大规模随机优化基础

张旋

中国科学技术大学

2025 年 3 月 27 日

目录

简介

大规模数据优化

- 数学优化是机器学习的基石之一。大规模机器学习中，训练数据和参数的数量都很大，这代表了一个与传统非线性优化方法通常会遇到困难的独特情境。
- 本次讨论班将简要介绍一些典型的优化问题，这些问题来自机器学习，随后将转向随机算法（以及其他流行方法），并给出适用的具体模型。

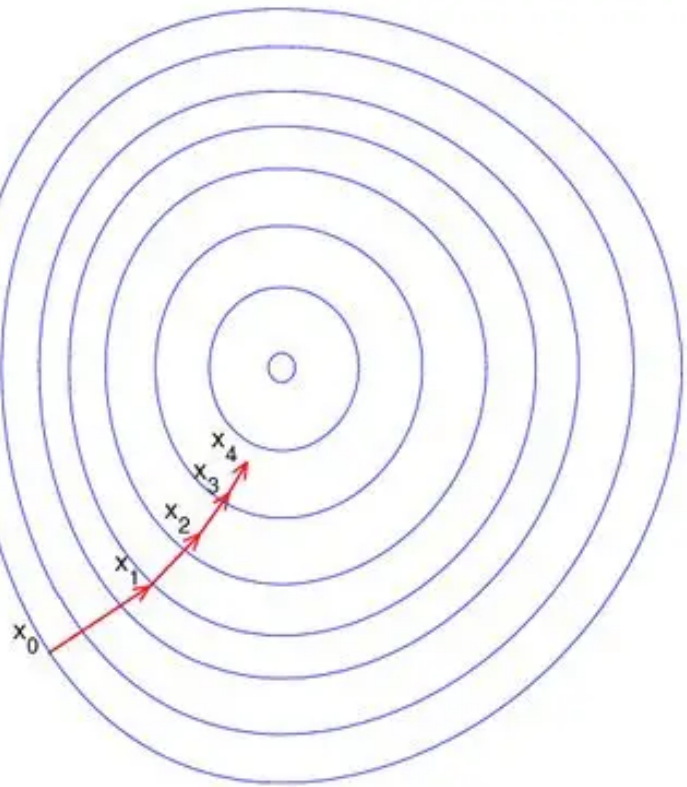


图 1. 大规模数据优化

典型的问题

简单起见，我们关注的是在有监督分类问题中出现的优化问题；也就是说，我们关注的是通过基于一组标注的训练数据中的信息，优化预测函数来标注未见过的数据。
这样分析是合理的，因为许多无监督学习和其他学习技术也可以归结为形式类似的优化问题。

典型问题举例

- **回归**：尽管回归的处理方法与分类不同，但回归与有监督分类有相似的模型。有监督分类和回归统称为有监督学习。
- **深度强化学习**：在深度 Q 网络（DQN）中，样本通过与环境的交互获得，训练代理（agent）的目标是以回归的方式求解贝尔曼方程。
- **生成对抗网络**：生成对抗网络（GAN）由生成器和判别器组成，通常是交替训练的。每个部分的训练过程可以视为有监督分类，其中标签表示样本是否来自数据分布。

基本原理

数学原理

- **目标**：确定一个预测函数 $h : \mathcal{X} \to \mathcal{Y}$ 其中从输入空间 $\mathcal{X}$ 到输出空间 $\mathcal{Y}$。- 这里的目标是通过选择一个合适的函数 h，使其能够根据输入数据进行预测输出。
- **要求**：h 应避免死记硬背，而是要能够从给定的一组样本中泛化出可以学习的概念。- 这意味着 h 不能仅仅记住训练数据，而要能够应用于未见过的新数据，并且要能够从数据中提取出有效的规律。
- **方案**：通过试图最小化风险度量来选择 h，该度量是在一个充分选择的预测函数族 $\mathcal{H}$ 上进行的。

核心思想是通过优化一个风险度量（例如，预测误差）来选择合适的预测函数 h，并确保这个函数来自一个经过合理选择的预测函数集合。

基本原理

1. 前提假设： 假设样本是从一个联合概率分布 $P(x, y)$ 中抽取的。
2. 目标函数： 目标是找到一个函数 h ，使得它对所有可能的输入 x 都能最小化误分类的期望风险。即要最小化下面的公式：

$$R(h) = \Pr[h(x) \neq y] = E[1[h(x) \neq y]]$$

这里，$R(h)$ 表示函数 h 的期望误分类风险，$\Pr[h(x) \neq y]$ 表示函数 $h(x)$ 预测错误的概率，$E[1[h(x) \neq y]]$ 是对误分类的期望。
3. 变分框架和随机性： 这个框架被称为 " 变分 " 的（variational），因为我们在优化一个函数集合。它也是 " 随机 " 的（stochastic），因为目标函数涉及期望值，表示在多个样本上的平均误分类概率。
4. 实践中的应用： 在实际应用中，期望值通常是通过对样本进行求和来近似的。给定样本 $\{(x_i, y_i)\}_{i=1}^{n}$ ，函数 h 应该最小化经验风险，即对样本的误分类进行求和：

$$R_n(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\left[h\left(x_i\right) \neq y_i\right], \quad \text{其中} \mathbf{1}[A] = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

预测函数族的选择

函数族 $\mathcal{H}$ 的确定应考虑三个潜在的竞争目标。

1. 足够的能力： $\mathcal{H}$ 应该包含能够在训练集上实现低经验风险的预测函数，以避免数据拟合不足。
这可以通过选择丰富的函数族或通过使用先验知识来选择目标明确的函数族来实现。

2. 低泛代误差： 期望风险与经验风险之间的差距 $R(h) - R_n(h)$ 应该对所有 $h \in \mathcal{H}$ 都很小。
通常，当使用更多的训练样本时，这个差距会减小，但当使用更丰富的函数族时，这个差距会增大，这是由于可能发生的过拟合问题。

3. 高效训练： 选择 $\mathcal{H}$ 可以有效地解决对应的优化问题，当使用更丰富的函数族或更大的训练集时，优化问题的难度可能会增加。

结构化误差

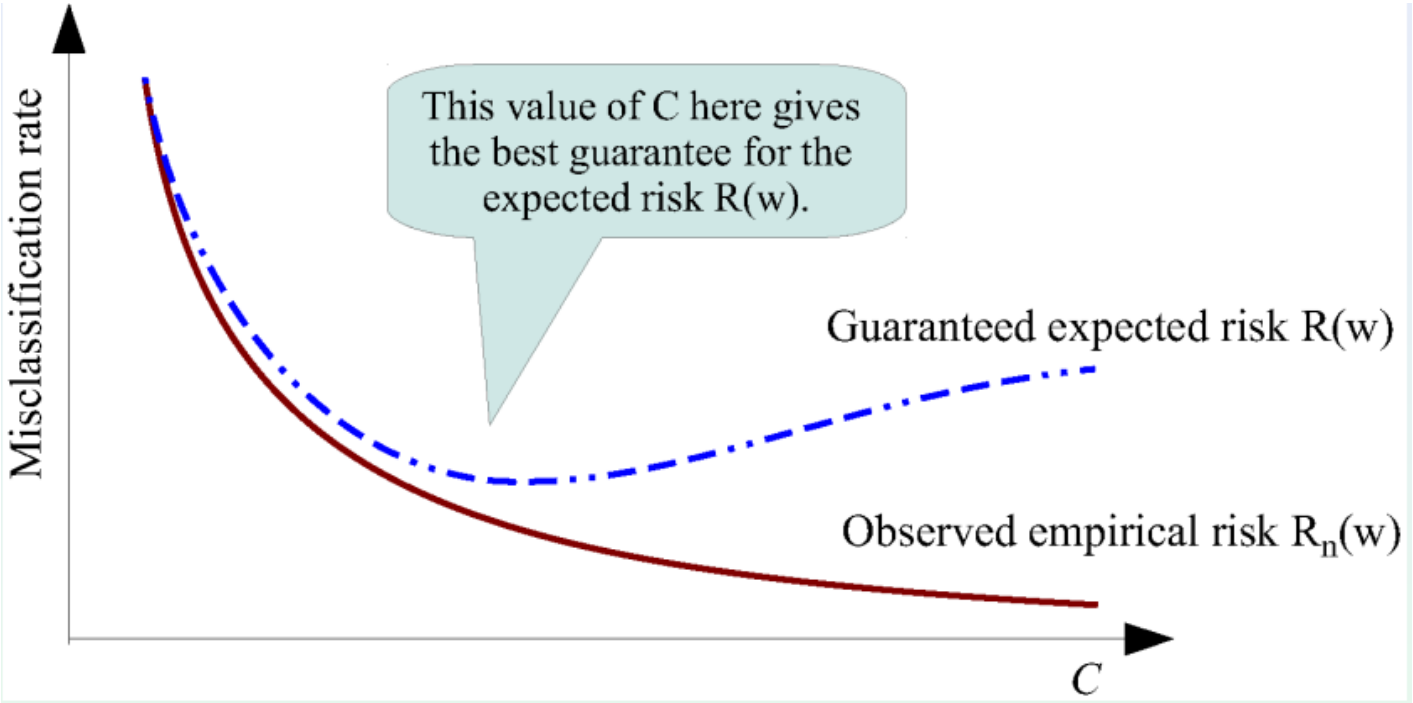


图 2. 结构化误差

- 蓝色虚线：期望风险 $R(w)$ 的变化趋势。
- 红色实线：经验风险 $R_n(w)$ 的变化趋势。
- 最优的 C 值，在这个点上，期望风险和经验风险之间的差距最小，且误分类率处于最低。

目录

简要介绍

随机算法介绍

- **批量梯度下降法**：为了最小化经验风险，权重 w 通过以下公式更新：

$$w_{k+1} = w_k - \alpha_k \nabla R_n(w_k) = w_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(w_k)$$

其中，$\alpha_k > 0$ 是步长。计算步骤 $-\alpha_k \nabla R_n(w_k)$ 是昂贵的，因为它需要访问所有样本。

- **随机梯度（SG）**：相比之下，随机梯度（SG）在每次迭代中只使用一个样本：

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k)$$

其中 i_k 是从 $\{1, \ldots, n\}$ 中随机选择的索引。虽然 $-\nabla f_{i_k}(w_k)$ 可能不是从 w_k 出发的下降方向，但如果它在期望上是下降方向，那么序列 $\{w_k\}$ 可以引导到 R_n 的最小值。

简要介绍

Q1：两者的本质区别？

迭代过程中样本的数量区别

- 批量梯度下降法（Batch Gradient Descent）：在每次迭代中，使用所有样本来计算梯度。这种方法需要计算整个样本集的平均梯度，因此在计算上非常昂贵。
- 随机梯度下降法（Stochastic Gradient Descent，SGD）：每次迭代仅使用一个样本来计算梯度。虽然每次计算的梯度方向可能不是最优的下降方向，但在多次迭代之后，SGD 能够有效地逼近经验风险

Q2：如何推广 SG 方法？

为了推广 SG 方法，我们考虑了两种方法：

- 通过生成一批样本而不是单个样本来减少每次迭代的噪声（方差）
- 利用二阶信息并计算随机牛顿或拟牛顿方向而不是梯度方向。

简要介绍

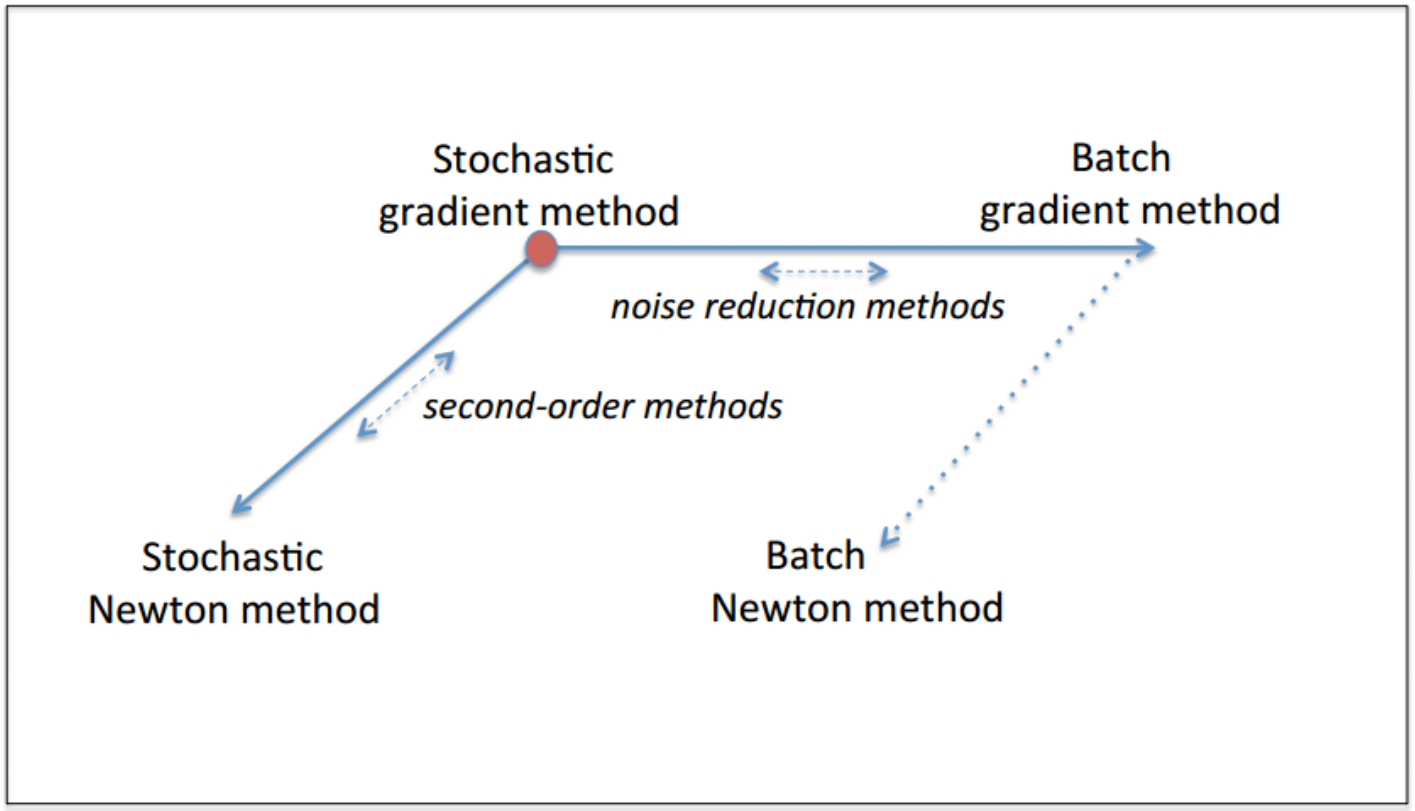


图 3. 机器学习优化方法的二维谱示意图

随机梯度下降法

随机梯度下降法（SGD）对训练数据做随机采样，其更新公式如下：

$$w^{t+1} = w^t - \eta^t \nabla f_{i^t}\left(w^t\right)$$

其中，i^t 是第 t 轮随机采样的数据标号。具体算法如下列的伪代码所示：

Algorithm 1: 随机梯度下降法

Input: 函数 $f(\cdot)$，初始参数 w^0，迭代轮数 T

Output: 终止参数 w^T

```
for t = 0, 1, ··· , T − 1 do
    随机采一个样本 i^t ∈ {1, ··· , n};
    计算梯度 ∇f_{i^t}(w^t);
    更新参数 w^{t+1} = w^t − η^t ∇f_{i^t}(w^t);
end
return w^T
```

随机梯度下降法

算法描述

- 我们知道，机器学习问题中的经验损失函数定义为所有样本数据对应的损失函数的平均值。而我们这里用有放回随机采用获得的数据来计算梯度，是对用全部数据来计算梯度的一个无偏估计，即 $\mathbb{E}_{i^t}\nabla_{i^t} f\left(w^t\right) = \nabla f\left(w^t\right)$ ，注意此处 $f\left(w^t\right) = \frac{1}{n}\sum_{i=1}^{n}\nabla f_i\left(w^t\right)$)。而由于每次更新只随机采一个样本，随机梯度中的计算量大大减小。因此，随机梯度可以作为梯度的自然替代，从而大大提高学习效率。
- 不过正如我们前面所说，优化算法的复杂度不仅包括单位计算复杂度，还包括迭代次数复杂度（与收敛率有关）。天下没有免费的午餐，随机梯度下降单位计算复杂度降低，那么也相应地会付出迭代次数复杂度增大的代价。

随机梯度下降法

算法描述

- 考虑实际每次只采一个样本比较极端，常用的形式是随机梯度下降法的一个推广：小批量（mini - batch）随机梯度下降法。该方法可以看做是在随机优化算法和确定性优化算法之间寻找某种折中，每次采一个较小的样本集合 $\mathcal{I}^t \in \{1, 2, \dots n\}$（多于单样本，少于全样本），然后执行更新公式：

$$w^{t+1} = w^t - \eta^t \nabla f_{\mathcal{I}^t}\left(w^t\right) = w^t - \frac{\eta^t}{|\mathcal{I}^t|} \sum_{i \in \mathcal{I}^t} \nabla f_i\left(w^t\right)$$

我们之后介绍的各种随机优化方法，都有相应的小批量版本。小批量采样可以有效减小方差，从而提高收敛速率。

收敛性和计算复杂度分析

与梯度下降法类似，对不同性质的目标函数，随机梯度下降法具有不同的收敛速率。

L - Lipschitz 连续凸函数收敛性

假设目标函数 $f\colon \mathbb{R}^d \to \mathbb{R}$ 是凸函数，并且 L - Lipschitz 连续，$w^* = \arg\min_{\|w\| \leqslant D} f(w)$，当步长 $\eta^t = \sqrt{\frac{D^2}{L^2 t}}$ 时，对于给定的迭代步数 T，随机梯度下降法具有 $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$ 的次线性收敛速率：

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} f\left(w^t\right) - f\left(w^*\right)\right] \leqslant \frac{LD}{\sqrt{T}}$$

收敛性和计算复杂度分析

光滑强凸函数收敛性：

假设目标函数 $f\colon \mathbb{R}^d \to \mathbb{R}$ 是 α - 强凸函数，并且 β 光滑，如果随机梯度的二阶矩有上界，即 $\mathbb{E}_{i^t}\left\|\nabla f_{i^t}\left(w^t\right)\right\|^2 \leqslant G^2$ ，当步长 $\eta^t = \frac{1}{\alpha t}$ 时，对于给定的迭代步数 T ，随机梯度下降法具有 $\mathcal{O}\left(\frac{1}{T}\right)$ 的次线性收敛速率：

$$\mathbb{E}\left[f\left(w^T\right) - f\left(w^*\right)\right] \leqslant \frac{2\beta G^2}{\alpha^2 T}$$

收敛性和计算复杂度分析

收敛速率对比

- 通过与梯度下降法的收敛速率进行对比，我们可以发现随机梯度下降法的收敛速率更慢。这主要是由于虽然随机梯度是是全梯度的无偏估计，但这种估计存在一定的方差，会引入不确定性，导致最终算法的收敛速率下降。
- 虽然随机梯度下降法的收敛速率慢于梯度下降法，但因为其每一轮的单位计算复杂度为 $\mathcal{O}(d)$ ，而梯度下降法每一轮的单位计算复杂度为 $\mathcal{O}(nd)$ ，所以当样本量很大时，随机梯度下降法的总计算复杂度 $\mathcal{O}\left(d\left(\frac{1}{\varepsilon}\right)\right)$ 比梯度下降法的总计算复杂度 $\mathcal{O}\left(ndQ\log\left(\frac{1}{\epsilon}\right)\right)$ 要低。

目录

方差缩减技术

方差缩减

方差缩减 (Variance Reduction) 是用于加速 SGD 收敛的一种技术，目的是减少单个样本梯度估计的噪声，改善收敛速度。

方差缩减技术

由于训练样本中包含大量冗余信息，因此自提出以来，SGD 方法就非常受欢迎。但是，随机梯度法只能以亚线性速率收敛，并且梯度的方差通常很大。如何减少方差并提高 SGD 到线性收敛一直是一个重要的问题。

方差缩减技术

以下是一些常见的方差缩减技术:

方差缩减技术

- SVRG(Stochastic Variance Reduced Gradient)SVRG 通过引入一个” 参考点”，即计算一次整个数据集的梯度，然后在每次迭代时使用这个参考点来减少方差。SVRG 通过减少噪声使得每次更新更加稳定，从而加速收敛。
- SAGA 是一种基于梯度累积的方差缩减方法，通过维护所有样本的历史梯度估计来减少方差。在每次迭代时，SAGA 更新的是每个样本的梯度平均值，这样能够减少梯度估计的声，提高收敛速度。

SVRG

SVRG (stochastic variance reduced gradient)

SVRG 方法是在一个循环中进行的，在每次循环之前，w_k 可用于计算

$$\nabla R_n(w_k) = \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(w_k)$$

初始化 $\tilde{w}_1 \leftarrow w_k$ ，进行一系列由 j 引导的 m 次内循环，其更新格式是:

$$\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha\tilde{g}_j$$
$$\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - \left(\nabla f_{i_j}(w_k) - \nabla R_n(w_k)\right)$$

$i_j \in \{1, 2, 3, \ldots\}$ 是随机选取的, 而且我们可以发现 $\tilde{g}_j$ 是 $\nabla R(\tilde{w}j)$ 的一个无偏估计, 这样我们就可以和全梯度下降 (GD) 联系起来。方差会减小, 所以最终该算法会以线, 收敛到最优值。

SVRG

算法 5.1 用于最小化经验风险 R_n 的 SVRG 方法

1. 选择初始迭代 $w_1 \in \mathbb{R}^d$，步长 $\alpha > 0$ 和正整数 m。
2. 对于 $k = 1, 2, \ldots$，执行以下操作:
3. 计算批量梯度 $\nabla R_n(w_k)$。
4. 初始化 $\tilde{w}_1 \leftarrow w_k$。
5. 对于 $j = 1, \ldots, m$，执行以下操作:
6. 从 $\{1, \ldots, n\}$ 中均匀选择 i_j。
7. 设 $\tilde{g}_j \leftarrow \nabla f_{i_j}(\tilde{w}_j) - (\nabla f_{i_j}(w_k) - \nabla R_n(w_k))$。
8. 设 $\tilde{w}_{j+1} \leftarrow \tilde{w}_j - \alpha \tilde{g}_j$。
9. 结束。
10. 选项 (a): 设 $w_{k+1} = \tilde{w}_{m+1}$。
11. 选项 (b): 设 $w_{k+1} = \frac{1}{m} \sum_{j=1}^{m} \tilde{w}_{j+1}$。
12. 选项 (c): 从 $\{1, \ldots, m\}$ 中均匀选择 j，并设 $w_{k+1} = \tilde{w}_{j+1}$。
13. 结束。

SAG

SAG（Stochastic Average Gradient）

提出了 SAG（Stochastic Average Gradient）方法。它的特点是与 SGD（随机梯度下降）算法计算量小，并且具有像 FGD（Faster Gradient Descent）那样的线性收敛速度。其一般形式如下：

$$W^{k+1} = W^k - \alpha_k \sum_{i=1}^{m} y_i^k$$

其中，y_i^k 是一个向量，每次迭代中随机选择 $i_k \in \{1, 2, \ldots, m\}$ ，我们有：

$$y_i^k = \begin{cases} \nabla f_i\left(W^k\right), & \text{if } i = i_k \\ y_i^{k-1}, & \text{otherwise} \end{cases}$$

SAG

SAG（Stochastic Average Gradient）

通过这种方式，我们可以看到，每次迭代时，算法通过选择一个索引 i_k 来更新对应的梯度，并为每个索引保留最新计算的梯度值。
最终的计算是通过以下公式来进行的：

$$d = \sum_{i=1}^{m} y_i$$

简单来说，SAG 方法是在每一步迭代中利用上一次计算的梯度来更新当前的权重，从而加速收敛过程。这种方法将每次更新的梯度保存下来，避免重复计算，从而有效提高计算效率。

SAG

理解：SAG 方法就是一开始我们在 x^0 处计算每个梯度 $\nabla f_i\left(x^0\right)$，然后把这些梯度全部存储到内存中。在每次迭代时，我们随机选择一个样本的损失函数 $f_i(x)$，计算其梯度 $\nabla f_i(x)$，并使用它替换当前存储位置的梯度，随后我们就得到了新的解。其公式为：

$$d = d - y_i + \nabla f_i(x)$$

它类似于 SGD（随机梯度下降），但每次迭代时我们需要选择两个梯度，并且都将计算出的梯度代入原来位置的梯度，并保持该梯度。

SAG 收敛性

对于 SAG，我们并没有实质性地说明它降低了方差，但是我们有理论证明它在步长 $\alpha = \frac{1}{16L}$ 时线性收敛并具有最优性。

SAGA

SAG（Stochastic Average Gradient）

SAGA 算法是 SAG 算法的一个加速版，和 SAG 一样，它既不在一个循环里面操作，也不计算批量梯度（除了在初始化时）。在每次迭代中，它都计算随机梯度 g_k 作为当前梯度的平均值。
特别地，在迭代过程中，算法将会存储每个样本的 $\nabla f_i(w[i])$ 对于前期所有的 $i \in \{1, 2, 3, \ldots\}$，其中 $w[i]$ 表示 $\nabla f_i(w)$ 计算出的值的前一次迭代。整数 $j \in \{1, 2, 3, \ldots\}$ 是选择的索引，且是随机选择的。然后更新规则如下：

$$g_k \leftarrow \nabla f_j\left(w_k\right) - \nabla f_j(w[j]) + \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w[i])$$

可以看出，g_k 的结构也表现出了它是 $\nabla R_n\left(w_k\right)$ 的一个无偏估计。每次迭代时我们都需要存储前期的梯度。

SAGA

他的具体算法如下：

Algorithm 5.2 SAGA Method for Minimizing an Empirical Risk R_n

Choose an initial iterate $w_1 \in \mathbb{R}^d$ and stepsize $\alpha > 0$.
for $i = 1, \ldots, n$ **do**
    Compute $\nabla f_i(w_1)$.
    Store $\nabla f_i(w_{[i]}) \leftarrow \nabla f_i(w_1)$.
end for
for $k = 1, 2, \ldots$ **do**
    Choose j uniformly in $\{1, \ldots, n\}$.
    Compute $\nabla f_j(w_k)$.
    Set $g_k \leftarrow \nabla f_j(w_k) - \nabla f_j(w_{[j]}) + \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_{[i]})$.
    Store $\nabla f_j(w_{[j]}) \leftarrow \nabla f_j(w_k)$.
    Set $w_{k+1} \leftarrow w_k - \alpha g_k$.
end for

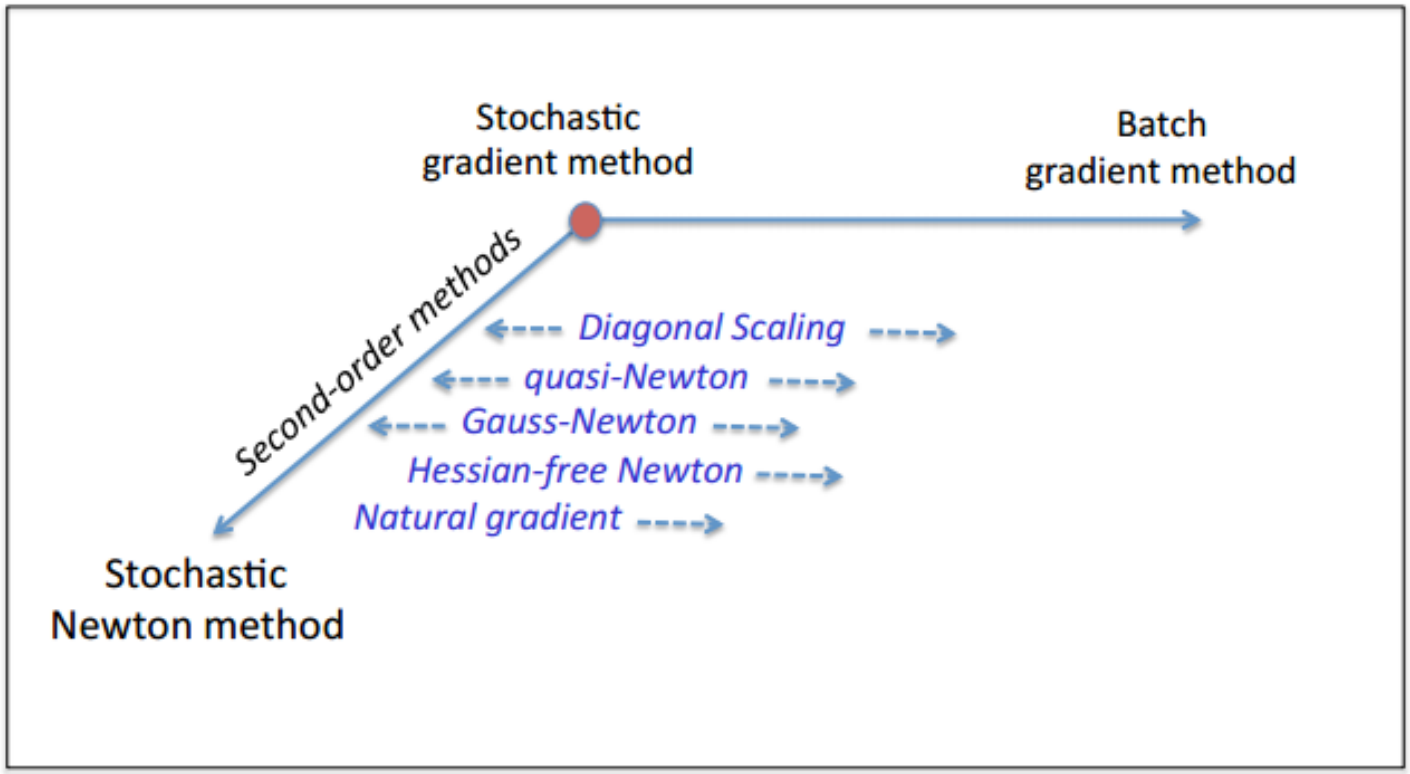
SAGA 也减少了噪音的影响。(SAGA 和 SAG 的区别就是一个是无偏估计，一个不是)

目录

二阶方法

除了减少随机方向中的噪声外，超越传统 SG 方法的另一种方式是通过使用二阶信息来解决目标函数的高非线性和病态条件的负面影响。
已知确定性方法能从二阶信息（Hessian 矩阵）的使用中获益；例如，牛顿法能够实现局部二次收敛，意味着在最优解附近，它的收敛速度非常快。

二阶方法



我们使用双向箭头表示那些可以在随机梯度和批量梯度方法之间的整个光谱中有效的方法。单向箭头表示那些仅在随机梯度估计中至少使用适度批量大小时有效的方法。

牛顿法

牛顿法：算法描述

牛顿法的基本思想是将目标函数在当前迭代点处进行二阶泰勒展开，然后最小化这个近似目标函数，即

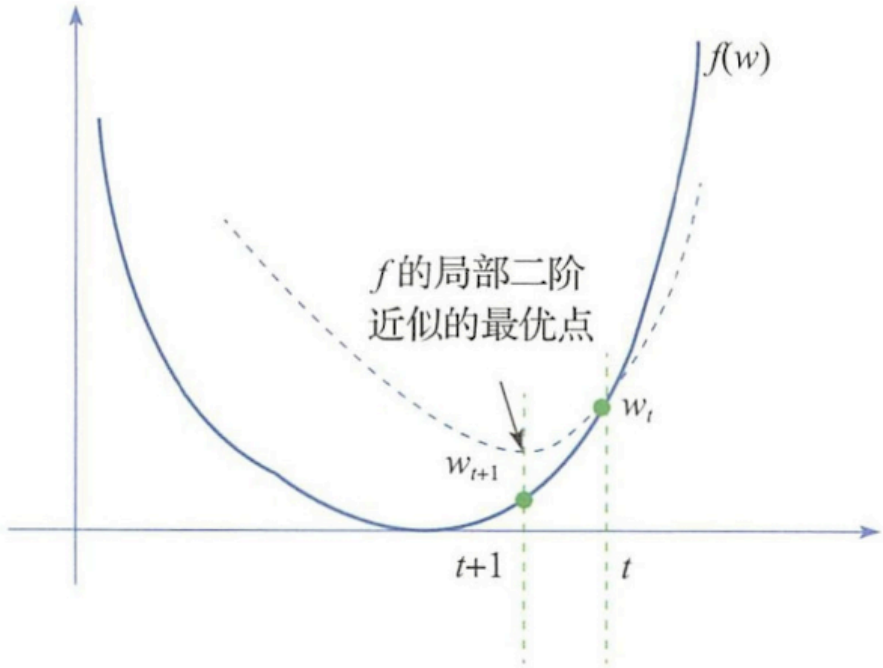
$$\min_{w\in\mathcal{W}} f(w) \approx \min_{w\in W} f\left(w^t\right)+\nabla f\left(w^t\right)^T\left(w-w^t\right)+\frac{1}{2}\left(w-w^t\right)^T\nabla^2 f\left(w^t\right)\left(w-w^t\right)$$

此处 $\nabla^2 f\left(w^t\right)$ 是目标函数在当前迭代点 w^t 处的海森矩阵。如果该海森矩阵是正定的，则上述问题的最优值 $w^t-\left[\nabla^2 f\left(w^t\right)\right]^{-1}\nabla f\left(w^t\right)$ 处取到，牛顿法将其做为下一时刻的状态，即

$$w^{t+1}=w^t-\left[\nabla^2 f\left(w^t\right)\right]^{-1}\nabla f\left(w^t\right)$$

牛顿法

下图给了一个简单实例：



牛顿法

牛顿法的伪代码如下所示：

Algorithm 1: 牛顿法

Input: 函数 $f(\cdot)$，初始参数 w^0，迭代轮数 T

Output: 终止参数 w^T

for $t = 0, 1, \cdots, T-1$ **do**
  计算梯度 $\nabla f(w^t)$;
  计算海森矩阵 $\nabla^2 f(w^t)$，并求逆矩阵 $[\nabla^2 f(w^t)]^{-1}$;
  更新参数 $w^{t+1} = w^t - [\nabla^2 f(w^t)]^{-1} \nabla f(w^t)$;
end
return w^T

收敛性和计算复杂度分析

相比于一阶方法中的梯度下降法，二阶的牛顿法提供了更为精细的步长调节，即利用当前状态海森矩阵的逆矩阵。因为步长更为精细，牛顿法的收敛速率比梯度下降法的收敛速率显著加快，具有二次收敛速率。

牛顿法收敛性

假设目标函数 $f\colon \mathbb{R}^d \to \mathbb{R}$ 的导数 $\nabla f(w)$ 是光滑的，存在二阶导数，并且在其最优点 x^* 处满足 $\nabla f(x^*) = 0, \nabla^2 f(x^*) \succ 0$ 且初始点 x^0 与最优点 x^* 足够近，那么牛顿法具有 $\mathcal{O}\left(e^{-2^t}\right)$ 的超线性（二阶）收敛速率：

$$\left\|w^t - w^*\right\| \leqslant \mathcal{O}\left(e^{-2^t}\right)$$

收敛性和计算复杂度分析

然而，事物总有两面性，相比一阶方法，虽然牛顿法的收敛速度更快，但是存在下面两个问题：

牛顿法的问题

- 在每个时刻需要计算当前状态的海森逆矩阵，计算量和存储量都显著增大；
- 海森矩阵不一定是正定的。

为了解决这个问题，人们提出了拟牛顿法。

拟牛顿法

既然海森矩阵不一定正定，那就构造一个与海森矩阵相差不太远的正定矩阵作为其替代品，这正是拟牛顿法的主要思想。此外，逆牛顿法可以迭代更新海森逆矩阵，而不是每次迭代都需要重新进行逆矩阵的计算。

拟牛顿法：算法描述

记 $H_t = \nabla^2 f\left(w^t\right)$，| $M^t = \left[\nabla^2 f\left(w^t\right)\right]^{-1}$ 。在第 $t+1$ 轮迭代，对第 t 轮迭代的海森矩阵 H_t 加上一个或者两个秩为 1 的矩阵作为对 $t+1$ 时刻的海森矩阵 H^{t+1} 的估计。例如：

$$H^{t+1} = H^t + aa^T + bb^T$$

当然，海森矩阵的更新需要满足一定的条件，比如下面推导的拟牛顿条件。

拟牛顿法

拟牛顿法：拟牛顿条件

对 $f(w)$ 在 w^t 处的二阶泰勒展开的左右两边计算梯度，可得

$$\nabla f(w) \approx \nabla f\left(w^t\right) + H^t\left(w - w^t\right)$$

令 $\delta_1^t = w^{t+1} - w^t$ 为模型的更新量，$\delta_2^t = \nabla f\left(w^{t+1}\right) - \nabla f\left(w^t\right)$ 为目标函数导数的更新量，则我们有：

$$\left(H^t\right)^{-1} \delta_2^t \approx \delta_1^t$$

拟牛顿法

拟牛顿法：算法描述

在拟牛顿条件下，更新规则为

$$H^0 = I$$

$$H^{t+1} = H^t + \frac{\delta_2^t \left(\delta_2^t\right)^T}{\left(\delta_2^t\right)^T \delta_1^t} - \frac{H^t \delta_1^t \left(H^t \delta_1^t\right)^T}{\left(\delta_1^t\right)^T H^t \delta_1^t}$$

$$M^{t+1} = \left(I - \frac{\delta_1^t \left(\delta_2^t\right)^T}{\left(\delta_2^t\right)^T \delta_1^t}\right) M^t \left(I - \frac{\delta_2^t \left(\delta_1^t\right)^T}{\left(\delta_2^t\right)^T \delta_1^t}\right) + \frac{\delta_1^t \left(\delta_1\right)^T}{\left(\delta_2^t\right)^T \delta_1^t}$$

此时，所对应的算法被称作 BFGS 算法。
可以证明，当初始点最优点足够近时，拟牛顿法和牛顿法同样具有二阶收敛速率。

拟牛顿法

此时，所对应的算法被称作 BFGS 算法，如下面伪代码所示：

Algorithm 2: BFGS 算法

Input: 函数 $f(\cdot)$，初始参数 w^0，迭代轮数 T，迭代步长 η

Output: 终止参数 w^T

$H^0 = I$;

计算 $\nabla f(w^0)$;

计算 $M^0 = [\nabla^2 f(w^0)]^{-1}$;

for $t = 0, 1, \cdots, T-1$ **do**

计算 $w^{t+1} = w^t - \eta(H^t)^{-1}\nabla f(w^t)$ 或 $w^{t+1} = w^t - \eta M^t \nabla f(w^t)$;

计算模型和导数的更新量：$\delta_1^t = w^{t+1} - w^t$，$\delta_2^t = \nabla f(w^{t+1}) - \nabla f(w^t)$;

利用更新量 δ_1^t 和 δ_2^t，迭代更新海森矩阵和海森逆矩阵：

$$H^{t+1} = H^t + \frac{\delta_2^t(\delta_2^t)^T}{(\delta_2^t)^T\delta_1^t} - \frac{H^t\delta_1^t(H^t\delta_1^t)^T}{(\delta_1^t)^T H^t \delta_1^t}$$

$$M^{t+1} = \left(I - \frac{\delta_1^t(\delta_2^t)^T}{(\delta_2^t)^T\delta_1^t}\right) M^t \left(I - \frac{\delta_2^t(\delta_1^t)^T}{(\delta_2^t)^T\delta_1^t}\right) + \frac{\delta_1^t(\delta_1)^T}{(\delta_2^t)^T\delta_1^t}$$

end

return w^T

随机拟牛顿法

随机拟牛顿法：算法描述

随机拟牛顿法的思想与一阶随机算法类似，随机采一个样本或者一个小批量样本来计算梯度，然后更新海森逆矩阵。小批量随机拟牛顿法的更新公式如下：

$$w^{t+1} = w^t - \eta^t M^t \left(\frac{1}{|\mathcal{I}^t|} \sum_{i \in \mathcal{I}^t} \nabla f_i \left(w^t \right) \right)$$

其中 $\mathcal{I}^t$ 是所采的小批量数据子集，M^t 为 $\mathcal{I}$ 上目标函数的海森逆矩阵。

类似于之前介绍的拟牛顿法，虽然直接计算海森逆矩阵 M^t 的复杂度很高，但是利用历史信息迭代更新上一轮海森逆矩阵 M^{t-1} 可以得到对 M^t 的良好逼近，其计算复杂度要低很多。

随机拟牛顿法

具体而言，首先在算法运行过程中记录下表征模型变化量和梯度变化量的修正组合 (δ_1^s, δ_2^s)，也就是

$$\delta_1^s = w^s - w^{s-1}$$

$$\delta_2^s = \frac{1}{|\mathcal{I}^t|} \sum_{i \in \mathcal{I}^t} \nabla^2 f_i\left(w^s\right)\left(w^s - w^{s-1}\right)$$

然后依据第 t 轮迭代之前的多个修正组合按照如下公式迭代更新海森逆矩阵：

$$M \leftarrow \left(I - \rho^s \delta_1^s \left(\delta_2^s\right)^T\right) M \left(I - \rho^s \delta_2^s \left(\delta_1^s\right)^T\right) + \rho^s \delta_1^s \left(\delta_1^s\right)^T$$

其中 $\rho^s = \frac{1}{\left(\delta_2^s\right)^T \delta_1^s}$。

随机拟牛顿法及海森逆矩阵更新算法

Algorithm 2: 随机拟牛顿法

Input: 函数 $f(\cdot)$，初始参数 w^0，迭代轮数 T，正常数 M、L

Output: 终止参数 w^T

设置 $s=-1$，$\overline{w}^s=0$;
for $t=0,1,\cdots,T-1$ **do**
  随机采一个样本子集 $\mathcal{I}_t\subset\{1,\cdots,n\}$;
  计算梯度 $\nabla f_{\mathcal{I}^t}(w^t)$;
  $\overline{w}^s=\overline{w}^s+w^t$;
  if $s\leqslant 2L$ **then**
    更新参数 $w^{t+1}=w^t-\eta^t\nabla f_{\mathcal{I}^t}(w^t)$; // 用随机梯度下降法做冷启动
  else
    更新参数 $w^{t+1}=w^t-\eta^t M^s\nabla f_{\mathcal{I}^t}(w^t)$;
  end
  if $\mod(s,L)=0$ **then**
    $s=s+1$;
    $\overline{w}^s=\overline{w}^s/L$;
    if $t>0$ **then**
      选择一个子集 $\mathcal{I}_H\subset\{1,\ldots,n\}$ 去计算 $\nabla^2 f_{\mathcal{I}_H}(\overline{w}^s)$;
      计算 $\delta_1^s=(\overline{w}^s-\overline{w}^{s-1}),\delta_2^s=\nabla^2 f_{\mathcal{I}_H}(\overline{w}^s-\overline{w}^{s-1})$;
    end
    $\overline{w}^s=0$;
  end
end
return w^T

随机拟牛顿法及海森逆矩阵更新算法

Algorithm 3: 海森逆矩阵 M^t 的更新算法

Input: s，t，内存参数 S，修正组合序列

Output: H^s

令 $H = \frac{(\delta_1^t)^T \delta_2^t}{(\delta_2^t)^T \delta_2^t} I$;

for $j = s - \min\{t, S\} + 1, \cdots, s$ **do**

  $\rho^j = \frac{1}{(\delta_2^j)^T \delta_1^j}$;

  $M \leftarrow (I - \rho^j \delta_1^j (\delta_2^j)^T) M (I - \rho^j \delta_2^j (\delta_1^j)^T) + \rho^j \delta_1^j (\delta_1^j)^T$

end

return $M^s \leftarrow M$

收敛性和计算复杂度分析

假设条件

在以上四条假设之下，我们可以证明随机拟牛顿法具有与随机梯度下降法相同的收敛速率：

- 目标函数 $f(w)$ 是二阶连续可微的
- 存在 $\lambda_2 > \lambda_1 > 0$，使得对于任意 $w \in \mathbb{R}^d$，$\lambda_1 I \prec \nabla^2 f(w) \prec \lambda_2 I$
- 存在 $\mu_2 > \mu_1 > 0$，使得对于任意 $w \in \mathbb{R}^d$，$\mu_1 I \prec \left(\nabla^2 f(w)\right)^{-1} \prec \mu_2 I$
- 存在 $G > 0$，使得对于任意 $w \in \mathbb{R}^d$，$\mathbb{E}\left[\|\nabla f_i(w)\|^2\right] \leqslant G^2$

假设上述条件成立，当步长 $\eta^t = \frac{a}{t}$ 并且 $a > \frac{1}{2\mu_1\lambda_1}$ 时，对于给定的迭代步数 T，随机拟牛顿法具有 $\mathcal{O}\left(\frac{1}{T}\right)$ 的次线性收敛速率：

$$\mathbb{E}\left[f\left(w^T\right) - f(w^*)\right] \leqslant \frac{Q(a)}{T}$$

其中 $Q(a) = \max\left\{\frac{\lambda_2 \mu_2^2 a^2 G^2}{2(2\mu_1\lambda_1 a - 1)}, f\left(w^1\right) - f(w^*)\right\}$。

谢谢！

敬请各位老师批评指正！