第十二章 优化算法

第 36 讲 复合优化算法

黄定江

DaSE @ ECNU
djhuang@dase.ecnu.edu.cn

1. 36.1 近似点梯度法
2. 36.2 分块坐标下降法
3. 36.3 交替方向乘子法
4. 36.4 随机梯度下降
5. 36.5 动量梯度下降
6. 36.6 自适应学习速率
7. 36.7 在线凸优化算法简介

本节主要考虑如下复合优化问题：

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \psi(\boldsymbol{x}) \overset{\text{def}}{=} f(\boldsymbol{x}) + h(\boldsymbol{x}), \tag{1}$$

其中 $f(\boldsymbol{x})$ 为可微函数（可能非凸），$h(\boldsymbol{x})$ 可能为不可微函数.

- 出现在很多应用领域中，例如压缩感知、图像处理、机器学习等
- 光滑化的思想处理不可微项 $h(\boldsymbol{x})$，但这种做法没有充分利用 $h(\boldsymbol{x})$ 的性质
- 引人针对该问题的近似点梯度法和 Nesterov 加速算法

36.1.1 邻近算子

邻近算子是处理非光滑问题的一个非常有效的工具，也与许多算法的设计密切相关. 首先给出邻近算子的定义.

定义 1

（邻近算子）对于一个凸函数 h，定义它的邻近算子为

$$\operatorname{prox}_h(\boldsymbol{x}) = \underset{\boldsymbol{u}\in\operatorname{dom} h}{\arg\min}\left\{h(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u}-\boldsymbol{x}\|^2\right\}.$$

可以看到，邻近算子的目的是求解一个距 $\boldsymbol{x}$ 不算太远的点，并使函数值 $h(\boldsymbol{x})$ 也相对较小.

36.1.2 邻近算子的存在唯一性

一个很自然的问题是，上面给出的邻近算子的定义是不是有意义的，即定义中的优化问题的解是不是存在唯一的. 下面的定理将给出定义中优化问题解的存在唯一性.

定理 1

（存在唯一性）如果 h 是适当的闭凸函数，则对任意的 $\boldsymbol{x} \in \mathbb{R}^n$，$\mathrm{prox}_h(\boldsymbol{x})$ 的值存在且唯一.

这样我们就可使用邻近算子去构建迭代格式.

36.1.3 邻近算子与次梯度

另外，根据最优性条件可以得到如下等价结论：

定理 2

（邻近算子与次梯度的关系）如果 h 是适当的闭凸函数，则

$$\boldsymbol{u} = \operatorname{prox}_h(\boldsymbol{x}) \Longleftrightarrow \boldsymbol{x} - \boldsymbol{u} \in \partial h(\boldsymbol{u})$$

证明.

若 $\boldsymbol{u}=\operatorname{prox}_h(\boldsymbol{x})$, 则由最优性条件得 $0\in\partial h(\boldsymbol{u})+(\boldsymbol{u}-\boldsymbol{x})$, 因此有 $\boldsymbol{x}-\boldsymbol{u}\in\partial h(\boldsymbol{u})$. 反之, 若 $\boldsymbol{x}-\boldsymbol{u}\in\partial h(\boldsymbol{u})$ 则由次梯度的定义可得到

$$h(\boldsymbol{v})\geqslant h(\boldsymbol{u})+(\boldsymbol{x}-\boldsymbol{u})^{\mathrm{T}}(\boldsymbol{v}-\boldsymbol{u}),\quad \forall \boldsymbol{v}\in\operatorname{dom} h.$$

两边同时加 $\frac{1}{2}\|\boldsymbol{v}-\boldsymbol{x}\|^2$, 即有

$$\begin{aligned} h(\boldsymbol{v})+\frac{1}{2}\|\boldsymbol{v}-\boldsymbol{x}\|^2 &\geqslant h(\boldsymbol{u})+(\boldsymbol{x}-\boldsymbol{u})^{\mathrm{T}}(\boldsymbol{v}-\boldsymbol{u})+\frac{1}{2}\|\boldsymbol{v}-\boldsymbol{x}\|^2 \\ &\geqslant h(\boldsymbol{u})+\frac{1}{2}\|\boldsymbol{u}-\boldsymbol{x}\|^2,\quad \forall \boldsymbol{v}\in\operatorname{dom} h \end{aligned}$$

因此我们得到 $\boldsymbol{u}=\operatorname{prox}_h(\boldsymbol{x})$. □

下面给出一些常见的 ℓ_1 范数和 ℓ_2 范数对应的邻近算子计算例子.

例 1

在下面所有例子中, 常数 $t>0$ 为正实数. ℓ_1 范数:

$$h(\boldsymbol{x})=\|\boldsymbol{x}\|_1, \quad \operatorname{prox}_{th}(\boldsymbol{x})=\operatorname{sign}(\boldsymbol{x})\max\{|\boldsymbol{x}|-t,0\}.$$

证明.

因为求解 ℓ_1 范数的邻近算子，所对应的优化问题是可以拆分的。因此，我们只需要考虑一维的情形。易知，邻近算子 $u=\operatorname{prox}_{th}(x)$ 的最优性条件为

$$x-u\in t\partial|u|=\begin{cases}\{t\}, & u>0,\\ [-t,t], & u=0,\\ \{-t\}, & u<0,\end{cases}$$

因此, 当 $x>t$ 时, $u=x-t$; 当 $x<-t$ 时, $u=x+t$; 当 $x\in[-t,t]$ 时, $u=0$, 即有 $u=\operatorname{sign}(x)\max\{|x|-t,0\}$. □

例 2

常数 $t>0$ 为正实数. ℓ_2 范数:

$$h(\boldsymbol{x})=\|\boldsymbol{x}\|_2,\quad \operatorname{prox}_{th}(\boldsymbol{x})=\begin{cases}\left(1-\frac{t}{\|\boldsymbol{x}\|_2}\right)\boldsymbol{x}, & \|\boldsymbol{x}\|_2\geqslant t,\\ 0, & \text{其他}.\end{cases}$$

证明.

邻近算子 $\boldsymbol{u}=\operatorname{prox}_{th}(\boldsymbol{x})$ 的最优性条件为

$$\boldsymbol{x}-\boldsymbol{u}\in t\partial\|\boldsymbol{u}\|_2=\begin{cases}\left\{\frac{t\boldsymbol{u}}{\|\boldsymbol{u}\|_2}\right\}, & \boldsymbol{u}\neq 0,\\ \{\boldsymbol{w}:\|\boldsymbol{w}\|_2\leqslant t\}, & \boldsymbol{u}=0,\end{cases}$$

因此, 当 $\|\boldsymbol{x}\|_2>t$ 时, $\boldsymbol{u}=\boldsymbol{x}-\frac{t\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$; 当 $\|\boldsymbol{x}\|_2\leqslant t$ 时, $\boldsymbol{u}=0$. □

另外一种比较常用的邻近算子是关于示性函数的邻近算子. 集合 C 的示性函数定义为

$$I_C(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in C, \\ +\infty, & \text{其他}, \end{cases}$$

它可以用来把约束变成目标函数的一部分.

例 3

(闭凸集上的投影) 设 C 为 $\mathbb{R}^n$ 上的闭凸集, 则示性函数 I_C 的邻近算子为点 $\boldsymbol{x}$ 到集合 C 的投影, 即

$$\begin{aligned} \operatorname{prox}_{I_C}(\boldsymbol{x}) &= \arg\min_{u} \left\{ I_C(\boldsymbol{u}) + \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{x}\|^2 \right\} \\ &= \arg\min_{\boldsymbol{u} \in C} \|\boldsymbol{u} - \boldsymbol{x}\|^2 = \mathcal{P}_C(\boldsymbol{x}). \end{aligned}$$

36.1.4 近似点梯度法

近似点梯度法的思想非常简单：注意到 $\psi(x)$ 有两部分，对于光滑部分 f 做梯度下降，对于非光滑部分 h 使用邻近算子，则近似点梯度法的迭代公式为

$$\boldsymbol{x}^{k+1} = \operatorname{prox}_{t_k h}\left(\boldsymbol{x}^k - t_k \nabla f\left(\boldsymbol{x}^k\right)\right),$$

其中 $t_k > 0$ 为每次迭代的步长，它可以是一个常数或者由线搜索得出.

近似点梯度法跟众多算法都有很强的联系，在一些特定条件下，近似点梯度法还可以转化为其他算法：当 $h(\boldsymbol{x})=0$ 时，迭代公式变为梯度下降法

$$\boldsymbol{x}^{k+1}=\boldsymbol{x}^{k}-t_k\nabla f\left(\boldsymbol{x}^{k}\right)$$

当 $h(\boldsymbol{x})=I_C(\boldsymbol{x})$ 时，迭代公式变为投影梯度法

$$\boldsymbol{x}^{k+1}=\mathcal{P}_C\left(\boldsymbol{x}^{k}-t_k\nabla f\left(\boldsymbol{x}^{k}\right)\right).$$

近似点梯度法可以总结为:

算法 1 近似点梯度法

Require: 函数 $f(\boldsymbol{x}), h(\boldsymbol{x})$, 初始点 $\boldsymbol{x}^0$. 初始化 $k=0$.

1: **while** 未达到停止准则 **do**
2: $\quad \boldsymbol{x}^{k+1} = \operatorname{prox}_{t_k h}\left(\boldsymbol{x}^k - t_k \nabla f\left(\boldsymbol{x}^k\right)\right)$.
3: $\quad k \leftarrow k+1$.
4: **end while**

如何理解近似点梯度法？根据邻近算子的定义，把迭代公式展开：

$$\begin{aligned}\boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{u}} \left\{ h(\boldsymbol{u}) + \frac{1}{2t_k} \left\| \boldsymbol{u} - \boldsymbol{x}^k + t_k \nabla f\left(\boldsymbol{x}^k\right) \right\|^2 \right\} \\ &= \arg\min_{\boldsymbol{u}} \left\{ h(\boldsymbol{u}) + f\left(\boldsymbol{x}^k\right) + \nabla f\left(\boldsymbol{x}^k\right)^{\mathrm{T}} \left(\boldsymbol{u} - \boldsymbol{x}^k\right) + \frac{1}{2t_k} \left\| \boldsymbol{u} - \boldsymbol{x}^k \right\|^2 \right\},\end{aligned}$$

可以发现，近似点梯度法实质上就是将问题的光滑部分线性展开，再加上二次项并保留非光滑部分，然后求极小来作为每一步的估计.

36.1.5 Nesterov 加速

- 上一小节介绍了近似点梯度算法，如果光滑部分的梯度是利普希茨连续的，则它的收敛速度可以达到 $\mathcal{O}\left(\frac{1}{k}\right)$.
- 一个自然的问题是如果仅用梯度信息，我们能不能取得更快的收敛速度.
- Nesterov 分别在 1983 年、1988 年和 2005 年提出了三种改进的一阶算法，收敛速度能达到 $\mathcal{O}\left(\frac{1}{k^2}\right)$.
- 实际上，这三种算法都可以应用到近似点梯度算法上. 限于篇幅，我们这里仅介绍其中一种加速算法。

36.1.6 FISTA 算法

Beck 和 Teboulle 就在 2008 年给出了 Nesterov 在 1983 年提出的算法的近似点梯度法版本—FISTA。FISTA 算法由两步组成：第一步沿着前两步的计算方向计算一个新点，第二步在该新点处做一步近似点梯度迭代，即

$$\begin{aligned}\boldsymbol{y}^k &= \boldsymbol{x}^{k-1} + \frac{k-2}{k+1}\left(\boldsymbol{x}^{k-1} - \boldsymbol{x}^{k-2}\right),\\ \boldsymbol{x}^k &= \operatorname{prox}_{t_k h}\left(\boldsymbol{y}^k - t_k \nabla f\left(\boldsymbol{y}^k\right)\right).\end{aligned}$$

FISTA 算法

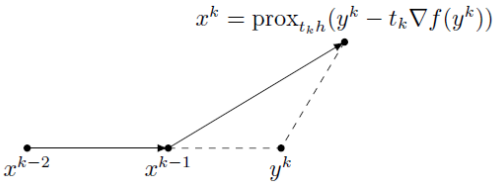

图 1: FISTA 算法迭代图示

从图 1 中可以直观地看出 FISTA 算法的迭代情况. 可以得到这一做法对每一步迭代的计算量几乎没有影响, 而带来的效果是显著的.

FISTA 算法

完整的 FISTA 算法如下：

算法 2 FISTA 算法

Require: $\boldsymbol{x}^0 = \boldsymbol{x}^{-1} \in \mathbb{R}^n, k \leftarrow 1$.

1: **while** 未达到停止准则 **do**
2: 计算 $\boldsymbol{y}^k = \boldsymbol{x}^{k-1} + \frac{k-2}{k+1}\left(\boldsymbol{x}^{k-1} - \boldsymbol{x}^{k-2}\right)$,
3: 选取 $t_k = t \in \left(0, \frac{1}{L}\right]$, 计算 $\boldsymbol{x}^k = \mathrm{prox}_{t_k h}\left(\boldsymbol{y}^k - t_k \nabla f\left(\boldsymbol{y}^k\right)\right)$.
4: $k \leftarrow k + 1$
5: **end while**

为了对算法做更好的推广, 可以给出 FISTA 算法的一个等价变形, 只是把原来算法中的第一步拆成两步迭代, 相应算法如下所示. 当 $\gamma_k = \frac{2}{k+1}$ 时, 并且取固定步长时, 两个算法是等价的.

算法 3 FISTA 等价变形

Require: $\boldsymbol{v}_0 = \boldsymbol{x}_0 \in \mathbb{R}^n, k \leftarrow 1$.

1: **while** 未达到停止准则 **do**
2: 计算 $\boldsymbol{y}^k = (1-\gamma_k)\,\boldsymbol{x}^{k-1} + \gamma_k \boldsymbol{v}^{k-1}$.
3: 选取 t_k, 计算 $\boldsymbol{x}^k = \mathrm{prox}_{t_k h}\left(\boldsymbol{y}^k - t_k \nabla f\left(\boldsymbol{y}^k\right)\right)$.
4: 计算 $\boldsymbol{v}^k = \boldsymbol{x}^{k-1} + \frac{1}{\gamma_k}\left(\boldsymbol{x}^k - \boldsymbol{x}^{k-1}\right)$.
5: $k \leftarrow k+1$.
6: **end while**

36.1.7 应用：近似点梯度法求低秩矩阵恢复

考虑将等式约束转化为二次罚函数项的低秩矩阵恢复模型：

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

其中 $\boldsymbol{M}$ 是想要恢复的低秩矩阵，但是只知道其在下标集 Ω 上的值. 令

$$f(\boldsymbol{X}) = \frac{1}{2}\sum_{(i,j)\in\Omega}(X_{ij}-M_{ij})^2, \quad h(\boldsymbol{X}) = \mu\|\boldsymbol{X}\|_*,$$

定义矩阵 $\boldsymbol{P} \in \mathbb{R}^{m\times n}$ **:**

$$P_{ij} = \begin{cases} 1, & (i,j) \in \Omega, \\ 0, & \text{其他}, \end{cases}$$

则

$$\begin{aligned} f(\boldsymbol{X}) &= \frac{1}{2}\|\boldsymbol{P} \odot (\boldsymbol{X} - \boldsymbol{M})\|_F^2, \\ \nabla f(\boldsymbol{X}) &= \boldsymbol{P} \odot (\boldsymbol{X} - \boldsymbol{M}), \\ \operatorname{prox}_{t_k h}(\boldsymbol{X}) &= \boldsymbol{U}\operatorname{Diag}\left(\max\left\{|\boldsymbol{d}| - t_k\mu, 0\right\}\right)\boldsymbol{V}^{\mathrm{T}}, \end{aligned}$$

其中 $\boldsymbol{X} = \boldsymbol{U}\operatorname{Diag}(\boldsymbol{d})\boldsymbol{V}^{\mathrm{T}}$ **为矩阵** $\boldsymbol{X}$ **的约化的奇异值分解**.

则近似点梯度法的迭代格式为

$$\boldsymbol{Y}^k = \boldsymbol{X}^k - t_k \boldsymbol{P} \odot \left(\boldsymbol{X}^k - \boldsymbol{M} \right),$$

$$\boldsymbol{X}^{k+1} = \operatorname{prox}_{t_k h}\left(\boldsymbol{Y}^k \right)$$

对应的加速算法 FISTA 的迭代格式为

$$\boldsymbol{Y}^k = \boldsymbol{X}^{k-1} + \frac{k-2}{k+1}\left(\boldsymbol{X}^{k-1} - \boldsymbol{X}^{k-2} \right) \boldsymbol{Z}^k = \boldsymbol{Y}^k - t_k \boldsymbol{P} \odot \left(\boldsymbol{Y}^k - \boldsymbol{M} \right),$$

$$\boldsymbol{X}^{k+1} = \operatorname{prox}_{t_k h}\left(\boldsymbol{Z}^k \right)$$

在许多实际的优化问题中, 人们所考虑的目标函数虽然有成千上万的自变量, 对这些变量联合求解目标函数的极小值通常很困难, 但有些情形自变量具有某种 “可分离” 的形式. 分块坐标下降法 (block coordinate descent, BCD) 正是利用了这样的思想来求解这种具有特殊结构的优化问题. 考虑具有如下形式的问题:

$$\min_{\boldsymbol{x}\in\mathcal{X}} F(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_s) = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_s) + \sum_{i=1}^{s} r_i(\boldsymbol{x}_i), \tag{2}$$

其中 $\mathcal{X}$ 是函数的可行域, 这里将自变量 $\boldsymbol{x}$ 拆分成 s 个变量块 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_s$, 每个变量块 $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$. 函数 f 是关于 $\boldsymbol{x}$ 的可微函数, 每个 $r_i(\boldsymbol{x}_i)$ 关于 $\boldsymbol{x}_i$ 是适当的闭凸函数, 但不一定可微.

36.2.1 具有“可分离”形式的实际问题

例 4

（聚类问题）前面我们介绍了 K-均值聚类问题的等价形式:

$$\begin{aligned} \min_{\boldsymbol{\Phi},\boldsymbol{H}} \quad & \|\boldsymbol{A}-\boldsymbol{\Phi}\boldsymbol{H}\|_{F^v}^2 \\ s.t. \quad & \boldsymbol{\Phi}\in\mathbb{R}^{n\times k}, \text{ 每一行只有一个元素为}1, \text{ 其余为}0, \\ & \boldsymbol{H}\in\mathbb{R}^{k\times p}. \end{aligned}$$

这是一个矩阵分解问题, 自变量总共有两块. 注意到变量 $\boldsymbol{\Phi}$ 取值在离散空间上, 因此聚类问题不是凸问题.

例 5

（非负矩阵分解）设 $\boldsymbol{M}$ 是已知矩阵，考虑求解如下极小化问题：

$$\min_{\boldsymbol{X},\boldsymbol{Y}\geqslant 0} \frac{1}{2}\|\boldsymbol{X}\boldsymbol{Y}-\boldsymbol{M}\|_F^2+\alpha r_1(\boldsymbol{X})+\beta r_2(\boldsymbol{X}).$$

在这个例子中自变量共有两块，且均有非负的约束.

- 上述的所有例子中，函数 f 关于变量全体一般是非凸的，这使得求解问题 (2) 变得很有挑战性.
- 首先，应用在非凸问题上的算法的收敛性不易分析，很多针对凸问题设计的算法通常会失效;
- 其次，目标函数的整体结构十分复杂，这使得变量的更新需要很大计算量.
- 对于这类问题，我们最终的目标是要设计一种算法，它具有简单的变量更新格式，同时具有一定的（全局）收敛性. 而分块坐标下降法则是处理这类问题较为有效的算法.

36.2.2 分块坐标下降法

考虑问题 (2), 分块坐标下降法主要思想: 按照 $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_s$ 的次序依次固定其他 $(s-1)$ 块变量极小化 F, 完成一块变量的极小化后, 它的值便立即被更新到变量空间中, 更新下一块变量时将使用每个变量最新的值.

根据这种更新方式定义辅助函数

$$f_i^k(\boldsymbol{x}_i) = f\left(\boldsymbol{x}_1^k, \cdots, \boldsymbol{x}_{i-1}^k, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}^{k-1}, \cdots, \boldsymbol{x}_s^{k-1}\right),$$

其中 $\boldsymbol{x}_j^k$ 表示在第 k 次迭代中第 j 块自变量的值, $\boldsymbol{x}_i$ 是函数的自变量.

在每一步更新中，通常使用以下三种更新格式之一：

$$\boldsymbol{x}_i^k = \underset{\boldsymbol{x}_i \in \mathcal{X}_i^k}{\arg\min} \left\{ f_i^k\left(\boldsymbol{x}_i\right) + r_i\left(\boldsymbol{x}_i\right) \right\}, \tag{3}$$

$$\boldsymbol{x}_i^k = \underset{\boldsymbol{x}_i \in \mathcal{X}_i^k}{\arg\min} \left\{ f_i^k\left(\boldsymbol{x}_i\right) + \frac{L_i^{k-1}}{2} \left\| \boldsymbol{x}_i - \boldsymbol{x}_i^{k-1} \right\|_2^2 + r_i\left(\boldsymbol{x}_i\right) \right\} \tag{4}$$

$$\boldsymbol{x}_i^k = \underset{\boldsymbol{x}_i \in \mathcal{X}_i^k}{\arg\min} \left\{ \left\langle \hat{\boldsymbol{g}}_i^k, \boldsymbol{x}_i - \hat{\boldsymbol{x}}_i^{k-1} \right\rangle + \frac{L_i^{k-1}}{2} \left\| \boldsymbol{x}_i - \hat{\boldsymbol{x}}_i^{k-1} \right\|_2^2 + r_i\left(\boldsymbol{x}_i\right) \right\}, \tag{5}$$

其中 $L_i^k > 0$ 为常数，

$$\mathcal{X}_i^k = \left\{ \boldsymbol{x} \in \mathbb{R}^{n_i} \mid \left(\boldsymbol{x}_1^k, \cdots, \boldsymbol{x}_{i-1}^k, \boldsymbol{x}, \boldsymbol{x}_{i+1}^{k-1}, \cdots, \boldsymbol{x}_s^{k-1} \right) \in \mathcal{X} \right\}.$$

在更新格式 (5) 中，$\hat{\boldsymbol{x}}_i^{k-1}$ 采用外推定义：

$$\hat{\boldsymbol{x}}_i^{k-1} = \boldsymbol{x}_i^{k-1} + \omega_i^{k-1} \left(\boldsymbol{x}_i^{k-1} - \boldsymbol{x}_i^{k-2} \right),$$

其中 $\omega_i^k \geqslant 0$ 为外推的权重，$\hat{\boldsymbol{g}}_i^k \overset{\text{def}}{=} \nabla f_i^k\left(\hat{\boldsymbol{x}}_i^{k-1}\right)$ 为外推点处的梯度.

我们可以通过如下的方式理解这三种格式:

- 格式 (3) 是最直接的, 即固定其他分量然后对单一变量求极小;
- 格式 (4) 则是增加了一个近似点项 $\frac{L_i^{k-1}}{2}\left\|\boldsymbol{x}_i-\boldsymbol{x}_i^{k-1}\right\|_2^2$ 来限制下一步迭代不应该与当前位置相距过远, 增加近似点项的作用是使得算法能够收敛;
- 格式 (5) 首先对 $f_i^k(\boldsymbol{x})$ 进行线性化以简化子问题的求解, 在此基础上引人了 Nesterov 加速算法的技巧加快收敛.

例 6

考虑二元二次函数的优化问题

$$\min f(x,y) = x^2 - 2xy + 10y^2 - 4x - 20y,$$

现在对变量 x, y 使用分块坐标下降法求解. 当固定 y 时, 可知当 $x = 2 + y$ 时函数取极小值; 当固定 x 时, 可知当 $y = 1 + \frac{x}{10}$ 时函数取极小值. 故采用格式 (3) 的分块坐标下降法为

$$x^{k+1} = 2 + y^k$$

$$y^{k+1} = 1 + \frac{x^{k+1}}{10}$$

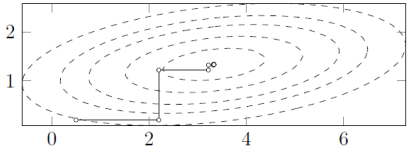

图 2: 迭代轨迹图

图 2 描绘了当初始点为 $(x, y) = (0.5, 0.2)$ 时的迭代点轨迹, 可以看到在进行了 7 次迭代后迭代点与最优解已充分接近. 回忆一下我们曾经对一个类似的问题使用过梯度法, 而梯度法的收敛相当缓慢. 一个直观的解释是: 对于比较病态的问题, 由于分块坐标下降法是对逐个分量处理, 它能较好地捕捉目标函数的各向异性, 而梯度法则会受到很大影响.

36.2.3 应用：分块坐标下降法求解 K 均值聚类

下面对聚类问题使用分块坐标下降法进行求解. 其目标函数为

$$\begin{aligned} \min_{\boldsymbol{\Phi},\boldsymbol{H}} \quad & \|\boldsymbol{A}-\boldsymbol{\Phi}\boldsymbol{H}\|_F^2, \\ \text{s.t.} \quad & \boldsymbol{\Phi}\in\mathbb{R}^{n\times k}, \text{每一行只有一个元素为} 1, \text{其余为} 0, \\ & \boldsymbol{H}\in\mathbb{R}^{k\times p}. \end{aligned}$$

接下来分别讨论在固定 $\boldsymbol{\Phi}$ 和 $\boldsymbol{H}$ 的条件下如何极小化另一块变量.

当固定 $\boldsymbol{H}$ 时，设 $\boldsymbol{\Phi}$ 的每一行为 $\boldsymbol{\phi}_i^{\mathrm{T}}$，那么根据矩阵分块乘法，

$$\boldsymbol{A}-\boldsymbol{\Phi}\boldsymbol{H}=\begin{bmatrix}\boldsymbol{a}_1^{\mathrm{T}}\\ \boldsymbol{a}_2^{\mathrm{T}}\\ \vdots\\ \boldsymbol{a}_n^{\mathrm{T}}\end{bmatrix}-\begin{bmatrix}\boldsymbol{\phi}_1^{\mathrm{T}}\\ \boldsymbol{\phi}_2^{\mathrm{T}}\\ \vdots\\ \boldsymbol{\phi}_n^{\mathrm{T}}\end{bmatrix}\boldsymbol{H}=\begin{bmatrix}\boldsymbol{a}_1^{\mathrm{T}}-\boldsymbol{\phi}_1^{\mathrm{T}}\boldsymbol{H}\\ \boldsymbol{a}_2^{\mathrm{T}}-\boldsymbol{\phi}_2^{\mathrm{T}}\boldsymbol{H}\\ \vdots\\ \boldsymbol{a}_n^{\mathrm{T}}-\boldsymbol{\phi}_n^{\mathrm{T}}\boldsymbol{H}\end{bmatrix}.$$

注意到 $\boldsymbol{\phi}_i$ 只有一个分量为 1，其余分量为 0，不妨设其第 j 个分量为 1，此时 $\boldsymbol{\phi}_i^{\mathrm{T}}\boldsymbol{H}$ 相当于将 $\boldsymbol{H}$ 的第 j 行取出，因此 $\left\|\boldsymbol{a}_i^{\mathrm{T}}-\boldsymbol{\phi}_i^{\mathrm{T}}\boldsymbol{H}\right\|$ 为 $\boldsymbol{a}_i^{\mathrm{T}}$ 与 $\boldsymbol{H}$ 的第 j 个行向量的距离. 我们的最终目的是极小化 $\|\boldsymbol{A}-\boldsymbol{\Phi}\boldsymbol{H}\|_F^2$，所以 j 应该选矩阵 $\boldsymbol{H}$ 中距离 $\boldsymbol{a}_i^{\mathrm{T}}$ 最近的那一行，即

$$\Phi_{ij}=\begin{cases}1, & j=\arg\min\limits_{l}\left\|\boldsymbol{a}_i-\boldsymbol{h}_l\right\|,\\ 0, & \text{其他}.\end{cases}$$

其中 $\boldsymbol{h}_l^{\mathrm{T}}$ 表示矩阵 $\boldsymbol{H}$ 的第 l 行.

当固定 $\boldsymbol{\Phi}$ 时，此时考虑 $\boldsymbol{H}$ 的每一行 $\boldsymbol{h}_j^{\mathrm{T}}$，根据目标函数的等价性有

$$\|\boldsymbol{A}-\boldsymbol{\Phi}\boldsymbol{H}\|_F^2=\sum_{j=1}^{k}\sum_{\boldsymbol{a}\in S_j}\|\boldsymbol{a}-\boldsymbol{h}_j\|^2,$$

因此只需要对每个 $\boldsymbol{h}_j$ 求最小即可. 设 $\bar{\boldsymbol{a}}_j$ 是目前第 j 类所有点的均值，则

$$\begin{aligned}\sum_{\boldsymbol{a}\in S_j}\|\boldsymbol{a}-\boldsymbol{h}_j\|^2&=\sum_{\boldsymbol{a}\in S_j}\|\boldsymbol{a}-\bar{\boldsymbol{a}}_j+\bar{\boldsymbol{a}}_j-\boldsymbol{h}_j\|^2\\&=\sum_{\boldsymbol{a}\in S_j}\left(\|\boldsymbol{a}-\bar{\boldsymbol{a}}_j\|^2+\|\bar{\boldsymbol{a}}_j-\boldsymbol{h}_j\|^2+2\langle\boldsymbol{a}-\bar{\boldsymbol{a}}_j,\bar{\boldsymbol{a}}_j-\boldsymbol{h}_j\rangle\right)\\&=\sum_{\boldsymbol{a}\in S_j}\left(\|\boldsymbol{a}-\bar{\boldsymbol{a}}_j\|^2+\|\bar{\boldsymbol{a}}_j-\boldsymbol{h}_j\|^2\right),\end{aligned}$$

这里利用了交叉项 $\sum_{\boldsymbol{a}\in S_j}\langle\boldsymbol{a}-\bar{\boldsymbol{a}}_j,\bar{\boldsymbol{a}}_j-\boldsymbol{h}_j\rangle=0$ 的事实. 因此容易看出，此时 $\boldsymbol{h}_j$ 直接取为 $\bar{\boldsymbol{a}}_j$ 即可达到最小值.

综上，我们得到了针对聚类问题的分块坐标下降法，它每一次迭代分为两步：

1. 固定参考点 $\boldsymbol{H}$，将每个样本点分到和其最接近的参考点代表的类中；
2. 固定聚类方式 $\boldsymbol{\Phi}$，重新计算每个类所有点的均值并将其作为新的参考点. 这个过程恰好就是经典的 K-均值聚类算法，因此可以得到结论：K-均值聚类算法本质上是一个分块坐标下降法.

36.2.4 应用：分块坐标下降法求解非负矩阵分解

非负矩阵分解问题也可以使用分块坐标下降法求解. 现在考虑最基本的非负矩阵分解问题

$$\min_{\boldsymbol{X},\boldsymbol{Y}\geqslant 0} \frac{1}{2}\|\boldsymbol{X}\boldsymbol{Y}-\boldsymbol{M}\|_F^2$$

它的一个等价形式为

$$\min_{\boldsymbol{X},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X}\boldsymbol{Y}-\boldsymbol{M}\|_F^2 + I_{\geqslant 0}(\boldsymbol{X}) + I_{\geqslant 0}(\boldsymbol{Y}),$$

其中 $I_{\geqslant 0}(\cdot)$ 为集合 $\{\boldsymbol{X} \mid \boldsymbol{X}\geqslant 0\}$ 的示性函数. 不难验证该问题具有形式 (2).

以下考虑求解方法：

- 注意到 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 耦合在一起，在固定 $\boldsymbol{Y}$ 的条件下，我们无法直接按照格式 (3) 或格式 (4) 的形式给出子问题的显式解.
- 若要采用这两种格式需要额外设计算法求解子问题，最终会产生较大计算量.
- 但我们总能使用格式 (5) 来对子问题进行线性化，从而获得比较简单的更新格式.

今 $f(\boldsymbol{X},\boldsymbol{Y})=\frac{1}{2}\|\boldsymbol{X}\boldsymbol{Y}-\boldsymbol{M}\|_F^2$, 则

$$\frac{\partial f}{\partial \boldsymbol{X}}=(\boldsymbol{X}\boldsymbol{Y}-\boldsymbol{M})\boldsymbol{Y}^{\mathrm{T}},\quad \frac{\partial f}{\partial \boldsymbol{Y}}=\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{Y}-\boldsymbol{M})$$

注意到在格式 (5) 中, 当 $r_i(\boldsymbol{X})$ 为凸集示性函数时即是求解到该集合的投影, 因此得到分块坐标下降法如下:

$$\boldsymbol{X}^{k+1}=\max\left\{\boldsymbol{X}^k-t_k^x\left(\boldsymbol{X}^k\boldsymbol{Y}^k-\boldsymbol{M}\right)\left(\boldsymbol{Y}^k\right)^{\mathrm{T}},0\right\}$$

$$\boldsymbol{Y}^{k+1}=\max\left\{\boldsymbol{Y}^k-t_k^y\left(\boldsymbol{X}^k\right)^{\mathrm{T}}\left(\boldsymbol{X}^k\boldsymbol{Y}^k-\boldsymbol{M}\right),0\right\}$$

其中 t_k^x, t_k^y 是步长, 分别对应格式 (5) 中的 $\frac{1}{L_i^k}, i=1,2.$

36.3.1 问题引入

本节考虑如下凸问题：

$$\begin{array}{ll}\min_{\boldsymbol{x}_1,\boldsymbol{x}_2} & f_1\left(\boldsymbol{x}_1\right)+f_2\left(\boldsymbol{x}_2\right) \\ \text{s.t.} & \boldsymbol{A}_1\boldsymbol{x}_1+\boldsymbol{A}_2\boldsymbol{x}_2=\boldsymbol{b},\end{array} \tag{6}$$

其中 f_1, f_2 是适当的闭凸函数，但不要求是光滑的，$\boldsymbol{x}_1 \in \mathbb{R}^n, \boldsymbol{x}_2 \in \mathbb{R}^m, \boldsymbol{A}_1 \in \mathbb{R}^{p\times n}, \boldsymbol{A}_2 \in \mathbb{R}^{p\times m}, \boldsymbol{b} \in \mathbb{R}^p$. 这个问题的特点是目标函数可以分成彼此分离的两块，但是变量被线性约束结合在一起.

常见的一些无约束和带约束的优化问题都可以表示成这一形式. 下面的一些例子将展示如何把某些一般的优化问题转化为适用交替方向乘子法求解的标准形式.

例 7

可以分成两块的无约束优化问题

$$\min_{\boldsymbol{x}} f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}).$$

为了将此问题转化为标准形式 (6), 需要将目标函数改成可分的形式. 我们可以通过引入一个新的变量 z 并令 $\boldsymbol{x} = \boldsymbol{z}$, 将问题转化为

$$\begin{array}{ll} \min_{\boldsymbol{x},\boldsymbol{z}} & f_1(\boldsymbol{x}) + f_2(\boldsymbol{z}), \\ s.t. & \boldsymbol{x} - \boldsymbol{z} = 0. \end{array}$$

例 8

带线性变换的无约束优化问题

$$\min_{\boldsymbol{x}} f_1(\boldsymbol{x}) + f_2(\boldsymbol{A}\boldsymbol{x}).$$

类似地，我们可以引入一个新的变量 $\boldsymbol{z}$，令 $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x}$，则问题变为

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{z}} \quad & f_1(\boldsymbol{x}) + f_2(\boldsymbol{z}), \\ s.t. \quad & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{z} = 0 \end{aligned}$$

对比问题 (6) 可知 $\boldsymbol{A}_1 = \boldsymbol{A}$ 和 $\boldsymbol{A}_2 = -\boldsymbol{I}$.

36.3.2 交替方向乘子法

下面给出交替方向乘子法 (alternating direction method of multipliers, ADMM) 的迭代格式, 首先写出问题 (6) 的增广拉格朗日函数

$$L_\rho(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}) = f_1(\boldsymbol{x}_1) + f_2(\boldsymbol{x}_2) + \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{A}_1\boldsymbol{x}_1 + \boldsymbol{A}_2\boldsymbol{x}_2 - \boldsymbol{b}) + \frac{\rho}{2}\|\boldsymbol{A}_1\boldsymbol{x}_1 + \boldsymbol{A}_2\boldsymbol{x}_2 - \boldsymbol{b}\|_2^2$$

其中 $\rho > 0$ 是二次罚项的系数.

常见的求解带约束问题的增广拉格朗日函数法为如下更新：

$$\left(\boldsymbol{x}_1^{k+1}, \boldsymbol{x}_2^{k+1}\right) = \arg\min_{\boldsymbol{x}_1, \boldsymbol{x}_2} L_\rho\left(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}^k\right)$$

$$\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \tau\rho\left(\boldsymbol{A}_1\boldsymbol{x}_1^{k+1} + \boldsymbol{A}_2\boldsymbol{x}_2^{k+1} - \boldsymbol{b}\right),$$

其中 τ 为步长. 在实际求解中，第一步迭代同时对 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 进行优化有时候比较困难，而固定一个变量求解关于另一个变量的极小问题可能比较简单，

因此我们可以考虑对 x_1 和 x_2 交替求极小, 这就是交替方向乘子法的基本思路. 其迭代格式可以总结如下:

$$\begin{aligned}
\boldsymbol{x}_1^{k+1} &= \arg\min_{\boldsymbol{x}_1} L_\rho\left(\boldsymbol{x}_1, \boldsymbol{x}_2^k, \boldsymbol{y}^k\right), \\
\boldsymbol{x}_2^{k+1} &= \arg\min_{\boldsymbol{x}_2} L_\rho\left(\boldsymbol{x}_1^{k+1}, \boldsymbol{x}_2, \boldsymbol{y}^k\right), \\
\boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \tau\rho\left(\boldsymbol{A}_1\boldsymbol{x}_1^{k+1} + \boldsymbol{A}_2\boldsymbol{x}_2^{k+1} - \boldsymbol{b}\right),
\end{aligned}$$

其中 τ 为步长.

观察交替方向乘子法的迭代格式, 第一步固定 $\boldsymbol{x}_2, \boldsymbol{y}$ 对 $\boldsymbol{x}_1$ 求极小; 第二步固定 $\boldsymbol{x}_1, \boldsymbol{y}$ 对 $\boldsymbol{x}_2$ 求极小; 第三步更新拉格朗日乘子 $\boldsymbol{y}$.

36.3.3 应用：交替方向乘子法求解矩阵分离问题

考虑矩阵分离问题：

$$\begin{aligned} \min_{\boldsymbol{X},\boldsymbol{S}} \quad & \|\boldsymbol{X}\|_* + \mu\|\boldsymbol{S}\|_1, \\ \text{s.t.} \quad & \boldsymbol{X} + \boldsymbol{S} = \boldsymbol{M}, \end{aligned}$$

其中 $\|\cdot\|_1$ 与 $\|\cdot\|_*$ 分别表示矩阵 ℓ_1 范数与核范数. 引入乘子 $\boldsymbol{Y}$ 作用在约束 $\boldsymbol{X}+\boldsymbol{S}=\boldsymbol{M}$ 上，我们可以得到此问题的增广拉格朗日函数

$$L_\rho(\boldsymbol{X},\boldsymbol{S},\boldsymbol{Y}) = \|\boldsymbol{X}\|_* + \mu\|\boldsymbol{S}\|_1 + \langle \boldsymbol{Y}, \boldsymbol{X}+\boldsymbol{S}-\boldsymbol{M}\rangle + \frac{\rho}{2}\|\boldsymbol{X}+\boldsymbol{S}-\boldsymbol{M}\|_F^2.$$

在第 $(k+1)$ 步，交替方向乘子法分别求解关于 $\boldsymbol{X}$ 和 $\boldsymbol{S}$ 的子问题来更新得到 $\boldsymbol{X}^{k+1}$ 和 $\boldsymbol{S}^{k+1}$. 对于 $\boldsymbol{X}$ 子问题，

$$
\begin{aligned}
\boldsymbol{X}^{k+1} &= \arg\min_{\boldsymbol{X}} L_\rho\left(\boldsymbol{X}, \boldsymbol{S}^k, \boldsymbol{Y}^k\right) \\
&= \arg\min_{\boldsymbol{X}} \left\{ \|\boldsymbol{X}\|_* + \frac{\rho}{2}\left\|\boldsymbol{X}+\boldsymbol{S}^k-\boldsymbol{M}+\frac{\boldsymbol{Y}^k}{\rho}\right\|_F^2 \right\}, \\
&= \arg\min_{\boldsymbol{X}} \left\{ \frac{1}{\rho}\|\boldsymbol{X}\|_* + \frac{1}{2}\left\|\boldsymbol{X}+\boldsymbol{S}^k-\boldsymbol{M}+\frac{\boldsymbol{Y}^k}{\rho}\right\|_F^2 \right\}, \\
&= \boldsymbol{U}\mathrm{Diag}\left(\mathrm{prox}_{(1/\rho)\|\cdot\|_1}(\sigma(\boldsymbol{A}))\right)\boldsymbol{V}^{\mathrm{T}},
\end{aligned}
$$

其中 $\boldsymbol{A}=\boldsymbol{M}-\boldsymbol{S}^k-\frac{\boldsymbol{Y}^k}{\rho}$, $\sigma(\boldsymbol{A})$ 为 $\boldsymbol{A}$ 的所有非零奇异值构成的向量并且 $\boldsymbol{U}\mathrm{Diag}(\sigma(\boldsymbol{A}))\boldsymbol{V}^{\mathrm{T}}$ 为 $\boldsymbol{A}$ 的约化奇异值分解.

对于 $\boldsymbol{S}$ 子问题，

$$
\begin{aligned}
\boldsymbol{S}^{k+1} &= \arg\min_{\boldsymbol{S}} L_\rho\left(\boldsymbol{X}^{k+1}, \boldsymbol{S}, \boldsymbol{Y}^k\right) \\
&= \arg\min_{\boldsymbol{S}} \left\{ \mu\|\boldsymbol{S}\|_1 + \frac{\rho}{2}\left\|\boldsymbol{X}^{k+1} + \boldsymbol{S} - \boldsymbol{M} + \frac{\boldsymbol{Y}^k}{\rho}\right\|_F^2 \right\} \\
&= \operatorname{prox}_{(\mu/\rho)\|\cdot\|_1}\left(\boldsymbol{M} - \boldsymbol{X}^{k+1} - \frac{\boldsymbol{Y}^k}{\rho}\right).
\end{aligned}
$$

对于乘子 $\boldsymbol{Y}$, 依然使用常规更新, 即

$$\boldsymbol{Y}^{k+1} = \boldsymbol{Y}^{k} + \tau\rho\left(\boldsymbol{X}^{k+1} + \boldsymbol{S}^{k+1} - \boldsymbol{M}\right).$$

那么, 交替方向乘子法的迭代格式为

$$\begin{aligned}
\boldsymbol{X}^{k+1} &= \boldsymbol{U}\mathrm{Diag}\left(\mathrm{prox}_{(1/\rho)\|\cdot\|_1}(\sigma(\boldsymbol{A}))\right)\boldsymbol{V}^{\mathrm{T}},\\
\boldsymbol{S}^{k+1} &= \mathrm{prox}_{(\mu/\rho)\|\cdot\|_1}\left(\boldsymbol{M} - \boldsymbol{X}^{k+1} - \frac{\boldsymbol{Y}^{k}}{\rho}\right),\\
\boldsymbol{Y}^{k+1} &= \boldsymbol{Y}^{k} + \tau\rho\left(\boldsymbol{X}^{k+1} + \boldsymbol{S}^{k+1} - \boldsymbol{M}\right).
\end{aligned}$$

值得说明的是，在实际中，大多数问题并不直接具有问题 (6) 的形式. 我们需要通过一系列拆分技巧将问题化成 ADMM 的标准形式，同时要求每一个子问题尽量容易求解. 需要指出的是，对同一个问题可能有多种拆分方式，不同方式导出的最终算法可能差异巨大，读者应当选择最容易求解的拆分方式.

在机器学习或深度学习中，通常利用经验风险最小化的原则，寻找最优的模型。
即极小化如下代价函数

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{P}_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{y}) \tag{7}$$

其中，L 是每个样本的损失函数，$f(\boldsymbol{x};\boldsymbol{\theta})$ 是输入为 $\boldsymbol{x}$ 时所预测的输出，$\hat{P}_{data}$ 是经验分布，监督学习中，$\boldsymbol{y}$ 是目标输出。

注：在本节中，不讨论结构风险最小化的问题。

结合前一章的内容可知，在求解极小化代价函数时，最常用的目标函数的性质是梯度：

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{P}_{data}} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{y}) \tag{8}$$

在一阶优化或二阶优化算法时，均需要用到梯度信息。

然而，由于机器学习或深度学习中的数据集非常大，因此，每次迭代计算梯度的运算量也非常大。该如何解决这个问题呢？

36.4.1 随机梯度下降

考虑到，梯度的计算实际上可以看作是求期望。根据期望的性质可知，任何一个样本都可看作是期望值的一个无偏估计。因此，考虑如下方式计算梯度：

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \tag{9}$$

这也被称为**随机梯度**，对应的优化算法被称为**随机梯度下降法**。

随机梯度可行的两个原因：

1. 首先，n 个样本均值的标准差是 $\sigma/\sqrt{n}$，其中 σ 是样本值真实的标准差，分母 $\sqrt{n}$。这表明使用更多样本来估计梯度的方法是**低于线性的**。

注：换言之，一个基于 100 个样本，另一个基于 10,000 个样本，后者用于梯度计算的计算量是前者的 100 倍，但却只降低了 10 倍的均值标准差。

随机梯度可行的两个原因：

2. 其次，从小数目样本中获得梯度的统计估计的动机是训练集的**冗余性**。

注：实践中，我们会发现大量样本都对梯度做出了非常相似的贡献。

36.4.2 一些概念的区分

梯度算法与随机梯度算法

- 使用整个训练集的梯度优化算法被称为 **batch** 或**确定性**梯度算法，简称**梯度算法**。
- 每次从一个固定大小的训练集中随机抽取单个样本的梯度优化算法被称为**随机梯度**算法。

随机梯度算法与在线算法

- 每次只使用单个样本的梯度优化算法有时被称为**随机梯度**或者**在线**算法。
- 其中“**在线**”通常是指从连续产生的数据流中抽取样本的情况，而不是从一个固定大小的训练集中遍历多次采样的情况。

minibatch 随机梯度算法与随机梯度算法

大多数用于深度学习的算法介于以上两者之间，使用一个以上，而又不是全部的训练样本。传统上，这些会被称为 **minibatch** 或 **minibatch 随机梯度**方法，通常也简单地称为**随机梯度**方法。

36.4.3 随机梯度下降法具体步骤

现将**随机梯度下降法**（SGD）具体步骤概括如下：

算法 4 随机梯度法

Require: 学习速率 ϵ_k，初始参数 $\boldsymbol{\theta}$

1: **repeat**
2: 从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ 的 minibatch，对应目标为 $\boldsymbol{y}^{(i)}$；
3: 计算梯度估计：$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right)$
4: 应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
5: **until** 达到停止准则

36.4.4 学习速率

SGD 算法中的一个关键参数是学习速率。在实践中，随着时间的推移有必要逐渐降低学习速率。

保证 SGD 收敛的一个**充分条件**是

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \tag{10}$$

且

$$\sum_{k=1}^{\infty} \epsilon_k^2 = \infty \tag{11}$$

其中 ϵ_k 为第 k 次迭代得学习速率。

实践中，一般会线性衰减学习速率到第 τ 次迭代：

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{12}$$

其中，$\alpha = \frac{k}{\tau}$。在 τ 步迭代之后，一般使 ϵ 保持常数。

- 学习速率可通过试验和误差来选取，通常最好的选择方法是画出目标函数值随时间变化的学习曲线。
- 通常，就总训练时间和最终损失值而言，最优初始学习速率会高于大约迭代 100 步后输出最好效果的学习速率。
- 一般，最好是检测最早的几次迭代，使用一个高于此时效果最佳学习速率的学习速率，但又不能太高以致严重的不稳定性。

36.4.5 SGD 的优点

SGD 的优点：

- SGD 和相关的 minibatch 或在线基于梯度的优化的最重要性质是每一步更新的计算时间不会随着训练样本数目而增加。即使训练样本数目非常大时，这也能收敛。
- 对于足够大的数据集，SGD 可能会在处理整个训练集之前就收敛到最终测试集误差的某个固定容差范围内。
- SGD 应用于凸问题时，k 步迭代后的额外误差量级是 $O(\frac{1}{\sqrt{k}})$，在强凸情况下是 $O(\frac{1}{k})$。

36.5.1 梯度下降的不足

虽然随机梯度下降是非常受欢迎的优化方法，但学习速率有时会很慢。可能出现如下图3所示“震荡”的现象。

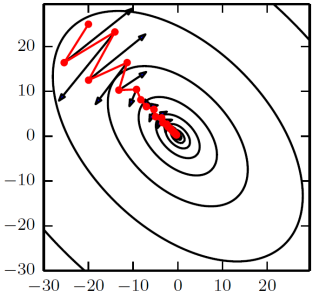

图 3

“震荡”的原因分析

分析出现“震荡”的原因：
在历史梯度的某些分量方向上，出现反复迂回的现象；而在一致前进的分量方向又没有得到加速，所以产生“震荡”。

“震荡”的原因分析

例如：假设第 $k-1$ 次迭代的梯度 $\nabla f(x_{k-1}) = [0.5\ 0.2]^T$，而在第 k 次迭代的梯度 $\nabla f(x_k) = [-0.4\ 0.1]^T$。显然，这两次迭代在第一维分量上产生震荡，第二维的分量方向行进一致。

这促使我们**思考**：如何在震荡的分量上抵消震荡，在行进一致的分量上进行加速？

36.5.2 动量梯度下降

一个简单的方法：将历史数据与当前的梯度进行某种加法，那么自然会抵消震荡部分的分量，并且加速行进一致的分量。

这便得到**动量法**的更新规则如下：

$$\begin{cases} \boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\dfrac{1}{m} \displaystyle\sum_{i=1}^{m} L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)} \right) \right) \\ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \end{cases} \tag{13}$$

其中第一个式子的第一项可以看作是历史梯度信息，第一个式子的第二项为当前的负梯度。然后将它们的加权和作为当前下降的方向。

36.5.3 动量梯度下降法具体步骤

将带动量的 SGD 算法总结如下：

算法 5 动量法

Require: 学习速率 ϵ，动量参数 α，初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

1: **repeat**
2: 从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ 的 minibatch，对应目标为 $\boldsymbol{y}^{(i)}$；
3: 计算梯度估计：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right)$
4: 计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
5: 应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
6: **until** 达到停止准则

36.5.4 动量法的几点注解

动量法的几点注解：

1. 物理学上的直观含义：从形式上看，动量算法引入了变量 v 充当速度的角色——它代表参数在参数空间移动的方向和速度。名称**动量** (momentum) 来自物理类比，根据牛顿运动定律，负梯度是移动参数空间中粒子的力。动量在物理学上是质量乘以速度。在动量学习算法中，我们假设是单位质量，因此速度向量 v 也可以看作是粒子的动量。

动量法的几点注解：

2. 实际上动量（速度）被设为历史负梯度信息的指数衰减平均，其中超参数 $\alpha \in [0,1)$ 决定了之前梯度的贡献衰减得有多快。这可以通过简单的展开得到：

$$\boldsymbol{v}_k = \alpha \boldsymbol{v}_{k-1} - \epsilon \boldsymbol{g}_k = \alpha^2 \boldsymbol{v}_{k-2} - \alpha\epsilon \boldsymbol{g}_{k-1} - \epsilon \boldsymbol{g}_k = \cdots .$$

动量法的几点注解：

3. 对动量法做一个最理想的假设：如果动量动量算法总是观测到梯度 g，那么它会在方向 $-g$ 上不停加速，直到达到最后速度的步长为

$$\frac{\epsilon\|\boldsymbol{g}\|}{1-\alpha} \tag{14}$$

因此，将动量的超参数视为 $\frac{1}{1-\alpha}$。

36.5.5 Nesterov 动量法

思考:
动量法中使用历史的梯度数据来改进搜索方向，那能不能使用**未来的梯度数据**来改进当前搜索方向呢？

实际上，这就是 Nesterov 加速梯度算法。假设我们试探性的向前迈进了一步，然后计算下一步的梯度，再利用它改善当前的前进方向。

受 Nesterov 加速梯度算法启发，Sutskever(2013) 提出了动量算法的一个变种。这种情况的更新规则如下：

$$\begin{cases} \boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right] \\ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \end{cases} \tag{15}$$

其中参数 α 和 ϵ 发挥了和标准动量方法中类似的作用。

注：$\boldsymbol{\theta} + \alpha \boldsymbol{v}$ 即表示试探性的下一步，那么第一个式子的第二项即为下一步的梯度。

36.5.6 Nesterov 动量法具体步骤

因此，可将 Nesterov 动量的随机梯度下降算法总结如下：

算法 6 Nesterov 动量法

Require: 学习速率 ϵ，动量参数 α，初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

1: **repeat**
2: 从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ 的 minibatch，对应目标为 $\boldsymbol{y}^{(i)}$；
3: 应用临时更新：$\widetilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
4: 计算梯度 (在临时点)：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\widetilde{\boldsymbol{\theta}}} \sum_i L\left(f(\boldsymbol{x}^{(i)}; \widetilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)}\right)$
5: 计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
6: 应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
7: **until** 达到停止准则

36.6.1 自适应学习速率

- 前面讨论的是如何优化搜索方向，实际上还有一个关键的问题是线搜索，即机器学习领域称之为学习速率。
- 神经网络研究员早就意识到学习速率肯定是难以设置的超参数之一，因为它对模型的性能有显著的影响。自然地**问题**：该如何优化学习速率？

事实上，**Delta-bar-delta** 算法 (Jacobs, 1988) 是一个早期的在训练时适应模型参数各自学习速率的启发式方法。该方法基于一个很简单的想法：

- 如果损失对于某个给定模型参数的偏导保持相同的符号，那么学习速率应该增加。
- 如果对于该参数的偏导变化了符号，那么学习速率应减小。

当然，这种方法只能应用于全 batch 优化中。而对于基于 minibatch 的算法是否有相应的启发式学习速率？

36.6.2 AdaGrad

再次回到下图4所示“震荡”的现象。

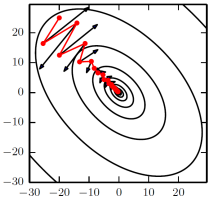

图 4

这次不从搜索方向的角度思考，而从学习速率的角度思考，直观地发现：

- 在左下至右上的维度前进的步伐较大，导致了反复迂回；
- 在左上至右下的维度前进的步伐较小，导致了下降不够。

因此，可以根据梯度历史值自适应地更新学习速率。这就是 **AdaGrad** 算法：按照每个参数的**梯度历史值的平方和的平方根成反比**缩放每个参数 (Duchi, 2011)，独立地适应所有模型参数的学习速率。

- 具有最大偏导的参数，由于变化剧烈，应相应地有一个较小下降的学习速率；
- 而具有小偏导的参数在学习速率上应相对较大。

36.6.3 AdaGrad 算法具体步骤

于是，AdaGrad 算法可总结如下：

算法 7 AdaGrad

Require: 全局学习速率 ϵ，初始参数 $\boldsymbol{\theta}$，小常数 δ（为了数值稳定大约设为 10^{-7}）

初始化梯度累计变量 $\boldsymbol{r} = \mathbf{0}$;

repeat

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ 的 minibatch，对应目标为 $\boldsymbol{y}^{(i)}$;

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right)$

累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$（$\odot$ 表示逐元素相乘）

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$（逐元素地应用除和求平方根）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

until 达到停止准则

注:
在凸优化背景中，AdaGrad 算法具有一些令人满意的理论性质。然而，经验上已经发现，对于训练深度神经网络模型而言，从训练开始时积累梯度平方会导致有效学习速率过早和过量的减小。AdaGrad 在某些深度学习模型上效果不错，但不是全部。

36.6.4 RMSProp

在了解动量法之后，我们会发现 AdaGrad 算法存在明显**不足**：

- 将所有的历史梯度值的平方直接相加，一方面可能使得学习速率在达到这样的凸结构前就变得太小了。
- 另一方面，对于遥远过去的历史数据没有衰减。

借助动量法的思想，**RMSProp** 算法 (Hinton, 2012) 修改 AdaGrad 算法，改变梯度积累为指数加权的移动平均：

$$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}.$$

注：RMSProp 使用指数衰减平均以丢弃遥远过去的历史，使其能够在找到碗状凸结构后快速收敛，它就像一个初始化于该碗状结构的 AdaGrad 算法实例。

36.6.5 RMSProp 算法具体步骤

因此，可得 RMSProp 算法如下：

算法 8 RMSProp

Require: 全局学习速率 ϵ，衰减速率 ρ，初始参数 $\boldsymbol{\theta}$，小常数 δ（为了数值稳定大约设为 10^{-6}）

初始化梯度累计变量 $\boldsymbol{r} = \mathbf{0}$;

repeat

  从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ 的 minibatch，对应目标为 $\boldsymbol{y}^{(i)}$;

  计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right)$

  累积平方梯度：$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \odot \boldsymbol{g}$

  计算更新：$\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

  应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

until 达到停止准则

36.6.6 Adam

- 首先，我们探讨了搜索方向上的优化，得到了动量法；
- 其次，我们得到了学习速率上的优化，即 AdaGrad 算法、RMSProp 算法；

显然，可以将二者结合起来，即接下来要介绍的 Adam 算法。它可看作是结合了 RMSProp 和动量法的变种。

Adam(kingma and Ba, 2014) 是另一种学习速率自适应的优化算法。“Adam” 这个名字派生自短语 “adaptive moments”。

- 首先，在 Adam 中，将动量加入 RMSProp 最直观的方法是将动量应用于缩放后的梯度。
- 其次，Adam 包括偏置修正，修正从原点初始化的一阶矩（动量项）和（非中心的）二阶矩的估计。

36.6.7 Adam 算法具体步骤

算法 9 Adam

Require: 步长 ϵ（建议默认为 0.001），矩估计的指数衰减速率 ρ_1, ρ_2（建议分别默认为 0.9 和 0.999），用于数值稳定的小常数 δ（建议默认为 10^{-8}），初始参数 $\boldsymbol{\theta}$

初始化一阶和二阶矩变量 $\boldsymbol{s} = \mathbf{0}, \boldsymbol{r} = \mathbf{0}, t = 0$

repeat

  从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ 的 minibatch，对应目标为 $\boldsymbol{y}^{(i)}$；

  计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}\right)$

  $t \leftarrow t + 1$

  更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

  更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

  修正一阶矩和二阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$，$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

  计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$

  应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

until 达到停止准则

- RMSProp 也采用了（非中心的）二阶矩估计，然而缺失了修正因子。因此，不像 Adam，RMSProp 二阶矩估计可能在训练初期有很高的偏置。Adam 通常被认为对超参数的选择相当鲁棒。

- RMSProp 算法还可以和 Nesterov 动量法结合，便可得到 Nadam 算法，这里不再详述。

36.6.8 随机优化算法应用：多层感知机

多层感知机也叫全连接神经网络，是一种基本的网络结构，在前面已经简要地介绍了这一模型. 考虑有 L 个隐藏层的多层感知机，给定输入 $\boldsymbol{x} \in \mathbb{R}^p$，则多层感知机的输出可用如下迭代过程表示:

$$\boldsymbol{f}^{(l)} = \sigma\left(\boldsymbol{A}^{(l)}\boldsymbol{f}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \quad l = 1, 2, \cdots, L+1$$

其中 $\boldsymbol{A}^{(l)} \in \mathbb{R}^{m_{l-1} \times m_l}$ 为系数矩阵，$\boldsymbol{b}^{(l)} \in \mathbb{R}^{m_l}$ 为非齐次项，$\sigma(\cdot)$ 为非线性激活函数. 该感知机的输出为 $\boldsymbol{f}^{(L+1)}$.

现在用非线性函数 $h(\boldsymbol{x};\boldsymbol{A})$ 来表示该多层感知机，其中

$$\boldsymbol{A}=\left(\boldsymbol{A}^{(1)},\boldsymbol{A}^{(2)},\cdots,\boldsymbol{A}^{(L)},\boldsymbol{b}^{(1)},\boldsymbol{b}^{(2)},\cdots,\boldsymbol{b}^{(L)}\right)$$

表示所有网络参数的集合，则学习问题可以表示成经验损失函数求极小问题:

$$\min\frac{1}{N}\sum_{i=1}^{N}L\left(h\left(\boldsymbol{x}_i;\boldsymbol{A}\right),\boldsymbol{y}_i\right).$$

同样地，由于目标函数表示成了样本平均的形式，我们可以用随机梯度算法:

$$\boldsymbol{A}^{k+1}=\boldsymbol{A}^{k}-\tau_k\nabla_{\boldsymbol{A}}L\left(h\left(\boldsymbol{x}_{s_k};\boldsymbol{A}^{k}\right),\boldsymbol{y}_{s_k}\right),$$

其中 s_k 为从 $\{1,2,\cdots,N\}$ 中随机抽取的一个样本.

算法最核心的部分为求梯度, 由于函数具有复合结构, 因此可以采用后传算法. 假定已经得到关于第 l 隐藏层的导数 $\frac{\partial L}{\partial \boldsymbol{f}^{(l)}}$, 然后可以通过下面递推公式得到关于第 l 隐藏层参数的导数以及关于前一个隐藏层的导数:

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} &= \frac{\partial L}{\partial \boldsymbol{f}^{(l)}} \odot \frac{\partial \sigma}{\partial \boldsymbol{z}} \\
\frac{\partial L}{\partial \boldsymbol{A}^{(l)}} &= \left(\frac{\partial L}{\partial \boldsymbol{f}^{(l)}} \odot \frac{\partial \sigma}{\partial \boldsymbol{z}} \right) \left(\boldsymbol{f}^{(l-1)} \right)^{\mathrm{T}}, \\
\frac{\partial L}{\partial \boldsymbol{f}^{(l-1)}} &= \left(\boldsymbol{A}^{(l)} \right)^{\mathrm{T}} \left(\frac{\partial L}{\partial \boldsymbol{f}^{(l)}} \odot \frac{\partial \sigma}{\partial \boldsymbol{z}} \right).
\end{aligned}$$

其中 $\odot$ 为逐元素相乘.

完整的后传算法如下：

算法 10 后传算法

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{f}}} L\left(\hat{\boldsymbol{f}}, \boldsymbol{y}_{s_k}\right)$.
for $l = L+1, L, \cdots, 1$ **do**
  $\boldsymbol{g} \leftarrow \boldsymbol{g} \odot \frac{\partial \sigma}{\partial \boldsymbol{z}}$.
  $\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{g}$.
  $\frac{\partial L}{\partial \boldsymbol{A}^{(l)}} = \boldsymbol{g}\left(\boldsymbol{f}^{(l-1)}\right)^{\mathrm{T}}$.
  $\boldsymbol{g} \leftarrow \left(\boldsymbol{A}^{(l)}\right)^{\mathrm{T}} \boldsymbol{g}$.
end for

36.7.1 在线凸优化模型

在在线凸优化（Online Convex Optimization，OCO）问题中，

- 一个在线参与者（玩家）迭代式的做出决策。
- 在做出每一个决策时，与他的选择相关的结果对参与者来说是未知的。
- 在做出决策后，决策者会付出一个代价。每一个可能的决策都会付出一个（可能是不同的）代价，这些代价是决策者无法提前预知的，可以由对手选择，甚至取决于决策者自身采取的行动。

在在线凸优化架构模型中，决策集被模型化为欧式空间中的一个凸集，记为 $\mathcal{K} \subset \mathbb{R}^n$。代价被模型化为 $\mathcal{K}$ 上的有界凸函数。OCO 架构可以看作一个有结构的、不断重复的博弈过程。

这一学习架构的规则为：

- 在第 t 次迭代时，在线参与者选择 $x_t \in \mathcal{K}$。
- 在参与者做出这一选择后，给出一个凸函数 $f_t \in \mathcal{F} : \mathcal{K} \mapsto \mathbb{R}$。此处 $\mathcal{F}$ 为对手可以使用的有界代价函数族。
- 在线参与者付出的代价为 $f_t(x_t)$，即选择 x_t 时代价函数的值。

一个自然地问题，是什么使得一个算法成为好的 OCO 算法呢？由于该架构为一个博弈过程，它天然具有对抗性，其合理的性能评估指标也将来自于博弈论：决策者的**遗憾**（regret）。它定义为在事后看来，决策者做出决策所付出的总代价与固定的最好决策总代价之间的差。

令 $\mathcal{A}$ 为一个 OCO 算法，它将某特定博弈中的历史决策映射到决策集中。经 T 次的迭代后，$\mathcal{A}$ 的遗憾的形式化定义为：

$$\text{遗憾}_T(\mathcal{A}) = \sup_{f_1,\cdots,f_T\subseteq\mathcal{F}} \{\sum_{t=1}^{T} f_t(x_t) - \min_{\boldsymbol{x}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\boldsymbol{x})\} \tag{16}$$

36.7.2 在线投资组合

- OCO 在最近几年成为在线学习的主要架构的原因得益于它极强的建模能力，使得它被广泛地应用于诸多领域，例如：在线路由，广告选择，垃圾邮件过滤和投资组合选择等。
- 这里详细介绍最近被广泛研究的投资组合选择问题，考虑一个对股票市场不做任何统计性假设（用以区别传统的几何布朗运动股票价格模型）的投资组合选择模型，并将其称为“通用投资组合选择”（universal portfolio selection）模型。

在线投资组合

- 在每一迭代 $t \in [T]$ 中，决策者在 n 种资产上选择其财富的一个分布 $\boldsymbol{x}_t \in \Delta_n$，此处 $\Delta_n = \{\boldsymbol{x} \in \mathbb{R}_+^n, \sum_i = 1\}$ 为 n 维单纯形。
- 对手独立地选择资产的回报，即一个所有元素都严格为正的向量 $\boldsymbol{r}_t \in \mathbb{R}^n$，其每一个分量 $\boldsymbol{r}_t(i)$ 表示资产在迭代 t 和 $t+1$ 之间的价格的比值。
- 令 W_t 为在迭代 t 时她的总财富，则在忽略交易成本的情况下，有

$$W_{t+1} = W_t \cdot \boldsymbol{r}_t^T \boldsymbol{x}_t$$

投资者在迭代 $t+1$ 和 t 时的财富的比值为 $\boldsymbol{r}_t^T \boldsymbol{x}_t$。

在线投资组合

- 这样经过 T 次迭代后，总资产财富可用下式给出：
$$W_T = W_1 \cdot \Pi_{t=1}^{T} \boldsymbol{r}_t^T \boldsymbol{x}_t$$
- 决策者的目标是最大化整体的财富收益 W_T / W_1，它可以通过最大化下面的对数值更为方便的求得：
$$\log \frac{W_T}{W_1} = \sum_{t=1}^{T} \log \boldsymbol{r}_t^T \boldsymbol{x}_t$$
- 尽管它被标示为收益最大化而不是代价最小化，但这并没有本质区别。这一设定下的收益就定义为该财富比例变化的对数，即
$$f_t(x) = \log(\boldsymbol{r}_t^T \boldsymbol{x}).$$

在线投资组合

在这种情况下，遗憾被定义为：

$$\text{遗憾}_T = \max_{\boldsymbol{x}^\star \in \Delta_n} \sum_{t=1}^{T} \log(\boldsymbol{r}_t^T \boldsymbol{x}^\star) - \sum_{t=1}^{T} \log(\boldsymbol{r}_t^T \boldsymbol{x}_t)$$

即

$$\text{遗憾}_T = \max_{\boldsymbol{x}^\star \in \Delta_n} \sum_{t=1}^{T} f_t(\boldsymbol{x}^\star) - \sum_{t=1}^{T} f_t(\boldsymbol{x}_t)$$

在这一设定下，通用投资者组合选择非常符合 OCO 架构。

36.7.3 一阶方法：在线梯度下降法

本小节将描述并分析在线凸优化中的一个最简单也最基本的算法：在线梯度下降法，它在实践中也非常有效。本小节中介绍的算法的目标都是最小化遗憾（regret），而不是优化误差（在在线假设下，它是病态的）。

一阶方法：在线梯度下降法

应用于多数一般设定的在线凸优化问题的最简单算法是在线梯度下降法。这一算法是基于离线优化中最基本的标准梯度下降法的，它首次由 zinkevich 引入到在线形式中。

该算法的基本思想：

- 在每一次迭代中，算法首先在上一次迭代的基础上、沿上一次代价函数的负梯度方向移动一个步长．这一步可能得到一个不在基本凸集中的点。
- 此时，算法将该点投影回凸集，即求凸集中与这一个点最接近的点.

一阶方法：在线梯度下降法

具体迭代算法如下：

算法 11 在线梯度下降法（OGD）

1: **输入**：凸集 $\mathcal{K}$, T, $\boldsymbol{x}_1 \in \mathcal{K}$, **步长序列** $\{\eta_t\}$
2: **for** $t = 1$ **到** T **do**
3: 执行 $\boldsymbol{x}_t$ 并考查代价函数 $f_t(\boldsymbol{x}_t)$.
4: 更新及投影：

$$\boldsymbol{y}_{t+1} = \boldsymbol{x}_t - \eta_t \nabla f_t(\boldsymbol{x}_t)$$

$$\boldsymbol{x}_{t+1} = \Pi_{\mathcal{K}}(\boldsymbol{y}_{t+1})$$

5: **end for**

一阶方法：在线梯度下降法

尽管下一个代价函数的值与当前得到的代价函数的值可能完全不同，但该算法得到的遗憾是次线性的。这一结论可形式化整理为下面的定理。

定理 3

步长大小为 $\{\eta_t = \frac{D}{G\sqrt{t}}, t \in [T]\}$ 的在线梯度下降算法保证对所有的 $T \geq 1$ 有如下结论：

$$\text{遗憾}_T = \sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - \min_{\boldsymbol{x}^\star \in \mathcal{K}} \sum_{t=1}^{T} f_t(\boldsymbol{x}^\star) \leq \frac{3}{2} GD\sqrt{T}$$

这里 D 表示凸集 $\mathcal{K}$ 的上界，G 表示函数 f_t 在凸集 $\mathcal{K}$ 上的次梯度的范数上界。

一阶方法：在线梯度下降法

证明.

令 $\boldsymbol{x}^\star \in \arg\min_{\boldsymbol{x}\in\mathcal{K}} \sum_{t=1}^{T} f_t(\boldsymbol{x})$。记 $\nabla_t \triangleq \nabla f_t(\boldsymbol{x}_t)$。由凸性可得

$$f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}^\star) \leq \nabla_t^T(\boldsymbol{x}_t - \boldsymbol{x}^\star) \tag{17}$$

首先，利用 $\boldsymbol{x}_{t+1}$ 的更新规则，可给出 $\nabla_t^T(\boldsymbol{x}_t - \boldsymbol{x}^\star)$ 的上界：

$$\|\boldsymbol{x}_{t+1} - \boldsymbol{x}^\star\|^2 = \|\Pi_{\mathcal{K}}(\boldsymbol{x}_t - \eta_t\nabla_t) - \boldsymbol{x}^\star\|^2 \leq \|\boldsymbol{x}_t - \eta_t\nabla_t - \boldsymbol{x}^\star\|^2 \tag{18}$$

于是，

$$\begin{aligned} \|\boldsymbol{x}_{t+1} - \boldsymbol{x}^\star\|^2 &\leq \|\boldsymbol{x}_t - \boldsymbol{x}^\star\|^2 + \eta_t^2\|\nabla_t\|^2 - 2\eta_t\nabla_t^T(\boldsymbol{x}_t - \boldsymbol{x}^\star) \\ 2\nabla_t^T(\boldsymbol{x}_t - \boldsymbol{x}^\star) &\leq \frac{\|\boldsymbol{x}_t - \boldsymbol{x}^\star\|^2 - \|\boldsymbol{x}_{t+1} - \boldsymbol{x}^\star\|^2}{\eta_t} + \eta_t G^2 \end{aligned} \tag{19}$$

□

一阶方法：在线梯度下降法

将式 (17) 和式 (19) 对 $t=1$ 到 T 求和，并令 $\eta_t = \frac{D}{G\sqrt{t}}$（其中 $\frac{1}{\eta_0} \triangleq 0$）：

$$
\begin{aligned}
2(\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}^\bigstar)) &\leq 2\sum_{t=1}^{T} \nabla_t^T(\boldsymbol{x}_t - \boldsymbol{x}^\bigstar) \\
&\leq \sum_{t=1}^{T} \frac{\|\boldsymbol{x}_t - \boldsymbol{x}^\bigstar\|^2 - \|\boldsymbol{x}_{t+1} - \boldsymbol{x}^\bigstar\|^2}{\eta_t} + G^2\sum_{t=1}^{T}\eta_t \\
&\leq \sum_{t=1}^{T} \|\boldsymbol{x}_t - \boldsymbol{x}^\bigstar\|^2(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) + G^2\sum_{t=1}^{T}\eta_t \\
&\leq D^2\sum_{t=1}^{T}(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) + G^2\sum_{t=1}^{T}\eta_t \leq D^2\frac{1}{\eta_T} + G^2\sum_{t=1}^{T}\eta_t \leq 3DG\sqrt{T}
\end{aligned}
\tag{20}
$$

得到最后一个不等式的原因是 $\eta_t = \frac{D}{G\sqrt{t}}$ 和 $\sum_{t=1}^{T}\frac{1}{\sqrt{t}} \leq 2\sqrt{T}$。

36.7.4 二阶方法：在线牛顿步方法

到目前为止，我们仅考虑了最小化遗憾的一阶方法。本节考虑一个拟牛顿结果，即一个在线凸优化算法，该算法估计了二阶导数，或在超过一维时，估计了其黑塞矩阵。但严格地讲，此处分析的算法仍然是一阶算法，因为它仅使用了梯度的信息。下面引入并分析的算法被称为在线牛顿步 (online Newton step) 算法，它主要是借鉴常规优化问题的拟牛顿法。

二阶方法：在线牛顿步方法

在线牛顿步方法基本思想：

- 在每一次迭代时，这一算法选择一个向量，该向量是前面各步迭代中使用的向量与一个附加向量之和的投影向量。
- 对在线梯度下降算法，这一附加的向量是前一个代价函数的梯度向量，
- 而对在线牛顿步算法，这一向量则是不同的：它保持了在使用前面的代价函数时，使用离线 Newton-Raphson 方法能得到的方向。Newton-Raphson 算法会沿着黑塞矩阵的逆与梯度向量乘积的方向移动。在在线牛顿步算法中，这一方向为 $A_t^{-1}\nabla_t$，其中矩阵 A_t 是与黑塞矩阵相关的。
- 由于在当前的向量中增加了一个牛顿向量 $A_t^{-1}\nabla_t$ 的倍数，最终得到的点可能会在凸集之外，因此需要一个附加的投影步来得到 $\boldsymbol{x}_t$，即在时刻 t 的决策向量。这一投影与前面在线梯度下降算法中使用的标准欧氏投影是不同的. 它是在用 A_t 定义的范数的基础上得到的，而不是在欧氏范数意义下的。

二阶方法：在线牛顿步方法

在线牛顿步方法具体如下：

算法 12 在线牛顿步算法

1: **输入**：凸集 $\mathcal{K}, T, \boldsymbol{x}_1 \in \mathcal{K} \subseteq \mathbb{R}^n$，**参数** $\gamma, \epsilon > 0, A_0 = \epsilon I_n$
2: **for** $t = 1$ 到 T **do**
3: 执行 $\boldsymbol{x}_t$ 并考代价函数 $f_t(\boldsymbol{x}_t)$
4: 秩 1 更新：$A_t = A_{t-1} + \nabla_t \nabla_t^T$
5: 牛顿步及投影：

$$\boldsymbol{y}_{t+1} = \boldsymbol{x}_t - \frac{1}{\gamma} A_t^{-1} \nabla_t$$

$$\boldsymbol{x}_{t+1} = \Pi_{\mathcal{K}}^{A_t}(\boldsymbol{y}_{t+1})$$

6: **end for**

二阶方法：在线牛顿步方法

在线牛顿步算法的优点是，它有着对数遗憾，正如下面的定理所示，给出了在线牛顿步算法遗憾的界。

定理 4

参数为 $\gamma = \frac{1}{2}\min\frac{1}{4GD}, \alpha, \epsilon - \frac{1}{\gamma^2 D^2}$ 及 $T > 4$ 的算法 *12* 保证了

$$\text{遗憾}_T \leq 5(\frac{1}{\alpha} + GD)n\log T$$

注：该定理的详细证明可以参阅教材。

二阶方法：在线牛顿步方法

关于该算法的实现与运行时间：

- 在线牛顿步算法需要 $O(n^2)$ 的存储空间来存储矩阵 A_t。每一次迭代需要经计算矩阵 A_t^{-1}、当前梯度、一个矩阵于向量的乘积，以及可能需要的向量基本凸集 $\mathcal{K}$ 上的投影。
- 在每一次迭代时计算矩阵 A_t 的逆时：根据前面章节中介绍矩阵求逆的定理可得，对可逆矩阵 A 和向量 $\boldsymbol{x}$，有

$$(A + \boldsymbol{x}\boldsymbol{x}^T) = A^{-1} - \frac{A^{-1}\boldsymbol{x}\boldsymbol{x}^T A^{-1}}{1 + \boldsymbol{x}^T A^{-1}\boldsymbol{x}}$$

  因此，给定 A_{t-1}^{-1} 和 ∇_t，可以仅使用矩阵和向量的乘法和向量于向量的乘法，用 $O(n^2)$ 时间给出 A_t^{-1}。
- 在线牛顿步算法也需要在 $\mathcal{K}$ 上投影，但与在线梯度下降算法和其他在线凸优化算法的情形有所不同。此处需要的投影（记为 $\Pi_{\mathcal{K}}^{A_t}$）为向量在矩阵 A_t 诱导范数下的投影，即 $\|\boldsymbol{x}\|_{A_t} = \sqrt{\boldsymbol{x}^T A_t \boldsymbol{x}}$。它等价于求向量 $\boldsymbol{x} \in \mathcal{K}$，使其最小化 $(\boldsymbol{x} - \boldsymbol{y})^T A_t (\boldsymbol{x} - \boldsymbol{y})$，其中 $\boldsymbol{y}$ 为被投影的点。这是一个凸优化，它可以使用多项式时间得到任意精度的解。

本讲小结

复合优化方法

- 近似点梯度法
- 分块坐标下降法
- 交替方向乘子法

在线优化算法

- 在线优化模型、投资组合选择
- 在线梯度下降法和在线牛顿步算法

随机优化算法

- SGD：一般方法及概念区分、收敛性、优缺点
- 动量法：动量梯度下降法、Nestrov 动量法
- 自适应学习速率
- AdaGrad、RMSProp、Adam

本节内容介绍了深度学习时代，梯度法的一些变种，用于解决处理大数据问题时，传统梯度方法面临的挑战。至此，本门课程的内容也全部结束。尽管如此，还是需要强调数据科学一直在不断地发展，研究结果日新月异。本门课程难免存在疏漏，欢迎大家提出宝贵意见。