第七章 复合优化算法

修贤超

https://xianchaoxiu.github.io

目录

监督学习模型

- 假定 (a, b) 服从概率分布 P，其中 a 为输入，b 为标签
  - 在自动邮件分类任务中，a 表示邮件内容，b 表示正常邮件或垃圾邮件
  - 在人脸识别任务中，a 表示人脸的图像信息，b 表示该人脸属于何人
- 实际问题中我们不知道真实的概率分布 P，而是随机采样得到一个数据集

$$\mathcal{D} = \{(a_1, b_1), (a_2, b_2), \cdots, (a_N, b_N)\}$$

数据集 $\mathcal{D}$ 对应经验分布

$$\hat{P} = \frac{1}{N} \sum_{n=1}^{N} \delta_{a_i, b_i}$$

监督学习模型

- 监督学习的任务是要给定输入 a 预测标签 b，即决定一个最优的函数 ϕ 使得期望风险最小

$$\mathbb{E}[L(\phi(a), b)]$$

- ℓ_2 损失函数

$$L(x, y) = \frac{1}{2}\|x - y\|_2^2$$

- 若 $x, y \in \mathbb{R}^d$ 为概率分布，则可定义互熵损失函数

$$L(x, y) = \sum_{i=1}^{d} x_i \log \frac{x_i}{y_i}$$

- 为了缩小目标函数的范围，需要将 $\phi(\cdot)$ 参数化为 $\phi(\cdot; x)$

监督学习模型

- 用经验风险来近似期望风险，即要求解下面的极小化问题

$$\min_{x} \quad \frac{1}{N}\sum_{i=1}^{N} L(\phi(a_i;x),b_i) = \mathbb{E}_{(a,b)\sim\hat{P}}[L(\phi(a;x),b)]$$

- 记 $f_i(x) = L(\phi(a_i;x),b_i)$, 则只需考虑如下随机优化问题

$$\min_{x\in\mathbb{R}^n} \quad f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$$

- 由于数据规模巨大，通过采样的方式只计算部分样本的梯度来进行梯度下降

随机梯度下降算法 (SGD)

- SGD 的基本迭代格式为

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k), \quad \nabla f(x^k) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x^k)$$

$$\Downarrow$$

$$x^{k+1} = x^k - \alpha_k \nabla f_{s_k}(x^k)$$

  - s_k 是从 $\{1, 2, \cdots, N\}$ 中随机等可能地抽取的一个样本
  - α_k 称为步长．在机器学习和深度学习领域中称为学习率

- 要保证随机梯度的条件期望恰好是全梯度，即

$$\mathcal{E}_{s_k}[\nabla f_{s_k}(x^k)|x^k] = \nabla f(x^k)$$

随机梯度法

- **小批量（mini-batch）随机梯度法** 每次迭代中，随机选择一个元素个数很少的集合 $I_k \subset \{1, 2, \cdots, N\}$，然后执行迭代格式

$$x^{k+1} = x^k - \alpha_k \nabla f_{s_k}(x^k)$$

$$\Downarrow$$

$$x^{k+1} = x^k - \frac{\alpha_k}{|\mathcal{I}_k|} \sum_{s \in \mathcal{I}_k} \nabla f_s(x^k)$$

- **随机次梯度法** 当 $f_i(x)$ 是凸函数但不一定可微时，可以用 $f_i(x)$ 的次梯度代替梯度进行迭代

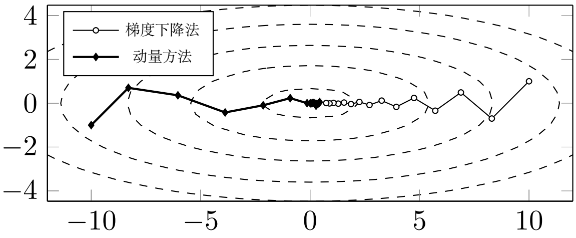
$$x^{k+1} = x^k - \alpha_k g^k$$

动量方法

- 动量方法的具体迭代格式如下

$$
\begin{aligned}
v^{k+1} &= \mu_k v^k - \alpha_k \nabla f_{s_k}(x^k) \\
x^{k+1} &= x^k + v^{k+1}
\end{aligned}
$$

- 参数 μ_k 的范围是 $[0,1)$，通常取 $\mu_k \geq 0.5$，其含义为迭代点带有较大惯性，每次迭代会在原始迭代方向的基础上做一个小的修正. 当 $\mu_k = 0$ 时退化成随机梯度下降法



Nesterov 加速算法

- 假设 $f(x)$ 为光滑的凸函数, 针对凸问题的 Nesterov 加速算法为

$$
\begin{aligned}
y^{k+1} &= x^k + \mu_k(x^k - x^{k-1}) \\
x^{k+1} &= y^k - \alpha_k \nabla f(y^k)
\end{aligned}
$$

- 针对光滑问题的 Nesterov 加速算法迭代的随机版本为

$$
\begin{aligned}
y^{k+1} &= x^k + \mu_k(x^k - x^{k-1}) \\
x^{k+1} &= y^{k+1} - \alpha_k \nabla f_{s_k}(y^{k+1})
\end{aligned}
$$

其中 $\mu_k = \frac{k-1}{k+2}$，步长 α_k 是一个固定值或者由线搜索确定

- 二者的唯一区别为随机版本将全梯度 $\nabla f(y^k)$ 替换为随机梯度 $\nabla f_{s_k}(y^{k+1})$

Nesterov 加速算法与动量方法的联系

- 引入速度变量 $v^k = x^k - x^{k-1}$，结合原始 Nesterov 加速算法的两步迭代得到

$$x^{k+1} = x^k + \mu_k(x^k - x^{k-1}) - \alpha_k \nabla f_k(x^k + \mu_k(x^k - x^{k-1}))$$

- 定义 $v^{k+1} = \mu_k v^k - \alpha_k \nabla f_k(x^k + \mu_k v^k)$, 于是关于 x^k 和 v^k 的等价迭代式

$$\begin{aligned} v^{k+1} &= \mu_k v^k - \alpha_k \nabla f_{s_k}(x^k + \mu_k v^k) \\ x^{k+1} &= x^k + v^{k+1} \end{aligned}$$

- 与动量方法相比

$$\begin{aligned} v^{k+1} &= \mu_k v^k - \alpha_k \nabla f_{s_k}(x^k) \\ x^{k+1} &= x^k + v^{k+1} \end{aligned}$$

Nesterov 加速算法先对点施加速度的作用再求梯度，即对动量方法做了校正

- 令 $g^k = \nabla f_{s_k}(x^k)$，引入

$$G^k = \sum_{i=1}^{k} g^i \odot g^i$$

- 当 G^k 的某分量较大时，认为该分量变化比较剧烈，应采用小步长，反之亦然
- AdaGrad 的迭代格式为

$$x^{k+1} = x^k - \frac{\alpha}{\sqrt{G^k + \varepsilon 1_n}} \odot g^k$$
$$G^{k+1} = G^k + g^{k+1} \odot g^{k+1}$$

AdaGrad 的收敛阶

- 如果在 AdaGrad 中使用真实梯度 $\nabla f(x^k)$，那么 AdaGrad 也可以看成是一种介于一阶和二阶的优化算法
- 考虑 $f(x)$ 在点 x^k 处的二阶泰勒展开

$$f(x) \approx f(x^k) + \nabla f(x^k)^\top (x - x^k) + \frac{1}{2}(x - x^k)^\top B^k (x - x^k)$$

- 选取不同的 B^k 可以导出不同的优化算法，例如 AdaGrad 选择

$$B^k = \frac{1}{\alpha}\mathrm{Diag}(\sqrt{G^k + \varepsilon 1_n})$$

- AdaGrad 会累加之前所有的梯度分量平方，这就导致步长是单调递减的，因此在训练后期步长会非常小，计算的开销较大

RMSProp

- RMSProp（root mean square propagation）是对 AdaGrad 的一个改进，在非凸问题上可能表现更好
- RMSProp 提出只需使用离当前迭代点比较近的项，同时引入衰减参数 ρ。具体地，令

$$M^{k+1} = \rho M^k + (1-\rho) g^{k+1} \odot g^{k+1}$$

再对其每个分量分别求根，就得到均方根 (root mean square)

$$R^k = \sqrt{M^k + \varepsilon 1_n}$$

最后将均方根的倒数作为每个分量步长的修正

- RMSProp 迭代格式

$$
\begin{aligned}
x^{k+1} &= x^k - \frac{\alpha}{\sqrt{G^k + \varepsilon 1_n}} \odot g^k \\
G^{k+1} &= G^k + g^{k+1} \odot g^{k+1}
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
x^{k+1} &= x^k - \frac{\alpha}{R^k} \odot g^k \\
M^{k+1} &= \rho M^k + (1-\rho) g^{k+1} \odot g^{k+1}
\end{aligned}
$$

- RMSProp 和 AdaGrad 的唯一区别是将 G^k 替换成了 M^k
- 一般取 $\rho = 0.9$, $\alpha = 0.001$

Adam

- Adam 选择了一个动量项进行更新

$$S^k = \rho_1 S^{k-1} + (1-\rho_1) g^k$$

- 类似 RMSProp，Adam 也会记录梯度的二阶矩

$$M^k = \rho_2 M^{k-1} + (1-\rho_2) g^k \odot g^k$$

- 与原始动量方法和 RMSProp 的区别是，由于 S^k 和 M^k 本身带有偏差，Adam 在更新前先对其进行修正

$$\hat{S}^k = \frac{S^k}{1-\rho_1^k}, \quad \hat{M}^k = \frac{M^k}{1-\rho_2^k}$$

- Adam 最终使用修正后的一阶矩和二阶矩进行迭代点的更新

$$x^{k+1} = x^k - \frac{\alpha}{\sqrt{\hat{M}^k + \varepsilon 1_n}} \odot \hat{S}^k$$

$Q\&A$

$Thank\ you!$

感谢您的聆听和反馈