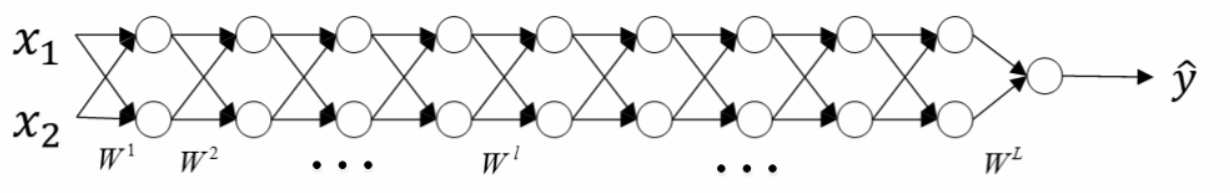
1. **梯度消失与梯度爆炸**

**意思是当训练一个层数非常多的神经网络时，计算得到的梯度可能非常小或非常大，甚至是指数级别的减小或增大。这样会让训练过程变得非常困难。**

**例如：假设一个多层的每层只包含两个神经元的深度神经网络模型，如下图所示：**



令激活函数为：，令b全部为零，则预测值Y帽有如下：

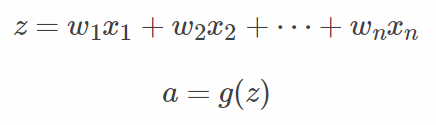


**会出现：数值爆炸：**如果各层权重W[l]的元素都稍大于1，例如1.5，则预测输出Y帽将正比于1.5L。L越大，Y帽越大，且呈指数型增长。

**数值消失：**如果各层权重W[l]的元素都稍小于1，例如0.5，则预测输出Y帽将正比于0.5L。L越大，Y帽越小，且呈指数型减小。

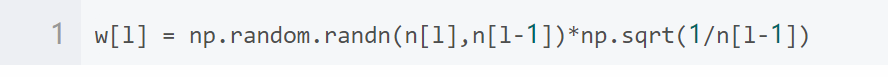
**解决梯度爆炸和梯度消失的方法-权重初始化：**

**例如，**以单个神经元为例，该层（l）的输入个数为n，其输出为：

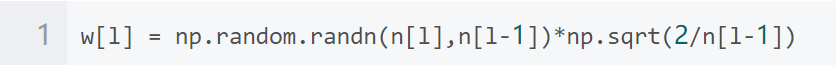


这里忽略了常数项b。为了让z不会过大或者过小，思路是让w与n有关，且n越大，w应该越小才好。这样能够保证z不会过大。

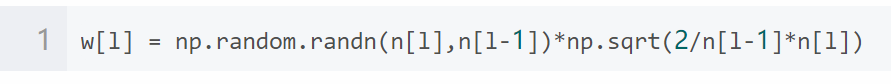
如果激活函数是tanh，初始化w时，令其方差为1/n。相应的python伪代码为：



如果激活函数是ReLU，初始化w时，一般令其方差为2/n。相应的python伪代码为：

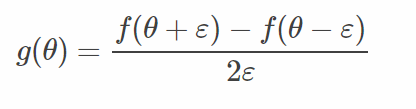


除此之外，Yoshua Bengio提出了另外一种初始化w的方法，相应的python伪代码为：



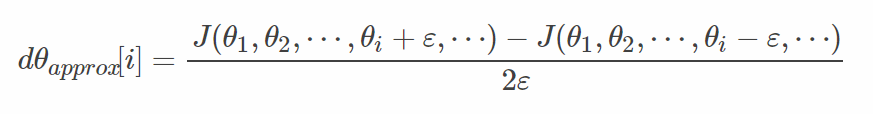
1. **梯度检查-验证反向传播过程中梯度下降算法是否正确。**

**求梯度值公式：**

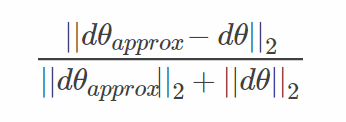


**第一步，分别将W[1],b[1],⋯,W[L],b[L]这些矩阵构造成一维向量，然后将这些一维向量组合起来构成一个更大的一维向量θ。这样cost function J(W[1],b[1],⋯,W[L],b[L])就可以表示成J(θ)。**

**第二步，将反向传播过程通过梯度下降算法得到的dW[1],db[1],⋯,dW[L],db[L]按照一样的顺序构造成一个一维向量dθ。dθ的维度与θ一致。**

**第三步，利用J(θ)对每个θi计算近似梯度，其值与反向传播算法得到的dθi相比较，检查是否一致。例如，对于第i个元素，近似梯度为：**

**第四步，计算完所有θi的近似梯度后，可以计算dθapprox与dθ的欧氏距离来比较二者的相似度。公式如下：**



**结论：如果欧氏距离越小，例如10−7，甚至更小，则表明dθapprox与dθ越接近，即反向梯度计算是正确的，没有bugs。如果欧氏距离较大，例如10−5，则表明梯度计算可能出现问题，需要再次检查是否有bugs存在。如果欧氏距离很大，例如10−3，甚至更大，则表明dθapprox与dθ差别很大，梯度下降计算过程有bugs，需要仔细检查。**

**梯度检查注意事项：**

**1.不要在整个训练过程中都进行梯度检查，仅仅作为debug使用。**

**2.如果梯度检查出现错误，找到对应出错的梯度，检查其推导是否出现错误。**

**3.注意不要忽略正则化项，计算近似梯度的时候要包括进去。**

**4.梯度检查时关闭dropout，检查完毕后再打开dropout。**

**5.随机初始化时运行梯度检查，经过一些训练后再进行梯度检查**

1. **几种重要的梯度下降优化算法**
2. **mini batch优化算法：**

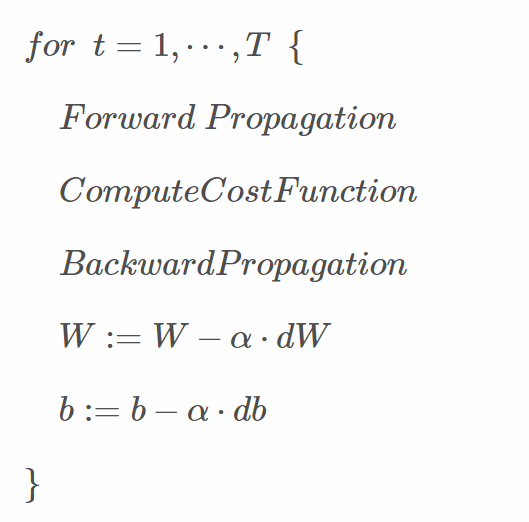
**解决问题：当样本数量达到百万级别时，训练速度往往会很慢。**

**解决方法：将很大的数据集等分成小份的数据集进行处理，一般是2的次方的个数会比较好，因为其符合计算机的二进制原理。**

**具体步骤：**

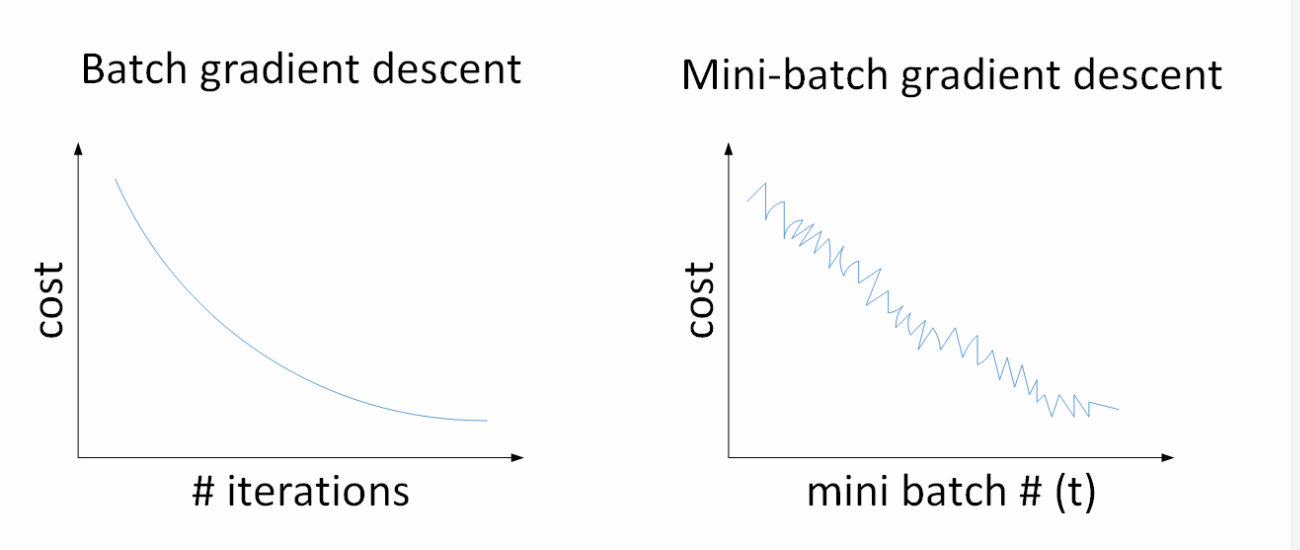
**假设总的训练样本个数m=5000000，其维度为(nx,m)。将其分成5000个子集，每个mini-batch含有1000个样本。我们将每个mini-batch记为X{t}，其维度为(nx,1000)。相应的每个mini-batch的输出记为Y{t}，其维度为(1,1000)，且t=1,2,⋯,5000。**

**训练代码：**



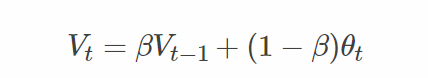
相当于进行了t次梯度下降，比没进行mini batch之前的batch算法更优。

以下为batch和mini batch的cost函数趋紧最优值的过程图：

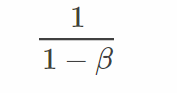


可以看到，采用batch算法的随着迭代次数增加，cost是不断减小的。而mini batch算法则是中间过程出现振荡，但总体呈下降趋势。之所以出现细微振荡的原因是不同的mini-batch之间是有差异的。例如可能第一个子集(X{1},Y{1})是好的子集，而第二个子集(X{2},Y{2})包含了一些噪声noise。出现细微振荡是正常的。

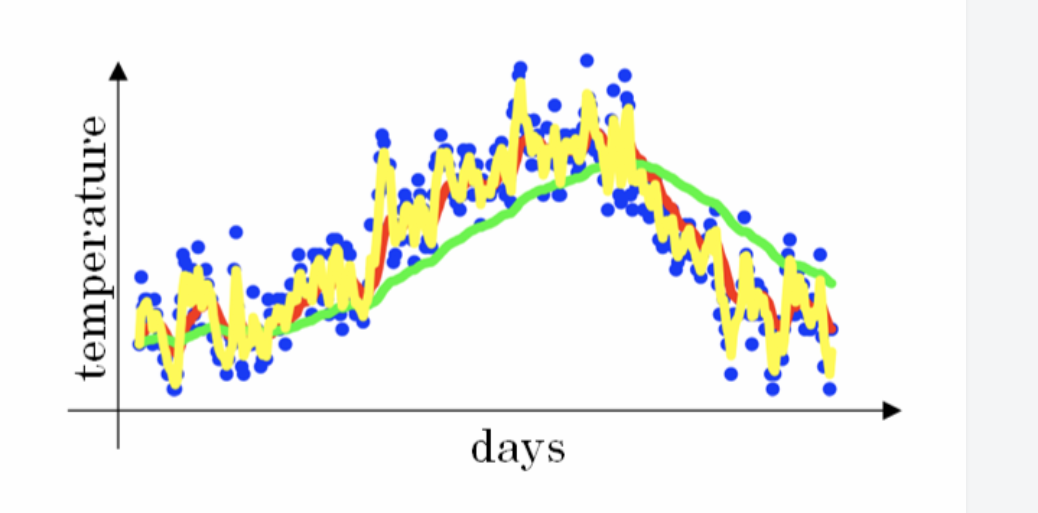
1. **加权平均法**



其中表示一个超参数，表示前一次的数据，表示本次的数据。

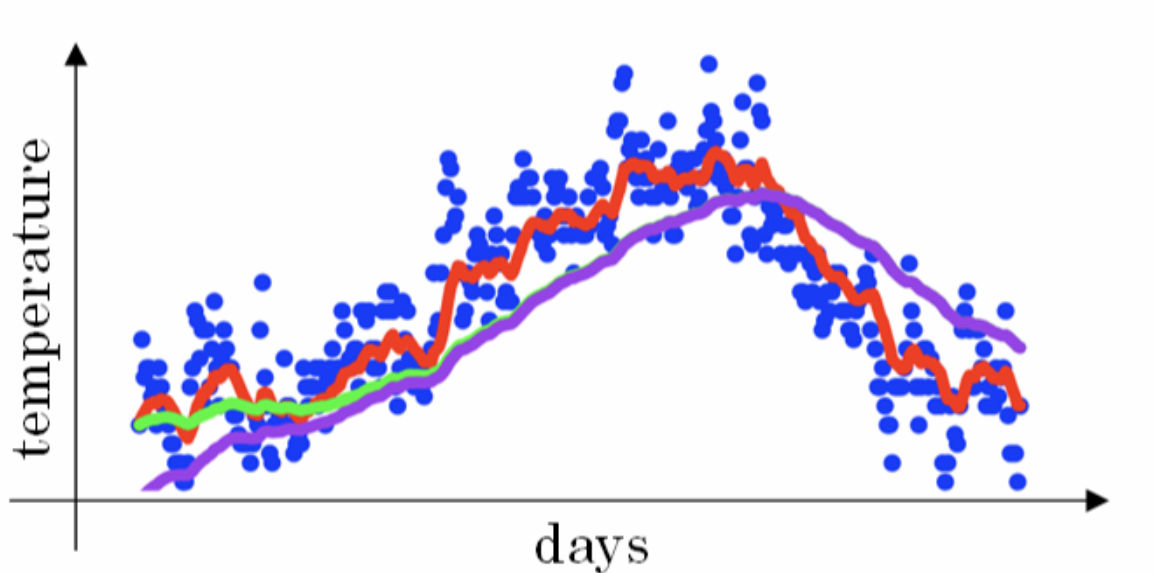
表示加权平均的次数。

**加权平均的作用：**



例如，当β=0.9，则1−β=10，表示将前10天进行指数加权平均。当β=0.98，则1−β=50，表示将前50天进行指数加权平均。β值越大，则指数加权平均的天数越多，平均后的趋势线就越平缓，但是同时也会向右平移。下图绿色曲线和黄色曲线分别表示了β=0.98和β=0.5时，指数加权平均的结果。

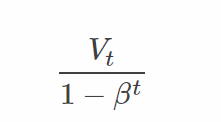
**加权平均的偏移校正：**



上文中提到当β=0.98时，指数加权平均结果如下图绿色曲线所示。但是实际上，真实曲线如紫色曲线所示。

我们注意到，紫色曲线与绿色曲线的区别是，紫色曲线开始的时候相对较低一些。这是因为开始时我们设置V0=0，所以初始值会相对小一些，直到后面受前面的影响渐渐变小，趋于正常。

修正这种问题的方法是进行偏移校正，即在每次计算完Vt后，对Vt进行下式处理：



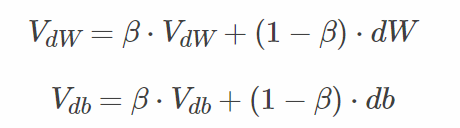
在刚开始的时候，t比较小，(1−βt)<1，这样就将Vt修正得更大一些，效果是把紫色曲线开始部分向上提升一些，与绿色曲线接近重合。随着t增大，(1−βt)≈1，Vt基本不变，紫色曲线与绿色曲线依然重合。这样就实现了简单的偏移校正，得到我们希望的绿色曲线。

**注意：**在机器学习中，偏移校正并不是必须的。因为，在迭代一次次数后（t较大），Vt受初始值影响微乎其微，紫色曲线与绿色曲线基本重合。所以，一般可以忽略初始迭代过程，等到一定迭代之后再取值，这样就不需要进行偏移校正了。

1. **动态梯度下降算法**

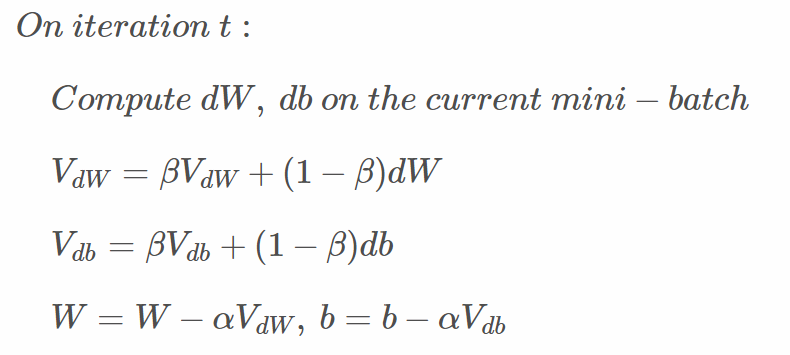
此算法速度要比传统的梯度下降算法快很多。做法是在每次训练时，对梯度进行指数加权平均处理，然后用得到的梯度值更新权重W和常数项b。

权重W和常数项b的指数加权平均表达式如下：



**为什么叫动量法**：从动量的角度来看，以权重W为例，VdW可以成速度V，dW可以看成是加速度a。指数加权平均实际上是计算当前的速度，当前速度由之前的速度和现在的加速度共同影响。而β<1，又能限制速度VdW过大。也就是说，当前的速度是渐变的，而不是瞬变的，是动量的过程。这保证了梯度下降的平稳性和准确性，减少振荡，较快地达到最小值处。

**其具体算法代码如下：**

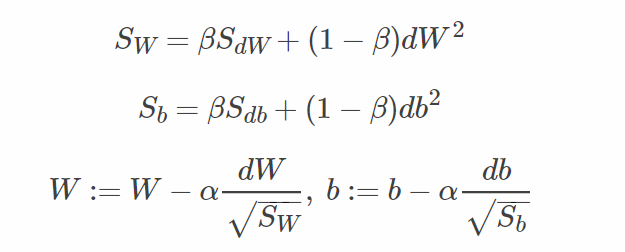


初始时，令VdW=0,Vdb=0。一般设置β=0.9，即指数加权平均前10天的数据，实际应用效果较好。

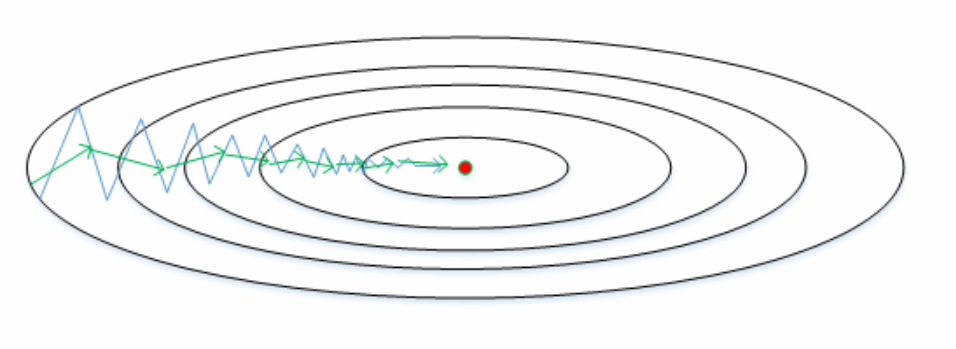
另外，关于偏移校正，可以不使用。因为经过10次迭代后，随着滑动平均的过程，偏移情况会逐渐消失。

1. **RMSprop算法**

此算法每次迭代训练过程中，其权重W和常数项b的更新表达式为：



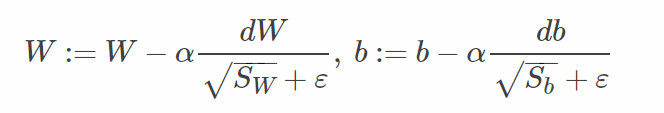
**RMSprop算法的原理：**



如图所示，令水平方向为W的方向，垂直方向为b的方向。

从图中可以看出，梯度下降（蓝色折线）在垂直方向（b）上振荡较大，在水平方向（W）上振荡较小，表示在b方向上梯度较大，即db较大，而在W方向上梯度较小，即dW较小。因此，上述表达式中Sb较大，而SW较小。在更新W和b的表达式中，变化值较大，而较小。也就使得W变化得多一些，b变化得少一些。即加快了W方向的速度，减小了b方向的速度，减小振荡，实现快速梯度下降算法，其梯度下降过程如绿色折线所示。总得来说，就是如果哪个方向振荡大，就减小该方向的更新速度，从而减小振荡。

**注意：**为了避免RMSprop算法中分母为零，通常可以在分母增加一个极小的常数ε：

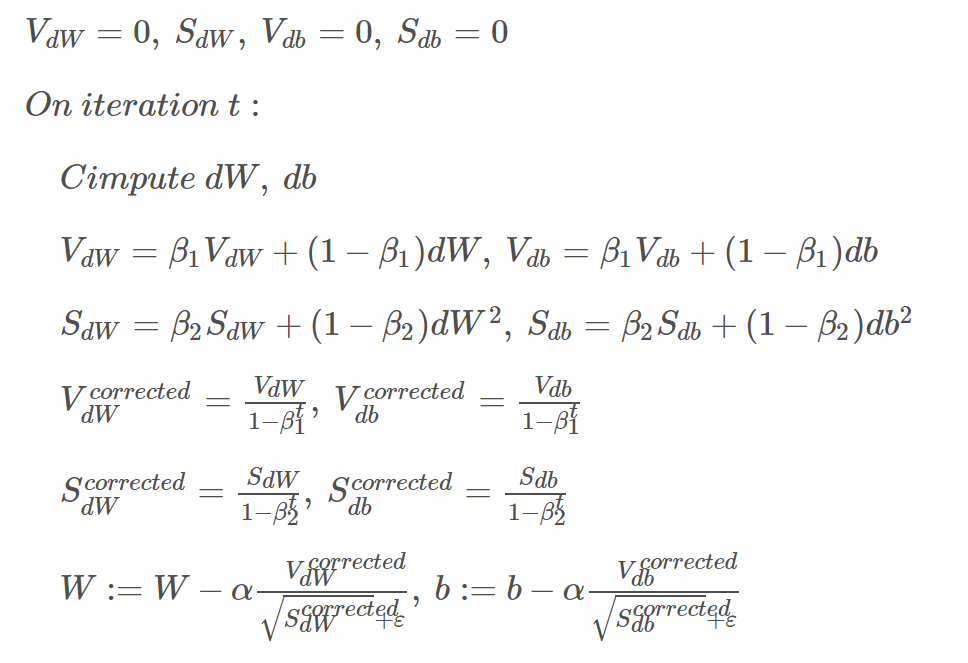


其中，ε=10−8，或者其它较小值。

1. **Adam算法**

此算法结合了动量梯度下降算法和RMSprop算法。

算法流程如下：



Adam算法包含了几个超参数，分别是：α,β1,β2,ε。其中，β1通常设置为0.9，β2通常设置为0.999，ε通常设置为10−8。一般只需要对β1和β2进行调试。

实际应用中，Adam算法结合了动量梯度下降和RMSprop各自的优点，使得神经网络训练速度大大提高。