L1 和 L2

作用
优化方法：
L1 坐标下降，LARS 角回归

优化方法

BGD

即 Batch Gradient Descent. 在训练中,每一步迭代都使用训练集的所有内容. 也就是说,利用现有参数对训练集中的每一个输入生成一个估计输出$\hat{y}_i$,然后跟实际输出y_i比较,统计所有误差,求平均以后得到平均误差,以此来作为更新参数的依据.

具体实现:

需要:学习速率 ϵ, 初始参数 θ
每步迭代过程:
1. 提取训练集中的所有内容$\{x_1,\cdots,x_n\}$,以及相关的输出y_i
2. 计算梯度和误差并更新参数:

$$\hat{g} \leftarrow +\frac{1}{n}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

$$\theta \leftarrow \theta - \epsilon\hat{g}$$

优点:

由于每一步都利用了训练集中的所有数据,因此当损失函数达到最小值以后,能够保证此时计算出的梯度为 0,换句话说,就是能够收敛.因此,使用 BGD 时不需要逐渐减小学习速率ϵ_k

缺点:

由于每一步都要使用所有数据,因此随着数据集的增大,运行速度会越来越慢.

SGD

SGD 全名 stochastic gradient descent， 即随机梯度下降。不过这里的 SGD 其实跟 MBGD(minibatch gradient descent)是一个意思,即随机抽取一批样本,以此为根

据来更新参数.

具体实现:

需要:学习速率 ϵ, 初始参数 θ
每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差并更新参数:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

$$\theta \leftarrow \theta - \epsilon\hat{g}$$

优点:

训练速度快,对于很大的数据集,也能够以较快的速度收敛.

缺点:

由于是抽取,因此不可避免的,得到的梯度肯定有误差.因此学习速率需要逐渐减小.否则模型无法收敛
因为误差,所以每一次迭代的梯度受抽样的影响比较大,也就是说梯度含有比较大的噪声,不能很好的反映真实梯度.

学习速率该如何调整:

那么这样一来,ϵ 如何衰减就成了问题.如果要保证 SGD 收敛,应该满足如下两个要求:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty$$

$$\sum_{k=1}^{\infty} {\epsilon_k}^2 < \infty$$

而在实际操作中,一般是进行线性衰减:

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

$$\alpha = \frac{k}{\tau}$$

其中ϵ_0是初始学习率, ϵ_τ是最后一次迭代的学习率. τ 自然代表迭代次数.一般来说, ϵ_τ 设为ϵ_0的 1%比较合适.而 τ 一般设为让训练集中的每个数据都输入模型上百次比较合适.那么初始学习率ϵ_0怎么设置呢?书上说,你先用固定的学习速率迭代 100 次,找出效果最好的学习速率,然后 ϵ0 设为比它大一点就可以了.

Momentum

上面的 SGD 有个问题,就是每次迭代计算的梯度含有比较大的噪音. 而 Momentum 方法可以比较好的缓解这个问题,尤其是在面对小而连续的梯度但是含有很多噪声的时候,可以很好的加速学习.Momentum 借用了物理中的动量概念,即前几次的梯度也会参与运算.为了表示动量,引入了一个新的变量 v(velocity).v 是之前的梯度的累加,但是每回合都有一定的衰减.

具体实现:

需要:学习速率 ϵ, 初始参数 θ, 初始速率 v，动量衰减参数α
每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差并更新速度 v 和参数θ:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

$$v \leftarrow \alpha v - \epsilon \hat{g}$$
$$\theta \leftarrow \theta - v$$

其中参数α表示每回合速率 v 的衰减程度.同时也可以推断得到,如果每次迭代得到的梯度都是 g,那么最后得到的 v 的稳定值为

$$\frac{\epsilon||g||}{1-\alpha}$$

也就是说,Momentum 最好情况下能够将学习速率加速$\frac{1}{1-\alpha}$倍.一般α的取值有 0.5,0.9,0.99 这几种.当然,也可以让 α 的值随着时间而变化,一开始小点,后来再加大.不过这样一来,又会引进新的参数.

特点:

前后梯度方向一致时,能够加速学习
前后梯度方向不一致时,能够抑制震荡

Nesterov Momentum

这是对之前的 Momentum 的一种改进,大概思路就是,先对参数进行估计,然后使用估计后的参数来计算误差

具体实现:

需要:学习速率 ϵ, 初始参数 θ, 初始速率 v，动量衰减参数α

每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差并更新速度 v 和参数θ:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta),y_i)$$

$$v \leftarrow \alpha v - \epsilon \hat{g}$$

$$\theta \leftarrow \theta + v$$

注意在估算 ĝ 的时候,参数变成了θ+αv 而不是之前的θ

AdaGrad

AdaGrad 可以自动变更学习速率,只是需要设定一个全局的学习速率 ϵ,但是这并非是实际学习速率,实际的速率是与以往参数的模之和的开方成反比的.也许说起来有点绕口,不过用公式来表示就直白的多:

$$\epsilon_n = \frac{\epsilon}{\delta + \sqrt{\sum_{i=1}^{n-1} g_i \odot g_i}}$$

其中 δ 是一个很小的常亮,大概在 10−7,防止出现除以 0 的情况.

具体实现:

需要:全局学习速率 ϵ, 初始参数 θ, 数值稳定量 δ
中间变量: 梯度累计量 r(初始化为 0)
每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差,更新 r,在根据 r 和梯度计算参数更新量:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta),y_i)$$

$$r \leftarrow r + \hat{g} \odot \hat{g}$$

$$\Delta\theta = -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$$

$$\theta \leftarrow \theta + \Delta\theta$$

优点:

能够实现学习率的自动更改。如果这次梯度大,那么学习速率衰减的就快一些;如果这次梯度小,那么学习速率衰减的就满一些。

缺点：

任然要设置一个变量 ϵ
经验表明，在普通算法中也许效果不错，但在深度学习中，深度过深时会造成训练提前结束。

RMSProp

RMSProp 通过引入一个衰减系数，让 r 每回合都衰减一定比例，类似于 Momentum 中的做法。

具体实现：

需要:全局学习速率 ϵ, 初始参数 θ, 数值稳定量 δ， 衰减速率ρ
中间变量: 梯度累计量 r(初始化为 0)
每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差,更新 r,在根据 r 和梯度计算参数更新量:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta),y_i)$$

$$r \leftarrow \rho r + (1-\rho)\hat{g} \odot \hat{g}$$

$$\Delta\theta = -\frac{\epsilon}{\delta+\sqrt{r}} \odot \hat{g}$$

$$\theta \leftarrow \theta + \Delta\theta$$

优点：

相比于 AdaGrad,这种方法很好的解决了深度学习中过早结束的问题
适合处理非平稳目标，对于 RNN 效果很好

缺点：

又引入了新的超参，衰减系数ρ
依然依赖于全局学习速率

RMSProp with Nesterov Momentum

当然，也有将 RMSProp 和 Nesterov Momentum 结合起来的

具体实现:

需要:全局学习速率 ϵ, 初始参数 θ, 初始速率 v，动量衰减系数 α，梯度累计量衰减速率ρ
中间变量: 梯度累计量 r(初始化为 0)
每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差,更新 r,在根据 r 和梯度计算参数更新量:

$$\tilde{\theta} \leftarrow \theta + \alpha v$$

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_{\tilde{\theta}} \sum_i L(f(x_i;\tilde{\theta}), y_i)$$

$$r \leftarrow \rho r + (1-\rho)\hat{g} \odot \hat{g}$$

$$v \leftarrow \alpha v - \frac{\varepsilon}{\sqrt{r}} \odot \hat{g}$$

$$\theta \leftarrow \theta + v$$

Adam

Adam(Adaptive Moment Estimation)本质上是带有动量项的 RMSprop，它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。Adam 的优点主要在于经过偏置校正后，每一次迭代学习率都有个确定范围，使得参数比较平稳。

具体实现:

需要:步进值 ϵ, 初始参数 θ, 数值稳定量 δ，一阶动量衰减系数ρ_1，二阶动量衰减系数ρ_2
其中几个取值一般为： δ =10−8, ρ_1,=0.9, ρ_2=0.999
中间变量：一阶动量 s，二阶动量 r,都初始化为 0
每步迭代过程:
1. 从训练集中的随机抽取一批容量为 m 的样本$\{x_1,\cdots,x_m\}$,以及相关的输出y_i
2. 计算梯度和误差,更新 r,在根据 r 和梯度计算参数更新量:

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_{\theta} \sum_i L(f(x_i;\theta), y_i)$$

$$s \leftarrow \rho_1 s + (1-\rho_1) g$$

$$r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$$

$$\hat{s} \leftarrow \frac{r}{1-\rho_1}$$

$$\hat{r} \leftarrow \frac{r}{1-\rho_2}$$

$$\theta \leftarrow \theta + \Delta\theta$$

参考：

http://blog.csdn.net/u014595019/article/details/52989301
http://www.360doc.com/content/17/0323/08/1489589_639370019.shtml

PCA，LDA，TSNE

有序数组的交集（要有代码）