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
- 例如在自动邮件分类任务中，a 表示邮件内容，b 表示邮件为正常邮件或垃圾邮件
- 又例如人脸识别任务中，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

- 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 $\mathbb{E}[L(\phi(a), b)]$ 最小，其中 $L(\cdot, \cdot)$ 表示损失函数，函数 ϕ 为某个函数空间中的预测函数
- 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（即各分量和为 1 的向量），则可定义互熵损失函数

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

监督学习模型

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

$$\phi(a) = pa + q$$

- 深度神经网络

$$\begin{aligned}
\phi_0(a) &= a \\
\hat{\phi}_l(a) &= W_l \phi_{l-1}(a) + b_h, \quad \phi_l(a) = \sigma(\phi_l(a)) \\
\phi(a) &= \hat{\phi}_L(a)
\end{aligned}$$

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

梯度下降算法

- 用假设每一个 $f_i(x)$ 是凸的、可微的
- 可以运用梯度下降算法

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

$$\nabla f(x^k) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(x^k)$$

- 计算 $\nabla f(x^k)$ 需要非常大的计算量

随机梯度下降算法 (SGD)

- SGD 的基本迭代格式为

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

其中 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，更多的时候被称为学习率 (learning rate)

- 随机梯度算法不去计算全梯度 $\nabla f(x^k)$，而是从众多样本中随机抽出一个样本 s_i，然后仅仅计算这个样本处的梯度 $\nabla f_{s_k}(x^k)$，以此作为 $\nabla f(x^k)$ 的近似

- 要保证随机梯度的条件期望恰好是全梯度，即

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

小批量随机梯度法

- 实际计算中每次只抽取一个样本 s_k 的做法比较极端，常用的形式是小批量（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 $_k \subset \{1, 2, \cdots, N\}$，然后执行迭代格式

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

其中 $|\mathcal{I}_k|$ 表示 $_k$ 中的元素个数

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，这就是随机次梯度算法.
- 迭代格式为

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

其中 α_k 为步长，$g^k \in \partial f_{s_k}(x^k)$ 为随机次梯度，其期望为真实的次梯度

动量方法

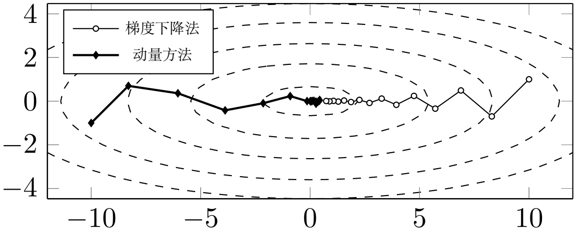
- 在算法迭代时一定程度上保留之前更新的方向，同时利用当前计算的梯度调整最终的更新方向
- 动量方法的具体迭代格式如下

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

- 在计算当前点的随机梯度 $\nabla f_{s_i}(x^k)$ 后，并不是直接将其更新到变量 x^k 上，而是将其和上一步更新方向 v^k 做线性组合来得到新的更新方向 v^{k+1}

动量方法

- 由动量方法迭代格式立即得出当 $\mu_k = 0$ 时该方法退化成随机梯度下降法. 在动量方法中，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
- 在普通的梯度法中，每一步迭代只用到了当前点的梯度估计，动量方法的更新方向还使用了之前的梯度信息
- 当许多连续的梯度指向相同的方向时，步长就会很大，这从直观上看也是非常合理的



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即版本将全梯度 $\nabla f(y^k)$ 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

- 若在第 k 步迭代引入速度变量 $v^k = x^k - x^{k-1}$，再合并原始 Nesterov 加速算法的两步迭代可以得到

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

- 定义有关 v^{k+1} 的迭代式

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

- 于是得到关于 x^k 和 v^k 的等价迭代

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

- 二者的主要差别在梯度的计算上，Nesterov 加速算法先对点施加速度的作用，再求梯度，可以理解为对标准动量方法做了校正

AdaGrad

- 在一般的随机梯度法中，调参是一个很大的难点．我们希望算法能在运行的过程中，根据当前情况自发地调整参数．
- 对无约束光滑凸优化问题，点 x 是问题的解等价于该点处梯度为零向量．但梯度的每个分量收敛到零的速度是不同的．传统梯度算法只有一个统一的步长 α_k 来调节每一步迭代，它没有针对每一个分量考虑
- 当梯度的某个分量较大时，可以推断出在该方向上函数变化比较剧烈，要用小步长；当梯度的某个分量较小时，在该方向上函数比较平缓，要用大步长．AdaGrad 就是根据这个思想设计的

AdaGrad

- 令$g^k=\nabla f_{s_k}(x^k)$，为了记录整个迭代过程中梯度各个分量的累积情况，引入

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

从 G^k 的定义可知 G^k 的每个分量表示在迭代过程中，梯度在该分量处的累积平方和．当 G^k 的某分量较大时，我们认为该分量变化比较剧烈，因此应采用小步长，反之亦然．

- AdaGrad 的迭代格式为

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

- 这里 $\frac{\alpha}{\sqrt{G^k+\varepsilon \mathbf{1}_n}}$ 中的除法和求根运算都是对向量每个分量分别操作的（下同），α 为初始步长，引入 $\varepsilon \mathbf{1}_n$ 这一项是为了防止除零运算

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．AdaGrad 是使用一个对角矩阵来作为 B^k．具体地，取

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

时导出的算法就是 AdaGrad

RMSProp

- RMSProp（root mean square propagation）是对 AdaGrad 的一个改进，该方法在非凸问题上可能表现更好．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 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

- RMSProp 迭代格式为

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

- 引入参数 ε 同样是为了防止分母为 0 的情况发生．一般取 $\rho = 0.9$, $\alpha = 0.001$
- 可以看到 RMSProp 和 AdaGrad 的唯一区别是将 G^k 替换成了 M^k.

AdaDelta

- AdaDelta 在 RMSProp 的基础上，对历史的 Δx^k 也同样累积平方并求均方根

$$D^k = \rho D^{k-1} + (1-\rho)\Delta x^k \odot \Delta x^k$$
$$T^k = \sqrt{D^k + \varepsilon 1_n}$$

  然后使用 T^{k-1} 和 R^k 的商对梯度进行校正

$$\Delta x^k = -\frac{T^{k-1}}{R^k} \odot g^k$$
$$x^{k+1} = x^k + \Delta x^k$$

- AdaDelta 的特点是步长选择较为保守，同时也改善了 AdaGrad 步长单调下降的缺陷

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