MATH 3011 运筹学 **秋季 2023**

Lecture 11: 无约束优化 随机梯度算法

Lecturer: 陈士祥 *Scribes:* 陈士祥

1 问题背景

我们以监督学习为例，作为随机梯度算法 (Stochastic gradient descent，简称 SGD) 的重要应用。

什么是监督学习？

监督学习是一种机器学习范式，其中模型在带标签的数据上进行训练。训练数据包含输入-输出对。给定这些对，算法学习输入和输出之间的映射，然后可以用于预测新的、以前未见过的输入的输出。

它是如何工作的？

设想一个老师和一个学生。老师有一本习题集。当学生尝试这些问题时，老师会指出正确的解答来纠正任何错误。随着时间的推移，学生变得擅长独立解决类似的问题。

在监督学习中，算法扮演学生的角色，数据充当问题集，标签是正确的解决方案。算法对训练数据进行预测，并在出错时根据真实标签调整其理解。

关键组件：

1. **训练数据**：由输入特征和正确的输出组成，这是监督学习的基础。

2. **模型**：这是用来根据输入特征预测输出的算法或算法集。

3. **损失函数**：监督学习的核心概念是损失函数。这是一个数学函数，用于测量我们模型的预测与真实值之间的差距。监督学习的目标是最小化这种损失。

损失函数的例子：最常用于回归任务（输出是连续值）的损失函数是**均方误差 (MSE)**。如果我们有 N 个数据点，对于每个点 a_i，模型 ϕ 预测的值是 $\hat{y_i}$，真实值（标签）是 y_i，那么 MSE 为：

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y_i} - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(\phi(a_i) - y_i)^2$$

简单来说，这计算了预测值和真实值之间的平均平方差。我们记模型 ϕ 的参数为 x，则优化的目标为

$$\min_x \frac{1}{N}\sum_{i=1}^{N}(\phi(x; a_i) - y_i)^2$$

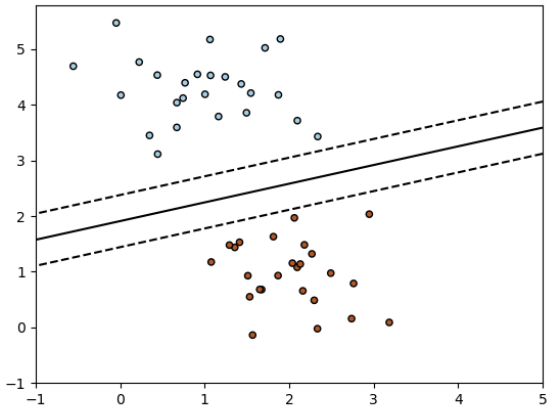


图 11.1: 分类任务示例。来源：https://scikit-learn.org/stable/modules/sgd.html

4. **优化技术**：这就是 SGD 等技术发挥作用的地方。它们的任务是调整模型的参数（如神经网络中的权重）以最小化损失。

深度神经网络 (DNN) 是一种前馈结构，通过堆叠多个神经元层来构建。每个神经元都会应用一个线性变换，然后再通过一个非线性激活函数进行激活。对于分类任务，最后一层通常后接一个 softmax 函数以产生类概率。然后，使用交叉熵损失来测量 DNN 的性能。
神经网络的函数如下：

- 第 l 层, 给定前一层的激活值 $a^{(l-1)}$ ：

$$z^{(l)} = \phi(W^{(l)}, b^{(l)}; a^{(l-1)}) = W^{(l)}a^{(l-1)} + b^{(l)} \tag{11.1}$$

- ReLU作为激活函数，得到第 l 层的激活值

$$a^{(l)} = \text{ReLU}(z^{(l)}) \tag{11.2}$$

- 对于最后一层，我们使用 softmax 函数得到类概率：

$$\text{softmax}(z_i^{(L)}) = \frac{e^{z_i^{(L)}}}{\sum_{j=1}^{C} e^{z_j^{(L)}}} \tag{11.3}$$

  其中，C 是类别的数量。

- 测量网络预测与实际标签之间差异的损失函数是交叉熵损失：

$$\text{CE}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\text{softmax}(z_{i,c}^{(L)})) \tag{11.4}$$

对于大规模神经网络训练，我们通常是求解下列形式的数学优化问题

$$\min_x f(x) = \frac{1}{N}\sum_{i=1}^{N} f(x, Y_i). \tag{11.5}$$

式子(11.5)中, 未知量 x 表示网络参数, $Y_i, i = 1, 2, \ldots, N$ 是训练数据, 它们组成总体训练集 $\{Y_1, \ldots, Y_N\}$。函数 $f(x)$ 表示的是模型的损失函数，例如在分类任务中，$f(x)$ 表示的是模型预测数据的分类与真实类别的误差。训练神经网络的目标即是寻找最优的参数 x 使得误差最小。一般来说, 求解优化问题(11.5)关注的是最小化训练误差，而机器学习关注的是降低模型的泛化误差。在本课程中，我们更关注的是优化算法求解训练模型的最优解。

神经网络模型具有以下两个特征：（1）参数量非常大；（2）训练样本数量非常大。表 11.1 和表 11.2 给出了一些经典的数据集和模型大小。模型规模和数据集大小制约了训练效率，大模型的训练有可能花费数天甚至数月时间。优化器的选择直接影响模型的训练效率。本节中，我们简单地探讨常用的模型训练优化器以及加速训练的并行分布式方法。下面，我们将会介绍常见的优化器的参数设定，如学习率，动量参数，批量大小等。

表 11.1: 大数据集数据量

数据集	数据集大小
Cifar10, Cifar100	60000 张图像
ImageNet	约 1400 万张图像
MS Coco	约 33 万张图像
GPT-3	45TB

表 11.2: 大模型参数量

模型名称	参数量
VGG16	1.4 亿
ResNet50	2500 万
DenseNet-201	2000 万
GPT-3	1750 亿
GPT-4	1.8 万亿

网络上爆料的 GPT-4 训练成本：一次的训练的成本为 6300 万美元，OpenAI 训练 GPT-4 的 FLOPS 约为 2.15e25，在大约 25000 个 A100 上训练了 90 到 100 天，利用率在 32% 到 36% 之间。

2 随机梯度算法 SGD

注意(11.5)的求和形式, 我们有

$$\nabla f(x) = \frac{1}{N}\sum_{i=1}^{N} \nabla f(x, Y_i).$$

因此当参数量非常大的时候，计算 $\nabla f(x, Y_i)$ 非常慢并且占用内存资源；当数据量 N 也非常大的时候，计算整体函数 $f(x)$ 的梯度是不可取的。所以，随机梯度算法应运而生。为了解决无法计算梯度的难点，SGD 的想法是随机抽取批量大小为 B 的子数据集 $\{Y_{i_1}, Y_{i_2}, \ldots, Y_{i_B}\}$，计算函数在子数据集上的梯度

$$\nabla f_B(x) = \frac{1}{B}\sum_{i=i_1,\ldots,i_B} \nabla f(x, Y_i).$$

在每一轮迭代（epoch）中，SGD 有两种方式选取子数据集（i）无重复（无放回）地选取子数据集进行批量梯度方向更新；（ii）可重复（有放回）地选取子数据集进行批量梯度方向更新。一般来说，实际中我们采用第（i）种方式，因为这种方式易于遍历所有数据集，使得训练得到的模型泛化能力更强。

Algorithm 1 随机梯度算法 SGD

Require: 算法迭代轮数 epochs，步长 (学习率)γ_t，批数据量 B

1: **for** t=1,2,..., epochs **do**
2: **for** i=1,2,...,n **do**
3: 随机选取大小为 B 的批数据集, 计算梯度 $\nabla f_B(x_{t,i})$
4: $x_{t,i+1} = x_{t,i} - \gamma_t \nabla f_B(x_{t,i})$
5: **end for**
6: $x_{t+1,1} = x_{t,n}$
7: **end for**

算法 1是 SGD 的迭代更新流程。其中，epochs 是总的更新轮数，t 是当前更新轮数。学习率 γ_t 是和 t 相关的参数。在每一轮更新中，我们采用无放回方式选取批数据，遍历全部数据集 $\{Y_1, Y_2, \ldots, Y_N\}$ 计算梯度进行更新。因此，共需要 $n = \lceil \frac{N}{B} \rceil$ 次内循环迭代（即算法 1中的第 2 到 5 行），每个内层循环更新即为批量梯度更新。

由于我们不知道整体梯度信息，批次梯度 $-\nabla f_B(x_{t,i})$ 并非下降方向。所以线性搜索在随机梯度法中无法使用。

2.1 参数选取

下面我们介绍 SGD 中常见的参数选取方式。

1. 迭代轮数设置。常见的 epochs 设定为 200 左右。如 Resnet 网络可在 100-300 次迭代得到很好的结果，Transformer 模型则需要更多的迭代，常用 300-500 次迭代。
2. 学习率设置。学习率 γ_t 和迭代轮数相关，随着 t 增大，γ_t 逐渐减小。一般有如下几种流行的方式。
   - γ_t 为一个恒定的常数。这种方式易于调参，但是最终训练误差会停在某个较大的值，所以不推荐此种方式。
   - γ_t 分段减小。例如，当 epochs 为 200 时，可以设定

$$\gamma_t = \begin{cases} 0.1, & t \leq 100 \\ 0.01, & 100 < t \leq 150 \\ 0.001, & 150 < t \leq 200 \end{cases} \tag{11.6}$$

   也就是说，在 $t = 100$ 和 $t = 150$ 时，我们缩小学习率 10 倍。这种分段缩小学习率的方法是最常见的设定方法。需要根据经验选择合适的初始学习率以及缩小阶段点。图 11.3展示的

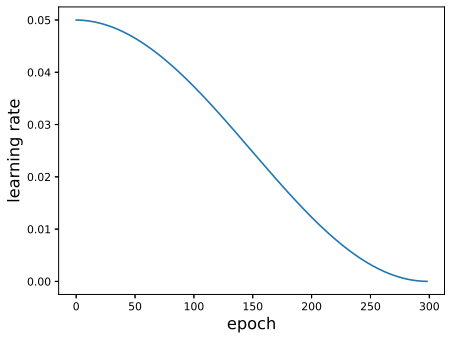


图 11.2: Cosine learning rate 示意图

是某神经网络，采用(11.6)学习率设定，函数值随着迭代轮数的变化情况。我们看到，在 100 和 150 次迭代时，由于学习率缩小 10 倍，函数值也有明显的快速下降。

3. 批量大小设置为了充分利用计算资源，我们尽可能选择最大的批量。因为大批量可以增加计算并行效率，提高训练速度。然而，有研究表明，批数据量大小直接影响泛化误差。因此，我们实际中也为设定一个上限，例如 $B = 128, 256$ 是最常见的选择。

4. 余弦函数学习率。由分段减小学习率函数变化启发，当迭代点接近最优点时，应当选取很小的学习率。训练初期，学习率应缓慢变小，起到热启动的作用。中期则快速下降，达到快速训练的目标。而训练末期又缓慢减小，逐步逼近最优点。利用余弦函数

$$\gamma_t = \gamma_{\min} + \frac{1}{2}(\gamma_{\max} - \gamma_{\min})(1 + \cos(\frac{t}{\text{epochs}}))$$

有效模拟了该想法。其中 $\gamma_{\min}$ 和 $\gamma_{\max}$ 分别是学习率的最小值和最大值。图 11.2所示为 cosine 学习率，以 $\gamma_{\min} = 10^{-5}$ 为最小，$\gamma_{\max} = 0.05$。此方法也是最常见的学习率设定方法。

3 动量随机梯度算法

(随机）梯度法简单易用，但解决困难的问题（例如优化函数条件数非常大）时，常出现“之”字轨迹现象。如图 4(a)所示的函数 $x_1^2 + 10x_2^2$ 等高线，由于函数在 x_2 方向变化更为陡峭，所以梯度更新轨迹在 x_2 方向过于“激进”，过于“信任”当前的梯度信息，而在 x_1 方向变化缓慢，造成了“之”字形轨迹，导致降低了算法收敛速度。动量随机梯度算法（SGD with momentum）可有效缓解这种情况，其描述见算法 2。动量随机梯度法更新方向为动量方向 $v_{t,i+1}$，该算法也多一个动量参数 η。易知，$\eta = 0$ 时，其等价于随机梯度法。实际中，η 常用的值为 0.9。动量迭代 $v_{t,i+1} = \eta v_{t,i} + \gamma_t \nabla f_B(x_{t,i})$ 可理解为利用了“历史”梯度信息，纠正当前过于“激进”的梯度方向。

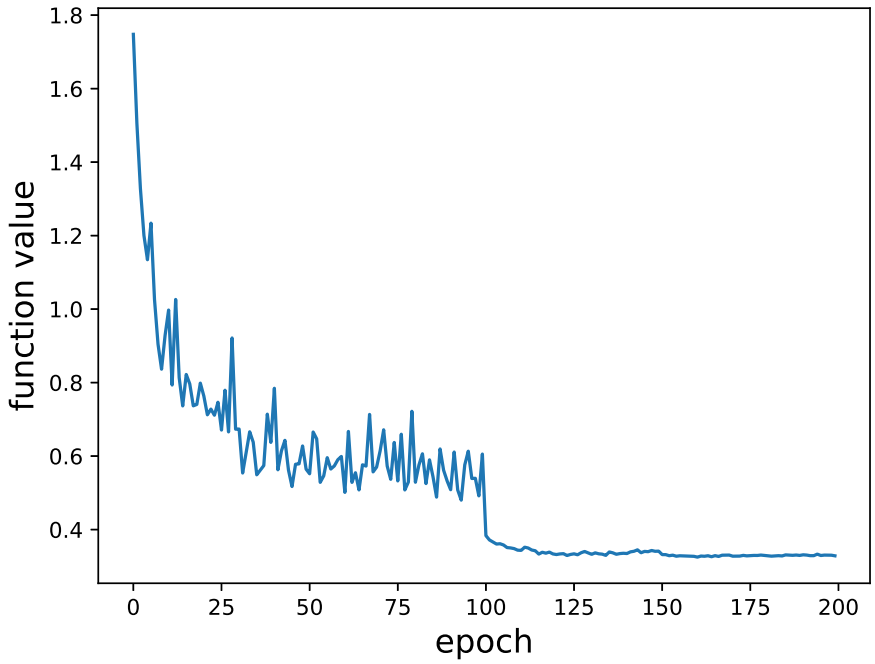


图 11.3: SGD 采用分段步长函数值变化示意图

Algorithm 2 动量随机梯度算法

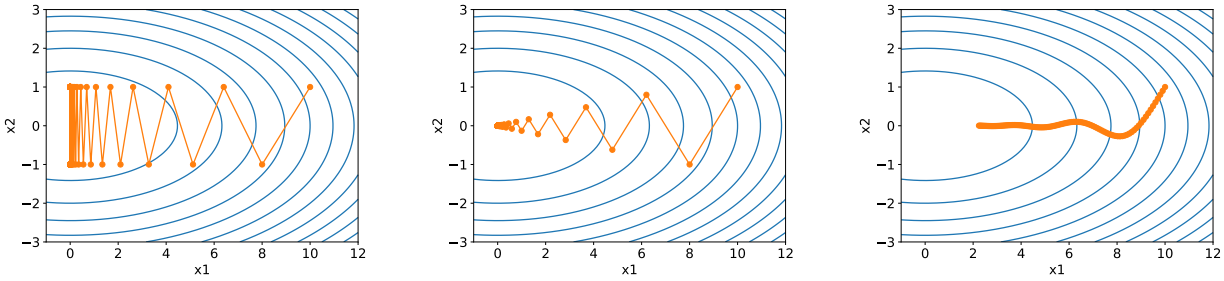
Require: 算法迭代轮数 epochs，学习率 γ_t，批数据量 B，初始动量 $v_{1,1}=0$, 动量参数 $0 \le \eta < 1$。

for t=1,2,..., epochs **do**
  for i=1,2,...,n **do**
    随机选取大小为 B 的批数据集, 计算梯度 $\nabla f_B(x_{t,i})$
    $v_{t,i+1} = \eta v_{t,i} + \gamma_t \nabla f_B(x_{t,i})$
    $x_{t,i+1} = x_{t,i} - v_{t,i+1}$
  end for
  $x_{t+1,1} = x_{t,n}, v_{t+1,1} = v_{t,n+1}$
end for

具体地，我们可以把迭代 $v_{t,i+1} = \eta v_{t,i} + \gamma_t \nabla f_B(x_{t,i})$ 展开为

$$\begin{aligned} v_{t,i+1} &= \gamma_t \nabla f_B(x_{t,i}) + \eta v_{t,i} \\ &= \gamma_t \nabla f_B(x_{t,i}) + \eta(\gamma_t \nabla f_B(x_{t,i-1}) + \eta v_{t,i-1}) \\ &= \dots \\ &= \gamma_t(\nabla f_B(x_{t,i}) + \eta \nabla f_B(x_{t,i-1}) + \eta^2 \nabla f_B(x_{t,i-2}) + \dots + \eta^j \nabla f_B(x_{t,i-j}) + \dots) \end{aligned}$$

图 4(b)展示的是动量梯度法求解函数 $x_1^2 + 10x_2^2$ 最小值的迭代轨迹。与图 4(a)中的梯度法相对比，动量梯度法轨迹在 x_2 更加缓和，因为动量随机梯度法在历史相同的梯度方向累积起来形成动量，而梯度变化快的方向相互抵消，缓解了振荡现象。总体上来说，动量梯度法只需添加一个参数 η, 其余参数设定方式可与 SGD 相似。因此，动量随机梯度法是训练神经网络非常流行的选择之一。



(a) 梯度法迭代轨迹示意图　(b) 动量梯度法迭代轨迹示意图　(c) Adam 迭代轨迹示意图

图 11.4: 三种优化器在函数 $x_1^2 + 10x_2^2$ 迭代轨迹

4 RMSProp 算法

前面我们知道，学习率很大程度上影响梯度算法的收敛表现。特别是对于如函数 $x_1^2 + 10x_2^2$，自变量 x_1, x_2 梯度方向大小不一致的情况，梯度法出现图 4(a)中“之”字振荡情况。RMSProp(全称为 Root Mean Square Prop, 由 Hinton 等提出）算法对于自变量 x 的不同坐标采用不同学习率，其算法描述见 3。

Algorithm 3 RMSProp 算法

Require: 算法迭代轮数 epochs，学习率 α，批数据量 B，初始向量 $s_{1,1} = 0$, 参数 $0 \leq \beta < 1$。

1: **for** t=1,2,..., epochs **do**
2: **for** i=1,2,...,n **do**
3: 随机选取大小为 B 的批数据集, 计算梯度 $\nabla f_B(x_{t,i})$
4: $s_{t,i+1} = \beta s_{t,i} + (1-\beta)\nabla f_B(x_{t,i}) \odot \nabla f_B(x_{t,i})$
5: $x_{t,i+1} = x_{t,i} - \frac{\alpha}{\sqrt{s_{t,i+1}+\epsilon}} \odot \nabla f_B(x_{t,i})$
6: **end for**
7: $x_{t+1,1} = x_{t,n}, s_{t+1,1} = s_{t,n+1}$
8: **end for**

初始 $s_{1,1}$ 是一个和变量 x 维度相同的全 0 向量。迭代 $s_{t,i+1} = \beta s_{t,i} + (1-\beta)\nabla f_B(x_{t,i}) \odot \nabla f_B(x_{t,i})$ 采用**指数平均**的方式更新。其中，符号 $\odot$ 表示向量之间按元素相乘。$s_{t,i+1}$ 追踪了函数梯度在不同坐标上的大小，指数平均的方式使得累加的梯度平方项变化更加平滑。具体地，我们考虑一般形式的指数平均

迭代

$$y_t = \beta y_{t-1} + (1-\beta) b_t.$$

上式可以展开为

$$\begin{aligned} y_t &= \beta y_{t-1} + (1-\beta) b_t \\ &= (1-\beta) b_t + (1-\beta)\beta b_{t-1} + \beta^2 y_{t-2} \\ &= (1-\beta) b_t + (1-\beta)\beta b_{t-1} + (1-\beta)\beta^2 b_{t-2} + \beta^3 y_{t-3} \\ &= \dots \end{aligned}$$

参数 β 作为权重系数，一般选为 0.9。因此，指数平均赋予了距离当前时间 t 近的参数相对大的权重系数，而逐渐忽略很久之前的参数，从而达到平滑化 y_t 随着时间的变化率。

因此，RMSProp 有效追踪了梯度在不同坐标分量的情况，并且变化率比较平滑。若是累积梯度平方项 s 较大，通过 $\frac{\gamma_t}{\sqrt{s_{t,i+1}+\epsilon}}$ 处理后使用较小的学习率, 这里的 $\epsilon > 0$ 是为了防止分母太小，保证数值稳定，例如可设置为 $\epsilon = 10^{-6}$。反之，若是某项梯度较小，则使用更大的学习率平衡各个方向学习率。特别注意到，机器学习中部分参数是稀疏的。在这种情况下，使用 RMSProp 将放大稀疏部分的更新量，从而加速收敛。

5 Adam 算法

由指数平均启发，动量随机梯度法也可以写成以下指数平均的形式：

$$v_{t,i+1} = (1-\eta)\frac{\gamma_t}{1-\eta}\nabla f_B(x_{t,i}) + \eta v_{t,i}.$$

结合 RMSProp 中的指数平均，可以组合动量估计 v_t 和梯度平方量估计 s_t 设计算法。因此，Adam(全称 adaptive moment estimation) 算法应运而生，其完整描述见算法 4, 它也是最为流行的训练神经网络的方法。β_1，β_2 分别是累积梯度的权重系数和动量权重系数，Adam 算法提出者建议设置为 $\beta_1 = 0.999, \beta_2 = 0.9$。算法中第 6 行为偏差修正：$\hat{v}_{t,i+1} = \frac{v_{t,i+1}}{1-\beta_1^t}$，$\hat{s}_{t,i+1} = \frac{s_{t,i+1}}{1-\beta_2^t}$，第 7 行更新使用修正后的 $\hat{v}_{t,i+1}$ 和 $\hat{s}_{t,i+1}$ 进行更新，其中 $\epsilon > 0$ 为保证数值稳定的常数，一般设置为 $\epsilon = 10^{-8}$。

Algorithm 4 Adam 算法

Require: 算法迭代轮数 epochs，学习率 α，批数据量 B，初始向量 $s_{1,1}=0$, 参数 $0\le\beta_1<1$，初始动量 $v_{1,1}=0$, 动量参数 $0\le\beta_2<1$。

for t=1,2,..., epochs **do**
  for i=1,2,...,n **do**
    随机选取大小为 B 的批数据集, 计算梯度 $\nabla f_B(x_{t,i})$
    $s_{t,i+1}=\beta_1 s_{t,i}+(1-\beta_1)\nabla f_B(x_{t,i})\odot\nabla f_B(x_{t,i})$
    $v_{t,i+1}=\beta_2 v_{t,i}+(1-\beta_2)\nabla f_B(x_{t,i})$
    偏差修正：$\hat{v}_{t,i+1}=\frac{v_{t,i+1}}{1-\beta_1^t}$, $\hat{s}_{t,i+1}=\frac{s_{t,i+1}}{1-\beta_2^t}$
    $x_{t,i+1}=x_{t,i}-\frac{\alpha}{\sqrt{\hat{s}_{t,i+1}}+\epsilon}\odot\hat{v}_{t,i+1}$
  end for
  $x_{t+1,1}=x_{t,n}, s_{t+1,1}=s_{t,n+1}, v_{t+1,1}=v_{t,n+1}$
end for

Adam 算法因结合了动量法和 RMSProp 算法，其优势在于学习率 α 更易于调节，算法在各种任务上表现更为稳定。实际中 α 对于不同任务可以采用大致相同量级的初始设定。另外，α 也可像 SGD 中的学习率 γ_t 一样随着时间改变，例如设为 cosine 学习率。图 4(c)也展示了 Adam 在求解 $x_1^2+10x_2^2$ 最小值的收敛轨迹。

6 其它随机优化算法

其他随机优化算法还有: 降方差的方法：SVRG、SAG、SAGA。自适应步长的算法：AdaGrad, AdaFactor 等。还有 Google 提出的用强化学习搜索得到的算法 lion。

7 随机梯度算法收敛性

我们只讨论随机梯度算法（SGD）的收敛性。
为了方便，我们记 $f(x,Y_i)=f_i(x)$. 每次随机选取 $i_k\in\{1,2,...,n\}$, SGD 迭代为

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

假设：我们将在以下假设下分析随机梯度率：

- f 有下界（不一定是凸的）。
- ∇f 是 L-Lipschitz 连续的。

- 对于某个常数 σ^2 和所有 x，有 $E[\|\nabla f_i(x)\|_2] \leq \sigma^2$（“方差” 有界）。

可以放宽噪声边界到更实际的假设 $E[\|\nabla f_i(x_k) - \nabla f(x_k)\|_2] \leq \sigma^2$。这只是在结果中引入了一些额外的项。

由’ 方差’ 有上界可得,

$$\begin{aligned}\mathbb{E}\left[f\left(x^{k+1}\right)\right] &\leq f\left(x^k\right) - \alpha_k \left\|\nabla f\left(x^k\right)\right\|^2 + \alpha_k^2 \frac{L}{2}\mathbb{E}\left[\left\|\nabla f_{i_k}\left(x^k\right)\right\|^2\right] \\ &\leq f\left(x^k\right) - \alpha_k \left\|\nabla f\left(x^k\right)\right\|^2 + \alpha_k^2 \frac{L\sigma^2}{2}\end{aligned}$$

- 整理可得

$$\alpha_k \left\|\nabla f\left(x^k\right)\right\|^2 \leq f\left(x^k\right) - \mathbb{E}\left[f\left(x^{k+1}\right)\right] + \alpha_k^2 \frac{L\sigma^2}{2}.$$

- 对 $k = 1, ...t$ 求和得

$$\sum_{k=1}^{t} \alpha_{k-1}\mathbb{E}\left\|\nabla f\left(x^{k-1}\right)\right\|^2 \leq \sum_{k=1}^{t}\left[\mathbb{E}f\left(x^{k-1}\right) - \mathbb{E}f\left(x^k\right)\right] + \sum_{k=1}^{t}\alpha_{k-1}^2\frac{L\sigma^2}{2}$$

继续处理上述不等式:

$$\sum_{k=1}^{t} \alpha_{k-1}\mathbb{E}\underbrace{\left\|\nabla f\left(x^{k-1}\right)\right\|^2}_{\text{bound by min}} \leq \sum_{k=1}^{t}\underbrace{\left[\mathbb{E}f\left(x^{k-1}\right) - \mathbb{E}f\left(x^k\right)\right]}_{\text{telescope}} + \sum_{k=1}^{t}\alpha_{k-1}^2\underbrace{\frac{L\sigma^2}{2}}_{\text{no } k}.$$

- 化简得

$$\min_{k=0,1,\ldots,t-1}\left\{\mathbb{E}\left\|\nabla f\left(x^k\right)\right\|^2\right\}\sum_{k=0}^{t-1}\alpha_k \leq f\left(x^0\right) - \mathbb{E}f\left(x^t\right) + \frac{L\sigma^2}{2}\sum_{k=0}^{t-1}\alpha_k^2.$$

- 因为 $\mathbb{E}f\left(x^k\right) \geq f^*$ ， 两边同时除以 $\sum_k \alpha_{k-1}$ 得

$$\min_{k=0,1,\ldots,t-1}\left\{\mathbb{E}\left\|\nabla f\left(x^k\right)\right\|^2\right\} \leq \frac{f\left(x^0\right) - f^*}{\sum_{k=0}^{t-1}\alpha_k} + \frac{L\sigma^2}{2}\frac{\sum_{k=0}^{t-1}\alpha_k^2}{\sum_{k=0}^{t-1}\alpha_k}$$

结果表明:

$$\min_{k=0,1,\ldots,t-1}\left\{\mathbb{E}\left\|\nabla f\left(x^k\right)\right\|^2\right\} \leq \frac{f\left(x^0\right) - f^*}{\sum_{k=0}^{t-1}\alpha_k} + \frac{L\sigma^2}{2}\frac{\sum_{k=0}^{t-1}\alpha_k^2}{\sum_{k=0}^{t-1}\alpha_k}.$$

- 若 $\sigma^2 = 0$, then we could use a constant step-size and would get a $O(1/t)$ rate.
- 由于随机性存在, 收敛速度取决于 $\sum_k \alpha_k^2 / \sum_k \alpha_k$.
- 单调下降步长: set $\alpha_k = \alpha/k$ for some α.
  - Gives $\sum_k \alpha_k = O(\log(t))$ and $\sum_k \alpha_k^2 = O(1)$, so error at t is $O(1/\log(t))$.
- 更大的下降步长: set $\alpha_k = \alpha/\sqrt{k}$ for some α.
  - Gives $\sum_k \alpha_k = O(\sqrt{k})$ and $\sum_k \alpha_k^2 = O(\log k)$, so error at t is $O(\log(t)/\sqrt{t})$.
- - 常数步长: set $\alpha_k = \alpha$ for some α.
  - Gives $\sum_k \alpha_k = k\alpha$ and $\sum_k \alpha_k^2 = k\alpha^2$, so error at t is $O(1/t) + O(\alpha)$