DCS5718 最优化理论与方法

第五章：随机梯度法及其变形

杨磊 (yanglei39@mail.sysu.edu.cn)

计算机学院，2025 春

- 问题背景
- 随机梯度法
- 随机梯度法的变形
- 应用举例
- 收敛性分析
- 方差减小技术

有限和形式优化问题

随机优化问题通常表示为如下形式：

$$\min_{x\in\mathbb{R}^n} \quad \mathbb{E}_\xi[F(x,\xi)]$$

- ξ 是一个服从概率分布 $\mathcal{P}$ 的随机变量（但分布 $\mathcal{P}$ 一般是未知的）；
- 优化目标是关于 ξ 的数学期望，由于分布 $\mathcal{P}$ 未知，其通常不可被计算。

思考：在实际中，分布 $\mathcal{P}$ 往往是未知的，因此数学期望 $\mathbb{E}_\xi[F(x,\xi)]$ 通常不可以计算，那么我们该如何解决？

回答：近似方法！我们可以利用 ξ 的经验分布来代替其真实分布 $\mathcal{P}$！假设

$$\xi_1,\ \xi_2,\ \cdots,\ \xi_N,$$

是从 $\mathcal{P}$ 中随机采样的 N 个样本，令 $f_i(x)=F\left(x,\xi_i\right)$，于是，可以得到如下有限和形式的优化问题：

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

在统计学和机器学习等领域中，大量实际应用问题可以被表示为该形式！

有限和形式优化问题

为了更好地理解问题(★)的形式，我们考虑机器学习中的一个基本模型 — 监督学习模型。

- 假定 (a, b) 服从概率分布 $\mathcal{P}$，其中 a 为输入，b 为标签；
- 我们的任务是要给定输入 a 预测标签 b，即要决定一个最优的预测函数 ϕ 使得期望风险 $\mathbb{E}\left[L\left(\phi(a), b\right)\right]$ 最小，其中 $L(\cdot, \cdot)$ 表示损失函数，用来衡量预测的准确度，函数 ϕ 为某个函数空间中的预测函数；
- 在实际问题中我们并不知道真实的概率分布 $\mathcal{P}$，而是随机采样得到的一个数据集 $D = \{(a_1, b_1), (a_2, b_2), \cdots, (a_N, b_N)\}$；

然后我们用经验风险来近似期望风险，并将预测函数 $\phi(\cdot)$ 参数化为 $\phi(\cdot\,; x)$ 以缩小要找的预测函数的范围，即要求解下面的极小化问题：

$$\min_x \quad \frac{1}{N} \sum_{i=1}^{N} L\left(\phi(a_i; x), b_i\right).$$

显然，该问题对应于问题(★)，有 $\xi_i = (a_i, b_i)$ 及 $f_i(x) = L\left(\phi(a_i; x), b_i\right)$。

梯度下降法的局限

假设问题 (★) 中的每一个函数 f_i 都是凸且连续可微的，那么我们自然想到利用梯度下降法来求解该问题：

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

其中，梯度的计算如下

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

可以看到：

- 为了计算 $\nabla f(x^k)$，我们必须计算所有的 $\nabla f_i(x^k)$ 再取平均，因此需要遍历所有的样本点；
- 如果样本量 N 巨大且样本自身的维度也很高，那么在实际计算 $\nabla f(x^k)$ 时，将付出高昂的计算成本和数据读取成本！

思考：面对如此大规模优化问题，如何改进梯度下降法，以节约计算成本？

回答：“近似计算梯度”！随机抽取一个或部分样本，近似计算 $\nabla f(x^k)$！

- 问题背景
- 随机梯度法
- 随机梯度法的变形
- 应用举例
- 收敛性分析
- 方差减小技术

随机梯度法

基于“近似计算梯度”的思想，我们可以得到

随机梯度法 (Stochastic Gradient Descent[1], SGD), 其迭代格式如下：

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

其中在第 k 轮迭代，从 $\{1, \cdots, N\}$ 中随机等可能地抽取一个样本 s_k，然后仅仅计算该样本处的梯度 $\nabla f_{s_k}(x^k)$ 作为 $\nabla f(x^k)$ 的近似。

回顾**梯度下降法**：

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

思考：随机性从何而来？来自随机抽取！同时，可以观察到，随机梯度 $\nabla f_{s_k}(x^k)$ 的条件期望恰好是全梯度，即

$$\mathbb{E}_{s_k} \left[\nabla f_{s_k}(x^k) \mid x^k \right] = \nabla f(x^k).$$

[1] 事实上，随机梯度法不能保证每轮迭代的单调下降性，因此称为随机梯度法更合理

随机梯度法

实际计算时，样本的选取方式可以灵活多样，可完全随机抽取，也可以每一轮“重洗牌”。

同时，每次只抽取一个样本 s_k 的做法显然比较极端，通常我们可以选取少量样本以近似计算全梯度，进而得到**小批量（mini-batch）随机梯度法**，即在每次迭代中，随机选择一个元素个数很少的集合 $\mathcal{I}_k \subset \{1, 2, \cdots, N\}$，然后执行迭代格式

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

其中 $|\mathcal{I}_k|$ 表示 $\mathcal{I}_k$ 中元素的个数。

当 $f_i(x)$ 是凸函数但不一定连续可微时，可以用 $f_i(x)$ 的次梯度代替梯度进行迭代，这就得到了**随机次梯度法**，其迭代格式：

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

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。

- 问题背景
- 随机梯度法
- 随机梯度法的变形
- 应用举例
- 收敛性分析
- 方差减小技术

动量方法

随机梯度算法在深度学习中得到了广泛的应用，本小节将简要介绍其在深度学习中的一些变形。传统的梯度算法在问题比较病态时收敛速度非常慢，而随机梯度法也会遇到类似问题。为了克服这一缺陷，人们提出了**动量法**：

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

- **设计思想**：在迭代时，一定程度上保留先前的更新方向 v^k，同时利用当前计算的梯度 $\nabla f_{s_k}(x^k)$ 调整最终的更新方向。也可以理解为，在计算当前的随机梯度 $\nabla f_{s_k}(x^k)$ 后，我们并不完全相信这个全新的随机梯度方向，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}。
- **好处**：增加迭代的稳定性；使算法具有一定摆脱局部最优解的能力；当许多连续的梯度指向相同的方向时，可以认为步长会变大。
- 参数 $\mu_k \in [0, 1)$，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，动量方法退化成随机梯度法。

Nesterov 加速算法

Nesterov 加速算法也有其随机版本。

- 回顾针对光滑凸优化问题的 Nesterov 加速算法：

$$y^{k+1} = x^k + \mu_k \left(x^k - x^{k-1}\right),$$
$$x^{k+1} = y^{k+1} - \alpha_k \nabla f(y^{k+1}),$$

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

- 其随机版本就是将全梯度替换为随机梯度，得到

$$y^{k+1} = x^k + \mu_k \left(x^k - x^{k-1}\right),$$
$$x^{k+1} = y^{k+1} - \alpha_k \nabla f_{s_k}(y^{k+1}).$$

Nesterov 加速算法

Nesterov 加速算法可看成是动量方法的变体。

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

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

- 然后，定义有关 v^{k+1} 的迭代式为：

$$v^{k+1} = \mu_k v^k - \alpha_k \nabla f_{s_k}\left(x^k + \mu_k v^k\right),$$

- 于是，得到关于 x^k 和 v^k 的等价迭代：

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

- 与动量方法相比，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变形，二者的主要差别在梯度的计算上。

AdaGrad 算法

在一般的随机梯度法中，调参是一个很大的难点，参数（如步长）设置的好坏将显著影响算法的实际性能，但是参数调优往往耗时耗力，所以希望算法能在运行的过程中自适应地调整参数。

观察到，对于一个无约束光滑凸优化问题 $\min_x \{f(x)\}$，x^* 是该问题的一个最优解等价于 $\nabla f(x^*) = 0$。然而，实际应用梯度类算法时，梯度的每个分量收敛到零的速度往往是不同的。

- 单纯随机梯度算法只有一个统一的步长 α_k，它没有针对每个分量考虑；
- 为此，人们提出 **AdaGrad** (**Ada**ptive Sub**grad**ient Methods)，它动态地调节梯度每个分量的步长，以期实现更快的收敛速率。设计思想：
  - 梯度的某个分量较大 $\Rightarrow$ 该方向上函数变化比较剧烈 $\Longrightarrow$ 用小步长
  - 梯度的某个分量较小 $\Rightarrow$ 该方向上函数变化比较平缓 $\Longrightarrow$ 用大步长

AdaGrad 算法

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

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

其中 $\odot$ 表示向量逐元素相乘。由 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。

由此启发，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均为对向量每个分量分别操作，而参数 ε 的引入是为了防止除零运算。注意到，AdaGrad 的步长大致反比于历史梯度累计值的算术平方根，所以梯度较大时步长下降很快，反之则下降较慢。

RMSProp 算法

AdaGrad 由于累加之前所有的梯度分量的平方，导致步长单调递减，容易过早或过多地减小。

为此，人们提出了改进方法 **RMSProp** (Root Mean Square **Prop**agation)。

- 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

$$M^{k+1} = \rho M^k + (1-\rho)\, g^{k+1} \odot g^{k+1}, \quad R^k = \sqrt{M^k + \varepsilon \mathbf{1}_n}.$$

- 将均方根的倒数作为每个分量步长的修正，得到 RMSProp 迭代格式：

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

  参数一般取 $\rho = 0.9, \alpha = 0.001$。

- 可以看到，RMSProp 和 AdaGrad 的唯一区别是将 G^k 替换成了 M^k。

AdaDelta 算法

AdaDelta (**Ada**ptive **Delta** Rule) 是 AdaGrad 算法的另一种改进。

- AdaDelta 在 RMSProp 的基础上，对历史的迭代点变化 $\Delta x^k = x^{k+1} - x^k$ 也同样累积平方，并求均方根：

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

- 然后使用 T^{k-1} 和 R^k 的商对步长进行校正，得到 AdaDelta 的迭代格式：

$$M^k = \rho M^{k-1} + (1-\rho)\, g^k \odot g^k, \quad R^k = \sqrt{M^k + \varepsilon \mathbf{1}_n},$$
$$\Delta x^k = -\frac{T^{k-1}}{R^k} \odot g^k,$$
$$D^k = \rho D^{k-1} + (1-\rho)\,\Delta x^k \odot \Delta x^k, \quad T^k = \sqrt{D^k + \varepsilon \mathbf{1}_n}$$
$$x^{k+1} = x^k + \Delta x^k,$$

Adam 算法

Adam (**Ada**ptive **M**oment Estimation) 可视为 “动量方法” + “RMSProp”

- Adam 不直接使用随机梯度 $g^k = \nabla f_{s_k}(x^k)$ 作为基础更新方向，而是选择一个动量项进行更新：

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

- 类似于 RMSProp，Adam 也会记录迭代过程中梯度的累积平方：

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

- 与原始动量方法和 RMSProp 的区别在于：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：

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

这里 ρ_1^k, ρ_2^k 分别表示 ρ_1, ρ_2 的 k 次方。

- Adam 使用修正后的 $\hat{S}^k$（更新方向）和 $\hat{M}^k$（梯度的分量累积）进行迭代点的更新：

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

- 问题背景
- 随机梯度法
- 随机梯度法的变形
- 应用举例
- 收敛性分析
- 方差减小技术

线性最小二乘回归

- 线性回归 (Linear Regression) 是机器学习和统计学中最基础的回归模型之一，旨在对自变量和因变量之间的线性关系进行建模。
- 假设某机构经过分析，认为影响本市房价 b 的因素有 m 个 (例如地段、房型大小、卧室数量等)。现从市场中随机选取了 N 处房源，考察并收集了它们的价格 $\{b_i\}_{i=1}^N$ 和相关因素 $\{a_i\}_{i=1}^N$，其中 $b_i \in \mathbb{R}$，$a_i \in \mathbb{R}^m$。
- 假设 b 和 a 之间的关系可以用线性函数 $b = \sum_{i=1}^m x_i a_i = a^\top x$ 来拟合，其中 $x \in \mathbb{R}^m$ 是线性函数的参数。

线性最小二乘回归

- 上述参数 x 可根据最小二乘规则来估计，即：

$$\min_{x\in\mathbb{R}^m} \quad \frac{1}{N}\sum_{i=1}^{N}\left(a_i^\top x - b_i\right)^2,$$

其中二次函数 $f_i(x)=\left(a_i^\top x - b_i\right)^2$ 用于衡量线性函数对样本 i 的拟合值 $a_i^\top x$ 与真值 b_i 之间的差异。

- 显然，线性最小二乘回归问题具有形如 (★) 的有限和形式：

$$\min_{x\in\mathbb{R}^m} \quad f(x) := \frac{1}{N}\sum_{i=1}^{N} f_i(x) = \frac{1}{N}\sum_{i=1}^{N}\left(a_i^\top x - b_i\right)^2.$$

现考虑应用随机梯度法求解该优化问题，其迭代格式为：

$$x^{k+1} = x^k - \alpha_k \nabla f_{s_k}(x^k) = x^k - 2\alpha_k\left(a_{s_k}^\top x - b_{s_k}\right),$$

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

逻辑回归

- 逻辑回归是最基本的线性分类模型。
- 给定数据集 $\{(a_i, b_i)\}_{i=1}^N$，带 ℓ_2 范数平方正则项的逻辑回归可以写成具有形如 (★) 的有限和形式：

$$\min_{x\in\mathbb{R}^n} \ f(x) = \frac{1}{N}\sum_{i=1}^N f_i(x) = \frac{1}{N}\sum_{i=1}^N \ln\left(1+\exp\left(-b_i\cdot a_i^\top x\right)\right) + \lambda\|x\|_2^2,$$

其中 $f_i(x) = \ln\left(1+\exp\left(-b_i\cdot a_i^\top x\right)\right) + \lambda\|x\|_2^2$.

- 应用随机梯度法求解该问题，其迭代格式为：

$$\begin{aligned} x^{k+1} &= x^k - \alpha_k \nabla f_{s_k}\left(x^k\right) \\ &= x^k - \alpha_k\left(\frac{-\exp\left(-b_{s_k}a_{s_k}^\top x^k\right) b_{s_k}a_{s_k}}{1+\exp\left(-b_{s_k}a_{s_k}^\top x^k\right)} + 2\lambda x^k\right), \end{aligned}$$

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

- 问题背景
- 随机梯度法
- 随机梯度法的变形
- 应用举例
- 收敛性分析
- 方差减小技术

一般凸函数下SGD的收敛性

随机梯度算法具有不确定性，这样的算法会有收敛性吗？本小节将简单讨论这一问题。我们将对目标函数分别为一般凸函数和连续可微强凸函数的情况进行讨论。

对于一般凸函数，随机梯度法此时实际上是随机次梯度法。首先，我们列出这一部分的基本假设：

假设 1：

1. 每个 $f_i(x)$ 是闭凸函数，存在次梯度；
2. 随机次梯度**二阶矩是一致有界的**，即存在 M，对任意的 $x \in \mathbb{R}^n$ 以及随机下标 s_k，有

$$\mathbb{E}_{s_k}\left[\left\|g^k\right\|^2\right] \leqslant M^2 < +\infty, \quad g^k \in \partial f_{s_k}(x^k);$$

3. 迭代的随机点列 $\{x^k\}$ 处处有界，即存在 $R > 0$，使得 $\left\|x^k - x^*\right\| \leq R,\ \forall k$，其中 x^* 是问题(★)的最优解。

一般凸函数下SGD的收敛性

定理 1 (随机次梯度法的收敛性 1)

在**假设 1**的条件下，令 $A_K = \sum_{i=1}^{K} \alpha_i$，定义 $\bar{x}_K = \frac{1}{A_K} \sum_{k=1}^{K} \alpha_k x^k$，则

$$\mathbb{E}\left[f\left(\bar{x}_K\right) - f\left(x^*\right)\right] \leqslant \frac{R^2 + \sum_{k=1}^{K} \alpha_k^2 M^2}{2 \sum_{k=1}^{K} \alpha_k}. \tag{T1}$$

- 从上述定理可以看到，当步长 α_k 满足以下条件时

$$\sum_{k=1}^{\infty} \alpha_k = +\infty, \qquad \frac{\sum_{k=1}^{K} \alpha_k^2}{\sum_{k=1}^{K} \alpha_k} \to 0,$$

随机次梯度法在步长加权平均意义下可以收敛。

一般凸函数下SGD的收敛性

- 对一个固定的步长 α，(T1)右侧有一个不随 K 递减的常数，因此固定步长随机次梯度法在函数值取期望意义下是不收敛的，它仅仅能找到一个次优解：

$$\mathbb{E}\left[f\left(\bar{x}_K\right)-f\left(x^*\right)\right] \leqslant \frac{R^2}{2K\alpha}+\frac{\alpha M^2}{2}.$$

- 对于给定的迭代次数 K，选取固定步长 $\alpha=\frac{R}{M\sqrt{K}}$，可以达到 $\mathcal{O}\left(\frac{1}{\sqrt{K}}\right)$ 的精度，即

$$\mathbb{E}\left[f\left(\bar{x}_K\right)-f\left(x^*\right)\right] \leqslant \frac{RM}{\sqrt{K}}.$$

一般凸函数下SGD的收敛性

定理 1 的收敛性是在步长加权平均意义下的收敛性，在步长单调不增的情况下，我们可以得到直接平均意义下的收敛性。

定理 2 (随机次梯度法的收敛性 2)

在**假设 1** 下，令 $\{\alpha_k\}$ 是一个单调不增的正步长序列，$\bar{x}_K = \frac{1}{K}\sum_{k=1}^{K} x^k$，则

$$\mathbb{E}\left[f(\bar{x}_K) - f(x^*)\right] \leq \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K}\alpha_k M^2.$$

- 选取 $\mathcal{O}\left(\frac{1}{\sqrt{k}}\right)$ 阶数的步长，可以得到目标函数的收敛速率为 $\mathcal{O}\left(\frac{1}{\sqrt{K}}\right)$。
- 特别地，选取步长 $\alpha_k = \frac{R}{M\sqrt{k}}$，则

$$\mathbb{E}\left[f(\bar{x}_K) - f(x^*)\right] \leqslant \frac{3RM}{2\sqrt{K}}.$$

可微强凸函数下 SGD 的收敛性

思考：如果 $f(x)$ 是可微强凸函数，随机梯度下降法的收敛速度会有改善吗？

回答：会有改善！

首先我们列出这一部分的统一假设：

假设 2

1. $f(x)$ 是可微函数，每个 $f_i(x)$ 存在梯度；
2. $f(x)$ 是梯度利普希茨连续的，相应常数为 L；
3. $f(x)$ 是强凸函数，强凸系数为 μ；
4. 随机梯度二阶矩是一致有界的，即存在 M，对任意的 $x \in \mathbb{R}^n$ 以及随机下标 s^k，有

$$\mathbb{E}_{s_k}\left[\left\|\nabla f_{s_k}(x)\right\|^2\right] \leq M^2 < +\infty.$$

可微强凸函数下 SGD 的收敛性

下面的定理将给出随机梯度法在固定步长下的收敛性分析。

定理 3 (随机梯度法的收敛性 4)

在**假设 2** 的条件下，定义 $\Delta_k = \|x^k - x^*\|$。对固定的步长 $\alpha_k \equiv \alpha$ 满足 $0 < \alpha < \frac{1}{2\mu}$，有

$$\mathbb{E}\left[f\left(x^{K+1}\right) - f\left(x^*\right)\right] \leq \frac{L}{2}\mathbb{E}\left[\Delta_{K+1}^2\right] \leq \frac{L}{2}\left[(1-2\alpha\mu)^K \Delta_1^2 + \frac{\alpha M^2}{2\mu}\right].$$

注意：对于固定步长，上式右端有不随 K 变化的常数，算法不能保证收敛。

可微强凸函数下 SGD 的收敛性

下面的定理将给出随机梯度法在递减步长下的收敛性分析。

定理 4 (随机梯度法的收敛性 5)

在定理 3 的结果中，取递减的步长

$$\alpha_k = \frac{\beta}{k+\gamma},$$

其中 $\beta > \frac{1}{2\mu}, \gamma > 0$，使得 $\alpha_1 \leq \frac{1}{2\mu}$，那么对于任意的 $k \geq 1$，都有

$$\mathbb{E}\left[f\left(x^k\right) - f\left(x^*\right)\right] \leqslant \frac{L}{2}\mathbb{E}\left[\Delta_k^2\right] \leqslant \frac{L}{2}\frac{v}{\gamma+k},$$

这里 $v = \max\left\{\frac{\beta^2 M^2}{2\beta\mu - 1}, (\gamma+1)\Delta_1^2\right\}$。

可以看到，取特定的递减步长 $\alpha_k = \frac{\beta}{k+\gamma}$，收敛速率可达 $\mathcal{O}\left(\frac{1}{k}\right)$。

SGD 的算法复杂度

次梯度算法和梯度下降法的普通版本和随机版本的算法复杂度如下表所示：

复杂度：计算梯度的次数　　ε：希望达到的精度　　N：样本数量

	f 凸 (次梯度算法)	f 可微强凸	f 可微强凸且 L-光滑
随机算法	$\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$	$\mathcal{O}\left(\frac{1}{\varepsilon}\right)$	$\mathcal{O}\left(\frac{1}{\varepsilon}\right)$
普通算法	$\mathcal{O}\left(\frac{N}{\varepsilon^2}\right)$	$\mathcal{O}\left(\frac{N}{\varepsilon}\right)$	$\mathcal{O}\left(N\ln\left(\frac{1}{\varepsilon}\right)\right)$

可以看到：

- 次梯度算法的普通版本和随机版本的收敛速度并没有差别，而每步的计算量能大大降低。
- 但是在 f 可微强凸且 L-光滑情形下，随机梯度法的收敛速度要慢于普通梯度算法。

思考： 这一差别的原因是什么？

- 问题背景
- 随机梯度法
- 随机梯度法的变形
- 应用举例
- 收敛性分析
- 方差减小技术

梯度下降法与随机梯度法的区别

- 上一小节说明了在次梯度情形以及 $f(x)$ 可微强凸的条件下, 随机梯度方法的算法复杂度要小于普通的梯度法。
- 但是，我们也发现随机方法容易受到梯度估计噪声的影响，这使得它在固定步长下不能收敛，并且在递减步长下也只能达到次线性收敛速率 $\mathcal{O}\left(\frac{1}{k}\right)$，而普通梯度法在强凸且梯度 L- 利普希茨连续的条件下可以达到 Q-线性收敛速率。
- 不难想象，主要原因在于随机梯度法中的梯度计算始终是有偏差（方差）的，即 $\left\|\nabla f_{s_k}(x^k) - \nabla f(x^k)\right\|^2$，因此无法充分“利用”梯度的良好性质。
- 因此，为了能获得比较快的渐进收敛速度，需要想办法减少梯度偏差（方差）！本节将简单介绍三种减小方差的算法：
  - SAG (**S**tochastic **A**verage **G**radient)
  - SAGA
  - SVRG (**S**tochastic **V**ariance **R**educed **G**radient)

  基本思想：利用历史梯度信息来减小方差，最终获得 Q-线性收敛速率。

SAG 算法

SAG 算法在迭代过程中将记录所有之前计算过的随机梯度，再与当前新计算的随机梯度求平均，最终作为下一步的梯度估计：

- SAG 算法在内存中开辟了存储 N 个随机梯度的空间

  $$\boxed{g_1^k} \quad \boxed{g_2^k} \quad \boxed{g_3^k} \quad \cdots\cdots\cdots \quad \boxed{g_N^k}$$

  分别用于记录和第 i 个样本相关的最新的随机梯度；

- 在第 k 步，记抽取的样本点下标为 s_k，计算随机梯度 $\nabla f_{s_k}(x^k)$，并将其赋值给 $g_{s_k}^k$，其它未抽取到的样本所对应的 g_i^k 保持不变；

- SAG 算法每次更新使用的随机梯度是所有 g_i^k 的平均值 $\frac{1}{N}\sum_{i=1}^N g_i^k$；

- 列表 $\left\{g_i^k\right\}_{i=1}^N$ 的初值可简单地取为零向量或中心化的随机梯度向量。

SAG 算法

- SAG 算法的迭代格式为：

$$x^{k+1} = x^k - \alpha_k \frac{1}{N} \sum_{i=1}^{N} g_i^k,$$

其中 g_i^k 的更新方式为：

$$g_i^k = \begin{cases} \nabla f_{s_k}(x^k), & i = s_k, \\ g_i^{k-1}, & \text{其他}, \end{cases}$$

- 由于每次迭代只有一个 g_i^k 发生了改变，故 SAG 迭代公式可写为

$$x^{k+1} = x^k - \alpha_k \left(\frac{1}{N} \left(\nabla f_{s_k}(x^k) - g_{s_k}^{k-1} \right) + \frac{1}{N} \sum_{i=1}^{N} g_i^{k-1} \right). \qquad \textbf{(SAG)}$$

SAG 算法的收敛性

定理 5 (SAG 算法的收敛性)

在**假设 2** 的条件下，取固定步长 $\alpha_k \equiv \frac{1}{16L}$，$g_i^k$ 的初值取为零向量，则对任意的 k，我们有

$$\mathbb{E}\left[f(x^k) - f(x^*)\right] \leq \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8N}\right\}\right)^k C_0,$$

其中常数 C_0 为与 k 无关的常数。

上述定理表明 SAG 算法确实有 Q-线性收敛速率。但是 SAG 算法需要存储 N 个梯度向量，当样本量 N 很大时，这无疑是一个很大的开销。因此 SAG 算法在实际中很少使用，它的主要价值在于算法的思想。很多其他实用算法都是根据 SAG 算法变形而来的。

SAGA 算法

SAGA 算法是 SAG 算法的一个修正，它使用无偏的随机梯度作为更新方向：

$$x^{k+1} = x^k - \alpha_k \left(\nabla f_{s_k}\left(x^k\right) - g_{s_k}^{k-1} + \frac{1}{N} \sum_{i=1}^{N} g_i^{k-1} \right). \qquad \textbf{(SAGA)}$$

对比 (**SAG**) 的更新公式：

$$x^{k+1} = x^k - \alpha_k \left(\frac{1}{N} \left(\nabla f_{s_k}\left(x^k\right) - g_{s_k}^{k-1} \right) + \frac{1}{N} \sum_{i=1}^{N} g_i^{k-1} \right). \qquad \textbf{(SAG)}$$

可以发现：(**SAGA**) 去掉了 $\nabla f_{s_k}\left(x^k\right) - g_{s_k}^{k-1}$ 前面的系数 $\frac{1}{N}$，可以证明，其每次迭代使用的梯度方向都是无偏的。

SAGA 算法的收敛性

SAGA 算法同样有 Q-线性收敛速率。

定理 6 (SAGA 算法的收敛性)

在**假设 2** 的条件下，取固定步长 $\alpha_k = \frac{1}{2(\mu N+L)}$。定义 $\Delta_k = \left\|x^k - x^*\right\|$，则对任意的 $k \geqslant 1$ 有

$$\mathbb{E}\left[\Delta_k^2\right] \leqslant \left(1 - \frac{\mu}{2(\mu N + L)}\right)^k \left(\Delta_1^2 + \frac{N\left(f\left(x^1\right) - f\left(x^*\right)\right)}{\mu N + L}\right).$$

SVRG 算法

与 SAG 算法和 SAGA 算法不同，SVRG 算法通过周期性缓存全梯度的方法来减小方差。它的具体步骤如下：

- 每经过 m 次迭代，设置一个“检查点”，计算一次全梯度
- 在之后的 m 次迭代中，将这个全梯度作为参考点来实现方差减小的目的
- 具体来说，令 $\tilde{x}^j$ 是第 j 个检查点，我们需要计算点 $\tilde{x}^j$ 处的全梯度：

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

在之后的迭代中使用方向 v^k 作为更新方向：

$$v^k = \nabla f_{s_k}(x^k) + (\nabla f(\tilde{x}^j) - \nabla f_{s_k}(\tilde{x}^j)),$$

其中 $s_k \in \{1, 2, \cdots, N\}$ 是随机选取的一个样本。

更新方向 v^k 的直观理解：我们希望用 $\nabla f_{s_k}(\tilde{x}^j)$ 去估计 $\nabla f(\tilde{x}^j)$，那么 $\nabla f(\tilde{x}^j) - \nabla f_{s_k}(\tilde{x}^j)$ 可以看作梯度估计的误差，所以在每步随机梯度迭代中，用该误差项来对 $\nabla f_{s_k}(x^k)$ 做一个校正。

SVRG 算法的收敛性

下面分析 SVRG 算法的收敛性。这里的收敛性是针对参考点序列 $\{\tilde{x}^j\}$ 的。

定理 7 (SVRG 算法的收敛性)

设每个 $f_i(x)$ 是可微凸函数，且梯度为 L-利普希茨连续的，函数 $f(x)$ 是强凸的，强凸参数为 μ。在 SVRG 算法中，取步长 $\alpha \in \left(0, \frac{1}{2L}\right]$，且 m 充分大使得

$$\rho = \frac{1}{\mu\alpha(1-2L\alpha)m} + \frac{2L\alpha}{1-2L\alpha} < 1,$$

那么 SVRG 算法对于参考点 $\tilde{x}^j$ 在函数值期望的意义下有 Q-线性收敛速率：

$$\mathbb{E}\left[f(\tilde{x}^j) - f(x^*)\right] \leq \rho \mathbb{E}\left[f(\tilde{x}^{j-1}) - f(x^*)\right].$$