# 一 mini batch梯度下降

## 为了解决什么问题?

为了提高训练速度.

具体实现如图:



将500万个样本,分成5000份, 每份1000个样本.这样是为一个mini batch, 每次对一个mini batch做梯度下降更新.



上图右侧规定了几个符号. X(i)是第i个样本. Z[l]是第l层的神经元的z值, X{t},Y{t}表示第t个mini batch的样本集.

## Minibatch的训练方法:

下图展示了一个epoch的训练顺序.

首先, 最外层轮询5000份mini batch

然后, 里层,每次执行1000个单位的Loss计算及bp分解以及权值更新(以向量化运行).

其中Loss公式为(带有正则化):





# 二 mini batch深入理解

对比batch梯度下降和minibatch 梯度下降, Loss收敛如下图.

1. Batch的收敛平滑, 如果出现不收敛可能是学习率太大的问题.
2. Minibatch的收敛有波动,但是趋势是下降的.波动原因是不同的batch的Loss的差异.



miniBatch的batchsize取值的影响:

1. 取值为样本数m,minibatch梯度下降就是batch梯度下降.

收敛如下图蓝色线段

1. 取值为1, 则为随机梯度下降.

收敛如下图紫色线段,不能在极值点收敛,而是在极值点震荡.



Batch梯度下降和mini batch梯度下降的选择.

1. M小于2000时(认为数据集太小),使用Batch效果好.
2. 数据集大的情况,采用mini batch效果好,但也要注意GPU/CPU是否有足够的内存能够让你用.
3. Batchsize**尽量选择2的次方.**

# 三 指数加权平均

介绍指数加权平均的目的是, 下面一些算法比梯度下降法快, 但它们的基础是指数加权平均.

## 3.1 指数加权平均是什么

**指数加权平均**(exponentially weighted averges)，也叫指数加权移动平均，是一种常用的序列数据处理方式。

它的计算公式如下：



其中，

* θt：为第 t 天的实际观察值，
* Vt: 是要代替 θt 的估计值，也就是第 t 天的指数加权平均值，
* β： 为 V\_{t-1} 的权重，是可调节的超参。( 0 < β < 1 )

比如例子

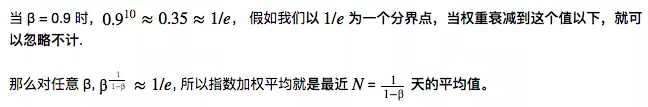
1. 蓝色点是过去一年的温度值.
2. 根据这个温度值,带入上面指数加权平均公式,就能求出加权平均之后的一种个平均值.用红色线表示.
3. 红色线(加权平均值)更加平滑,能够反映出温度趋势.



## 3.2 指数加权平均的简化及特点应用

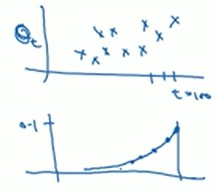
* 1. 展开如下式.
  2. 考虑一个指数特性. 必然有一个**ax**指数在**x**取到**X**时,使得**ax**约等于1/e.
  3. 实际使用过程中,可以忽略那些小于1/e的项.这样可以做到简化.
  4. 特别的, 得到一个简化的推论;

加权平均就是最近天的均值.



另一中理解加权均值

如下图, 将β和θ展开后,输出v可以认为是两者做的卷积.



## 3.3 指数加权平均推论应用

在推论下:

加权平均就是最近天的均值.

当β=0.9时, υ取的最近10天的均值.

当β=0.98时, υ取的最近50天的均值.

当β=0.5时, υ取的最近2天的均值.如下图黄色曲线,由于是最近两天的均值,波动很大.





## 3.4 指数加权平均的前期误差修正.

指数加权平均会有一个问题:

1. 按如下公式.
   1. 当v0=0时, **v1=0.02θ1, v1<<θ1**
   2. V2=0.0196θ1=0.02θ2,也远远小于θ1和θ2.会导致如下曲线图的异常.
      1. 红色曲线是β=0.9的
      2. 紫色曲线是β=0.98的
      3. 绿色曲线是**期望(正确的,目标的)β**=0.98的, 紫色和绿色有差距,但是在中后期会重合.





为了解决这个偏差问题.引入一个修改方法:

**Vt修改=vt原始/(1-βt) , t为天数.**



1. 这个修改可以弥补t较小时的偏差
2. 在t较大的时候,分母**(1-βt)**接近1,不会导致影响后期行为.

# 四 动量梯度下降法

为了解决什么问题?

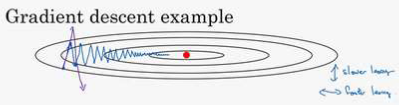
它比标准梯度下降法快很多.(标准梯度下降的学习率不能太大,避免发散).

是什么?

本质利用在梯度上应用指数加权平均,用该”指数加权平均”的梯度更新权重.

和标准梯度下降法的对比:

1. 蓝色描述的是标准梯度下降法.无论minibatch,batch都会有波动.也需要太多下降法的lr不能太大,太大就会导致发散(如紫色箭头的波动).
2. 期望在y轴上学习慢点,在x轴上加快学习.

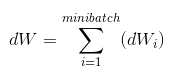
​

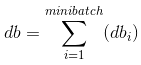
目的是加速横着收敛,减小纵轴震荡.

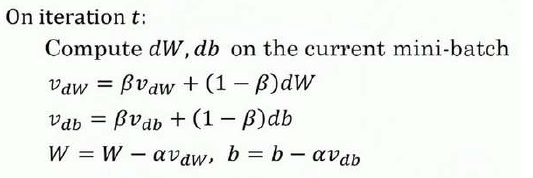
引入动量梯度下降法, 就是在梯度下降法上应用指数加权平均算法.

具体计算是:

在一次minibatch中. 计算这次minibatch的dW和db.

​

​

​

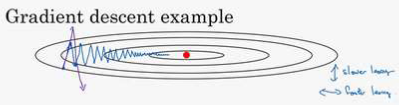
1. **有两个超参数α和β.β推荐值是0.9,采用最近10次的平均.**
2. **有些版本中(1-β)项退化成1.**

https://img-blog.csdnimg.cn/20190409190503263.png​

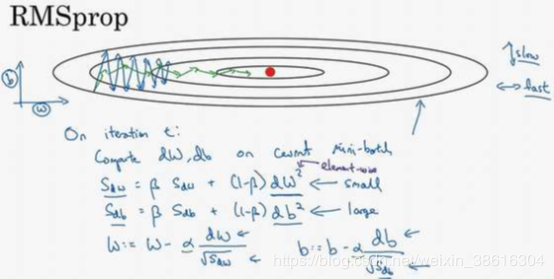
1. 物理效果上看, **近10次(β取0,9)的平均中,纵轴值有正有负,** 导致本次纵轴的值接近0. 因此纵轴上震荡不明显. 在横轴上, 向右的值一致在增加, 所以会加速向右的梯度更新效果

# 五 RMSprop

为了解决如下在纵轴上的梯度下降的震荡

​

引入RMSprop算法(root mean square prop).具体思路如下:

​

https://img-blog.csdnimg.cn/20190409190529922.png​

https://img-blog.csdnimg.cn/20190409190540973.png​

我们不妨可以设置W描述横轴, b描述纵轴.

b方向(纵轴)变化大,因此(db)2大, 因此Sdb大, 导致b的更新小. 因此b方向(纵轴)震荡小.

同理W方向(横轴). 更新量大.加速梯度下降.

## 5.1 RMSprop和momentum

相似点:

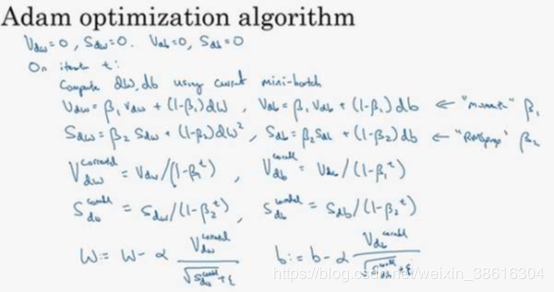
1. 都能消除(削弱)”纵轴”的震荡(波动,摆动).
2. 允许使用一个较大的学习率(某个方向,纵轴,的摆动被削弱了).
3. 两者结合会有一个更好的表现.

# 六 Adam优化算法

梯度优化算法有一个问题, 某些算法不能应用于其他的网络.

RMSprop和Adam优化算法是可以应用于其他网络的,并且表现还不错的.

**Adam是把Momentum和RMSprop结合在一起的算法.**

​

具体步骤是:

1. 先据minibatch 计算dW,db.
2. 计算”Momentum”下的Vdw和Vdb及修正偏差.

https://img-blog.csdnimg.cn/20190409190617296.png​

https://img-blog.csdnimg.cn/20190409190617309.png​

https://img-blog.csdnimg.cn/20190409190617311.png​

https://img-blog.csdnimg.cn/20190409190617321.png​

1. 计算RMSprop下的Sdw和Sdb

https://img-blog.csdnimg.cn/20190409190617338.png​

https://img-blog.csdnimg.cn/20190409190617341.png​

https://img-blog.csdnimg.cn/20190409190617350.png​

https://img-blog.csdnimg.cn/20190409190617363.png​

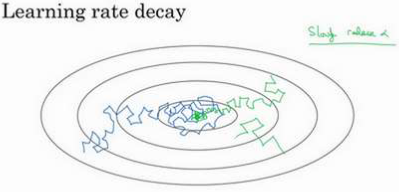
1. 组合成Adam算法的更新量:

https://img-blog.csdnimg.cn/20190409190617366.png​

https://img-blog.csdnimg.cn/20190409190617378.png​

# 七 学习率衰减

为了解决什么问题?

​

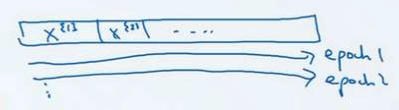
1. 图中,蓝线表示学习率(Lr)不变的情况下收敛的”轨迹”. 在极小值附件有震荡.
2. 震荡的原因是, 采用minibatch来做的权重更新, 每组minibatch的分布不同(会有一些噪声).
3. 震荡的另一个原因是: lr是固定值.
4. 绿线是衰减LR的收敛轨迹.

Lr的特点:

1. 在训练初期, lr需要较大步伐.
2. 在训练末期, lr需要较小的步伐.

结合训练的过程.

1. 下图X={x1,.., Xm}是训练集.
2. 训练集是按照minibatch的方式, 每次用一组minibatch来更新权重.
3. 轮询(遍历,使用,应用)一次所有数据集就叫一个epoch.
4. 需要训练多次(epochNum次数).
5. 每次epoch的lr应该是衰减的.

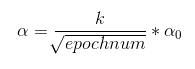
​

**一个可用的lr衰减公式**:

[[RH(1]](" \l "_msocom_1) https://img-blog.csdnimg.cn/20190418185040618.png​

1. 有三个参数, a0, decayrate, epochnum.
2. a0(初值)可以设置为0.2
3. decayrate可为常数1.
4. Epochnum是1,2,…
5. 对于第一代(第一次遍历) epochnum=1,则 a=0.1.

**其他lr衰减公式:**

[[RH(2]](" \l "_msocom_2) ​

[[RH(3]](" \l "_msocom_3) https://img-blog.csdnimg.cn/20190418185048225.png​

上式t为minibatch 数值.

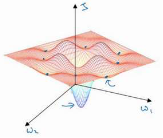
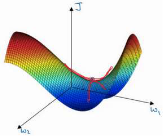
**离散Lr衰减.**

比如和global\_step挂钩的Lr衰减函数. 设置一个以global\_step为输入变量, Lr为输出变量的离散分段函数.

# 八 understanding 局部最优解

一个不合适的观念. 之前人们会认为训练会陷入到局部最优解. 但是实际上对于高维网络(比较深的网络)往往难以陷入局部最优解(如左下图)

比如, 有2万维度权重的网络. 难以让2万个维度方向上的导数都是0. 这个概率还是比较低的. 更常见的是陷入鞍点.(如右下图).

​​

鞍点会有什么问题?

鞍点附件的导数为0(某些维度上是凸函数, 某些维度上是凹函数), 在鞍点附近的更新会更慢.

Momentum, RMSprop, Adam算法能够很好地解决掉鞍点更新慢的问题.

 [[RH(1]](#_msoanchor_1)\alpha =\frac{1}{1+decayrate\*epochnum}\*\alpha\_0

 [[RH(2]](#_msoanchor_2)\alpha =\frac{k}{\sqrt[]{epochnum}}\*\alpha\_0

 [[RH(3]](#_msoanchor_3)\alpha =\frac{k}{\sqrt[]{t}}\*\alpha\_0