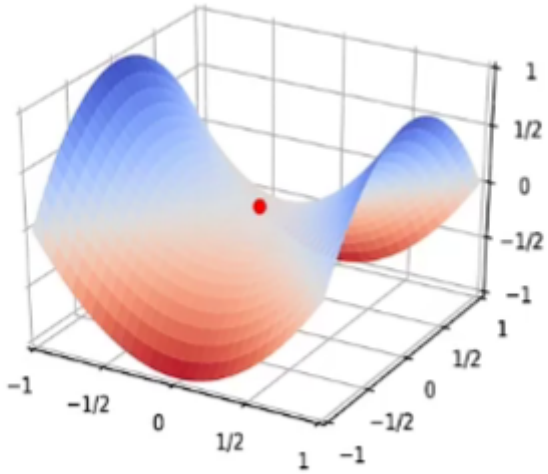
网络优化与正则化

网络优化的难点

1. 结构差异大
    - 没有通用的优化算法
    - 超参数多
2. 非凸优化
    - 参数初始化
    - 逃离局部最优或鞍点
3. 梯度消失(爆炸)问题

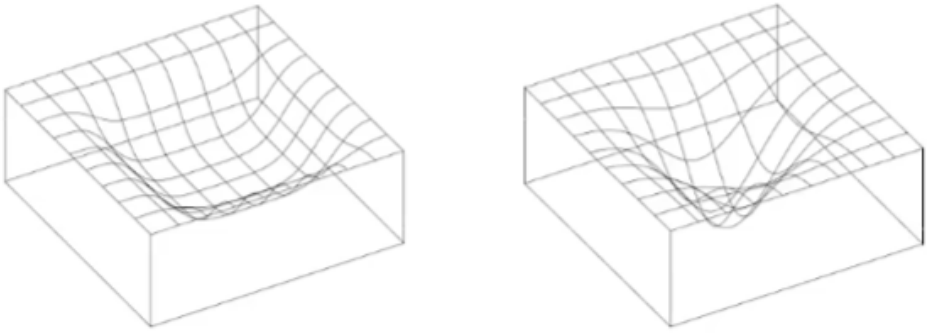
高维空间的非凸优化问题

1. 鞍点(Saddle Point) :高维空间中,在某些维度是最低点,在有些维度并不是.



2. 平坦最小值**(Flat Minima)**
    - 一个平坦最小值的邻域内,所有点对应的训练损失都比较接近

- 大部分的局部最小解是等价的
- 局部最小解对应的训练损失都可能非常接近于全局最小解对应的训练损失.



(a) 平坦最小值　　(b) 尖锐最小值

神经网络优化的改善方法

1. 更有效的优化算法来提高优化方法的效率和稳定性.
    - 动态学习率调整
    - 梯度估计修正
2. 更好的参数初始化、数据预处理方法来提高优化效率.
3. 修改网络结构来得到更好的优化地形.
    - 好的优化地形通常比较光滑
    - 使用ReLU激活函数、残差连接、逐层归一化等
4. 使用更好的超参数优化方法.

优化算法

优化算法:小批量随机梯度下降(MiniBatch)

1. 选取K个训练样本 $\{x^{(k)}, y^{(k)}\}_{k=1}^{K}$,计算偏导数

$$\mathfrak{g}_t(\theta) = \frac{1}{K} \sum_{(x,y)\in\delta_t} \frac{\partial \mathcal{L}(y, f(x;\theta))}{\partial \theta} \tag{1}$$

2. 定义梯度

$$g_t \overset{\Delta}{=} \mathfrak{g}_t(\theta t-1) \tag{2}$$

3. 更新参数

$$\theta_t \leftarrow \theta_{t-1} - \alpha g_t \tag{3}$$

> 几个关键因素:
>
> - 小批量样本数量
> - 梯度
> - 学习率

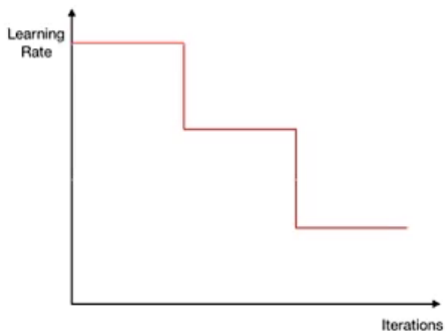
批量大小

批量大小不会影响随机梯度的期望,但会影响随机梯度的方差.

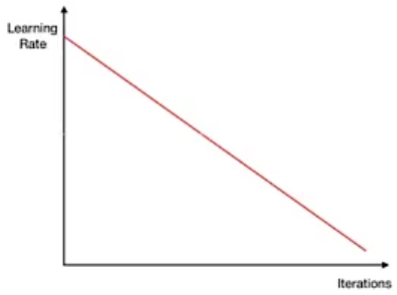
- 批量越大,随机梯度的方差越小,引入的噪声也越小,训练越稳定,可以设置较大的学习率
- 批量较小,需要设置较小的学习率,否则模型会不收敛.

学习率衰减

1. 梯级衰减



2. 线性衰减



3. 逆时衰减

$$\alpha_t = \alpha_0 \frac{1}{1+\beta \times t} \tag{4}$$

4. 指数衰减

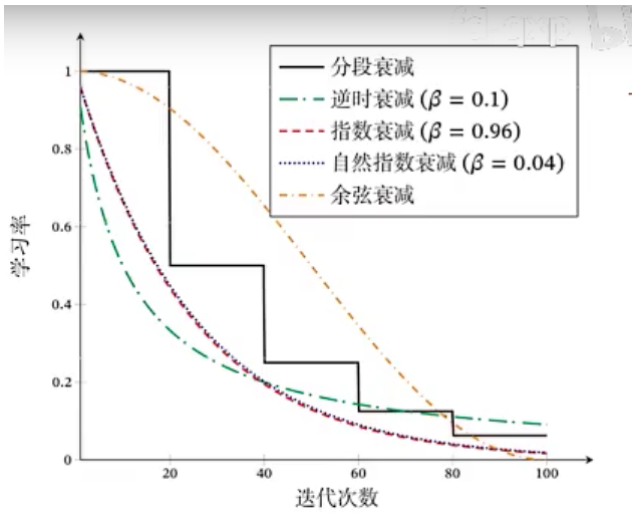
$$\alpha_t = \alpha_0 \beta^t \tag{5}$$

5. 自然指数衰减

$$\alpha_t = \alpha_0 \exp(-\beta \times t) \tag{6}$$

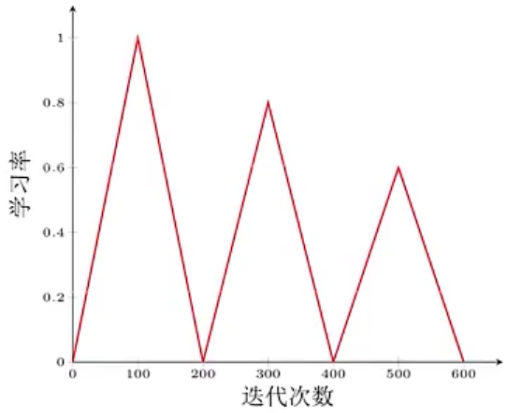
6. 余弦衰减

$$\alpha_t = \frac{1}{2}\alpha_0(1+\cos(\frac{t\pi}{T})) \tag{7}$$

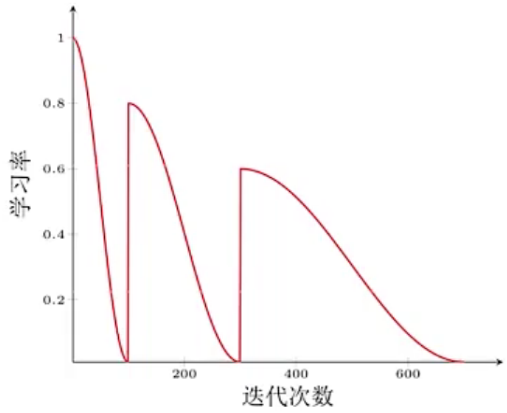


周期性学习率衰减

就是学习率不是单调递减的,而是周期性变化的(但整体上是衰减的).



(a) 三角循环学习率



(b) 带热重启的余弦衰减

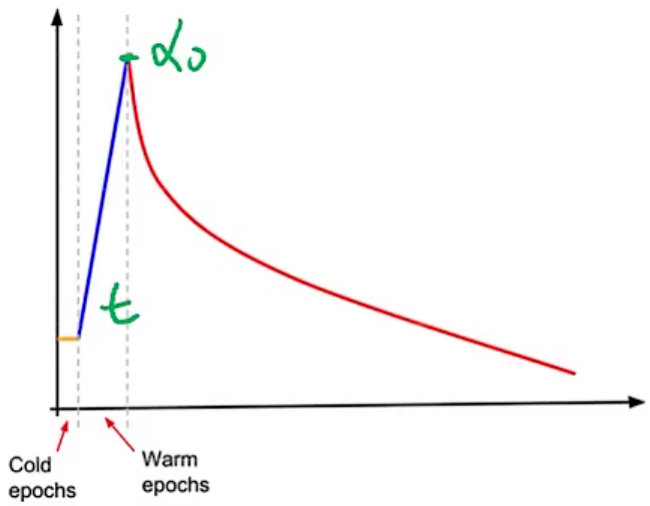
其他学习率调整方法

- 增大批量大小

前面讲过了批量和学习率之间的关系,增大批量大小,相对的,就是变相减小了学习率.

- 学习率预热

设定一个学习率,设定一个预热的epoch,在这个地方,学习率增大,然后之后学习率在减小.



自适应学习率

针对α

1. Adagrad

$$\begin{aligned}\Delta\theta_t &= -\frac{\alpha}{\sqrt{G_t+\epsilon}}\odot g_t \\ G_t &= \sum_{\tau=1}^{t} g_\tau \odot g_\tau\end{aligned} \tag{8}$$

其中g_τ表示历史的梯度,即随着不断的迭代,G_t会不断的增大,则实现了学习率衰减.

2. RMSprop

$$\begin{aligned}\Delta\theta_t &= -\frac{\alpha}{\sqrt{G_t+\epsilon}}\odot g_t \\ G_t &= \beta G_{t-1} + (1-\beta) g_t \odot g_t \\ &= (1-\beta)\sum_{\tau=1}^{t}\beta^{t-\tau} g_\tau \odot g_\tau\end{aligned} \tag{9}$$

将G_{t-1}不断迭代下来就得到最后的式子

RMSprop其实就是Adagrad的指数加权的移动平均.

优势:Adagrad是一直累加的,有可能一开始太大,很有可能直接变为0,而RMSprop由于指数加权移动平均,不一定会一直增大,通常不会衰减到0.

3. Adadelta

$$\begin{aligned}\Delta\theta_t &= -\frac{\sqrt{\Delta X_{t-1}^2+\epsilon}}{\sqrt{G_t+\epsilon}}\odot g_t \\ G_t &= \beta G_{t-1} + (1-\beta) g_t \odot g_t \\ &= (1-\beta)\sum_{\tau=1}^{t}\beta^{t-\tau} g_\tau \odot g_\tau \\ \Delta X_{t-1}^2 &= \beta_1 \Delta X_{t-2}^2 + (1-\beta_1)\Delta\theta_{t-1}\odot\Delta\theta_{t-1} \\ &= (1-\beta_1)\sum_{\tau=1}^{t}\beta_1^{t-\tau}\theta_\tau\odot\theta_\tau\end{aligned} \tag{10}$$

Adadelta比起RMSprop来说,就是不只是分母会变化,分子也会变化

优势:比起RMSprop来说,如果某一个方向的梯度很小,则优化的速度会很慢,而通过增大分子的该项,可以使慢的那一步走的快.

梯度估计修正

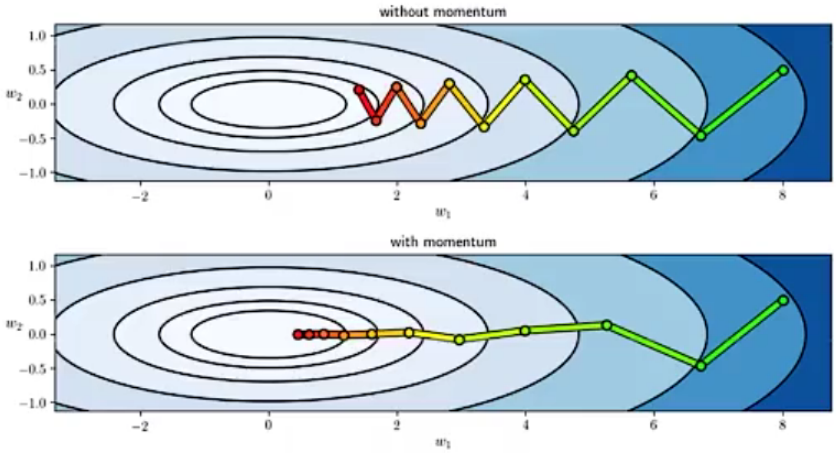
针对g_t

1. 动量法
   - 用之前累计的动量来代替真正的梯度

$$\begin{aligned}\Delta\theta_t &= \rho\Delta\theta_{t-1} - \alpha g_t \\ &= -\alpha\sum_{\tau=1}^{t}\rho^{t-\tau}g_\tau\end{aligned} \tag{11}$$

   其中ρ为动量因子.

   好处:若梯度在来回走的话,加权平均,可以让你在无效的方向上进行抵消.若与 上一次的方向一致,则可以加速训练,若与上一次不一致,可以在无效的方向上进行抵消.



2. Nesterov加速梯度

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha g_t(\theta_{t-1} + \rho\Delta\theta_{t-1}) \tag{12}$$

是对动量法的一点点优化(具体就是在更新的时候,梯度用更新后的带入).

3. Adam≈动量法+RMSprop
   - 先计算两个移动平均

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)g_t \tag{13}$$

$$G_t = \beta_2 G_{t-1} + (1-\beta_2) g_t \odot g_t$$

- 偏差修正

$$\begin{aligned} \hat{M}_t &= \frac{M_t}{1-\beta_2^t} \\ \hat{G}_t &= \frac{G_t}{1-\beta_2^t} \end{aligned} \tag{14}$$

- 更新

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}} \hat{M}_t \tag{15}$$

梯度截断

梯度截断是一种比较简单的启发式方法,按梯度的模限定在一个区间,当梯度的模小于或大于这个区间时就进行截断.

- 按值截断

$$g_t = \max(\min(g_t, b), a) \in [a, b] \tag{16}$$

- 按模截断

$$g_t = \frac{b}{||g_t||} g_t \tag{17}$$

参数初始化

1. 参数不能初始化为0.
    - 对称权重问题
2. 初始化方法

- 预训练初始化
- 随机初始化
- 固定值初始化
- 偏置项通常用0初始化.

随机初始化

1. Gaussian分布初始化
    - Gaussian初始化方法是最简单的初始化方法,参数从一个固定均值(比如0)和固定方差(比如0.01)的Gaussian分布进行随机初始化
2. 均匀分布初始化
    - 参数可以在区间 $[-r,r]$ 内采用均匀分布进行随机初始化.

范数保持性

- 一个M层的等宽线性网络,即矩阵维度不变,然后没有激活函数(为了方便分析).

$$y=W^{(l)}W^{(l-1)}\cdots W^{(1)}x \tag{18}$$

- 为了避免梯度消失或梯度爆炸,则我们希望误差项

$$||\delta^{(l-1)}||=||\delta^{(l)}||=||W^{(l)}\delta^{(l)}||\Rightarrow W^{(l)}(W^{(l)})^T=I \tag{19}$$

蓝色的部分:就是说希望比例是1,这样连乘,传播下去,可以防止梯度消失或者梯度爆炸.

红色的部分:就是说由$l-1$层的误差传播而来的.

绿色的部分:说明我们希望权重矩阵相乘是一个单位阵.

根据上述理论可以得到例子(邱锡鹏,神经网络与深度学习,7.5参数初始化中有计算过程,这里不赘述)

因此,有一个基于方差缩放的参数初始化.

初始化方法	激活函数	均匀分布[-r,r]	高斯分布N(0,σ^2)
Xavier初始化	logistics	$r=4\sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2=16\times\frac{2}{M_{l-1}+M_l}$
Xavier初始化	tanh	$r=\sqrt{\frac{6}{M_{l-1+M_l}}}$	$\sigma^2=\frac{2}{M_l-1+M_l}$
He初始化	ReLU	$r=\sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2=\frac{2}{M_{l-1}}$

其中M_l为l层神经元的个数.

还有一种更加直接的方式:正交初始化

1. 用均值为0,方差为1的高斯分布初始化一个矩阵
2. 将这个矩阵用奇异值分解得到两个正交矩阵(使用其中之一作为权重矩阵)

数据预处理

尺度不变性

尺度不变性是指:机器学习算法在缩放全部或部分特征后不影响学习和预测,则称其有尺度不变性(KNN不具有)

举个例子!

若某一层是这样的函数,其中$x_1\in[0,1],x_2\in[0,1000]$

$$y=w_1x_1+w_2x_2+b \tag{20}$$

那么w_1,w_2若服从一个分布,那么将很难初始化.

规范化

1. 最小最大值规范化:

$$\hat{x}^{(n)} = \frac{x^{(n)} - \min_n(x^{(n)})}{\max_n(x^{(n)}) - \min_n(x^{(n)})} \in [0,1] \tag{21}$$

2. 标准化:

$$\hat{x}^{(n)} = \frac{x^{(n)} - \mu}{\sigma} \tag{22}$$

3. PCA白化

标准化有一个弊端,就是相关性,如果用PCA,则可以得到没有相关性的新的特征. 但是由于PCA在实际应用中算法复杂度较高,一般使用标准化.

逐层归一化

目的

1. 更好的尺度不变性
2. 更平滑的优化地形

规范化方法

1. 批量规范化(Batch Normalization,BN)
2. 层规范化(Layer Normalization)
3. 权重规范化(Weight Normalization)
4. 局部响应规范化(Local Response Normalization, LRN)

这里主要讲批量规范化和层规范化

批量规范化(也就是BN层)

第l层网络为

$$a^{(l)} = f(z^{(l)}) = f(Wa^{(l-1)} + b) \tag{23}$$

给定一个包含K个样本的小批量样本集合,计算均值和方差:

$$\begin{aligned} \mu_{\mathcal{B}} &= \frac{1}{K}\sum_{k=1}^{K} z^{(k,l)} \\ \sigma_{\mathcal{B}}^2 &= \frac{1}{K}\sum_{k=1}^{K}(z^{(k,l)} - \mu_{\mathcal{B}}) \odot (z^{(k,l)} - \mu_{\mathcal{B}}) \end{aligned} \tag{24}$$

批量规范化:

$$\hat{z}^{(l)} = \frac{z^{(l)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \odot \gamma + \beta \tag{25}$$

其中γ和β是要学习的参数,那么$\gamma = \sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}, \beta = \mu_{\mathcal{B}}$,这样的话,就是不规范化,由于是学习的参数,所以,可以自动的判断标准化成什么样或者不标准化.

优点:

1 提高优化效率

2 隐形的正则化

在训练的时候,神经网络对一个样本的预测不仅和样本自身相关,也和同一批中的其他样本相关.由于在选择批次具有随机性.因此神经网络不会过拟合到一个样本上,从而提高样本的泛化能力.

缺点:

1 小批量样本的数量不能太小

2 无法应用到循环神经网络

> BN层为什么在训练和测试时不一样?(即model.train(),和model.eval()对BN层生效)
>
> 训练时,采用滑动来计算平均值和方差,在测试时,直接拿训练集的均值和方差使用.

层规范化

第l层的神经元净输入为$z^{(l)}$

$$\begin{aligned}\mu^{(l)} &= \frac{1}{M_l}\sum_{i=1}^{M_l} z_i^{(l)} \\ \sigma^{(l)^2} &= \frac{1}{M_l}\sum_{i=1}^{M_l}(z_i^{(l)} - \mu^{(l)})^2\end{aligned} \tag{26}$$

层规范化定义:

$$\begin{aligned}\hat{z}^{(l)} &= \frac{z^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)^2} + \epsilon}} \odot \gamma + \beta \\ &\overset{\Delta}{=} LN_{\gamma,\beta}(z^{(l)})\end{aligned} \tag{27}$$

批量规范化VS层规范化

- BN层是取不同的样本的同一个通道的特征做一个归一化,逐特征进行归一化
- LN层是取同一个样本,不同的通道的特征做一个归一化,逐个样本进行归一化

超参数优化

超参数:

1. 层数
2. 每层的神经元个数
3. 激活函数
4. 学习率
5. 正则化系数
6. batch_size

优化方法:

1. 网格搜索
2. 随机搜索
3. 贝叶斯优化
4. 动态资源分配
5. 神经架构搜索

正则化

1. 早停法(Early-Stop)

我们使用一个验证集来测试每一次迭代的参数在验证集上是否最优.如果验证集的错误率不在下降,就停止迭代.

2. 权重衰减

通过限制权重的取值范围来干扰随机化过程,降低模型能力.

在每次参数更新时,引入一个衰减系数β

$$\theta_t \leftarrow (1-\beta)\theta_{t-1} - \alpha g_t \tag{28}$$

3. Dropout

神经层 $y=f(Wx+b)$

引入一个遮掩函数使得

$$\begin{aligned} y&=f(Wmask(x)+b)\\ mask(x)&=m\odot x\\ m&\in\{0,1\}^D \end{aligned} \tag{29}$$

其中m由贝努利分布随机生成.

> 通过Dropout层之后,训练和测试的模型是不一样的.
>
> 在测试的时候,$mask(x)=px$,是直接将整个x的活性值降低到p倍.

集成学习的解释:

每做一次dropout层,相当于从原始的网络中采样到一个子网络.如果一个神经网络有n个神经元,那么总共可以采样出2^n个子网络.

贝叶斯学习的解释:

$$\begin{aligned} \mathbb{E}_{q(\theta)}[y]&=\int_q f(x;\theta)q(\theta)d\theta\\ &\approx\frac{1}{M}\sum_{m=1}^{M}f(x,\theta_m) \end{aligned} \tag{30}$$

上述说明,应用了dropout层相当于取了一次平均.

循环神经网络不能直接使用Dropout,这样会损害循环神经网络在时间维度上记忆能力.要在循环神经网络上使用的话,需要使用变分的Dropout,就是每个时刻使用的随机掩码需要一样.

4. l_1 和 $l2$ 正则化

优化问题可以写为:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y^{(n)}, f(x^{(n)}; \theta) + \lambda l_p(\theta)) \tag{31}$$

> **Note**:
>
> 在标准的随机梯度下降中,l_2正则化与权重衰减是等价的,参数更新的公式是一样的.($\beta = \lambda$);
>
> 但在复杂的优化方法中(如Adam)中,权重衰减与l_2正则化不等价.

5. 数据增强

1 人工构造新的样本来增加训练了数据的数量和多样性

A 图像数据

- 旋转:将图像按顺时针或逆时针方向随机旋转一定角度
- 翻转:将图像沿水平或垂直方向随机翻转一定的角度
- 缩放:将图形放大或缩小一定比例
- 平移:将图像沿水平或垂直方法平移一定步长
- 加噪声:加入随机噪声

B 文本数据

- 词汇替换
- 回译(Back Translation):例如:将中文翻译成英文,再将英文翻译回中文
- 随机编译噪声
    - 增删改查
    - 句子乱序

2 标签平滑(Label Smoothing)

在输出标签中添加噪声来避免模型过拟合.

一个样本x的标签一般用one-hot表示

$$y = [0, \ldots, 0, 1, 0, \ldots, 0]^T \tag{32}$$

引入一个噪声对标签进行平滑,即假设样本以ϵ的概率为其他类.平滑后的标签为

$$\hat{y} = [\frac{\epsilon}{K-1}, \ldots, \frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \ldots, \frac{\epsilon}{K-1}]^T \tag{33}$$