. 图 1.3 给出了用户电影评级矩阵 M 的一个简单示例. 令 Ω 是矩阵 M 中所有已知评级元素的下标的集合, 则该问题可以初步描述为构造一个矩阵 X, 使得在给定位置的元素等于已知评级元素, 即满足 $X_{ij}=M_{ij},\ (i,j)\in\Omega$. 不难看出满足这个条件的矩阵 X 有无穷多个, 那么如何得到一个真正有价值的 X 呢? 这就需要分析 X 应该具有什么样的结构. 类型相似的电影获得的评分往往是类似的, 这意味着这些电影在矩阵 M 中所对应的列也是相似的, 因此矩阵 M 的列可能是亏秩的; 同样地, 相似人群对不同电影的评分也可能是相似的, 它们在矩阵 M 中所对应的行也是相似的, 因此矩阵 M 的行也可能是亏秩的. 因此寻找一个低秩矩阵 X 可能给出很好的解. 令 $\mathrm{rank}(X)$ 为矩阵 X 的秩, 该问题可以表达为

$$\begin{aligned}\min_{X\in\mathbb{R}^{m\times n}} \quad & \mathrm{rank}(X),\\ \text{s.t.} \quad & X_{ij}=M_{ij},\ (i,j)\in\Omega.\end{aligned} \tag{1.3.1}$$

这类问题称为低秩矩阵恢复 (low rank matrix completion). 其约束条件保证了构造的低秩矩阵 X 与 M 中的所有已知元素完全相同. 但是极小化矩阵的秩是 NP 难的问题, 如何将其化成一个容易求解的问题呢? 这里仍然沿用稀疏优化的思想. 在稀疏优化问题中, 我们将 ℓ_0 范数换成了 ℓ_1 范数. 而 $\mathrm{rank}(X)$ 正好是矩阵 X 所有非零奇异值的个数, 根据稀疏优化的思想, 我们将其更换成所有非零奇异值 σ_i 的和, 即矩阵 X 的核范

数 (nuclear norm): $\|X\|_* = \sum_i \sigma_i(X)$. 因此问题 (1.3.1) 就变成

$$\begin{aligned} \min_{X\in\mathbb{R}^{m\times n}} \quad & \|X\|_*, \\ \text{s.t.} \quad & X_{ij} = M_{ij},\ (i,j)\in\Omega. \end{aligned} \tag{1.3.2}$$

	电影1	电影2	电影3	电影4	⋯	电影n
用户1	4	?	?	3	⋯	?
用户2	?	2	4	?	⋯	?
用户3	3	?	?	?	⋯	?
用户4	2	?	5	?	⋯	?
⋮	⋮	⋮	⋮	⋮		⋮
用户m	?	3	?	4	⋯	?

图 1.3　用户电影评级矩阵 M 示例

可以证明问题 (1.3.2) 是一个凸优化问题, 并且在一定条件下它与问题 (1.3.1) 等价. 也可以将问题 (1.3.2) 转换为一个半定规划问题, 但是目前半定规划算法所能有效求解的问题规模限制了这种技术的实际应用. 同样地, 考虑到观测可能出现误差, 对于给定的参数 $\mu > 0$, 我们也写出该问题的二次罚函数形式:

$$\min_{X\in\mathbb{R}^{m\times n}} \quad \mu\|X\|_* + \frac{1}{2}\sum_{(i,j)\in\Omega}(X_{ij} - M_{ij})^2. \tag{1.3.3}$$

类似于稀疏优化问题 (1.2.3) 和 (1.2.5), 本书大部分数值算法都可以针对问题 (1.3.2) 或问题 (1.3.3) 给出具体形式.

1.4　实例: 深度学习

深度学习 (deep learning) 的起源可以追溯至 20 世纪 40 年代, 其雏形出现在控制论中. 近十年来深度学习又重新走入了人们的视野, 深度学习问题和算法的研究也经历了一次新的浪潮. 虽然卷积网络的设计受到了生物学和神经科学的启发, 但深度学习目前的发展早已超越了机器学习模型中的神经科学观点. 它用相对简单的函数来表达复杂的表示, 从低层特征概括到更加抽象的高层特征, 让计算机从经验中挖掘隐含的信息和价值. 本节我们将通过介绍多层感知机和卷积神经网络来了解优化模型在深度学习中的应用.

1.4.1　多层感知机

多层感知机 (multi-layer perceptron, MLP) 也叫作深度前馈网络 (deep feedforward network) 或前馈神经网络 (feedforward neural network), 它通过已有的信息或者知识来

对未知事物进行预测. 在神经网络中, 已知的信息通常用数据集来表示. 数据集一般分为训练集和测试集: 训练集用来训练神经网络, 从而使得神经网络能够掌握训练集上的信息; 测试集用来测试训练完的神经网络的预测准确性. 一个常见的任务是分类问题. 假设我们有一个猫和狗的图片集, 将其划分成训练集和测试集 (保证集合中猫和狗图片要有一定的比例). 神经网络是想逼近一个从图片到 $\{0,1\}$ 的函数, 这里 0 表示猫, 1 表示狗. 因为神经网络本身的结构和大量的训练集信息, 训练得到的函数与真实结果具有非常高的吻合性.

具体地, 给定训练集 $D = \{\{a_1, b_1\}, \{a_2, b_2\}, \cdots, \{a_m, b_m\}\}$, 假设数据 $a_i \in \mathbb{R}^p$, $b_i \in \mathbb{R}^q$. 为了方便处理模型里的偏差项, 还假设 a_i 的第一个元素等于 1, 即 $a_{i1} = 1$. 图 1.4 给出了一种由 p 个输入单元和 q 个输出单元构成的 $(L+2)$ 层感知机, 其含有一个输入层, 一个输出层和 L 个隐藏层. 该感知机的第 l 个隐藏层共有 $m^{(l)}$ 个神经元, 为了方便我们用 $l = 0$ 表示输入层, $l = L+1$ 表示输出层, 并定义 $m^{(0)} = p$ 和 $m^{(L+1)} = q$. 设 $y^{(l)} \in \mathbb{R}^{m^{(l)}}$ 为第 l 层的所有神经元, 同样地, 为了能够处理每一个隐藏层的信号偏差, 除输出层外, 我们令 $y^{(l)}$ 的第一个元素等于 1, 即 $y_1^{(l)} = 1, 0 \leqslant l \leqslant L$, 而其余的元素则是通过上一层的神经元的值进行加权求和得到. 令参数 $x = (x^{(1)}, x^{(2)}, \cdots, x^{(L+1)})$ 表示网络中所有层之间的权重, 其中 $x_{i,k}^{(l)}$ 是第 $(l-1)$ 隐藏层的第 k 个单元连接到第 l 隐藏层的第 i 个单元对应的权重, 则在第 l 隐藏层中, 第 i 个单元 ($i > 1$, 当 $l = L+1$ 时可取为 $i \geqslant 1$) 计算输出信息 $y_i^{(l)}$ 为

$$y_i^{(l)} = t(z_i^{(l)}), \quad z_i^{(l)} = \sum_{k=1}^{m^{(l-1)}} x_{i,k}^{(l)} y_k^{(l-1)}. \tag{1.4.1}$$

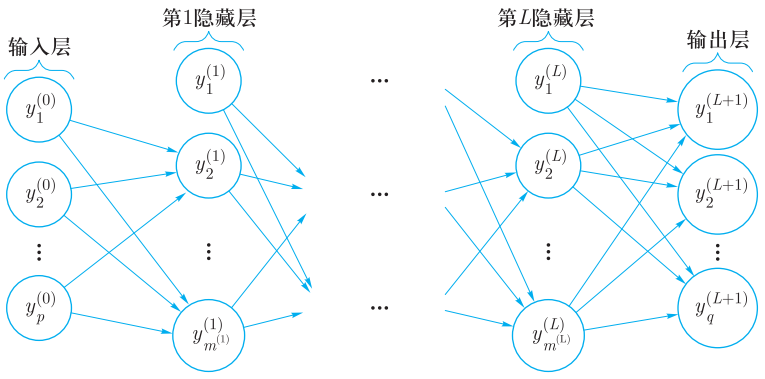


图 1.4 带 p 个输入单元和 q 个输出单元的 $(L+2)$ 层感知机的网络图, 第 l 个隐藏层包含 $m^{(l)}$ 个神经元

这里函数 $t(\cdot)$ 称为激活函数, 常见的类型有 Sigmoid 函数

$$t(z) = \frac{1}{1+\exp(-z)},$$

Heaviside 函数

$$t(z)=\begin{cases}1, & z\geqslant 0,\\ 0, & z<0,\end{cases}$$

以及 ReLU 函数

$$t(z)=\max\{0,z\}. \tag{1.4.2}$$

整个过程可以描述为

$$y^{(0)}\overset{x^{(1)}}{\to} z^{(1)}\overset{t}{\to} y^{(1)}\overset{x^{(2)}}{\to}\cdots\overset{t}{\to} y^{(L+1)}.$$

容易看出, 多层感知机的每一层输出实际就是由其上一层的数值作线性组合再逐分量作非线性变换得到的. 若将 $y^{(0)}$ 视为自变量, $y^{(L+1)}$ 视为因变量, 则多层感知机实际上定义了一个以 x 为参数的函数 $h(a;x):\mathbb{R}^p\to\mathbb{R}^q$, 这里 a 为输入层 $y^{(0)}$ 的取值. 当输入数据为 a_i 时, 其输出 $h(a_i;x)$ 将作为真实标签 b_i 的估计. 若选择平方误差为损失函数, 则我们得到多层感知机的优化模型:

$$\min_x \quad \sum_{i=1}^m \|h(a_i;x)-b_i\|_2^2+\lambda r(x), \tag{1.4.3}$$

其中 $r(x)$ 是正则项, 用来刻画解的某些性质, 如光滑性或稀疏性等; λ 称为正则化参数, 用来平衡模型的拟合程度和解的性质. 如果 λ 太小, 那么对解的性质没有起到改善作用; 如果 λ 太大, 那么模型与原问题相差很大, 可能是一个糟糕的逼近.

1.4.2　卷积神经网络

卷积神经网络 (convolutional neural network, CNN) 是一种深度前馈人工神经网络, 专门用来处理如时间序列数据或是图像等网格数据. CNN 在计算机视觉、视频分析、自然语言处理等诸多领域有大量成功的应用. 与图 1.4 对应的全连接网络 (相邻两层之间的节点都是相连或相关的) 不同, 卷积神经网络的思想是通过局部连接以及共享参数的方式来大大减少参数量, 从而减少对数据量的依赖以及提高训练的速度. 典型的 CNN 网络结构通常由一个或多个卷积层、下采样层 (subsampling)[①]和顶层的全连接层组成. 全连接层的结构与多层感知机的结构相同. 卷积层是一种特殊的网络层, 它首先对输入数据进行卷积操作产生多个特征映射, 之后使用非线性激活函数 (比如 ReLU) 对每个特征进行变换. 下采样层一般位于卷积层之后, 它的作用是减小数据维数并提取数据的多尺度信息, 其结果最终会输出到下一组变换.

① 有时也称为“池化”(pooling).

给定一个二维图像 $I \in \mathbb{R}^{n\times n}$ 和卷积核 $K \in \mathbb{R}^{k\times k}$, 我们定义一种简单的卷积操作 $S = I*K$, 它的元素是

$$S_{i,j} = \langle I(i:i+k-1, j:j+k-1), K\rangle\,, \tag{1.4.4}$$

其中两个矩阵 X, Y 的内积是它们相应元素乘积之和, 即 $\langle X, Y\rangle = \sum_{i,j} X_{ij}Y_{ij}$, $I(i:i+k-1, j:j+k-1)$ 是矩阵 I 从位置 (i,j) 开始的一个 $k\times k$ 子矩阵. 图 1.5 给出了一个例子. 生成的结果 S 可以根据卷积核的维数、I 的边界是否填充、卷积操作时滑动的大小等相应变化.

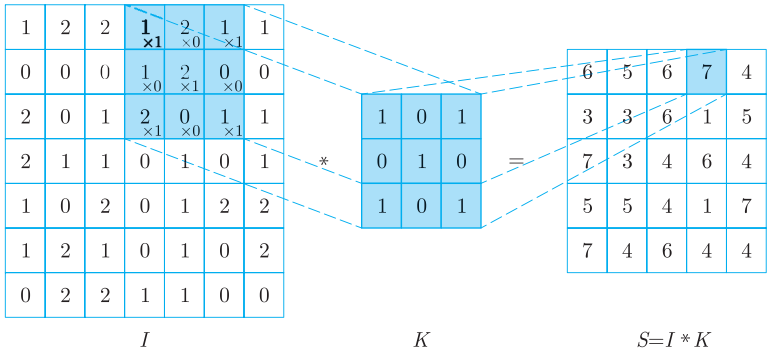


图 1.5 卷积操作

图 1.6 给出了下采样层的一个示例. 第 l 特征层是第 $(l-1)$ 特征层的下采样层. 具体地, 我们先将第 $(l-1)$ 层的每个矩阵划分成若干子矩阵, 之后将每个子矩阵里所有元素按照某种规则 (例如取平均值或最大值) 变换成一个元素. 因此, 第 $(l-1)$ 特征层每个小框里所有元素的平均值或最大值就对应于第 l 特征层的一个元素. 容易看出, 下采样层实际上是用一个数代表一个子矩阵, 经过下采样层变换后, 前一特征层矩阵的维数会进一步降低. 图 1.7 给出了一个简单的卷积神经网络示意图. 输入图片通过不同的卷积核生成的不同矩阵, 再经过非线性激活函数作用后生成第 1 层的特征; 第 2 层是第 1 层的下采样层; 第 3 层和第 4 层又是卷积层和下采样层; 第 5 层是全连接层; 第 6 层为

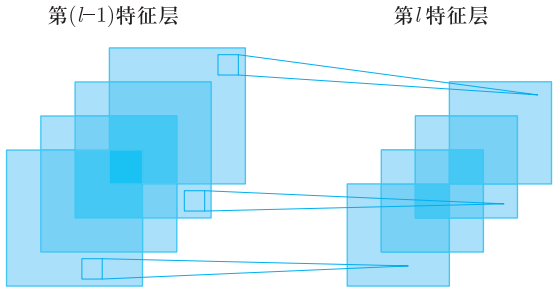


图 1.6 下采样层示例

输出层. 实际的卷积神经网络可达几十层甚至更多, 卷积核的大小、网络节点之间的连接方式也可以有很多变化, 从而生成不一样的模型.

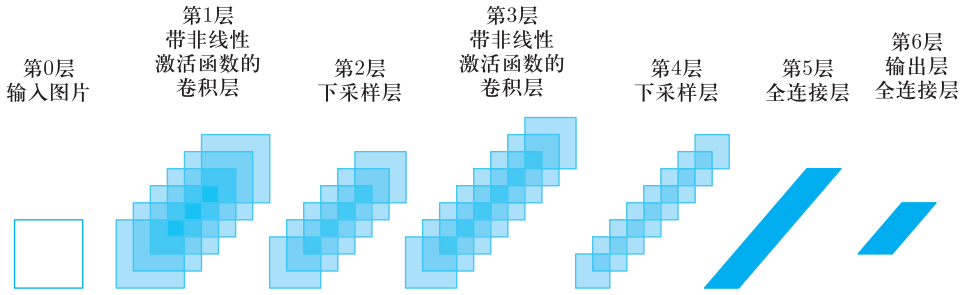


图 1.7　卷积神经网络一种示意图

给定一个训练集 $D = \{\{a_1, b_1\}, \{a_2, b_2\}, \cdots, \{a_m, b_m\}\}$, 其中 a_i 是训练图片, b_i 是其对应的标签. 卷积神经网络对应的优化问题的形式仍可套用 (1.4.3), 但函数 $h(a_i; x)$ 由卷积神经网络构成, 而 x 是卷积神经网络的参数.

1.5　最优化的基本概念

一般来说, 最优化算法研究可以分为: 构造最优化模型、确定最优化问题的类型和设计算法、实现算法或调用优化算法软件包进行求解. 最优化模型的构造和实际问题紧密相关, 比如说, 给定二维欧几里得 (Euclid) 空间的若干个离散点, 假定它们可以通过一条直线分成两部分, 也可以通过一条曲线分成两部分, 那么分别使用直线与曲线所得到的最优化模型是不同的. 在问题 (1.1.1) 中, 目标函数 f 和约束函数 c_i 都是由模型来确定的. 在确定模型之后, 我们需要对模型对应的优化问题进行分类. 这里, 分类的必要性是因为不存在对于所有优化问题的一个统一的算法. 因此我们需要针对具体优化问题所属的类别, 来设计或者调用相应的算法求解器. 最后就是模型的求解过程. 同一类优化问题往往存在着不同的求解算法. 对于具体的优化问题, 我们需要充分利用问题的结构, 并根据问题的需求 (求解精度和速度等) 来设计相应的算法. 另外, 根据算法得到的结果, 我们可以来判别模型构造是否合理或者进一步地改进模型. 如果构造的模型比较复杂, 那么算法求解起来相对困难 (时间慢或者精度差). 此时算法分析可以帮助我们设计替代模型, 以确保快速且比较精确地求出问题的解.

这三个部分的研究对于形成完备的最优化体系是必要的. 实际应用导出的各种各样的最优化模型给最优化学科不断注入新鲜的血液, 对现有的优化算法进行挑战并推动其向前发展. 最优化算法的设计以及理论分析帮助实际问题建立更鲁棒稳定的模型. 模型与算法相辅相成, 使得最优化学科不断发展.

1.5.1 连续和离散优化问题

最优化问题可以分为连续和离散优化问题两大类. 连续优化问题是指决策变量所在的可行集合是连续的, 比如平面、区间等. 如稀疏优化问题 (1.2.2) — (1.2.5) 的约束集合就是连续的. 离散优化问题是指决策变量能在离散集合上取值, 比如离散点集、整数集等. 常见的离散优化问题有整数规划, 其对应的决策变量的取值范围是整数集合.

在连续优化问题中, 基于决策变量取值空间以及约束和目标函数的连续性, 我们可以从一个点处目标和约束函数的取值来估计该点可行邻域内的取值情况. 进一步地, 可以根据邻域内的取值信息来判断该点是否最优. 离散优化问题则不具备这个性质, 因为决策变量是在离散集合上取值. 因此在实际中往往比连续优化问题更难求解. 实际中的离散优化问题往往可以转化为一系列连续优化问题来进行求解. 比如线性整数规划问题中著名的分支定界方法, 就是松弛成一系列线性规划问题来进行求解. 因此连续优化问题的求解在最优化理论与算法中扮演着重要的角色. 本书后续的内容也将围绕连续优化问题展开介绍.

1.5.2 无约束和约束优化问题

最优化问题的另外一个重要的分类标准是约束是否存在. 无约束优化问题的决策变量没有约束条件限制, 即可行域 $\mathcal{X} = \mathbb{R}^n$. 相对地, 约束优化问题是指带有约束条件的问题. 在实际应用中, 这两类优化问题广泛存在. 无约束优化问题对应于在欧几里得空间中求解一个函数的最小值点. 比如在 ℓ_1 正则化问题 (1.2.5) 中, 决策变量的可行域是 $\mathbb{R}^n$, 其为一个无约束优化问题. 在问题 (1.2.2) — (1.2.4) 中, 可行域为 $\{x \mid Ax = b\}$, 其为约束优化问题.

因为问题 (1.1.1) 可以通过将约束 ($\mathcal{X} \neq \mathbb{R}^n$) 罚到目标函数上转化为无约束问题, 所以在某种程度上, 约束优化问题就是无约束优化问题. 很多约束优化问题的求解也是转化为一系列的无约束优化问题来做, 常见方式有增广拉格朗日函数法、罚函数法等. 尽管如此, 约束优化问题的理论以及算法研究仍然是非常重要的. 主要原因是, 借助于约束函数, 我们能够更好地描述可行域的几何性质, 进而更有效地找到最优解.

1.5.3 随机和确定性优化问题

伴随着近年来人工智能的发展, 随机优化问题的研究得到了长足的发展. 随机优化问题是指目标或者约束函数中涉及随机变量而带有不确定性的问题. 不像确定性优化问题中目标和约束函数都是确定的, 随机优化问题中总是包含一些未知的参数. 在实际问题中, 我们往往只能知道这些参数的某些估计. 随机优化问题在机器学习、深度学习以

及强化学习中有着重要应用, 其优化问题的目标函数是关于一个未知参数的期望的形式. 因为参数的未知性, 实际中常用的方法是通过足够多的样本来逼近目标函数, 得到一个新的有限和形式的目标函数. 由于样本数量往往非常大, 我们还是将这个问题看作相对于指标随机变量的期望形式, 然后通过随机优化方法来进行求解.

相比于确定性优化问题, 随机优化问题的求解往往涉及更多的随机性. 很多确定性优化算法都有相应的随机版本. 随机性使得这些算法在特定问题上具有更低的计算复杂度或者更好的收敛性质. 以目标函数为多项求和的优化问题为例, 如果使用确定性优化算法, 每一次计算目标函数的梯度都会引入昂贵的复杂度. 但是对于随机优化问题, 我们每次可能只计算和式中的一项或者几项, 这大大减少了计算时间. 同时还能保证算法求解的足够精确. 确定性优化算法会在第三—五章中给出, 随机优化算法会在第五章中介绍.

1.5.4　线性和非线性规划问题

线性规划是指问题 (1.1.1) 中目标函数和约束函数都是线性的. 当目标函数和约束函数至少有一个是非线性的时, 对应的优化问题称为非线性规划问题. 线性规划问题在约束优化问题中具有较为简单的形式. 类似于连续函数可以用分片线性函数来逼近, 线性规划问题的理论分析与数值求解可以为非线性规划问题提供很好的借鉴和基础.

线性规划问题的研究很早便得到了人们的关注. 在 1946—1947 年, George Bernard Dantzig 提出了线性规划的一般形式并提出了至今仍非常流行的单纯形方法.虽然单纯形方法在实际问题中经常表现出快速收敛, 但是其复杂度并不是多项式的. 1979 年, Leonid Khachiyan 证明了线性规划问题多项式时间算法的存在性.1984 年, Narendra Karmarkar 提出了多项式时间的内点法. 后来, 内点法也被推广到求解一般的非线性规划问题. 目前, 求解线性规划问题最流行的两类方法依然是单纯形法和内点法.

1.5.5　凸和非凸优化问题

凸优化问题是指最小化问题 (1.1.1) 中的目标函数和可行域分别是凸函数和凸集. 如果其中有一个或者两者都不是凸的, 那么相应的最小化问题是非凸优化问题. 因为凸优化问题的任何局部最优解都是全局最优解, 其相应的算法设计以及理论分析相对非凸优化问题简单很多.

注 1.1　若问题 (1.1.1) 中的 min 改为 max, 且目标函数和可行域分别为凹函数和凸集, 我们也称这样的问题为凸优化问题. 这是因为对凹函数求极大等价于对其相反数 (凸函数) 求极小.

在实际问题的建模中, 我们经常更倾向于得到一个凸优化模型. 另外, 判断一个问题是不是凸问题也很重要. 比如, 给定一个非凸优化问题, 一种方法是将其转化为一系

列凸优化子问题来求解. 此时需要清楚原非凸问题中的哪个或哪些函数导致了非凸性, 之后考虑的是如何用凸优化模型来逼近原问题. 在压缩感知问题中, ℓ_0 范数是非凸的, 原问题对应的解的性质难以直接分析, 相应的全局收敛的算法也不容易构造. 利用 ℓ_0 范数和 ℓ_1 范数在某种意义上的等价性, 我们将原非凸问题转化为凸优化问题. 在一定的假设下, 我们通过求解 ℓ_1 范数对应的凸优化问题得到了原非凸优化问题的全局最优解.

1.5.6 全局和局部最优解

在求解最优化问题之前, 先介绍最小化问题 (1.1.1) 的最优解的定义.

定义 1.1 (最优解) 对于可行点 $\bar{x}$ (即 $\bar{x} \in \mathcal{X}$), 定义如下概念:

(1) 如果 $f(\bar{x}) \leqslant f(x), \forall x \in \mathcal{X}$, 那么称 $\bar{x}$ 为问题 (1.1.1) 的全局极小解 (点), 有时也称为 (全局) 最优解或最小值点;

(2) 如果存在 $\bar{x}$ 的一个 ε 邻域 $N_\varepsilon(\bar{x})$ 使得 $f(\bar{x}) \leqslant f(x), \forall x \in N_\varepsilon(\bar{x}) \cap \mathcal{X}$, 那么称 $\bar{x}$ 为问题 (1.1.1) 的局部极小解 (点), 有时也称为局部最优解;

(3) 进一步地, 如果有 $f(\bar{x}) < f(x), \forall x \in N_\varepsilon(\bar{x}) \cap \mathcal{X}, x \neq \bar{x}$ 成立, 那么称 $\bar{x}$ 为问题 (1.1.1) 的严格局部极小解 (点).

如果一个点是局部极小解, 但不是严格局部极小解, 我们称之为**非严格局部极小解**. 在图 1.8 中, 我们以一个简单的函数为例, 指出了其全局与局部极小解.

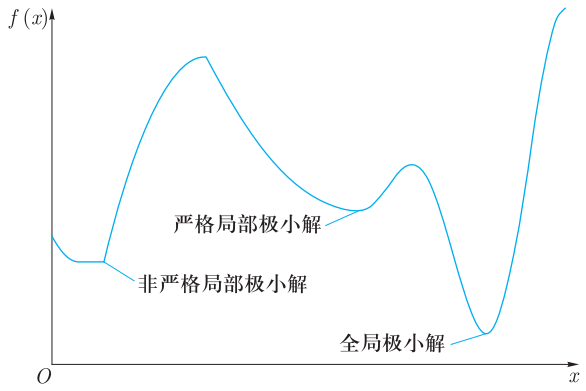


图 1.8 函数的全局极小解、严格局部极小解和非严格局部极小解

在问题 (1.1.1) 的求解中, 我们想要得到的是其全局最优解, 但是由于实际问题的复杂性, 往往只能够得到其局部最优解.

1.5.7 优化算法

在给定优化问题之后, 我们要考虑如何求解. 根据优化问题的不同形式, 其求解的困难程度可能会有很大差别. 对于一个优化问题, 如果我们能用代数表达式给出其最优

解, 那么这个解称为显式解, 对应的问题往往比较简单. 例如二次函数在有界区间上的极小化问题, 我们可以通过比较其在对称轴上和区间两个端点处的值得到最优解, 这个解可以显式地写出. 但实际问题往往是没有办法显式求解的, 因此常采用迭代算法.

迭代算法的基本思想是: 从一个初始点 x^0 出发, 按照某种给定的规则进行迭代, 得到一个序列 $\{x^k\}$. 如果迭代在有限步内终止, 那么希望最后一个点就是优化问题的解. 如果迭代点列是无穷集合, 那么希望该序列的极限点 (或者聚点) 则为优化问题的解. 为了使算法能在有限步内终止, 我们一般会通过一些收敛准则来保证迭代停在问题的一定精度逼近解上. 对于无约束优化问题, 常用的收敛准则有

$$\frac{f(x^k)-f^*}{\max\{|f^*|,1\}}\leqslant\varepsilon_1,\quad \|\nabla f(x^k)\|\leqslant\varepsilon_2, \tag{1.5.1}$$

其中 $\varepsilon_1,\varepsilon_2$ 为给定的很小的正数, $\|\cdot\|$ 表示某种范数$\Big($这里可以简单理解为 ℓ_2 范数: $\|x\|_2=\left(\sum\limits_{i=1}^{n}x_i^2\right)^{1/2}$, 第二章将会给出范数的一般定义$\Big)$, f^* 为函数 f 的最小值 (假设已知或者以某种方式估计得到) 以及 $\nabla f(x^k)$ 表示函数 f 在点 x 处的梯度 (光滑函数在局部最优点处梯度为零向量, 第三章中会给出更多介绍). 对于约束优化问题, 还需要考虑约束违反度. 具体地, 要求最后得到的点满足

$$c_i(x^k)\leqslant\varepsilon_3,\ i=1,2,\cdots,m,$$

$$|c_i(x^k)|\leqslant\varepsilon_4,\ i=m+1,m+2,\cdots,m+l,$$

其中 $\varepsilon_3,\varepsilon_4$ 为很小的正数, 用来刻画 x^k 的可行性. 除了约束违反度之外, 我们也要考虑 x^k 与最优解之间的距离, 如 (1.5.1) 式中给出的函数值与最优值的相对误差. 由于一般情况下事先并不知道最优解, 在最优解唯一的情形下一般使用某种基准算法来得到 x^* 的一个估计, 之后计算其与 x^k 的距离以评价算法的性能. 因为约束的存在, 我们不能简单地用目标函数的梯度来判断最优性, 实际中采用的判别准则是点的最优性条件的违反度 (关于约束优化的最优性条件, 会在第四章中给出).

对于一个具体的算法, 根据其设计的出发点, 我们不一定能得到一个高精度的逼近解. 此时, 为了避免无用的计算开销, 我们还需要一些停机准则来及时停止算法的进行. 常用的停机准则有

$$\frac{\|x^{k+1}-x^k\|}{\max\{\|x^k\|,1\}}\leqslant\varepsilon_5,\quad \frac{|f(x^{k+1})-f(x^k)|}{\max\{|f(x^k)|,1\}}\leqslant\varepsilon_6,$$

这里的各个 ε_l 一般互不相等. 上面的准则分别表示相邻迭代点和其对应目标函数值的相对误差很小. 在算法设计中, 这两个条件往往只能反映迭代点列接近收敛, 但不能代表收敛到优化问题的最优解.

在算法设计中, 一个重要的标准是算法产生的点列是否收敛到优化问题的解. 对于问题 (1.1.1), 其可能有很多局部极小解和全局极小解, 但所有全局极小解对应的目标函数值, 即优化问题的最小值 f^* 是一样的. 考虑无约束的情形, 对于一个算法, 给定初始点 x^0, 记其迭代产生的点列为 $\{x^k\}$. 如果 $\{x^k\}$ 在某种范数 $\|\cdot\|$ 的意义下满足

$$\lim_{k\to\infty}\|x^k - x^*\| = 0,$$

且收敛的点 x^* 为一个局部 (全局) 极小解, 那么我们称该点列收敛到局部 (全局) 极小解, 相应的算法称为是**依点列收敛到局部 (全局) 极小解**的.

在算法的收敛分析中, 初始迭代点 x^0 的选取也尤为重要. 比如一般的牛顿法, 只有在初始点足够接近局部 (全局) 最优解时, 才能收敛. 但是这样的初始点的选取往往比较困难, 此时我们更想要的是一个从任何初始点出发都能收敛的算法. 因此优化算法的研究包括如何设计全局化策略, 将已有的可能发散的优化算法修改得到一个新的全局收敛到局部 (全局) 最优解的算法. 比如通过采用合适的全局化策略, 我们可以修正一般的牛顿法使得修改后的算法是全局收敛到局部 (全局) 最优解的.

进一步地, 如果从任意初始点 x^0 出发, 算法都是依点列收敛到局部 (全局) 极小解的, 我们称该算法是**全局依点列收敛到局部 (全局) 极小解**的 . 相应地, 如果记对应的函数值序列为 $\{f(x^k)\}$, 我们还可以定义算法的 **(全局) 依函数值收敛到局部 (全局) 极小值**的概念. 对于凸优化问题, 因为其任何局部最优解都为全局最优解, 算法的收敛性都是相对于其全局极小而言的. 除了点列和函数值的收敛外, 实际中常用的还有每个迭代点的最优性条件 (如无约束优化问题中的梯度范数, 约束优化问题中的最优性条件违反度等) 的收敛.

对于带约束的情形, 给定初始点 x^0, 算法产生的点列 $\{x^k\}$ 不一定是可行的 (即 $x^k \in \mathcal{X}$ 未必对任意 k 成立). 考虑到约束违反的情形, 我们需要保证 $\{x^k\}$ 在收敛到 x^* 的时候, 其违反度是可接受的. 除此要求之外, 算法的收敛性的定义和无约束情形相同.

在设计优化算法时, 我们有一些基本的准则或技巧. 对于复杂的优化问题, 基本的想法是将其转化为一系列简单的优化问题 (其最优解容易计算或者有显式表达式) 来逐步求解. 常用的技巧有:

(1) **泰勒 (Taylor) 展开**. 对于一个非线性的目标或者约束函数, 我们通过其泰勒展开用简单的线性函数或者二次函数来逼近, 从而得到一个简化的问题. 因为该简化问题只在小邻域内逼近原始问题, 所以我们需要根据迭代点的更新来重新构造相应的简化问题.

(2) **对偶**. 每个优化问题都有对应的对偶问题. 特别是凸的情形, 当原始问题比较难解的时候, 其对偶问题可能很容易求解. 通过求解对偶问题或者同时求解原始问题和对偶问题, 我们可以简化原始问题的求解, 从而设计更有效的算法.

(3) **拆分**. 对于一个复杂的优化问题, 我们可以将变量进行拆分, 比如 $\min\limits_x h(x) +$

$r(x)$, 可以拆分成

$$\min_{x,y} h(x)+r(y), \quad \text{s.t.} \quad x=y.$$

通过引入更多的变量, 我们可以得到每个变量的简单问题 (较易求解或者解有显式表达式), 从而通过交替求解等方式来得到原问题的解.

(4) **块坐标下降**. 对于一个 n 维空间 (n 很大) 的优化问题, 我们可以通过逐步求解分量的方式将其转化为多个低维空间中的优化问题. 比如, 对于 $n=100$, 我们可以先固定第 2—100 个分量来求解 x_1; 接着固定下标为 $1,3$—100 的分量来求解 x_2; 依次类推.

关于这些技巧的具体应用, 读者可以进一步阅读本书中的算法部分.

对于同一个优化问题, 其求解算法可以有很多. 在设计和比较不同的算法时, 另一个重要的指标是算法的渐近收敛速度. 我们以点列的 **Q-收敛速度** (Q 的含义为 quotient) 为例 (函数值的 Q-收敛速度可以类似地定义). 设 $\{x^k\}$ 为算法产生的迭代点列且收敛于 x^*, 若对充分大的 k 有

$$\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|} \leqslant a, \quad a\in(0,1),$$

则称算法 (点列) 是 **Q-线性收敛**的; 若满足

$$\lim_{k\to\infty} \frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|} = 0,$$

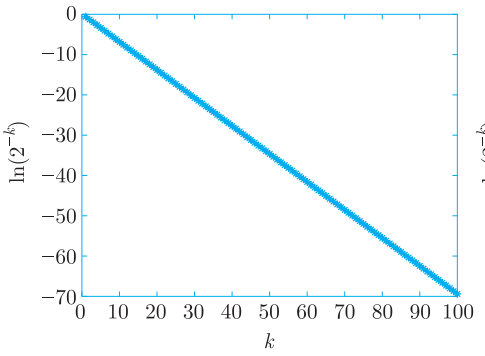
则称算法 (点列) 是 **Q-超线性收敛**的; 若满足

$$\lim_{k\to\infty} \frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|} = 1,$$

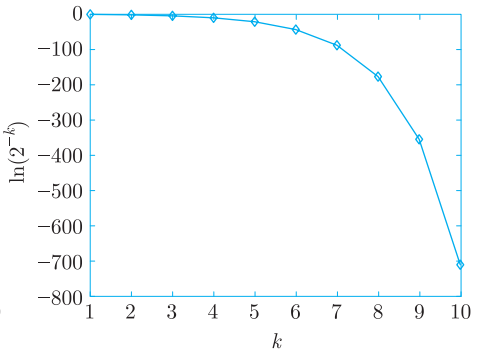
则称算法 (点列) 是 **Q-次线性收敛**的. 若对充分大的 k 满足

$$\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|^2} \leqslant a, \quad a>0,$$

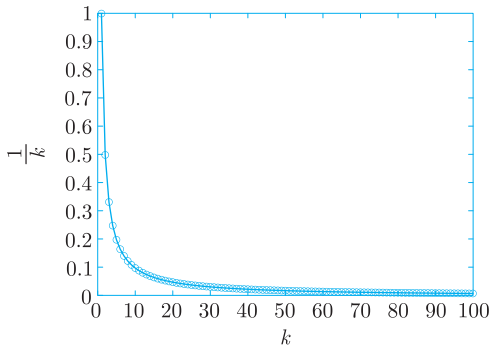
则称算法 (点列) 是 **Q-二次收敛**的. 类似地, 也可定义更一般的 Q-r 次收敛 ($r>1$). 我们举例来更直观地展示不同的 Q-收敛速度, 参见图 1.9(图中对所考虑的点列作了适当的变换). 点列 $\{2^{-k}\}$ 是 Q-线性收敛的, 点列 $\{2^{-2^k}\}$ 是 Q-二次收敛的 (也是 Q-超线性收敛的), 点列 $\left\{\dfrac{1}{k}\right\}$ 是 Q-次线性收敛的. 一般来说, 具有 Q-超线性收敛速度和 Q-二次收敛速度的算法是收敛较快的.



(a) Q–线性收敛



(b) Q–二次收敛



(c) Q-次线性收敛

图 1.9 不同 Q-收敛速度比较

除 Q-收敛速度外, 另一常用概念是 **R-收敛速度** (R 的含义为 root). 以点列为例, 设 $\{x^k\}$ 为算法产生的迭代点且收敛于 x^*, 若存在 Q-线性收敛于 0 的非负序列 t_k, 并且

$$\|x^k - x^*\| \leqslant t_k$$

对任意的 k 成立, 则称算法 (点列) 是 **R-线性收敛**的. 类似地, 可定义 **R-超线性收敛**和 **R-二次收敛**等收敛速度. 从 R-收敛速度的定义可以看出序列 $\{\|x^k - x^*\|\}$ 被另一趋于 0 的序列 $\{t_k\}$ 控制. 当知道 t_k 的形式时, 我们也称算法 (点列) 的收敛速度为 $\mathcal{O}(t_k)$.

与收敛速度密切相关的概念是优化算法的**复杂度** $N(\varepsilon)$, 即计算出给定精度 ε 的解所需的迭代次数或浮点运算次数. 在实际应用中, 这两种定义复杂度的方式均很常见. 若能较准确地估计每次迭代的运算量, 则可以由算法所需迭代次数推出所需浮点运算次数. 我们用具体的例子来进一步解释算法复杂度. 设某一算法产生的迭代序列 $\{x^k\}$ 满足

$$f(x^k) - f(x^*) \leqslant \frac{c}{\sqrt{k}}, \quad \forall\, k > 0,$$

其中 $c>0$ 为常数, x^* 为全局极小点. 如果需要计算算法满足精度 $f(x^k)-f(x^*)\leqslant\varepsilon$ 所需的迭代次数, 只需令 $\dfrac{c}{\sqrt{k}}\leqslant\varepsilon$ 就得到 $k\geqslant\dfrac{c^2}{\varepsilon^2}$, 因此该优化算法对应的 (迭代次数) 复杂度为 $N(\varepsilon)=\mathcal{O}\left(\dfrac{1}{\varepsilon^2}\right)$. 注意, 渐近收敛速度更多的是考虑迭代次数充分大的情形, 而复杂度给出了算法迭代有限步之后产生的解与最优解之间的定量关系, 因此近年来受到人们广泛关注.

1.6　总结

本章简要介绍了优化问题的应用背景、一般形式以及一些基本概念. 对于优化问题的更多分类、优化领域关心的热点问题, 可以参考文献 [162] 中的介绍. 对于优化算法的收敛准则、收敛性以及收敛速度, 我们会在介绍算法的时候再具体展开. 本书也会围绕上面介绍的优化算法的四个设计技巧, 针对不同类别的问题, 来具体地展示相应的算法构造以及有效性分析.

习题 1

1.1　考虑稀疏优化问题, 我们已经直观地讨论了在 ℓ_0, ℓ_1, ℓ_2 三种范数下问题的解的可能形式. 针对一般的 ℓ_p 范数:

$$\|x\|_p\overset{\text{def}}{=\!=}\left(\sum_{i=1}^{n}|x_i|^p\right)^{1/p},\quad 0<p<2,$$

我们考虑优化问题:

$$\begin{aligned}\min\quad & \|x\|_p,\\ \text{s.t.}\quad & Ax=b.\end{aligned}$$

试着用几何直观的方式 (类似于图 1.2) 来说明当 $p\in(0,2)$ 取何值时, 该优化问题的解可能具有稀疏性.

1.2　给定一个函数 $f(x):\mathbb{R}^n\to\mathbb{R}$ 及其一个局部最优点 x^*, 则该点沿任何方向 $d\in\mathbb{R}^n$ 也是局部最优的, 即 0 为函数 $\phi(\alpha)\overset{\text{def}}{=\!=}f(x^*+\alpha d)$ 的一个局部最优解. 反之, 若 x^* 沿任何方向 $d\in\mathbb{R}^n$ 都是局部最优解, 则 x^* 是否为 $f(x)$ 的一个局部最优解? 若是, 请给出证明; 若不是, 请给出反例.

1.3　试给出如下点列的 Q-收敛速度:

(a) $x^k = \dfrac{1}{k!}, k = 1, 2, \cdots$;

(b) $x^k = \begin{cases} \left(\dfrac{1}{4}\right)^{2^k}, & k \text{ 为偶数}, \\ \dfrac{x^{k-1}}{k}, & k \text{ 为奇数}, \end{cases} \quad k = 1, 2, \cdots$.

1.4　考虑函数 $f(x) = x_1^2 + x_2^2$, $x = (x_1, x_2) \in \mathbb{R}^2$, 以及迭代点列 $x^k = \left(1 + \dfrac{1}{2^k}\right) \cdot (\cos k, \sin k)^{\mathrm{T}}, k = 1, 2, \cdots$, 请说明

(a) $\{f(x^{k+1})\}$ 是否收敛? 若收敛, 给出 Q-收敛速度;

(b) $\{x^{k+1}\}$ 是否收敛? 若收敛, 给出 Q-收敛速度.