# 基础神经网络--优化器

[[TOC]]

## 0.资料网址：

- 飞桨文档
- Deep Learning Book
- AdamDelta的参考Dive into deep learning
- 本文所有图片的主要参考
- 另一个好的参考

## 1. 问题定义

优化算法(Optimization Algorithm)是机器学习理论中重要的组成部分，每年ICML会议中，总会有很多关于优化算法Paper以及优化算法的Workshop。

机器学习应用到实际场景中，第一个步骤需要数学建模，将实际问题抽象转化成机器学习问题。

- 问题定义
  我们会针对特定的应用场景会提出假设：$h(w)$，$h(w)$ 可以很好的刻画真实的场景。
  虽然我们对 $h(w)$ 的参数 $w$ 一无所知，但是我们可以观测到该场景中的现象(特征)和结果(标签)，即样本。
  我们需要基于观测的样本，求解一个最优的 $w$ 使 $h(w)$ 最好的拟合观测到的数据；
  这里就需要优化算法求解最优的 $w$，如，最常见的优化算法—梯度下降.

- 定义目标函数
  假设 $h(\theta)$ 是要拟合的函数，$J(\theta)$ 为目标函数（或者，叫损失函数）；其中 $\theta$ 是参数，迭代求解的值。$(x,y)$ 是样本数据。 那么，

$$J(\theta) = L(h(\theta,x),y) = \frac{1}{m}\sum_{i=1}^{m} f(h(\theta,x_i),y_i)$$

  则，我们的优化问题是：

$$\theta = \min_{\theta \in R^n} J(\theta)$$

  求得一个 $\theta$ 值，使 $J(\theta)$ 最小。

# 1 梯度下降(Gradient Descent)

梯度下降是求解无约束最优化问题的一种最常用的优化算法。其思想非常简单，即假设目标函数 $J(\theta)$ 梯度的反方向是目标函数减小最快的方向。
因此，$\theta$ 迭代优化公式如下

$$\theta^{k+1} \leftarrow \theta^k - \lambda \nabla J(\theta^k)$$

其中 $\nabla J(\theta^k)$ 是 $J(\theta)$ 在 $\theta^k$ 处的一阶导数； $\lambda$ 是步长。

$\theta \in R^n$ 属于 $n$ 维向量，上式中，可以对每个维度分别求偏导数，更新每个维度 $\theta_i$。

# 2.普通梯度下降

# 2 Batch VS. Stochastic

梯度下降法最核心的的部分是计算梯度，每轮迭代得到更新后 $\theta$ ，需要重复计算梯度。
在大规模的机器学习算法中，样本的数量很大，维度很高（样本量和维度都会达到亿级别），因此这里会变成非常耗时、耗资源的计算部分。

由此，梯度下降演变成两种不同类别的算法：批量梯度下降法(Batch Gradient Descent, BGD)和随机梯度下降法(Stochastic Gradient Descent, SGD)

## 2.1 批量梯度下降法(BGD)

批量梯度下降法是每轮迭代的时候使用了所有的样本算平均梯度，即

$$\theta^{k+1} \leftarrow \theta^k - \lambda \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta} f(h(\theta, x_i), y_i)$$

其中，$m$是样本量，用**所有**样本的梯度均值作为 $\theta$ 更新的梯度方向，因此每轮迭代都需要遍历所有的样本。

特点:

- 算法的收敛性好
- BGD每轮迭代用上所有的样本，可以得到当前最优的参数更新方向，算法的鲁棒性强
- 由于梯度是所有样本梯度的累加，因此BGD可以很好地做分布式并行化。

## 2.2 随机梯度下降法(SGD)

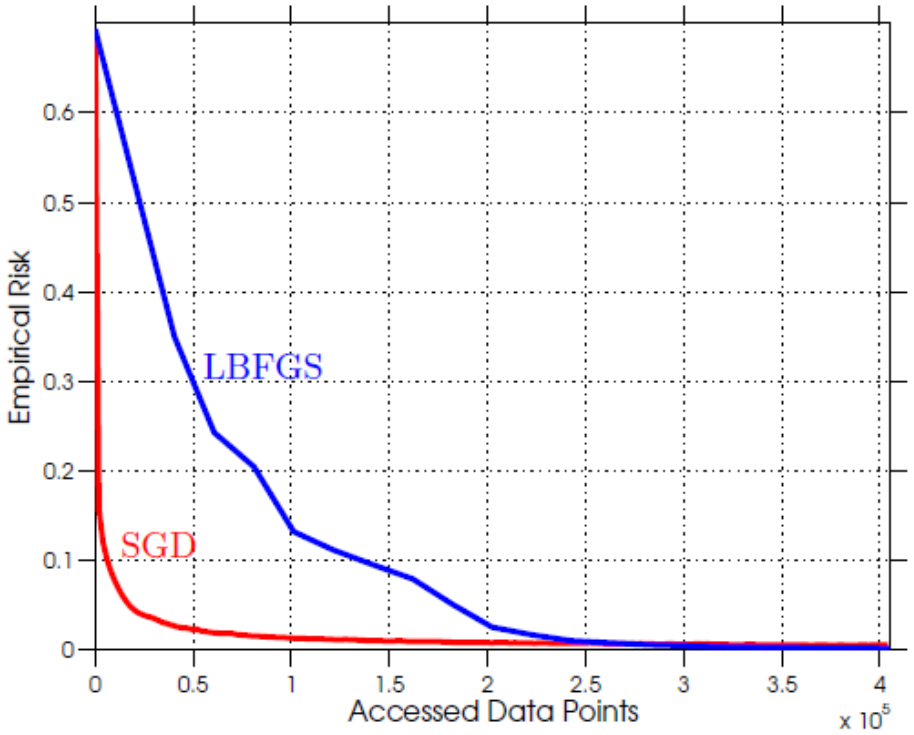
随机梯度下降法用单个样本的梯度值作为梯度更新方向，即

$$\theta^{k+1} \leftarrow \theta^k - \lambda \frac{\partial}{\partial \theta} f(h(\theta, x_i), y_i)$$

每轮迭代只用到一个样本，每轮迭代随机选取中**一个**样本。因此，需要保证更新的次数必须是$m$的数倍，$m$ 为样本数。

特点：

- SGD更充分地使用了样本携带的信息，每个样本计算所得的梯度都会尽可能地用于修正参数；
- 每轮迭代只用到一个样本，迭代效率更高；
- 在某些情况下，可以不需要全部的训练数据，就能得到不错的优化解；因此没有必要通过BGD用所有的数据计算梯度，只需要SGD即可高效地完成迭代；
- 在真实的应用场景中，SGD的收敛速度很快，甚至超过LBFGS。纵坐标是目标函数，横坐标是参与梯度更新的样本累计总数。LBFGS每轮迭代是用全量更新，每轮迭代是一个epoch。




- SGD每次只使用一个样本更新梯度，这样会给算法引入随机性；这种随机性即可以在一定程度上避免陷入局部最优，但同时也引入了噪声。
  由上图亦可知，SGD最终的收敛性不如LBFGS。

## 2.3 mini-batch梯度下降法(mini-batch GD)

mini-batch Gradient Descent 综合BGD和SGD的优点，将$m$个样本的数据集随机划分成$b$个batch，每个batch包含 $\frac{m}{b}$ 个样本。每轮迭代时，用一个batch的样本计算梯度，即

$$\theta^{k+1} \leftarrow \theta^k - \lambda \frac{1}{m/b} \sum_{i=1}^{m/b} \frac{\partial}{\partial \theta} f(h(\theta, x_i), y_i)$$

如果$b = 1$，mini-batch GD就是BGD；同样，如果$b = m$，mini-batch GD就变成了SGD。
目前，大规模数据下的优化算法，在离线部分模型训练会选择mini-batch GD，这既保证模型快速更新，同时也能保证较好的收敛性。

PS. 在实际场景中，我们所说的SGD就是指mini-batch GD。

# 3.梯度下降变种

## 3 基于动量的梯度下降(Gradient Descent with Momentum)

虽然SGD是非常受欢迎的优化方法，但在复杂的场景下，其收敛过程很慢，如大规模的神经网络模型。同时，SGD很难通过陡谷，即在一个维度上的表面弯曲度远大于其他维度的区域，这种情况通常是在局部最优点附近。在这种情况下，SGD摇摆地通过陡谷的斜坡，同时，沿着底部到局部最优点的路径上只是缓慢地前进。如下图(a)所示，不仅参数更新速度慢，而且容易陷入局部最优解。
基于动量的梯度下降可以帮助SGD在相关方向上加速并抑制在陡谷中的摇摆。如下图(b)所示

(a)

(b)

**动量** (Momentum)在物理学上定义为质量乘以速度；当我们假设质量是单位质量，那么速度就可以看成粒子的动量。

$$v^{k+1} \leftarrow \alpha v^k + \lambda \nabla J(\theta^k)$$

$$\theta^{k+1} \leftarrow \theta^k - v^{k+1}$$

当参数 $\alpha$ 和 $\lambda$ 越大，则表示之前的梯度对现在的影响越大。
算法迭代步长取决于梯度序列的大小和排序。当许多连续的梯度指向相同的方向是，步长就会很大。

在实践中，$\alpha$ 一般取0.5，0.9或者0.99. $\alpha$ 也会随时间不断调整，一般初始值是一个较小的值，随后会慢慢变大。随时间推移，调整 $\alpha$ 没有收缩 $\lambda$ 重要。

从本质上说，动量法就像我们从山上推下一个球，球在滚下来的过程中累积动量，变得越来越快。同样的事情也发生在参数的更新过程中：对于在梯度点处具有相同的方向的维度，其动量项增大，对于在梯度点处改变方向的维度，其动量项减小。因此，我们可以得到更快的收敛速度，同时可以减少摇摆。

受Nesterov加速梯度算法(Nesterov, 1983, 2004)启发，Sutskever et al.(2013)提出了动量算法的一种变种。更新公式如下：

$$v^{k+1} \leftarrow \alpha v^k + \lambda \nabla J(\theta^k + \alpha v^k)$$

$$\theta^{k+1} \leftarrow \theta^k - v^{k+1}$$

# 4.AdaGrad, RMSprop,AdaDelta, Adam

# 4 自适应学习率算法

梯度下降算法中，学习率是一个很难设置的超参数，而且它对模型的性能有很明显的影响。如果学习率太大，会导致目标函数值抖动，最终无法得到一个较好的收敛值；如果学习率太小，算法不但会收敛慢，而且极易陷入局部最优解。
基于动量的算法可以在一定程度上缓解这个问题，但是其代价是引入了额外的超参数。

## 4.1 AdaGrad(Adaptive Gradient Descent)

AdaGrad算法的学习率适应于模型参数 $\theta$ ，对于不常出现的维度，用较大的学习率；对于频繁出现的维度用较小的学习率。正因如此，AdaGrad非常适合稀疏的数据，AdaGrad大大提高了SGD算法的鲁棒性，同时也被用于训练大规模的神经网络。

$$g^k = \nabla J(\theta^k)$$

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

$$\theta^{k+1} \leftarrow \theta^k - \frac{\eta}{\sqrt{G^k} + \epsilon} \odot g^k$$

$\odot$ 表示向量和向量之间进行element-wise运算。

其中 $G^k$ 的第 $i$ 维是 $\theta$ 的第 $i$ 维 $\theta_i$ 从第 $0$ 轮到 $k$ 轮梯度平方和； $\epsilon$ 是平滑参数，避免出现除0的情况，一般 $\epsilon$ 取 1e-8 。

AdaGrad的优点之一是避免了人工对学习率的调参，大多数情况下，设 $\eta = 0.01$ 即可。
AdaGrad的缺点是随着不断的迭代，梯度平方和不断增加，甚至会趋向于无穷小；导致有效学习率过早或者过度的减小。

## 4.2 RMSProp

RMSProp算法(Hinton, 2012)改进了AdaGrad使其在非凸设定下有更好的效果，将AdaGrad的梯度平方和变成梯度指数衰减的移动平均。 AdaGrad在凸优化问题下可以快速收敛。RMSProp使用指数衰减的移动平均，丢弃较早的历史值，使算法的收敛性更好。

$$g^k = \nabla J(\theta^k)$$

$$E^k = \gamma E^{k-1} + (1-\gamma) g^k \cdot g^k$$

$$\theta^{k+1} \leftarrow \theta^k - \frac{\eta}{\sqrt{E[g^2]} + \epsilon} \odot g^k$$

Hinton建议 $\gamma = 0.9$， $\eta = 0.001$ ，通用$\epsilon$ 取 1e-8 即可。

### 11.9.1 Adadelta算法

简而言之，Adadelta使用两个状态变量，$\mathbf{s}_t$用于存储梯度二阶导数的泄露平均值，$\Delta\mathbf{x}_t$用于存储模型本身中参数变化二阶导数的泄露平均值。请注意，为了与其他出版物和实现的兼容性，我们使用作者的原始符号和命名（没有其它真正理由让大家使用不同的希腊变量来表示在动量法、AdaGrad、RMSProp和Adadelta中用于相同用途的参数）。

以下是Adadelta的技术细节。鉴于参数du jour是$\rho$，我们获得了与 11.8节类似的以下泄漏更新：

$$\mathbf{s}_t = \rho \mathbf{s}_{t-1} + (1 - \rho)\mathbf{g}_t^2. \tag{11.9.1}$$

与 11.8节的区别在于，我们使用重新缩放的梯度$\mathbf{g}_t'$执行更新，即

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \mathbf{g}_t'. \tag{11.9.2}$$

那么，调整后的梯度$\mathbf{g}_t'$是什么？我们可以按如下方式计算它：

$$\mathbf{g}_t' = \frac{\sqrt{\Delta\mathbf{x}_{t-1} + \epsilon}}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t, \tag{11.9.3}$$

其中$\Delta\mathbf{x}_{t-1}$是重新缩放梯度的平方$\mathbf{g}_t'$的泄漏平均值。我们将$\Delta\mathbf{x}_0$初始化为0，然后在每个步骤中使用$\mathbf{g}_t'$更新它，即

$$\Delta\mathbf{x}_t = \rho \Delta\mathbf{x}_{t-1} + (1 - \rho)\mathbf{g}_t'^2, \tag{11.9.4}$$

和$\epsilon$（例如$10^{-5}$这样的小值）是为了保持数字稳定性而加入的。

## 4.3 Adam(Adaptive Moment Estimation)

Adam算法(Kingma and Ba, 2014)是另外一种自适应学习率的优化方法。Adam融合了RMSProp和动量方法两种特性，即通过RMSProp做学习率的自适应，同时也通过动量法防止算法的抖动摇摆。

$$m^{k+1} = \alpha m^k + (1-\alpha) g^k$$

$$v^{k+1} = \beta v^k + (1-\beta) g^k \odot g^k$$

$m^k$ 是一阶梯度的运算(如 动量方法，代表着梯度的均值)，而 $v^k$ 是二阶梯度的运算(代表着梯度的方差)

$$M^k = \frac{m^k}{1-\alpha^k}$$

$$V^k = \frac{v^k}{1-\beta^k}$$

$$\theta^{k+1} \leftarrow \theta^k - \frac{\eta}{\sqrt{V^k} + \epsilon} M^k$$

Adam作者建议相关参数的默认值，$\beta_1 = 0.9, \beta_2 = 0.999, \eta = 1e-8$，实践中，Adam优化算法比其他自适应算法效果更好。

# 参考文献


[1] 李航，统计学习方法

[2] An overview of gradient descent optimization algorithms

[3] Optimization Methods for Large-Scale Machine Learning