# 第八章 深度模型中的优化

2019/03/07

# 8.1 学习和纯优化有什么不同

- 8.1.1 经验风险最小化
  - 目标函数基于真实分布 $$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y).$$
  - 实际上算用的样本分布 $$\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}}[L(f(\boldsymbol{x};\boldsymbol{\theta}), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}),$$
- 8.1.2 代理损失函数和提前终止
  - 有时，我们真正关心的损失函数（比如分类误差）并不能被高效地优化。在这种情况下，我们通常会优化代理损失函数
  - 如之前章节，可能会提前终止
- 8.1.3 批量算法和小批量算法
  - 机器学习算法和一般优化算法不同的一点是，机器学习算法的目标函数通常可 以分解为训练样本上的求和。机器学习中的优化算法在计算参数的每一次更新时通 常仅使用整个代价函数中一部分项来估计代价函数的期望值。

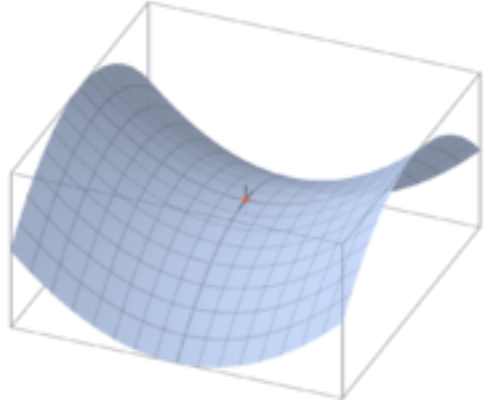
# 8.1.3 批量算法和小批量算法

- 使用整个训练集的优化算法被称为批量（batch）或确定性（deterministic）梯度算法，每次只使用单个样本的优化算法有时被称为随机（stochastic）或者在线（online）算法。
- 大多数用于深度学习的算法介于以上两者之间，使用一个以上，而又不是全部 的训练样本。传统上，这些会被称为小批量（minibatch）或小批量随机（minibatch stochastic）方法，现在通常将它们简单地称为随机（stochastic）方法。
- 小批量的大小通常由以下几个因素决定:

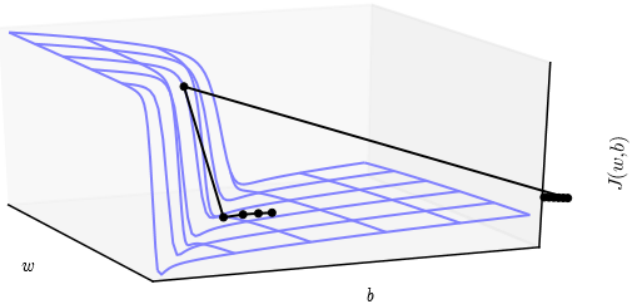
- 更大的批量会计算更精确的梯度估计，但是回报却是小于线性的。
- 极小批量通常难以充分利用多核架构。这促使我们使用一些绝对最小批量，低 于这个值的小批量处理不会减少计算时间。
- 如果批量处理中的所有样本可以并行地处理（通常确是如此），那么内存消耗 和批量大小会正比。对于很多硬件设施，这是批量大小的限制因素。
- 在某些硬件上使用特定大小的数组时，运行时间会更少。尤其是在使用GPU时， 通常使用 2 的幂数作为批量大小可以获得更少的运行时间。一般，2 的幂数的 取值范围是 32 到 256，16 有时在尝试大模型时使用。
- 可能是由于小批量在学习过程中加入了噪声，它们会有一些正则化效果(Wilson and Martinez, 2003)。泛化误差通常在批量大小为 1 时最好。因为梯度估计的 高方差，小批量训练需要较小的学习率以保持稳定性。因为降低的学习率和消 耗更多步骤来遍历整个训练集都会产生更多的步骤，所以会导致总的运行时间非常大。

# 8.2 神经网络优化中的挑战

- 8.2.1 病态
  - 小扰动导致逆改变很大的矩阵
- 8.2.2 局部极小值
  - 对于实际中感兴趣的网络，是否存在大量代价很高的局部极小值，优化算法是 否会碰到这些局部极小值，都是尚未解决的公开问题。
- 8.2.3 高原、鞍点和其他平坦区域
  - 低维空间中，局部极小值很普遍。在更高维空间中，局部极小值很罕见，而鞍点则很常见。
    - Hessian矩阵在局部极小点处只有正特征值。
  - 牛顿法会被鞍点影响，梯度下降法不会



# 8.2 神经网络优化中的挑战




- 8.2.4 悬崖和梯度爆炸
  - 我们可以用使 用第10.11.1节介绍的启发式梯度截断（gradient clipping）来避免其严重的后果。其基本想法源自梯度并没有指明最佳步长，只说明了在无限小区域内的最佳方向。当 传统的梯度下降算法提议更新很大一步时，启发式梯度截断会干涉来减小步长
- 8.2.5 长期依赖
  - 由于变深的结构使模型丧失了学习到先前信息的能力，让优化变得极 其困难。
- 8.2.6 非精确梯度
  - 几乎每一个深度学习算法都需要基 于采样的估计，至少使用训练样本的小批量来计算梯度。

# 8.2 神经网络优化中的挑战

- 8.2.7 局部和全局结构间的弱对应
  - 我们有可能在单点处克服以上所有困难，但仍然表现不佳。
- 8.2.8 优化的理论限制
  - 一些理论结果表明，我们为神经网络设计的任何优化算法都有性能限制。

# 8.3 基本算法

- 随机梯度下降（SGD）

**算法 8.1** 随机梯度下降（SGD）在第 $k$ 个训练迭代的更新

**Require:** 学习率 $\epsilon_k$

**Require:** 初始参数 $\boldsymbol{\theta}$

**while** 停止准则未满足 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，其中 $\boldsymbol{x}^{(i)}$ 对应目标为 $\boldsymbol{y}^{(i)}$。

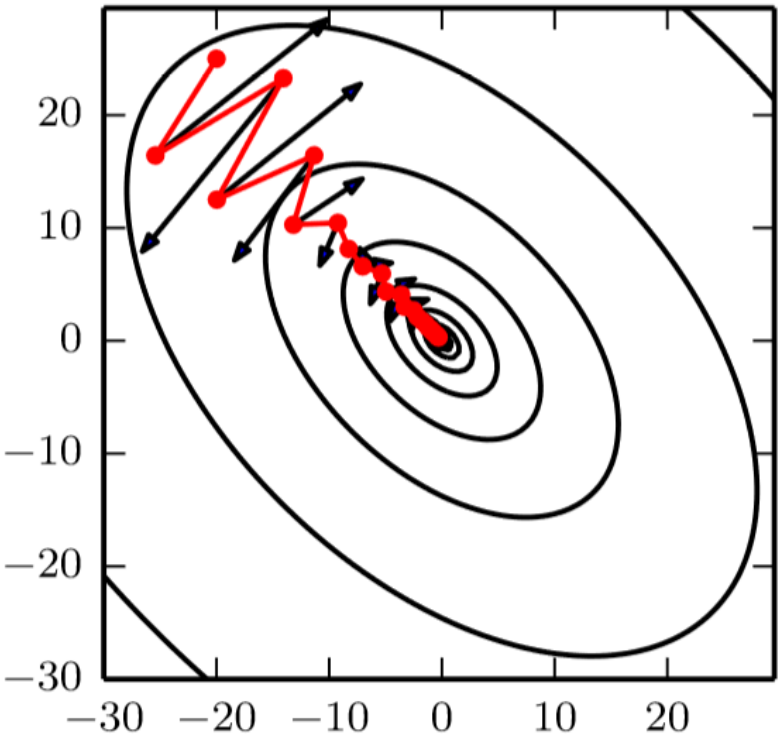
计算梯度估计：$\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$

**end while**

# 8.3 基本算法

- 8.3.2 动量




**算法 8.2** 使用动量的随机梯度下降（SGD）

**Require:** 学习率 $\epsilon$，动量参数 $\alpha$

**Require:** 初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度估计：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

**end while**

# 8.3 基本算法

- 8.3.3 Nesterov 动量

**算法 8.3** 使用Nesterov 动量的随机梯度下降（SGD）

**Require:** 学习率 $\epsilon$，动量参数 $\alpha$

**Require:** 初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

应用临时更新：$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

计算梯度（在临时点）：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

**end while**

# 8.4 参数初始化策略

- 我们可以明确地搜索一大组彼此互不相同的基函数，但这经常会导致明显的计算代价。例如，如果我们有和输出一样多的输入，我们可以使用 Gram-Schmidt 正交化于初始的权重矩阵，保证每个单元 计算彼此非常不同的函数。在高维空间上使用高熵分布来随机初始化，计算代价小 并且不太可能分配单元计算彼此相同的函数。
- 我们几乎总是初始化模型的权重为高斯或均匀分布中随机抽取的值。

# 8.4 参数初始化策略

- 标准初始化 $W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$
- 可惜，这些初始权重的最佳准则往往不会带来最佳效果。这可能有三种不同的 原因。首先，我们可能使用了错误的标准——它实际上并不利于保持整个网络信号 的范数。其次，初始化时强加的性质可能在学习开始进行后不能保持。最后，该标 准可能成功提高了优化速度，但意外地增大了泛化误差。在实践中，我们通常需要 将权重范围视为超参数，其最优值大致接近，但并不完全等于理论预测。

# 8.5 自适应学习率算法

- 学习率是难以设置的超参数之一，因为它对模 型的性能有显著的影响。

# 8.5.1 AdaGrad

- 缩放每 个参数反比于其所有梯度历史平方值总和的平方根
- 对于训练深度神经网络模型而言，从训练开始时积累梯度平方会导致有效学习率过早和过量的减小。AdaGrad只在某些深度学习模型上效果不错。

**算法 8.4** AdaGrad 算法

**Require:** 全局学习率 $\epsilon$

**Require:** 初始参数 $\boldsymbol{\theta}$

**Require:** 小常数 $\delta$，为了数值稳定大约设为 $10^{-7}$

初始化梯度累积变量 $\boldsymbol{r} = 0$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ （逐元素地应用除和求平方根）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end while**

# 8.5.2 RMSProp

- 修改 AdaGrad 以在非凸设定下效果更好，改变梯度积累为指数加权的移动平均

**算法 8.5** RMSProp 算法

**Require:** 全局学习率 $\epsilon$，衰减速率 $\rho$，动量系数 $\alpha$

**Require:** 初始参数 $\boldsymbol{\theta}$，初始参数 $\boldsymbol{v}$

初始化累积变量 $\boldsymbol{r} = 0$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算临时更新：$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

累积梯度：$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$　($\frac{1}{\sqrt{\boldsymbol{r}}}$ 逐元素应用)

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

**end while**

# 8.5.3 Adam

**算法 8.7** Adam 算法

**Require:** 步长 $\epsilon$（建议默认为：0.001）

**Require:** 矩估计的指数衰减速率，$\rho_1$ 和 $\rho_2$ 在区间 $[0, 1)$ 内。（建议默认为：分别为 0.9 和 0.999）

**Require:** 用于数值稳定的小常数 $\delta$（建议默认为：$10^{-8}$）

**Require:** 初始参数 $\boldsymbol{\theta}$

初始化一阶和二阶矩变量 $\boldsymbol{s} = 0$, $\boldsymbol{r} = 0$

初始化时间步 $t = 0$

**while** 没有达到停止准则 **do**

从训练集中采包含 $m$ 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

修正一阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$（逐元素应用操作）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end while**

# 8.5.4 选择正确的优化算法

- 但没有哪个算法能脱颖而出。
- 目前，最流行并且使用很高的优化算法包括 SGD、具动量的 SGD、RMSProp、 具动量的 RMSProp、AdaDelta 和 Adam。此时，选择哪一个算法似乎主要取决于使用者对算法的熟悉程度（以便调节超参数）。

# 8.6 二阶近似方法

- 8.6.1 牛顿法
  - 遇到鞍点出问题

可正则化海森矩阵，通常是在对角线增加常数α

  - 二阶算法计算量比较大

**算法 8.8** 目标为 $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$ 的牛顿法

**Require:** 初始参数 $\boldsymbol{\theta}_0$

**Require:** 包含 $m$ 个样本的训练集

**while** 没有达到停止准则 **do**

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算 Hessian 矩阵：$\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算 Hessian 逆：$\boldsymbol{H}^{-1}$

计算更新：$\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$

应用更新：$\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end while**

# 8.6 二阶近似方法

- 8.6.2 共轭梯度
  - 共轭方向 $\boldsymbol{d}_t^\top \boldsymbol{H}\boldsymbol{d}_{t-1} = 0$

**算法 8.9** 共轭梯度方法

**Require:** 初始参数 $\boldsymbol{\theta}_0$

**Require:** 包含 $m$ 个样本的训练集

初始化 $\boldsymbol{\rho}_0 = 0$

初始化 $g_0 = 0$

初始化 $t = 1$

**while** 没有达到停止准则 **do**

初始化梯度 $\boldsymbol{g}_t = 0$

计算梯度：$\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算 $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)

(非线性共轭梯度：视情况可重置 $\beta_t$ 为零，例如 $t$ 是常数 $k$ 的倍数时，如 $k = 5$)

计算搜索方向：$\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$

执行线搜索寻找：$\epsilon^* = \text{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon\boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$

（对于真正二次的代价函数，存在 $\epsilon^*$ 的解析解，而无需显式地搜索）

应用更新：$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$

$t \leftarrow t + 1$

**end while**

# 8.7 优化策略和元算法

- 许多优化技术并非真正的算法，而是一般化的模板，可以特定地产生算法，或 是并入到很多不同的算法中。
- 8.7.1 批标准化
  - 设 H 是需要标准化的某层的小批量激活函数
  - 为了标准化 H，我们将其替换为 $\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}},$

$$\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{H}_{i,:}$$

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2},$$

- 8.7.2 坐标下降
  - 一次优化一个坐标叫坐标下降, 一次优化几个坐标叫块坐标下降
- 8.7.3 Polyak 平均
  - 平均优化算法在参数空间访问轨迹中的几个点。 $\hat{\boldsymbol{\theta}}^{(t)} = \alpha\hat{\boldsymbol{\theta}}^{(t-1)} + (1-\alpha)\boldsymbol{\theta}^{(t)}.$

# 8.7 优化策略和元算法

- 8.7.4 监督预训练
  - 在直接训练目标模型求解目标问题之前，训练简单模型求解简 化问题的方法统称为预训练
- 8.7.5 设计有助于优化的模型
  - 具体来说，现代神经网络的设计选择体现在层之间的线性变换，几乎处处可导的激活函数，和大部分定义域都有明显的梯度。
- 8.7.6 延拓法和课程学习
  - 为了最小化代价函数 J(θ)，我们构建新的代价函数 {J(0),...,J(n)}。这些代价函数的难度逐步提高。