提纲

最优化问题的一般形式

最优化问题一般可以描述为

$$\begin{aligned} \min \quad & f(x), \\ \text{s.t.} \quad & x \in \mathcal{X}, \end{aligned} \tag{1}$$

- $x = (x_1, x_2, \cdots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$是**决策变量**
- $f: \mathbb{R}^n \to \mathbb{R}$ 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) 称为无约束优化问题.
- 集合$\mathcal{X}$通常可以由约束函数$c_i(x): \mathbb{R}^n \to \mathbb{R}, i = 1, 2, \cdots, m + l$ 表达为如下具体形式：

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

最优化问题的一般形式

- 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) 的最优解，即对任意$x \in \mathcal{X}$ 都有

$$f(x) \geq f(x^*).$$

- 如果求解在约束集合$\mathcal{X}$ 上目标函数$f(x)$的最大值，则问题(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) 中的“min(max)” 改为“inf(sup)”.

- 为了叙述简便，问题(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.

最优化问题的类型

最优化问题的具体形式非常丰富，可以按照目标函数、约束函数以及解的性质将其分类.

- 当目标函数和约束函数均为线性函数时，问题称为**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**；
- 如果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.

最优化问题的类型

- 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) 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.

最优化问题的应用

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.
- 如果是极小化收益的方差，则该问题是典型的二次规划.
- 如果极小化风险价值(value at risk)函数，则该问题是混合整数规划
- 如果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.

投资组合优化

- r_i, 随机变量, 股票的回报率i
- x_i, 投资于股票的相对金额i
- 回报: $r = r_1x_1 + r_2x_2 + \ldots + r_nx_n$
- 期望回报: $R = E(r) = \sum E(r_i)x_i = \sum \mu_i x_i$
- 风险: $V = Var(r) = \sum_{i,j} \sigma_{ij}x_ix_j = x^\top \Sigma x$

$$\min \frac{1}{2} x^\top \Sigma x, \qquad \min \quad \text{risk measure},$$

$$\text{s.t.} \sum \mu_i x_i \geq r_0 \qquad \text{s.t.} \sum \mu_i x_i \geq r_0$$

$$\sum x_i = 1, \qquad \sum x_i = 1,$$

$$x_i \geq 0 \qquad x_i \geq 0$$

风险度量

- 风险值(VaR): $F(\cdot)$ 为损失变量X的分布函数. 对于给定的$\alpha \in (0,1)$, X 的水平为α 的VaR 定义为

$$\mathrm{VaR}_\alpha(X) := \inf\{x \mid F(x) \geq \alpha\} = F^{-1}(\alpha).$$

- 条件风险值(CVaR): α尾的分布函数定义为

$$F_{\alpha,X}(x) := \begin{cases} 0, & \text{for } x < \mathrm{VaR}_\alpha(X), \\ \frac{F_X(x)-\alpha}{1-\alpha}, & \text{for } x \geq \mathrm{VaR}_\alpha(X). \end{cases}$$

$$\begin{aligned} \mathrm{CVaR}_\alpha(X) &:= \text{X的}\alpha\text{尾分布的均值 } X \\ &= \int_{-\infty}^{\infty} x dF_{\alpha,X}(x). \end{aligned}$$

- 市场风险的资本要求

$$
\begin{aligned}
c_t &= \max\left\{\mathsf{VaR}_{\alpha,t-1}, \frac{k}{60}\sum_{s=1}^{60}\mathsf{VaR}_{\alpha,t-s}\right\} \\
&\quad + \max\left\{\mathsf{sVaR}_{\alpha,t-1}, \frac{\ell}{60}\sum_{s=1}^{60}\mathsf{sVaR}_{\alpha,t-s}\right\}
\end{aligned}
$$

- $\mathsf{sVaR}_{\alpha,t-s}$ 称为在$t-s$日的置信水平为$\alpha=99\%$ 的压力 VaR，它是在金融市场承受重大压力的情况下计算的，如2007年至2008年期间发生的那种情况。

投资组合优化

- 使用CVaR(或等价的ES)代替VaR
- 股权、信贷、利率、货币等主要风险因素相似的一组交易部门的资本金要求定义为该组交易部门可能发生的损失的CVaR;
- CVaR应在有压力的情况下计算，而不是在当前市场条件下。
- 市场风险的资本要求

$$c_t = \max\left\{\text{sCVaR}_{\alpha,t-1}, \frac{\ell}{60}\sum_{s=1}^{60}\text{sCVaR}_{\alpha,t-s}\right\},$$

- $\text{scVaR}_{\alpha,t-s}$ 称为在$t-s$日的置信水平为$\alpha = 99\%$ 的压力 CVaR

提纲

稀疏优化

给定$b \in \mathbb{R}^m$，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$m \ll n$. 考虑线性方程组求解问题：

$$Ax = b. \tag{2}$$

- 注意到由于$m \ll n$，方程组(2) 是欠定的，因此存在无穷多个解，重构出原始信号看似很难.
- 这些解当中大部分是不重要的，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，那么可以通过求解稀疏优化问题把u与方程组(2) 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

$$(\ell_0)\begin{cases}\min\limits_{x\in\mathbb{R}^n} & \|x\|_0,\\ \text{s.t.} & Ax=b.\end{cases}\quad(\ell_2)\begin{cases}\min\limits_{x\in\mathbb{R}^n} & \|x\|_2,\\ \text{s.t.} & Ax=b.\end{cases}\quad(\ell_1)\begin{cases}\min\limits_{x\in\mathbb{R}^n} & \|x\|_1,\\ \text{s.t.} & Ax=b.\end{cases}\tag{3}$$

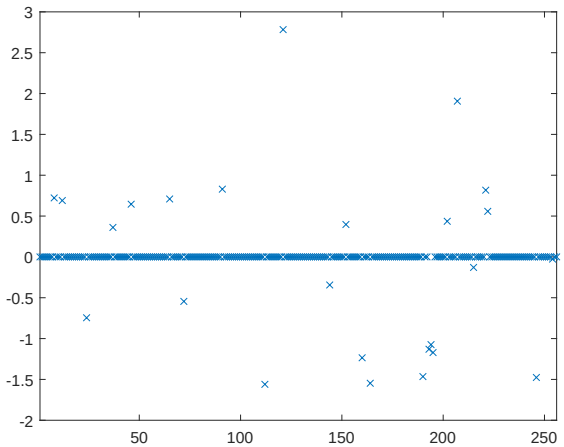
- 其中$\|x\|_0$是指x 中非零元素的个数. 由于$\|x\|_0$是不连续的函数，且取值只可能是整数，ℓ_0问题实际上是NP难的，求解起来非常困难.
- 若定义ℓ_1范数：$\|x\|_1=\sum_{i=1}^n|x_i|$，则得到了另一个形式上非常相似的问题，又称$\ell_1$ 范数优化问题，基追踪问题
- ℓ_2范数：$\|x\|_2=\left(\sum_{i=1}^n x_i^2\right)^{1/2}$

稀疏优化

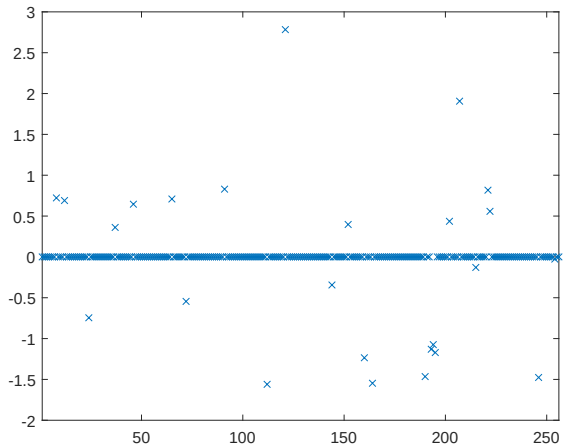
在MATLAB环境里构造A, u 和b:

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

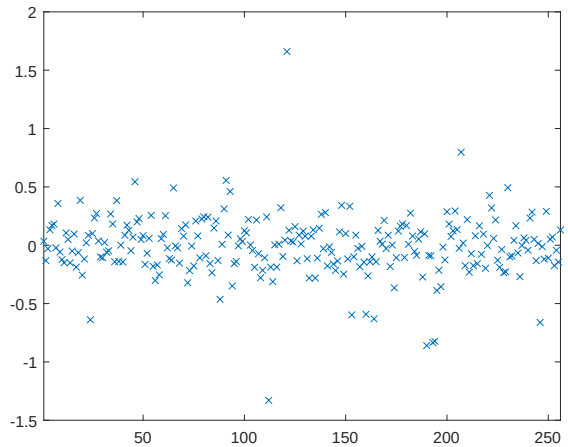
构造一个128×256矩阵A，它的每个元素都服从高斯（Gauss）随机分布．精确解u只有10%的元素非零，每一个非零元素也服从高斯分布



(a) 精确解u



(b) ℓ_1问题的解



(c) ℓ_2问题的解

Figure: 稀疏优化的例子

- 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），此时向量u也是ℓ_1范数优化问题(3)的唯一最优解．这一发现的重要之处在于，虽然问题(3) 仍没有显式解，但与零范数情形相比难度已经大大降低．

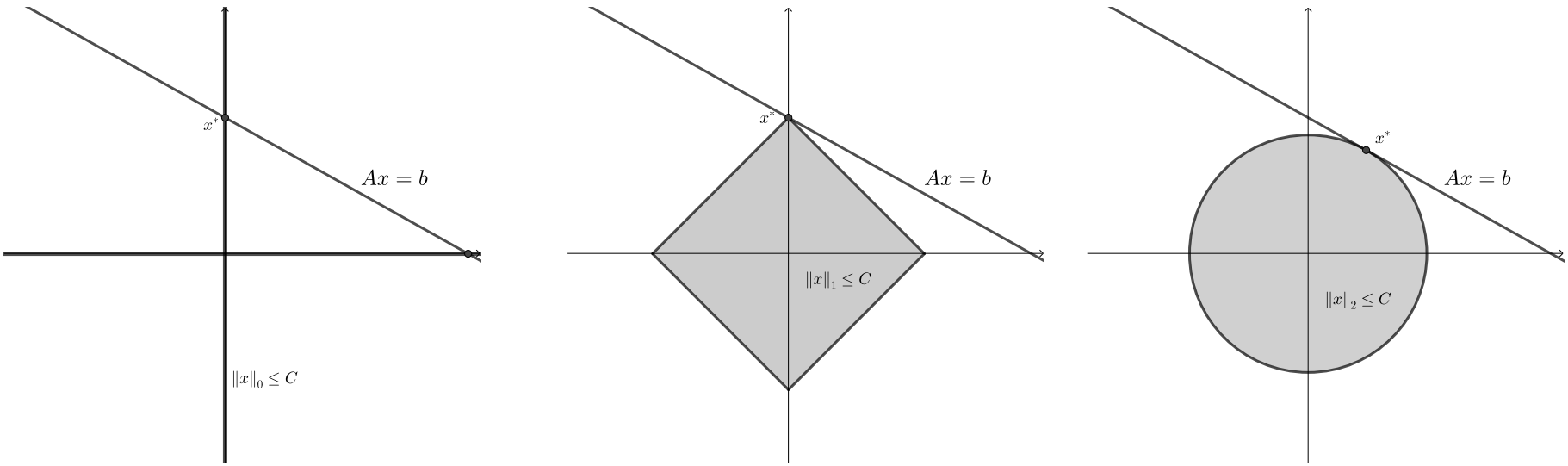
- 前面提到ℓ_0范数优化问题是NP 难问题，但ℓ_1范数优化问题的解可以非常容易地通过现有优化算法得到！从这个例子不难发现，优化学科的研究能够极大程度上帮助我们攻克现有的困难问题．但如果简单地把ℓ_1范数修改为ℓ_2范数：$\|x\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$，即求解如下优化问题：

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|x\|_2, \\ \text{s.t.} \quad & Ax = b. \end{aligned} \tag{4}$$

但二者是否相等是一个问题

稀疏优化

- 问题(4) 是原点到仿射集$Ax = b$的投影，可以写出它的显式表达式．但u并不是问题(4) 的解，也可从图1(a)-(c)看出该结论
- 下图可说明ℓ_1范数问题的解具有稀疏性而ℓ_2范数问题的解不具有该性质．



(a) ℓ_0范数　　(b) ℓ_1范数　　(c) ℓ_2范数

Figure: 三种范数优化问题求解示意图

稀疏优化

- 问题(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(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，保证稀疏解的存在性.
- 注意到绝对值函数在零点处不可微，问题(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(3).
- 需要强调的是，问题(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.

LASSO问题

考虑带ℓ_1范数正则项的优化问题

$$\min_{x\in\mathbb{R}^n} \quad \mu\|x\|_1 + \frac{1}{2}\|Ax-b\|_2^2, \tag{5}$$

其中$\mu > 0$是给定的正则化参数.

- 问题(5) 又称为LASSO（least absolute shrinkage and selection operator），该问题可以看成是问题(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(3) 简单. 课件关注的大部分数值算法都将针对问题(3) 或问题(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.

提纲

低秩矩阵恢复

- 某视频网站提供了约48万用户对1万7千多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

	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	…	?

低秩矩阵恢复问题的性质

该问题在推荐系统、图像处理等方面有着广泛的应用。

- 由于用户对电影的偏好可进行分类，按年龄可分为：年轻人，中年人，老年人；且电影也能分为不同的题材：战争片，悬疑片，言情片等。故这类问题隐含的假设为补全后的矩阵应为低秩的。即矩阵的行与列会有“合作”的特性，故该问题具有别名“collaborative filtering”。
- 除此之外，由于低秩矩阵可分解为两个低秩矩阵的乘积，所以低秩限制下的矩阵补全问题是比较实用的，这样利于储存且有更好的诠释性。
- 有些用户的打分可能不为自身真实情况，对评分矩阵有影响，所以原矩阵是可能有噪声的。

低秩矩阵恢复

由上述分析可以引出该问题：

- 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$X_{ij} = M_{ij},\ (i,j) \in \Omega$.

- 低秩矩阵恢复（low rank matrix completion）

$$\begin{aligned} \min_{X \in \mathbb{R}^{m \times n}} \quad & \operatorname{rank}(X), \\ \text{s.t.} \quad & X_{ij} = M_{ij},\ (i,j) \in \Omega. \end{aligned} \tag{6}$$

  $\operatorname{rank}(X)$正好是矩阵X所有非零奇异值的个数

- 矩阵X的核范数（nuclear norm）为矩阵所有奇异值的和，即：$\|X\|_* = \sum_i \sigma_i(X)$:

$$\begin{aligned} \min_{X \in \mathbb{R}^{m \times n}} \quad & \|X\|_*, \\ \text{s.t.} \quad & X_{ij} = M_{ij},\ (i,j) \in \Omega. \end{aligned} \tag{7}$$

低秩矩阵恢复

- 可以证明问题(7) 是一个凸优化问题，并且在一定条件下它与问题(6) 等价.
- 也可以将问题(7) 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.
- 考虑到观测可能出现误差，对于给定的参数$\mu > 0$，给出该问题的二次罚函数形式：

$$\min_{X \in \mathbb{R}^{m \times n}} \quad \mu \|X\|_* + \frac{1}{2} \sum_{(i,j) \in \Omega} (X_{ij} - M_{ij})^2. \tag{8}$$

低秩矩阵恢复

- 秩r情形：$X = LR^T$,其中$L \in \mathbb{R}^{m\times r}$,$R \in \mathbb{R}^{n\times r}$ 并且$r \ll \min(m,n)$.则可将问题写为

$$\min_{\boldsymbol{L},\boldsymbol{R}} \sum_{(i,j)\in\omega} \left(\left[\boldsymbol{L}\boldsymbol{R}^T\right]_{ij} - \boldsymbol{M}_{ij} \right)^2 + \alpha \|\boldsymbol{L}\|_F^2 + \beta \|\boldsymbol{R}\|_F^2$$

- 在该问题中，矩阵X在定义中已为秩r矩阵，所以没有必要再加上秩约束正则项。α, β为正则化参数，这里正则化的作用是消除解L, R在放缩意义下的不唯一性.

- 此时L, R矩阵中的数字之和为$(m+n)r$,远小于np,不过此时问题是非凸的.

- 尽管这个该问题是非凸的，但在某种意义上它是一个可处理问题的近似：如果对X 有一个完整的观察，那么秩-r近似可以通过X的奇异值分解来找到，并根据r导出的左奇异向量和右奇异向量定义L和R。

提纲

深度学习

深度学习（deep learning）是机器学习的一个子领域，它采用了一个特定的模型：一族通过某种方式连接起来的简单函数。由于这类模型的结构是受到人类大脑结构的启发而创造出来的，因此我们通常把它们称为神经网络（neural networks）。神经网络中的函数链条能够将复杂的概念分解为多个层次的更简单的概念，这就是深度学习的核心思想。

- 深度学习的起源可以追溯至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．

机器学习中典型问题形式

很多机器学习中的问题可以写为：

$$\min_{x\in\mathcal{W}} \quad \sum_{i=1}^{N} \frac{1}{2}\|a_i^\top x - b_i\|_2^2 + \mu\varphi(x) \quad \text{线性回归}$$

$$\min_{x\in\mathcal{W}} \quad \frac{1}{N}\sum_{i=1}^{N} \log(1+\exp(-b_i a_i^\top x)) + \mu\varphi(x) \quad \text{逻辑回归}$$

$$\min_{x\in\mathcal{W}} \quad \frac{1}{N}\sum_{i=1}^{N} \ell(f(a_i,x),b_i) + \mu\varphi(x) \quad \text{一般形式}$$

- (a_i, b_i) 是给定的数据对，b_i 是数据a_i对应的标签
- $\ell_i(\cdot)$: 度量模型拟合数据点i的程度(避免拟合不足)
- $\varphi(x)$: 避免过拟合的正则项: $\|x\|_2^2$ 或者$\|x\|_1$等等
- $f(a,x)$: 线性函数或者由深度神经网络构造的模型

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：训练集是用来训练神经网络，从而使得神经网络能够掌握训练集上的信息;测试集是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.

多层感知机模型介绍

给定训练集$D = \{\{a_1, b_1\}, \{a_2, b_2\}, \cdots, \{a_m, b_m\}\}$.

- 假设数据$a_i \in \mathbb{R}^p$, $b_i \in \mathbb{R}^q$. $a_{i1} = 1$. 图3 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，令$y_1^{(l)} = 1, 0 \le l \le L$,
- 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$i \ge 1$）计算输出信息$y_i^{(l)}$为

$$y_i^{(l)} = t(z_i^{(l)}), \quad z_i^{(l)} = \sum_{k=1}^{m^{(l-1)}} x_{i,k}^{(l)} y_k^{(l-1)}. \tag{9}$$

这里函数$t(\cdot)$称为激活函数.

常见的激活函数

常见的激活函数类型有Sigmoid 函数

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

Heaviside函数

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

以及ReLU函数

$$t(z) = \max\{0, z\}. \tag{10}$$

整个过程可以描述为

$$y^{(0)} \xrightarrow{x^{(1)}} z^{(1)} \xrightarrow{t} y^{(1)} \xrightarrow{x^{(2)}} \dots \xrightarrow{t} y^{(L+1)}.$$

上述过程可用下图表示:

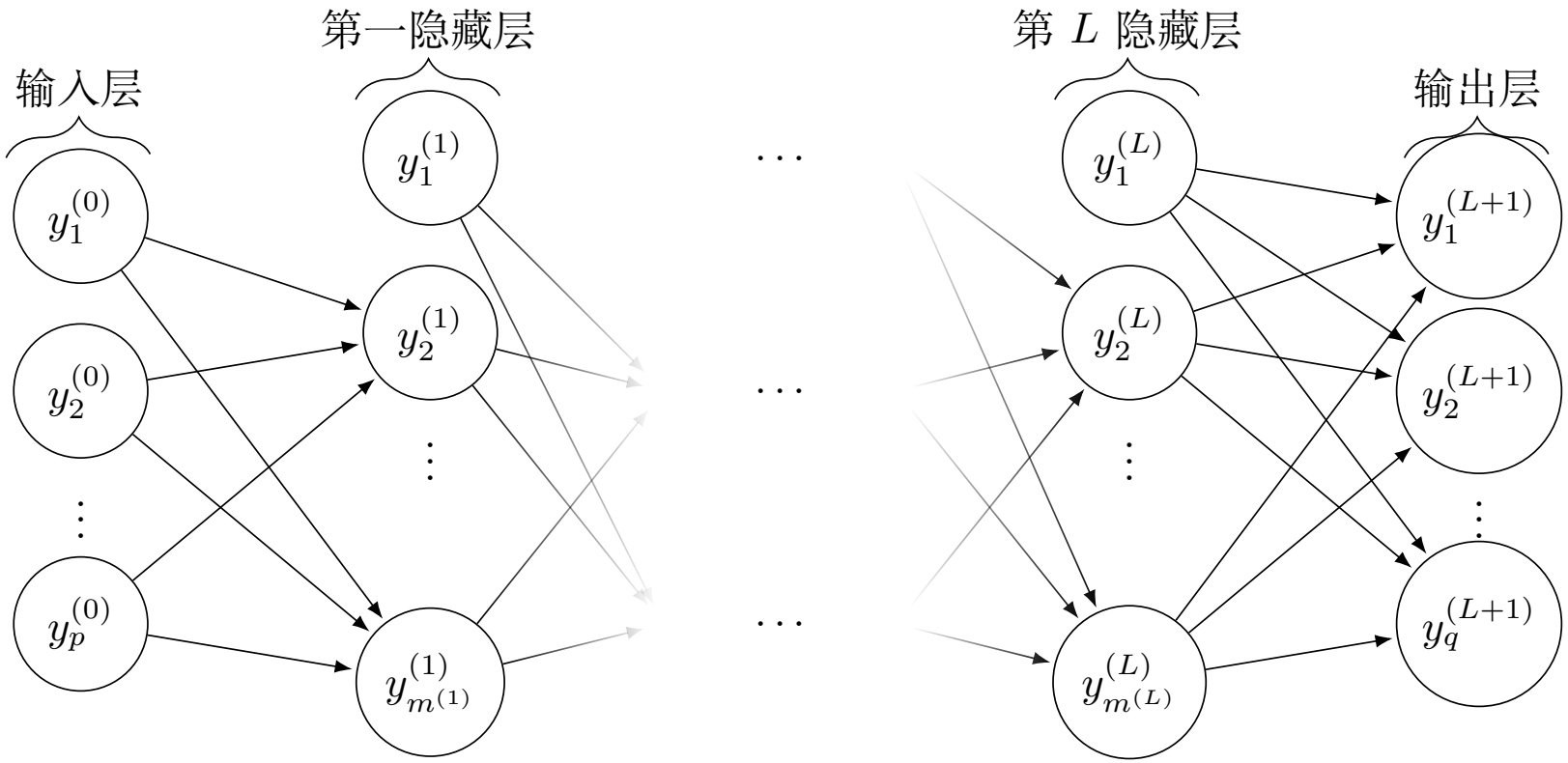


Figure: 带p个输入单元和q个输出单位的$(L+2)$层感知机的网络图，第l个隐藏层包含$m^{(l)}$个神经元.

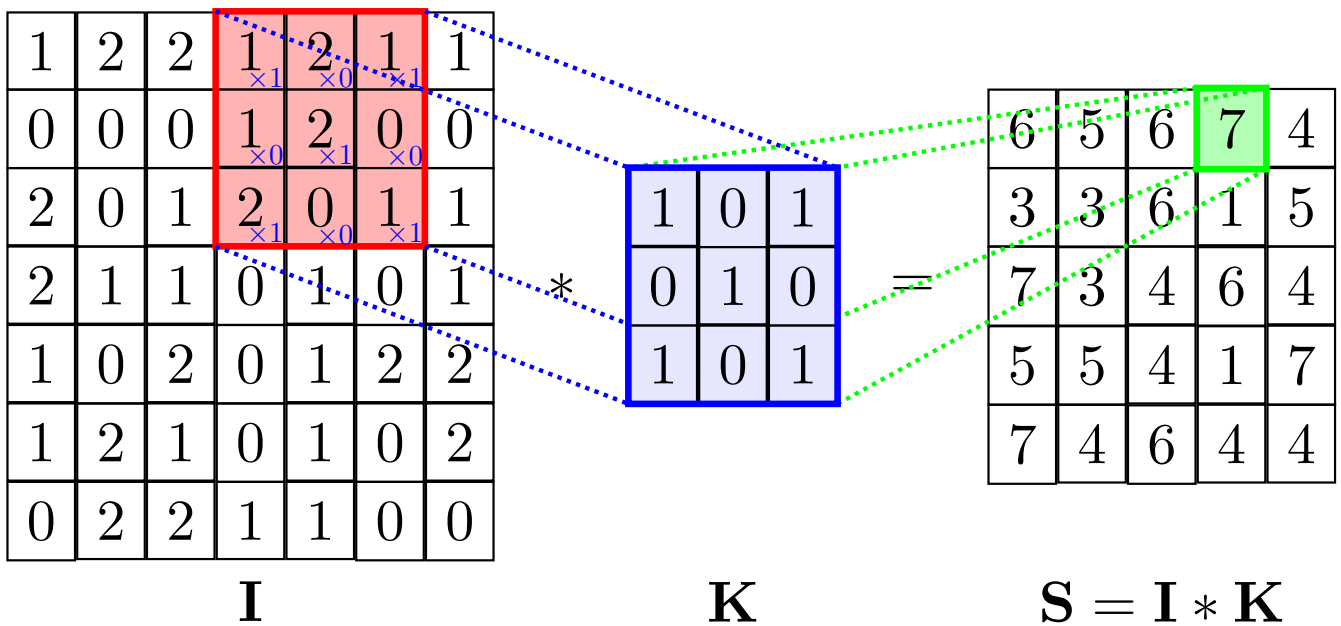
卷积神经网络Convolutional neural network (CNN)

- 给定二维图像$\boldsymbol{I} \in \mathbb{R}^{n\times n}$和卷积核$\boldsymbol{K} \in \mathbb{R}^{k\times k}$，定义卷积操作$\boldsymbol{S} = \boldsymbol{I} * \boldsymbol{K}$，它的元素是

$$\boldsymbol{S}_{i,j} = \langle \boldsymbol{I}(i : i+k-1, j : j+k-1), \boldsymbol{K} \rangle,$$

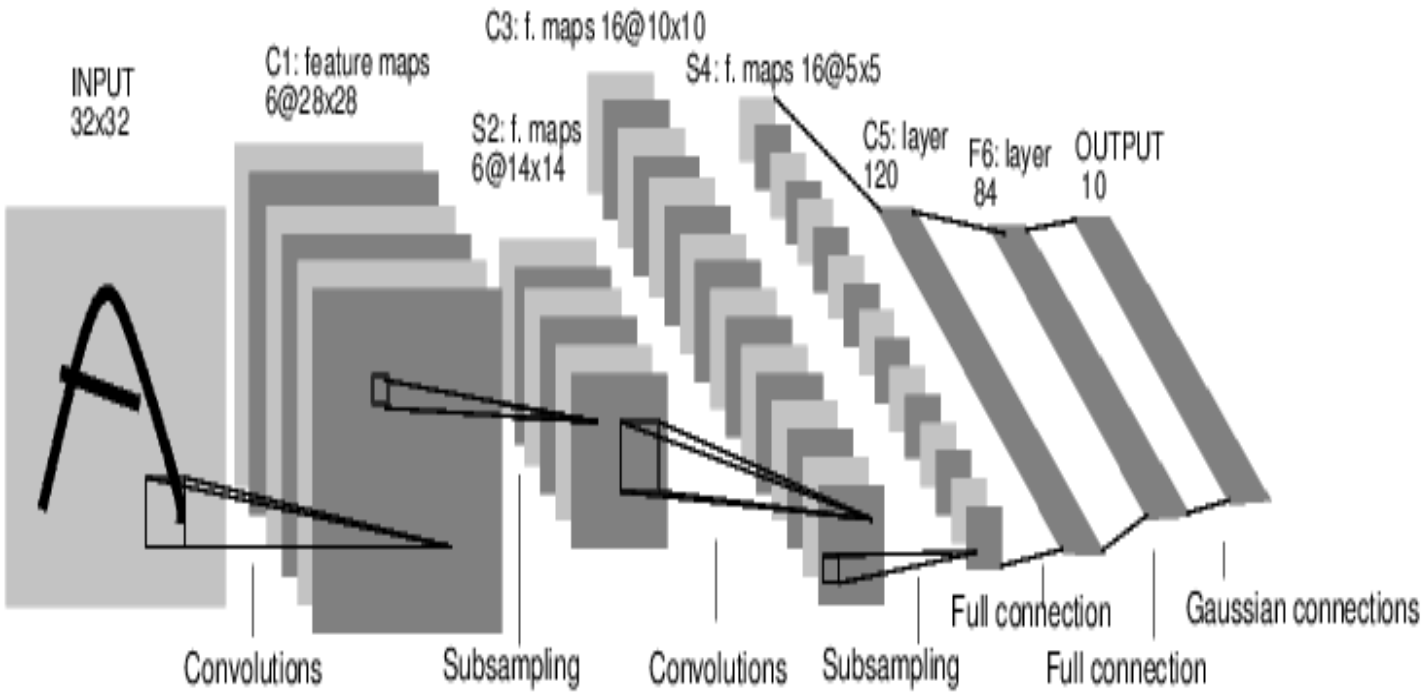
其中两个矩阵X, Y的内积是它们相应元素乘积之和

- 生成的结果$\boldsymbol{S}$可以根据卷积核的维数、$\boldsymbol{I}$的边界是否填充、卷积操作时滑动的大小等相应变化.



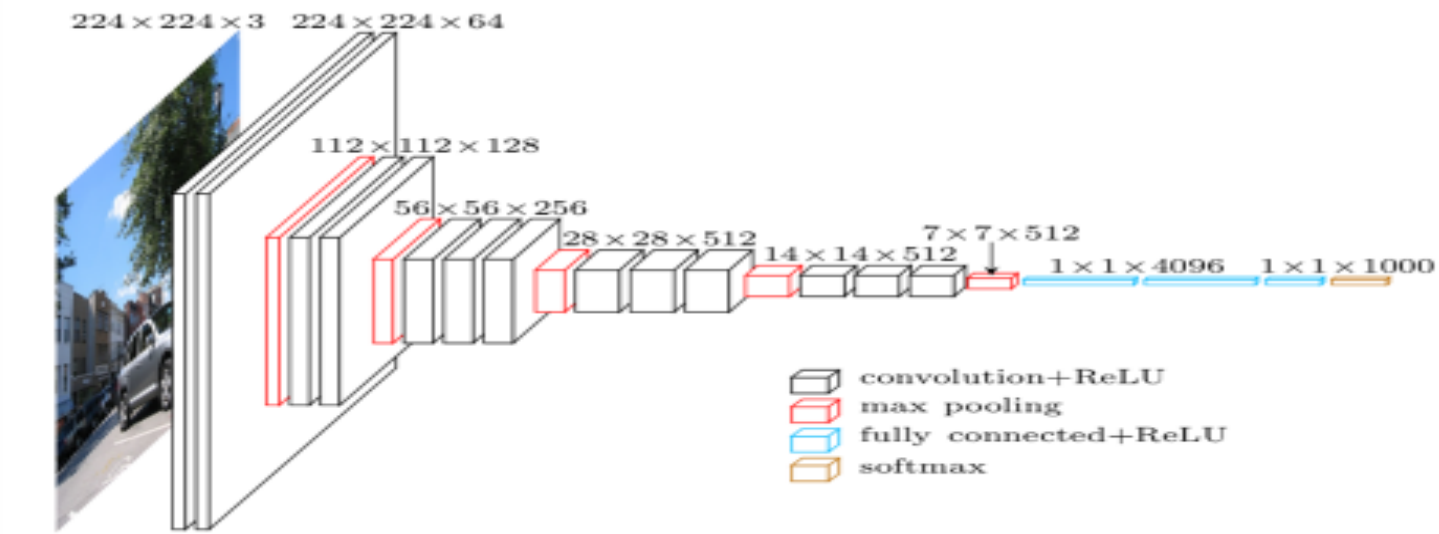
卷积神经网络Convolutional neural network (CNN)

LeCun等人开创性的建立了数字分类的神经网络。几家银行使用它来识别支票上的手写数字。



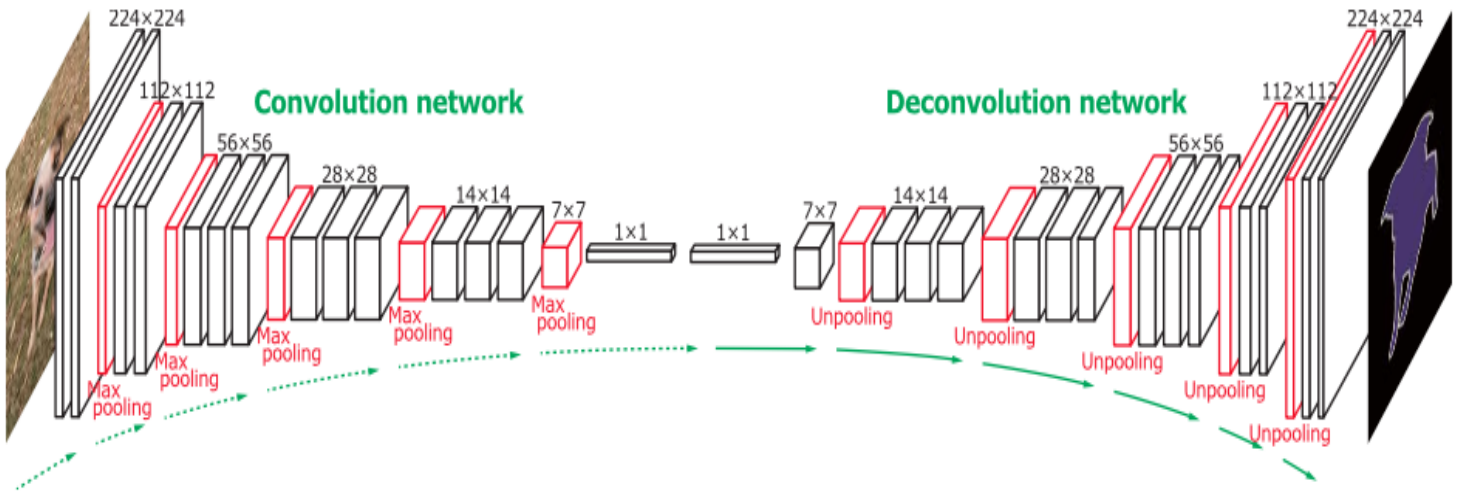
VGGNet

VGG小组（牛津）给出的19层卷积神经网络比AlexNet更简单层数更深。AlexNet中的所有大型过滤器都被3x3 过滤器的级联所取代（中间是非线性的）。在进行两次或三次卷积后放置最大合并，每次合并之后，过滤器的数量总是增加一倍。



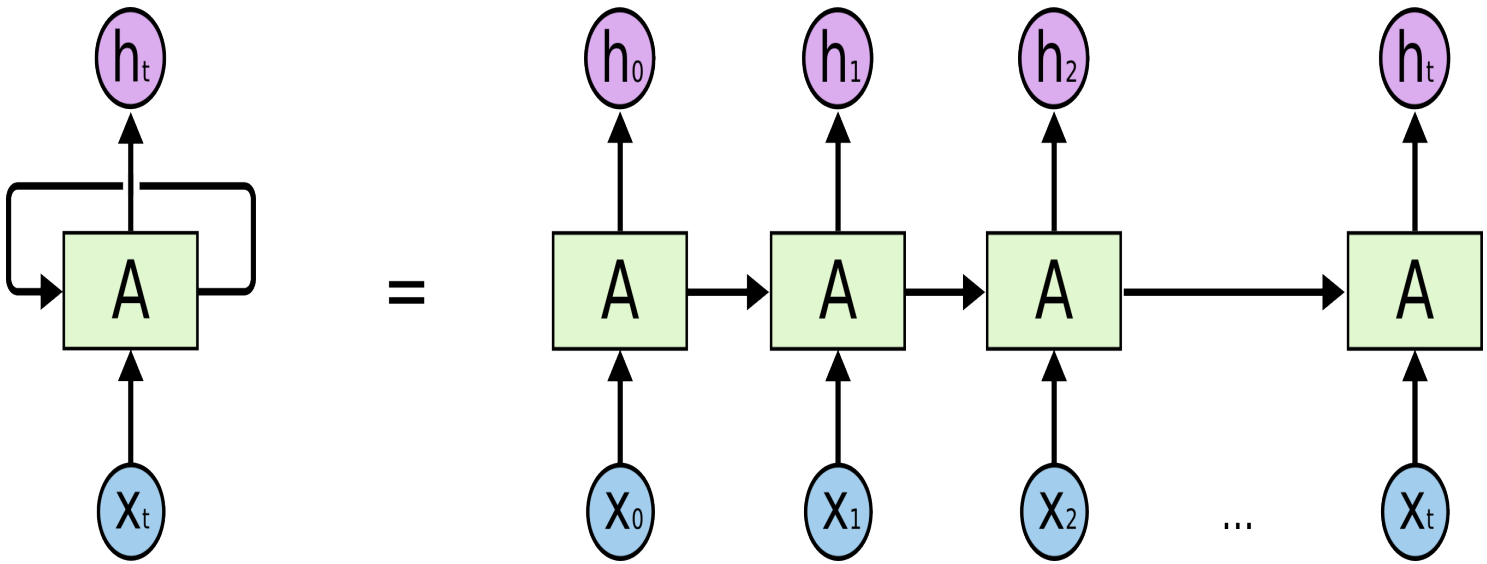
反卷积网络

生成网络是一种特殊的卷积网络，它使用转置卷积，也称为反卷积层。



递归神经网络Recurrent neural networks (RNN)

RNN建立在与前馈神经网络相同的计算单元上。RNN不必分层组织，并且允许定向循环。这样一来，他们就可以拥有内部存储器，从而可以处理顺序数据。如图所示，可以将RNN按顺序“展开”，从而将RNN转换为常规前馈神经网络。



深度学习中的优化算法

随机梯度类算法

- pytorch/caffe2 里实现的算法有adadelta, adagrad, adam, nesterov, rmsprop, YellowFin
  `https://github.com/pytorch/pytorch/tree/master/caffe2/sgd`
- pytorch/torch 里有：sgd, asgd, adagrad, rmsprop, adadelta, adam, adamax
  `https://github.com/pytorch/pytorch/tree/master/torch/optim`
- tensorflow 实现的算法有：Adadelta, AdagradDA, Adagrad, ProximalAdagrad, Ftrl, Momentum, adam, Momentum, CenteredRMSProp
  具体实现:
  `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/kernels/training_ops.cc`

深度学习中的随机优化算法

考虑问题$\min_{x\in\mathbb{R}^d}\frac{1}{n}\sum_{i=1}^n f_i(x)$

参考: chapter 8 in

http://www.deeplearningbook.org/

- 梯度下降

$$x^{t+1}=x^t-\frac{\alpha^t}{n}\sum_{i=1}^n\nabla f_i(x^t)$$

- 随机梯度算法SGD

$$x^{k+1}=x^t-\alpha^t\nabla f_i(x^t)$$

- 带动量项的SGD

$$v^{t+1}=\mu^t v^t-\alpha^t\nabla f_i(x^t)$$
$$x^{t+1}=x^t+v^{t+1}$$

提纲

最优化算法

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，在问题(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，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，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，可以来判别模型构造是否合理或者进一步地改进模型.

优化算法设计

在设计优化算法时，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：

- **泰勒（Taylor）展开**对于一个非线性的目标或者约束函数，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，所以需要根据迭代点的更新来重新构造相应的简化问题．
- **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，可以简化原始问题的求解，从而设计更有效的算法．

- **拆分**对于一个复杂的优化问题，可以将变量进行拆分，比如$\min_x h(x)+r(x)$，可以拆分成

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

  通过引入更多的变量，则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.

- **块坐标下降**对于一个n维空间（n 很大）的优化问题，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$n=100$，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.

凸和非凸优化问题

- 凸优化问题是指最小化问题(1) 中的目标函数和可行域分别是凸函数和凸集. 如果其中有任何一个不是凸的，则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.
- 若问题(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，经常更倾向于得到一个凸优化模型. 除此之外,判断一个问题是否是凸问题也很重要. 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.

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) 的最优解的定义.

定义 (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$f(\bar{x}) \leq f(x), \forall x \in \mathcal{X}$，那么称$\bar{x}$ 为问题(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$f(\bar{x}) \leq f(x), \forall x \in N_\varepsilon(\bar{x}) \cap \mathcal{X}$, 那么称$\bar{x}$ 为问题(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$f(\bar{x}) < f(x), \forall x \in N_\varepsilon(\bar{x}) \cap \mathcal{X}$, 且$x \neq \bar{x}$ 成立，则称$\bar{x}$ 为问题(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，则称为**非严格局部极小解**. 在图5 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.

全局和局部最优解

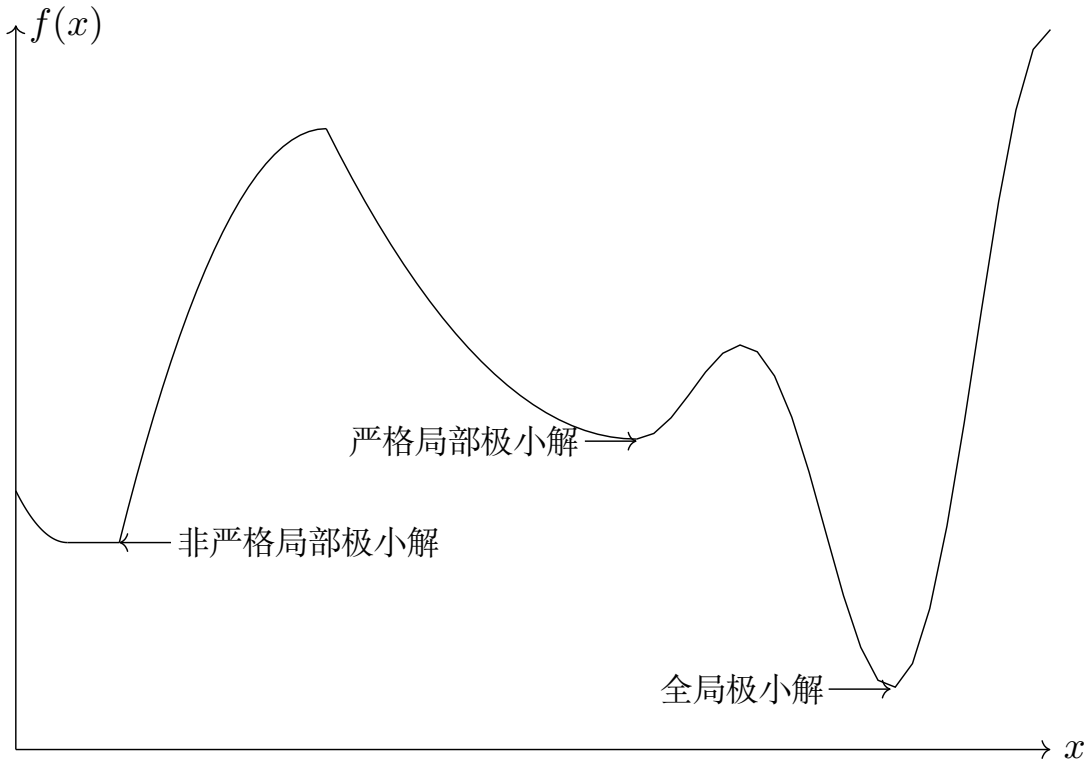


Figure: 函数的全局极小、严格局部极小和非严格局部极小解

在问题(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.

优化算法收敛性

由于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.
- 在算法设计中，还需考虑算法产生的点列是否收敛到优化问题的解. 给定初始点x^0，记算法迭代产生的点列为$\{x^k\}$. 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.

优化算法收敛性

- 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．记对应的函数值序列$\{f(x^k)\}$，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．

算法的渐进收敛速度

设$\{x^k\}$为算法产生的迭代点列且收敛于x^*

- 算法（点列）**Q-线性收敛**：对充分大的k有

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

- 算法（点列）**Q-超线性收敛**：对充分大的k有

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

- 算法（点列）**Q-次线性收敛**：对充分大的k有

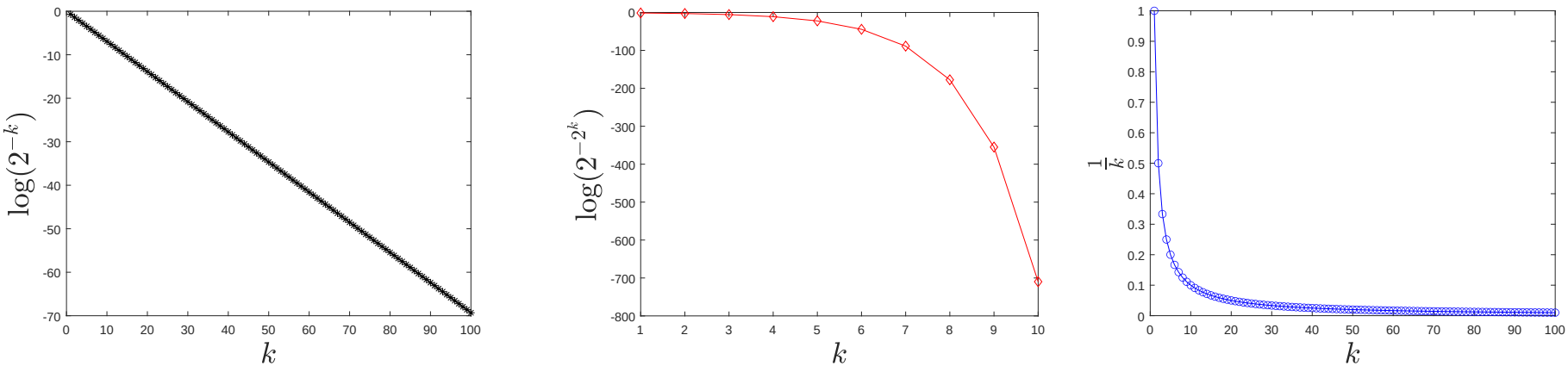
$$\lim_{k\to\infty}\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|}=1,$$

- 算法（点列）**Q-二次收敛**：对充分大的k有

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

算法的渐进收敛速度

我们举例来更直观地展示不同的Q-收敛速度，参见下图（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$\{\frac{1}{k}\}$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

Figure: 不同Q-收敛速度比较

算法的渐进收敛速度

- 算法（点列）**R-线性收敛**: 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

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

对任意的k成立．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$\mathcal{O}(t_k)$.

- **算法复杂度**．设x^*为全局极小点，某一算法产生的迭代序列$\{x^k\}$满足

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

其中$c > 0$为常数。如果需要计算算法满足精度$f(x^k) - f(x^*) \leq \varepsilon$所需的迭代次数，只需令$\frac{c}{\sqrt{k}} \leq \varepsilon$ 则得到$k \geq \frac{c^2}{\varepsilon^2}$，因此该优化算法对应的（迭代次数）复杂度为$N(\varepsilon) = \mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$.

优化算法的收敛准则

为了使算法能在有限步内终止，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

$$\frac{f(x^k)-f^*}{\max\{|f^*|,1\}} \le \varepsilon_1, \quad \|\nabla f(x^k)\| \le \varepsilon_2, \tag{11}$$

其中$\varepsilon_1, \varepsilon_2$为给定的很小的正数，$\|\cdot\|$表示某种范数（这里可以简单理解为$\ell_2$范数：$\|x\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$，$f^*$为函数$f$的最小值以及$\nabla f(x^k)$表示函数$f$在点$x$处的梯度.

- 对于约束优化问题，还需要考虑约束违反度. 也即要求最后得到的点满足

$$c_i(x^k) \le \varepsilon_3, \ i = 1, 2, \cdots, m,$$
$$|c_i(x^k)| \le \varepsilon_4, \ i = m+1, m+2, \cdots, m+l,$$

其中$\varepsilon_3, \varepsilon_4$为很小的正数，用来刻画$x^k$的可行性.

优化算法的收敛准则

- 除了约束违反度之外，也要考虑x^k与最优解之间的距离，如(11)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，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．
- 对于具体的算法，根据其设计的出发点，不一定能得到一个高精度的逼近解．为了避免无用的计算开销，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

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

其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．