# Adam优化算法

* 算法目的：减小计算机资源消耗，加速学习速度

Adam算法是Momentum算法和RMSProp算法共同作用的一种方法。

首先回归一下梯度下降法

## 梯度下降法（GD）

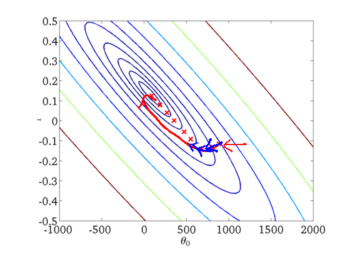
在应用机器学习算法时，我们通常采用梯度下降法来对采用的算法进行训练。其实，常用的梯度下降法还具体包含有三种不同的形式，它们也各自有着不同的优缺点，分别有BGD、SGD、MBGD。

### 批量梯度下降法BGD（Batch Gradient Descent）

使用批量梯度下降法，**每一次**更新参数都将会使用到**整个训练数据集**，如果训练数据非常多的话，是非常耗时的。如下表，每次更新都会用到所有的训练样本。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 特征数  *n* | | | | | 标签 |
| 1 | 2 | 3 | ... | *n* | Y |
| 样本数  *m* | 1 |  |  |  |  |  | 1 |
| 2 |  |  |  |  |  | 1 |
| 3 |  |  |  |  |  | 0 |
| .  .  . |  |  |  |  |  | .  .  . |
| *m* |  |  |  |  |  | 0 |

典型的BGD收敛图如下：



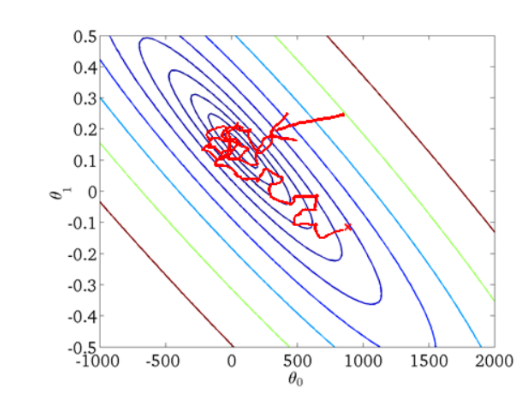
### 随机梯度下降法SGD（Stochastic Gradient Descent）

由于批梯度下降每一次更新参数都要用到所有的样本数，所以训练速度会随着样本数量的增加而变得**非常缓慢**。随机梯度下降正是为了解决这个办法而提出的，它**每一次更新参数**只利用随机选取的**单个训练样本**得到对应的梯度，从而更新参数。

随机梯度下降是通过每个样本来迭代更新一次，对比上面的批量梯度下降，迭代一次需要用到所有训练样本（**往往如今真实问题训练数据都是非常巨大**），一次迭代不可能最优，如果迭代10次的话就需要遍历训练样本10次。**但是，SGD伴随的一个问题是噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。**如下表，每次更新都会用到随机抽取的训练样本，例如某次更新参数抽取到了第3个样本，并根据3号样本进行参数更新。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 特征数  *n* | | | | | 标签 |
| 1 | 2 | 3 | ... | *n* | Y |
| 样本数  *m* | 1 |  |  |  |  |  | 1 |
| 2 |  |  |  |  |  | 1 |
| 3 |  |  |  |  |  | 0 |
| .  .  . |  |  |  |  |  | .  .  . |
| *m* |  |  |  |  |  | 0 |

典型的SGD收敛图如下：



我们可以从图中看出SGD迭代的次数较多，在解空间的搜索过程看起来很盲目。但是大体上是往着最优值方向移动，总花费时间肯定更少。

### 小批量梯度下降法MBGD（Mini-batch Gradient Descent）

小批量梯度下降法是MGD和SGD的折中方法。**算法的训练过程比较快，而且也要保证最终参数训练的准确率，**这正是小批量梯度下降法的初衷。

**每一次更新随机**在训练集中选取一个mini-batch，每个mini-batch包含p个样本，（p为**Mini-batch的Size；**p<n，n为训练集样本总数；**不同的任务，不同的模型，Mini-batch的Size完全不同**），在mini-batch里计算梯度然后更新参数。相当于，像SGD一样随机抽取一部分样本（只是SGD抽取1个，MBGD抽取p个），将这部分样本当做一个batch进行类似于BGD的操作。如下表，每次更新都会用到随机抽取的训练样本集mini-batch，例如mini-batch size为2，某次更新参数抽取到了第2、3个样本，形成一个mini-batch，并进行参数更新。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 特征数  *n* | | | | | 标签 |
| 1 | 2 | 3 | ... | *n* | Y |
| 样本数  *m* | 1 |  |  |  |  |  | 1 |
| 2 |  |  |  |  |  | 1 |
| 3 |  |  |  |  |  | 0 |
| .  .  . |  |  |  |  |  | .  .  . |
| *m* |  |  |  |  |  | 0 |

## Optimizor in Neural Network

使用Momentum算法前先要了解移动指数加权平均

### 移动指数加权平均

**移动指数加权平均**就是根据同一个移动段内不同时间的数据对预测值的影响程度，分别给予不同的权数，然后再进行平均移动以预测未来值。假定给定一系列数据值x1, x2, x3, …, xn。那么，我们根据这些数据来拟合一条曲线，所得的值v1, v2…..就是如下的公式：

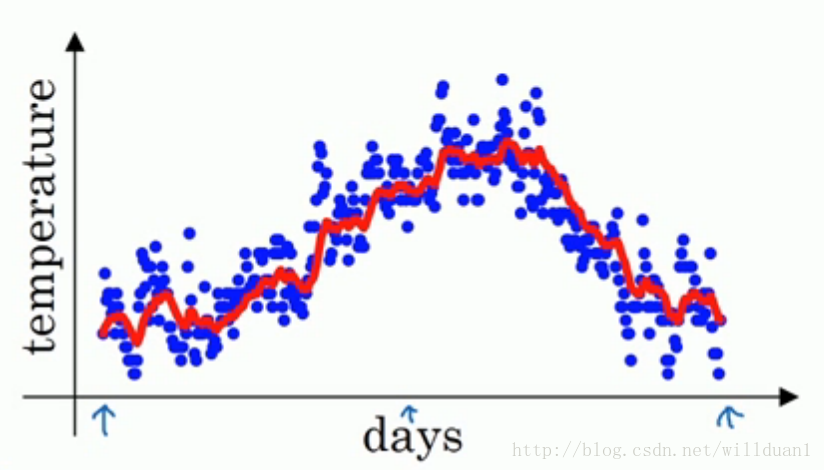
*v0＝0*

*v1=βv0+(1−β)x1*

*v2=βv1+(1−β)x2*

*.............*

其中，在上面的公式中，*β*等于历史值的加权率。如果把公式详细的展开，就会得到一个指数形式的公式，这里就不详细讲了。根据这个公式我们可以根据给定的数据，拟合出下图类似的一条比较平滑的曲线。



### Momentum算法

通常情况在训练深度神经网络的时候常用MBGD算法，虽然这种算法能够带来很好的**训练速度**，但是在到达最优点的时候**并不能够总是真正到达最优点**，而是在最优点附近徘徊。另一个缺点就是这种算法需要我们挑选一个**合适的学习率**，当我们采用小的学习率的时候，会导致网络在训练的时候收敛太慢；当我们采用大的学习率的时候，会导致在训练过程中优化的幅度跳过函数的范围，也就是可能跳过最优点。我们所希望的仅仅是网络在优化的时候网络的损失函数有一个很好的收敛速度同时又不至于摆动幅度太大。

**Momentum算法**可以解决我们所面临的问题，它主要是基于梯度的移动指数加权平均。假设在当前的迭代步骤第 *t* 步中，那么基于Momentum优化算法可以写成下面的公式：

*v*dw = *β*·*v*dw + (1−*β*)·d*W* (1)

*v*db *= β*·*v*db + (1−*β*)·d*b* (2)

*W* = *W* − *α*·*v*dw (3)

*b* = *b* − *α*·*v*db (4)

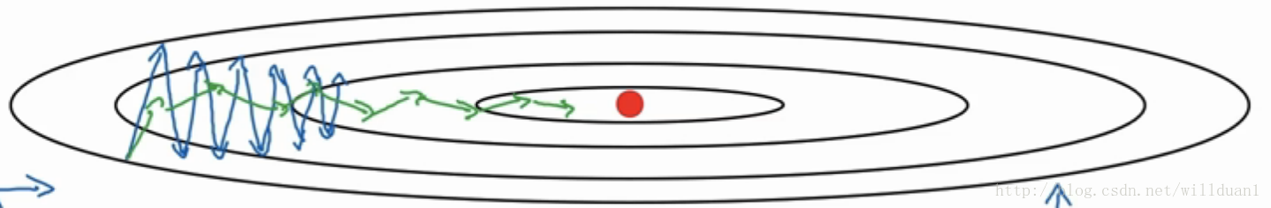
其中，在上面的公式中*v*dw和*v*db分别是损失函数在前 *t*−1 次迭代过程中累积的梯度动量，*β*是梯度累积的一个指数，这里我们一般设置值为0.9。类似于之前的移动指数加权平均中，算出来的“梯度值”类似于“每天的气温”，得到的“梯度动量值”类似于“每天的气温拟合值”。

所以Momentum Optimizor的主要思想就是利用了类似与**移动指数加权平均**的方法来对网络的参数进行平滑处理的，让梯度的波动幅度变得更小。

d*W*和d*b*分别是损失函数反向传播时候所求得的梯度，公式(3)和(4)是网络权重向量和偏置向量的更新公式，α是学习率。当我们使用Momentum优化算法的时候，可以解决mini-batch SGD优化算法更新幅度摆动大的问题，同时可以使得网络的收敛速度更快。

### RMS Prop算法

RMS Prop算法的全称叫 Root Mean Square Prop，是Geoffrey E. Hinton在Coursera课程中提出的一种优化算法，在上面的Momentum优化算法中，虽然初步解决了优化中摆动幅度大的问题。所谓的摆动幅度就是在优化中经过更新之后参数的变化范围，如下图所示，蓝色的为Momentum优化算法所走的路线，绿色的为RMSProp优化算法所走的路线。



为了进一步优化损失函数在更新中存在摆动幅度过大的问题，并且进一步加快函数的收敛速度，**RMS Prop算法**对权重 *W* 和偏置 *b* 的梯度使用了微分平方加权平均数。

其中，在第 *t* 轮迭代过程中，各个公式如下所示：

(1)

(2)

(3)

(4)

在上面的公式中和分别是损失函数在前 *t*−1 次迭代过程中累积的梯度动量，*β*是梯度累积的一个指数。与Momentum算法不同的是，RMS Prop算法对梯度计算了微分平方加权平均数。这种做法有利于消除了摆动幅度大的方向，用来修正摆动幅度，使得各个维度的摆动幅度都较小。另一方面也使得网络函数收敛更快。（比如当d*W*或者d*b*中有一个值比较大的时候，那么我们在更新权重或者偏置的时候除以它之前累积的梯度的平方根，这样就可以使得更新幅度变小）。为了防止分母为零，使用了一个很小的数值来进行平滑，一般取值为10−8。

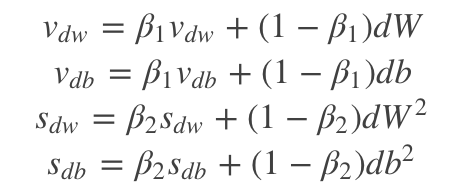
观察上面的公式可以看到，s是对梯度的平方做了一次平滑。在更新参数时，先用梯度除以梯度动量，相当于对梯度做了一次归一化。如果某个方向上梯度震荡很大，应该减小其步长；而震荡大，则这个方向的s也较大，除完之后，归一化的梯度就小了；如果某个方向上梯度震荡很小，应该增大其步长；而震荡小，则这个方向的s也较小，归一化的梯度就大了。因此，通过RMS prop，我们可以调整不同维度上的步长，加快收敛速度。

### Adam算法

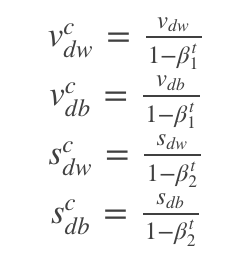
有了上面两种优化算法，一种可以使用类似于物理中的动量来累积梯度，另一种可以使得收敛速度更快同时使得波动的幅度更小。那么讲两种算法结合起来所取得的表现一定会更好。Adam（Adaptive Moment Estimation）算法是将Momentum算法和RMS Prop算法结合起来使用的一种算法，我们所使用的参数基本和上面讲的一致，在训练的最开始我们需要初始化梯度的累积量和平方累积量。

*v*dw = 0， *v*db = 0； *s*dw = 0, *s*db = 0

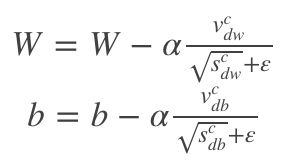
在第*t*次训练中，



由于移动指数平均在迭代开始的初期会导致和开始的值有较大的差异，所以我们需要对上面求得的几个值做偏差修正(correct)。



通过上面的公式，我们就可以求得在第 *t* 轮迭代过程中，参数梯度累积量的修正值，从而接下来就可以根据Momentum和RMS Prop算法的结合来对权重和偏置进行更新。



上面的所有步骤就是Momentum算法和RMS Prop算法结合起来从而形成Adam算法。Adam算法相当于先把原始梯度做一个指数加权平均，再做一次归一化处理，然后再更新梯度值。

在Adam算法中，参数 *β*1 所对应的就是Momentum算法中的 *β* 值，一般取0.9，参数 *β*2 所对应的就是RMSProp算法中的 *β* 值，一般我们取0.999，而是一个平滑项，我们一般取值为10−8，而学习率 *α* 则需要我们在训练的时候进行微调。

## 全新优化算法AdaBound

拳打Adam，脚踢SGD：北大提出全新优化算法AdaBound

<https://mp.weixin.qq.com/s?__biz=MzA3MzI4MjgzMw==&mid=2650757787&idx=1&sn=e5be45031edd8a6de06334177544a186&chksm=871a9ee5b06d17f3aab5e8252337f8639306a36423b0b880847396b437f98ef6b6facf7a0a6e&mpshare=1&scene=1&srcid=0416RPgmdrKZcooAN3jgKmAN#rd>

* 项目地址：<https://github.com/Luolc/AdaBound>

**论文：Adaptive Gradient Methods with Dynamic Bound of Learning Rate**

论文地址：<https://openreview.net/pdf?id=Bkg3g2R9FX>