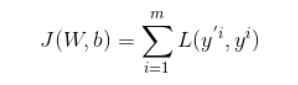
### 优化器(Optimizer)算法

Optimizer有GD, BGD, SGD, MBGD, Momentum, Adagrad, Adadelta, RMSprop, Adam

自适应学习率优化算法有Adagrad, Adadelta, RMSprop, Adam，针对模型的学习率，根据变化调节学习率。

在深度学习中，通过优化代价函数J来训练神经网络。正向传播得到y`值，在反向传播中通过优化算法更新可训练参数W，b值，使得代价函数J值收敛，最小化。其中J的变量为θ



目录

[1. GD(Gradient Descent) 2](#_Toc2808)

[2. BGD(Batch Gradient Descent) 3](#_Toc8124)

[3. SGD(Stochastic Gradient Descent) 3](#_Toc18169)

[4. BGD和SGD对比 5](#_Toc9880)

[5. MBGD(Mini-Batch Gradient Descent) 5](#_Toc13801)

[6. 指数加权平均 6](#_Toc13018)

[7. Momentum 8](#_Toc23307)

[8. Adagrad(Adaptive gradient algorithm) 8](#_Toc26204)

[9. Adadelta 9](#_Toc21994)

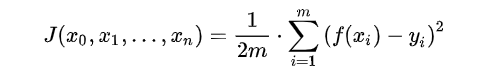
[10. RMSprop(Root Mean Squared prop) 9](#_Toc15152)

[11. Adam(Adaptive moment estimation) 10](#_Toc30264)

[12. 如何选择优化算法 10](#_Toc12418)

# **GD(Gradient Descent)**

J与样本数m有关，为了不超越数值型边界，

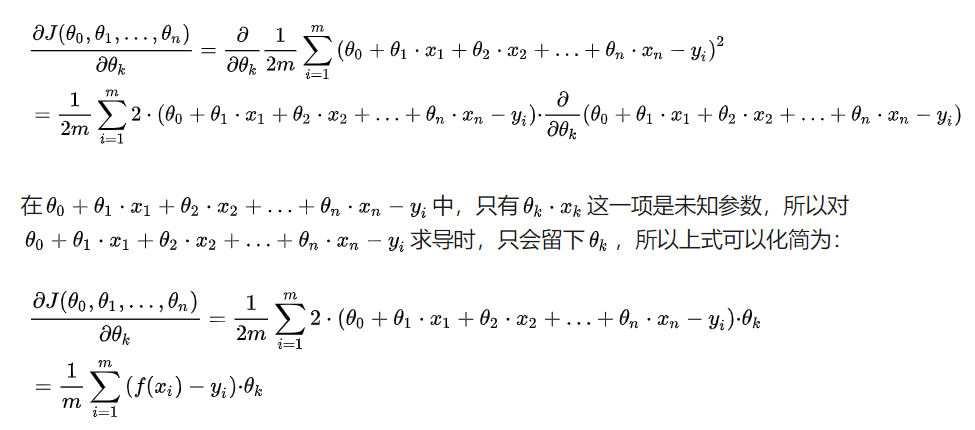


更新参数为：



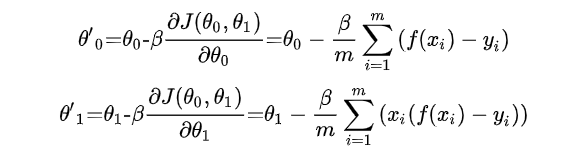
基本策略可以理解为“在有限视距内寻找最快路径下山”，因此每走一步，参考当前位置最陡的方向(即梯度)进而迈出下一步。

推导过程：



以一元线性方程为例：假设本次参数初始取值为θ0、θ1，下一次参数取值θ’0和θ’1：



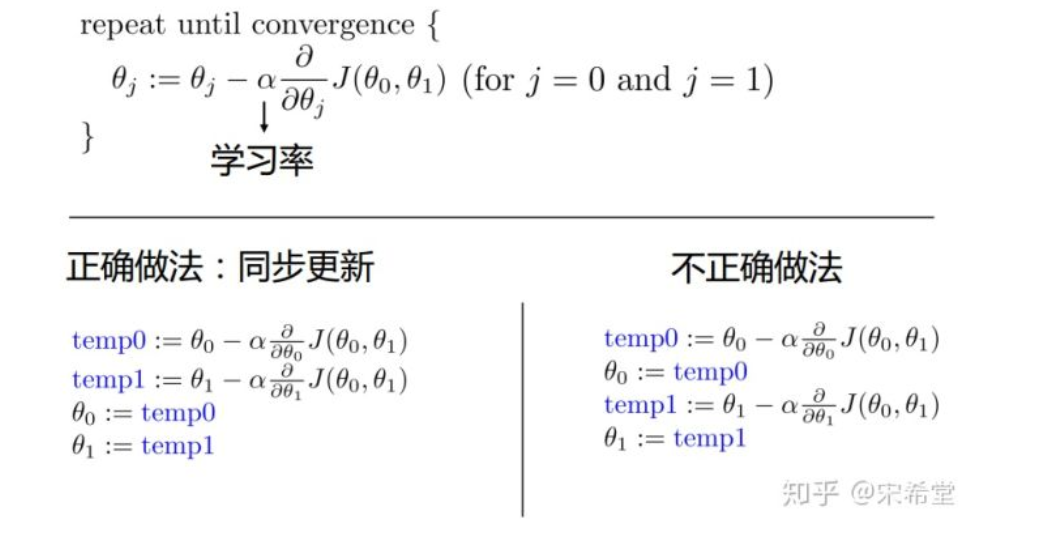


标准梯度下降法主要有两个缺点:

训练速度慢：每走一步都要要计算调整下一步的方向，下山的速度变慢。在应用于大型数据集中，每输入一个样本都要更新一次参数，且每次迭代都要遍历所有的样本。会使得训练过程极其缓慢，需要花费很长时间才能得到收敛解。

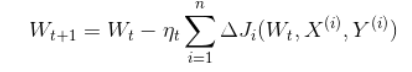
容易陷入局部最优解：由于是在有限视距内寻找下山的反向。当陷入平坦的洼地，会误以为到达了山地的最低点，从而不会继续往下走。所谓的局部最优解就是鞍点。落入鞍点，梯度为0，使得模型参数不在继续更新。

待讨论（不正确的做法有问题？）：



# **BGD(Batch Gradient Descent)**

假设批量训练样本总数为n，每次输入和输出的样本分别为X(i), Y(i)，模型参数为W，代价函数为J(W)，每输入一个样本i代价函数关于W的梯度为，学习率为ηt，则使用批量梯度下降法更新参数表达式为：

    （ Wt表示t时刻的模型参数。)

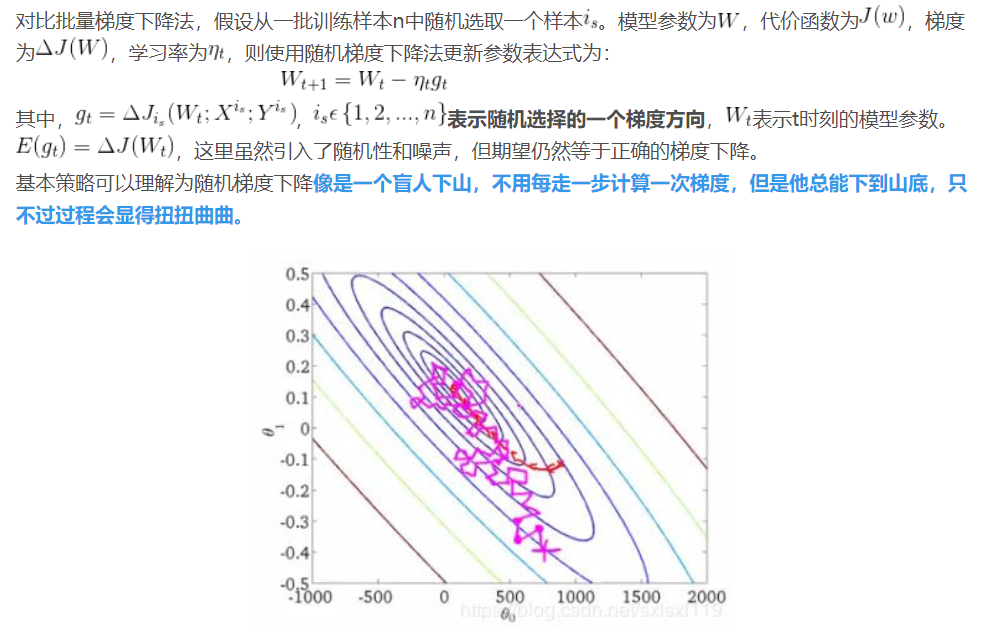
从表达式来看，模型参数的调整更新与全部输入样本的代价函数的和（即批量/全局误差）有关。即每次权值调整发生在批量样本输入之后，而不是每输入一个样本就更新一次模型参数。这样就会大大加快训练速度。基本策略可以理解为“在下山之前掌握了附近的地势情况，选择总体平均梯度最小的方向下山”。

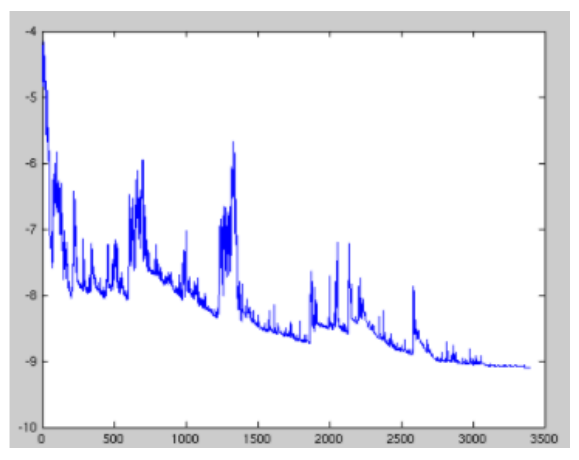
批量梯度下降法优缺点：

优点：批量梯度下降法比标准梯度下降法训练时间短，且每次下降的方向都很正确。

缺点：由于这种方法是在一次更新中，就对整个数据集计算梯度，所以计算起来非常慢，遇到很大量的数据集也会非常棘手，而且不能投入新数据实时更新模型。

# **SGD(Stochastic Gradient Descent)**



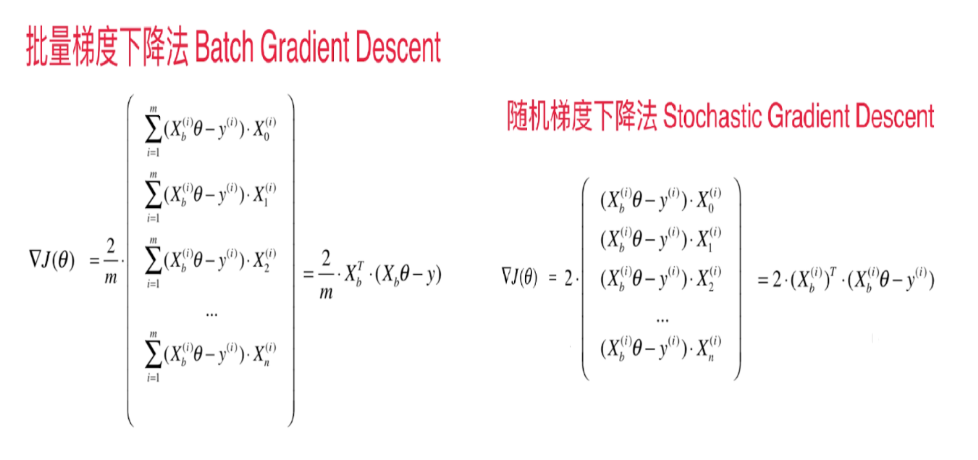


随机梯度下降法优缺点：

优点：虽然SGD需要走很多步的样子，但是对梯度的要求很低（计算梯度快）。而对于引入噪声，大量的理论和实践工作证明，只要噪声不是特别大，SGD都能很好地收敛。应用大型数据集时，训练速度很快。比如每次从百万数据样本中，取几百个数据点，算一个SGD梯度，更新一下模型参数。相比于标准梯度下降法的遍历全部样本，每输入一个样本更新一次参数，要快得多。

缺点：SGD在随机选择梯度的同时会引入噪声，使得权值更新的方向不一定正确。此外，SGD也没能单独克服局部最优解的问题。

# **BGD和SGD对比**



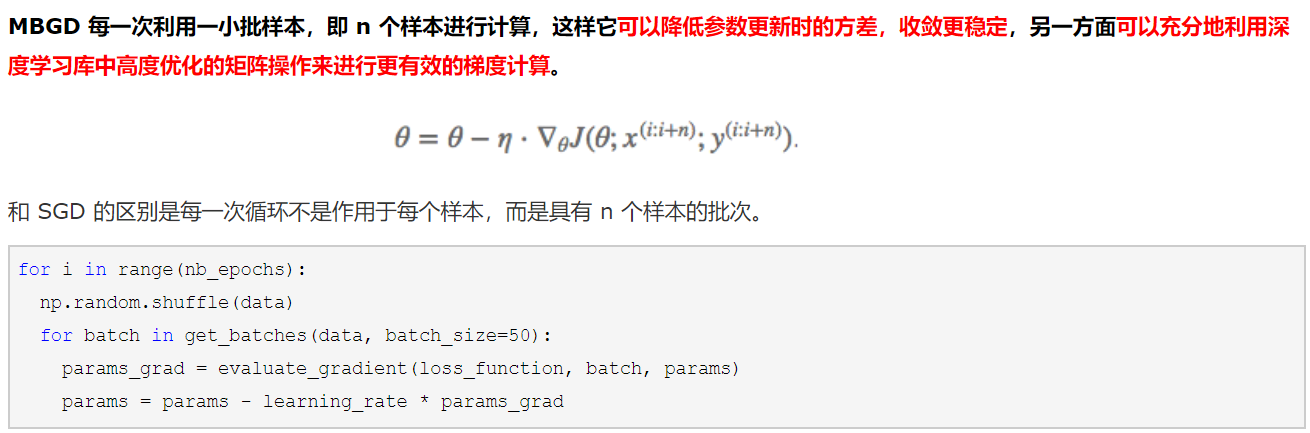
SGD的噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。所以虽然训练速度快，但是准确度下降，并不是全局最优。虽然包含一定的随机性，但是从期望上来看，它是等于正确的导数的。

SGD 因为更新比较频繁，会造成 cost function 有严重的震荡。

BGD 可以收敛到局部极小值，当然 SGD 的震荡可能会跳到更好的局部极小值处。

当我们稍微减小 learning rate，SGD 和 BGD 的收敛性是一样的。

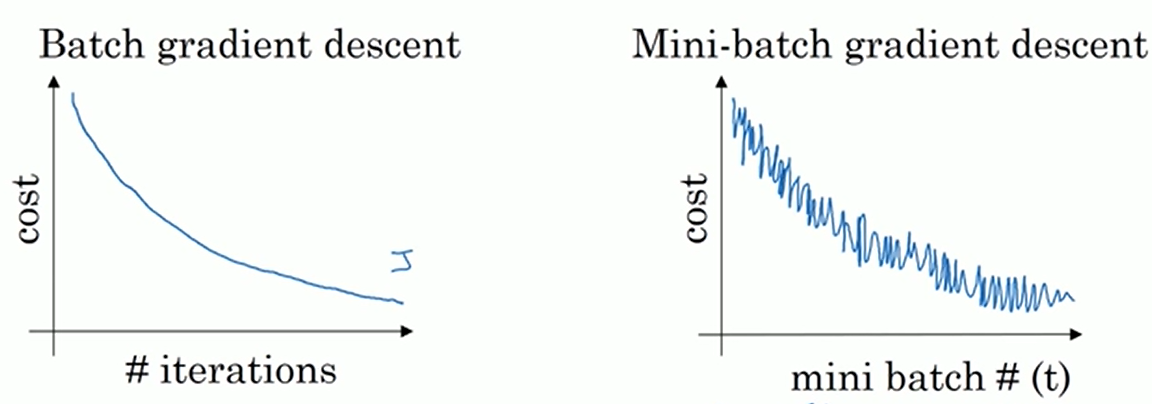
# **MBGD(Mini-Batch Gradient Descent)**



优点：速度比BGD快

缺点：不能保证很好的收敛性，lr太小，收敛慢，lr太大，loss极小值处振荡甚至偏离。

参考：[https://mooc.study.163.com/learn/2001281003?tid=2001391036&\_trace\_c\_p\_k2\_=7b3b26de31574179930df7ce28f4e446#/learn/content?type=detail&id=2001701050](https://mooc.study.163.com/learn/2001281003?tid=2001391036&_trace_c_p_k2_=7b3b26de31574179930df7ce28f4e446" \l "/learn/content?type=detail&id=2001701050)



If batch\_size = 1 MBGD = SGD

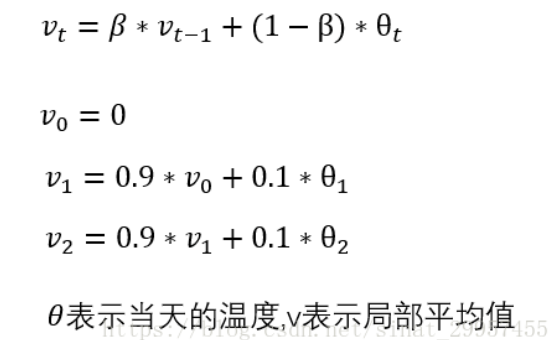
If batch\_size = m MBGD = BGD

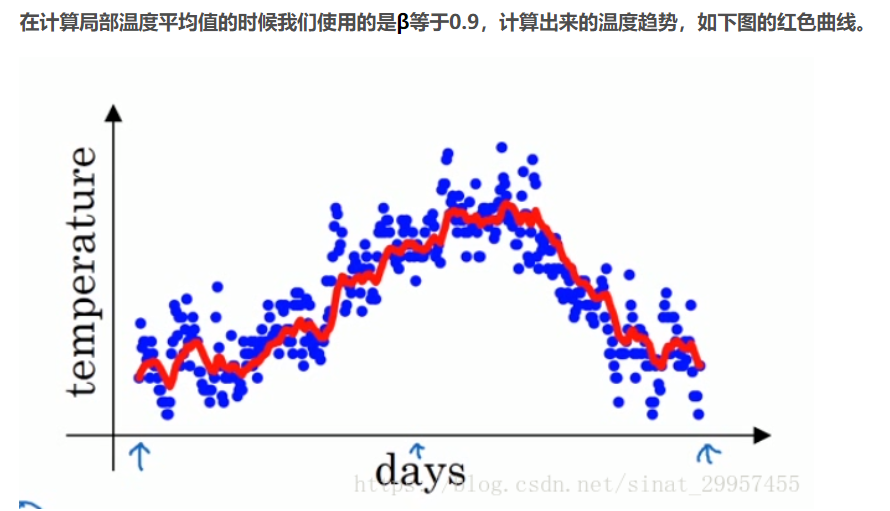
Batch\_size(n)一般取值有64，128，256，512

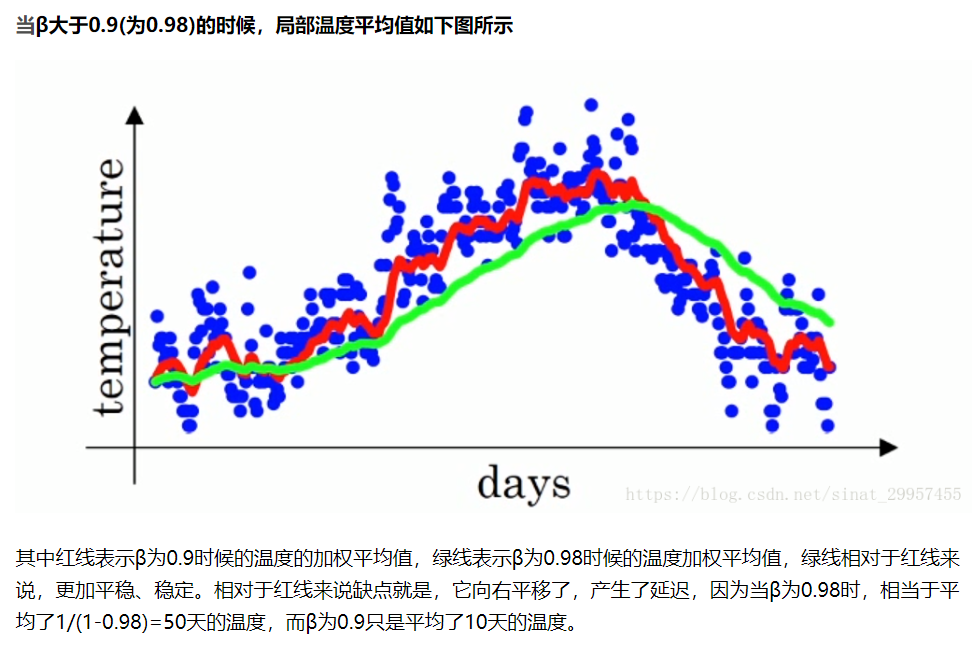
# **指数加权平均**

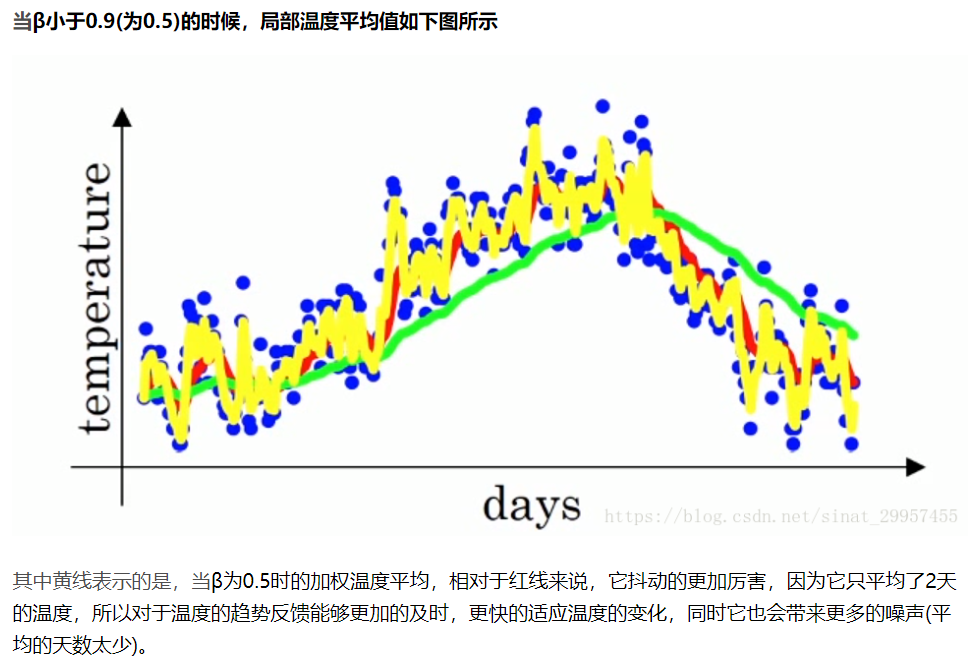
我们可以看到指数加权平均的求解过程实际上是一个递推的过程，那么这样就会有一个非常大的好处，每当我要求从0到某一时刻（n）的平均值的时候，我并不需要像普通求解平均值的作为，保留所有的时刻值，类和然后除以n。

而是只需要保留0-(n-1)时刻的平均值和n时刻的温度值即可。也就是每次只需要保留常数值，然后进行运算即可，这对于深度学习中的海量数据来说，是一个很好的减少内存和空间的做法。



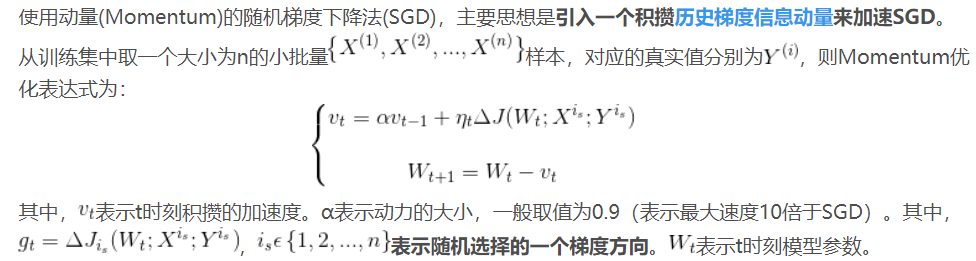


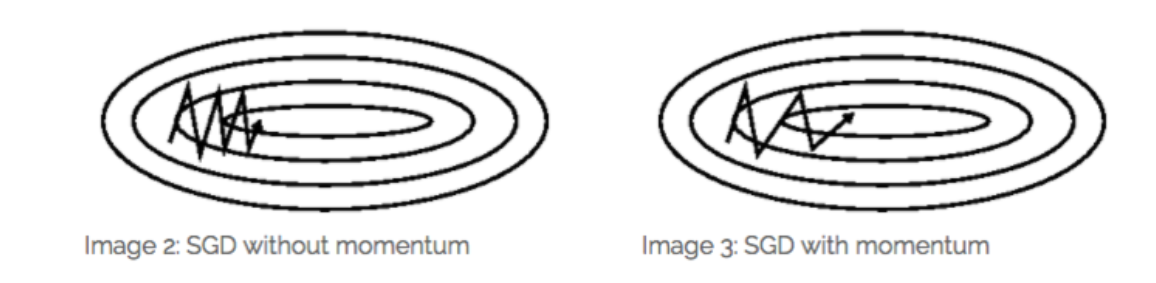




# **Momentum**

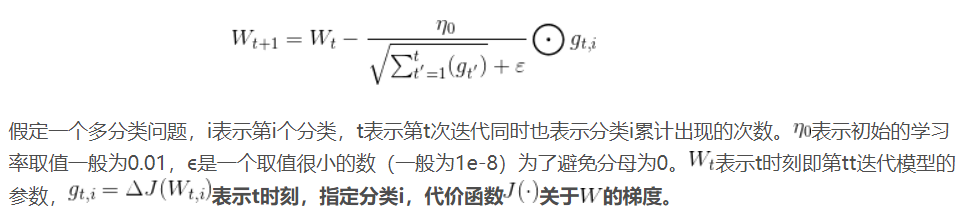
由于当前权值的改变会受到上一次权值改变的影响，类似于小球向下滚动的时候带上了惯性。这样可以加快小球向下滚动的速度。





纵轴幅度变小，横轴幅度变大，训练速度加快。

# **Adagrad(Adaptive gradient algorithm)**

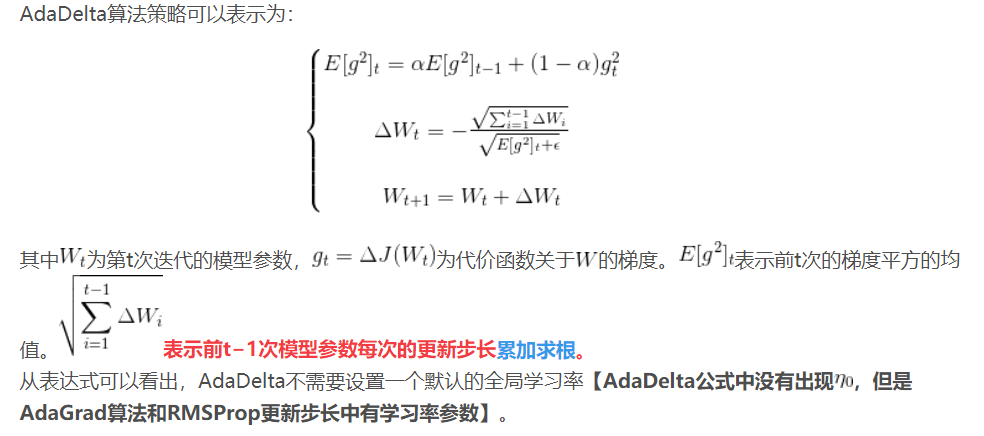


优点：在数据分布稀疏的场景，能更好利用稀疏梯度的信息，比标准的SGD算法更有效地收敛。

不需要人为的调节学习率，它可以自动调节。

缺点：来自分母项的对梯度平方不断累积，随之时间步地增加，分母项越来越大，最终导致学习率收缩到太小无法进行有效更新。

# **Adadelta**

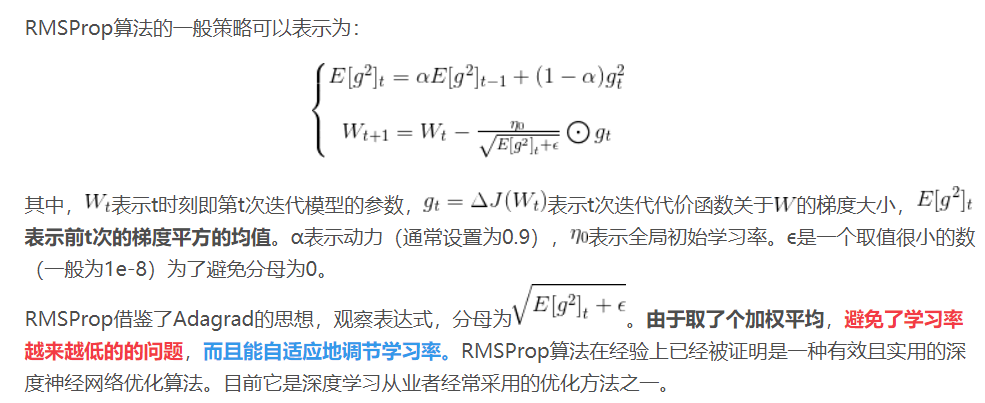


优缺点：

在模型训练的初期和中期，Adadelta表现很好，加速效果不错，训练速度快。

在模型训练的后期，模型会反复地在局部最小值附近抖动。

# **RMSprop(Root Mean Squared prop)**

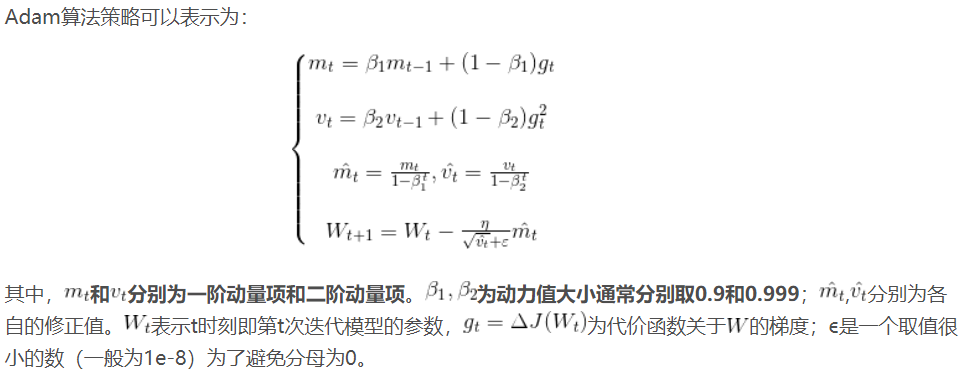


特点：使用的是指数加权平均，旨在消除梯度下降中的摆动，与Momentum的效果一样，某一维度的导数比较大，则指数加权平均就大，某一维度的导数比较小，则其指数加权平均就小，这样就保证了各维度导数都在一个量级，进而减少了摆动。允许使用一个更大的学习率η

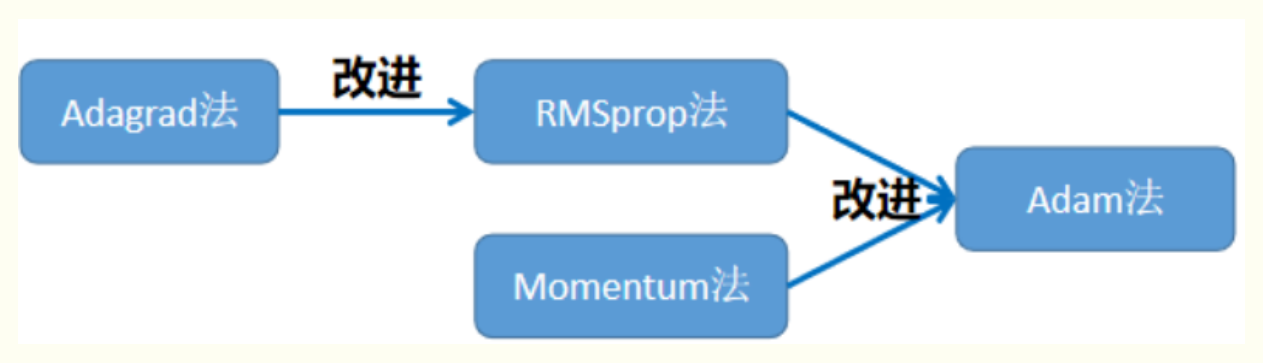
超参数值：α= 0.9， η= 0.001

# **Adam(Adaptive moment estimation)**

相当于 RMSprop + Momentum



# **如何选择优化算法**



如果数据是稀疏的，就用自适用方法，即 Adagrad, Adadelta, RMSprop, Adam。

RMSprop, Adadelta, Adam 在很多情况下的效果是相似的。

Adam 就是在 RMSprop 的基础上加了 bias-correction 和 momentum，

随着梯度变的稀疏，Adam 比 RMSprop 效果会好。

整体来讲，Adam 是最好的选择。

很多论文里都会用 SGD，没有 momentum 等。SGD 虽然能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。

如果需要更快的收敛，或者是训练更深更复杂的神经网络，需要用一种自适应的算法。