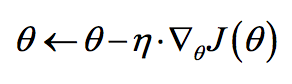
**概述**

梯度下降算法（Gradient Descent Optimization）是神经网络模型训练最常用的优化算法。对于深度学习模型，基本都是采用梯度下降算法来进行优化训练的。梯度下降算法背后的原理：目标函数640?wx_fmt=png&wxfrom=5&wx_lazy=1关于参数640?wx_fmt=png&wxfrom=5&wx_lazy=1的梯度将是目标函数上升最快的方向。对于最小化优化问题，只需要将参数沿着梯度相反的方向前进一个步长，就可以实现目标函数的下降。这个步长又称为学习速率640?wx_fmt=png&wxfrom=5&wx_lazy=1。参数更新公式如下：



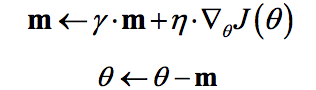
其中640?wx_fmt=png&wxfrom=5&wx_lazy=1是参数的梯度，根据计算目标函数640?wx_fmt=png&wxfrom=5&wx_lazy=1采用数据量的不同，梯度下降算法又可以分为批量梯度下降算法（Batch Gradient Descent），随机梯度下降算法（Stochastic GradientDescent）和小批量梯度下降算法（Mini-batch Gradient Descent）。对于批量梯度下降算法，其640?wx_fmt=png&wxfrom=5&wx_lazy=1是在整个训练集上计算的，如果数据集比较大，可能会面临内存不足问题，而且其收敛速度一般比较慢。随机梯度下降算法是另外一个极端，640?wx_fmt=png&wxfrom=5&wx_lazy=1是针对训练集中的一个训练样本计算的，又称为在线学习，即得到了一个样本，就可以执行一次参数更新。所以其收敛速度会快一些，但是有可能出现目标函数值震荡现象，因为高频率的参数更新导致了高方差。小批量梯度下降算法是折中方案，选取训练集中一个小批量样本计算640?wx_fmt=png&wxfrom=5&wx_lazy=1，这样可以保证训练过程更稳定，而且采用批量训练方法也可以利用矩阵计算的优势。这是目前最常用的梯度下降算法。

对于神经网络模型，借助于BP算法可以高效地计算梯度，从而实施梯度下降算法。但梯度下降算法一个老大难的问题是：不能保证全局收敛。如果这个问题解决了，深度学习的世界会和谐很多。梯度下降算法针对凸优化问题原则上是可以收敛到全局最优的，因为此时只有唯一的局部最优点。而实际上深度学习模型是一个复杂的非线性结构，一般属于非凸问题，这意味着存在很多局部最优点（鞍点），采用梯度下降算法可能会陷入局部最优，这应该是最头疼的问题。这点和进化算法如遗传算法很类似，都无法保证收敛到全局最优。因此，我们注定在这个问题上成为“高级调参师”。可以看到，梯度下降算法中一个重要的参数是学习速率，适当的学习速率很重要：学习速率过小时收敛速度慢，而过大时导致训练震荡，而且可能会发散。理想的梯度下降算法要满足两点：收敛速度要快；能全局收敛。为了这个理想，出现了很多经典梯度下降算法的变种，下面将分别介绍它们。

01

Momentum optimization

冲量梯度下降算法是BorisPolyak在1964年提出的，其基于这样一个物理事实：将一个小球从山顶滚下，其初始速率很慢，但在加速度作用下速率很快增加，并最终由于阻力的存在达到一个稳定速率。对于冲量梯度下降算法，其更新方程如下：

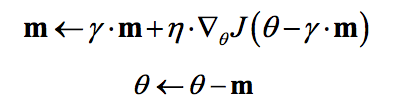


可以看到，参数更新时不仅考虑当前梯度值，而且加上了一个积累项（冲量），但多了一个超参0?wx_fmt=png，一般取接近1的值如0.9。相比原始梯度下降算法，冲量梯度下降算法有助于加速收敛。当梯度与冲量方向一致时，冲量项会增加，而相反时，冲量项减少，因此冲量梯度下降算法可以减少训练的震荡过程。TensorFlow中提供了这一优化器：tf.train.MomentumOptimizer(learning\_rate=learning\_rate,momentum=0.9)。

02

NAG

NAG算法全称Nesterov Accelerated Gradient,是YuriiNesterov在1983年提出的对冲量梯度下降算法的改进版本，其速度更快。其变化之处在于计算“超前梯度”更新冲量项，具体公式如下：



既然参数要沿着0?wx_fmt=png更新，不妨计算未来位置0?wx_fmt=png的梯度，然后合并两项作为最终的更新项，其具体效果如图1所示，可以看到一定的加速效果。在TensorFlow中，NAG优化器为：tf.train.MomentumOptimizer(learning\_rate=learning\_rate,momentum=0.9, use\_nesterov=True)

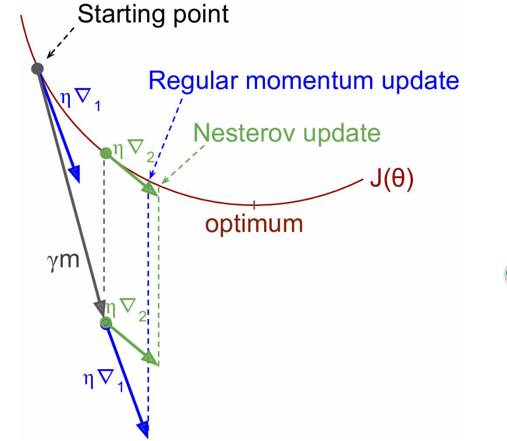
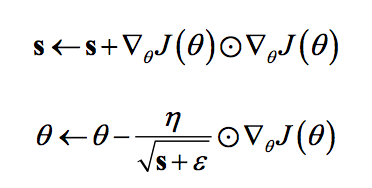


图1 NAG效果图

03

AdaGrad

AdaGrad是Duchi在2011年提出的一种学习速率自适应的梯度下降算法。在训练迭代过程，其学习速率是逐渐衰减的，经常更新的参数其学习速率衰减更快，这是一种自适应算法。其更新过程如下：



其中是梯度平方的积累量，在进行参数更新时，学习速率要除以这个积累量的平方根，其中加上一个很小值是为了防止除0的出现。由于是该项逐渐增加的，那么学习速率是衰减的。考虑如图2所示的情况，目标函数在两个方向的坡度不一样，如果是原始的梯度下降算法，在接近坡底时收敛速度比较慢。而当采用AdaGrad，这种情况可以被改观。由于比较陡的方向梯度比较大，其学习速率将衰减得更快，这有利于参数沿着更接近坡底的方向移动，从而加速收敛。

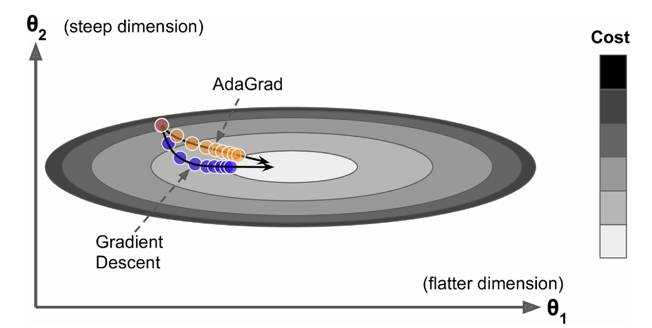


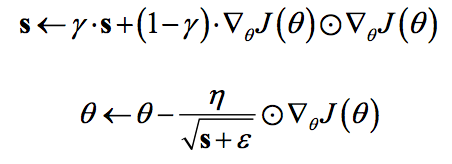
图2 AdaGrad效果图

前面说到AdaGrad其学习速率实际上是不断衰减的，这会导致一个很大的问题，就是训练后期学习速率很小，导致训练过早停止，因此在实际中AdaGrad一般不会被采用，下面的算法将改进这一致命缺陷。不过TensorFlow也提供了这一优化器：tf.train.AdagradOptimizer。

04

RMSprop

RMSprop是Hinton在他的课程上讲到的，其算是对Adagrad算法的改进，主要是解决学习速率过快衰减的问题。其实思路很简单，类似Momentum思想，引入一个超参数，在积累梯度平方项进行衰减：



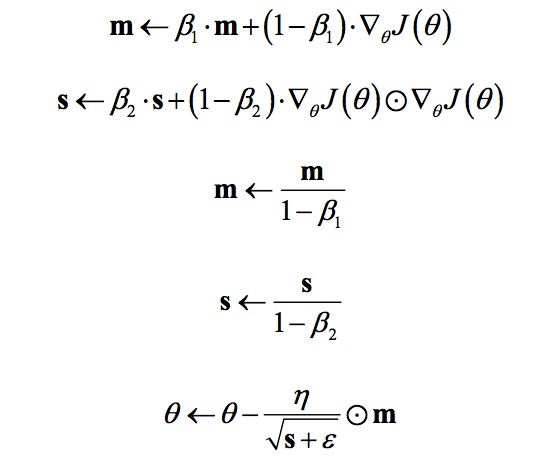
可以认为仅仅对距离时间较近的梯度进行积累，其中一般取值0.9，其实这样就是一个指数衰减的均值项，减少了出现的爆炸情况，因此有助于避免学习速率很快下降的问题。同时Hinton也建议学习速率设置为0.001。RMSprop是属于一种比较好的优化算法了，在TensorFlow中当然有其身影：tf.train.RMSPropOptimizer(learning\_rate=learning\_rate,momentum=0.9, decay=0.9, epsilon=1e-10)。

不得不说点题外话，同时期还有一个Adadelta算法，其也是Adagrad算法的改进，而且改进思路和RMSprop很像，但是其背后是基于一次梯度近似代替二次梯度的思想，感兴趣的可以看看相应的论文，这里不再赘述。

05

Adam

Adam全称Adaptive moment estimation，是Kingma等在2015年提出的一种新的优化算法，其结合了Momentum和RMSprop算法的思想。相比Momentum算法，其学习速率是自适应的，而相比RMSprop，其增加了冲量项。所以，Adam是两者的结合体：



可以看到前两项和Momentum和RMSprop是非常一致的，由于和的初始值一般设置为0，在训练初期其可能较小，第三和第四项主要是为了放大它们。最后一项是参数更新。其中超参数的建议值是0?wx_fmt=png。Adm是性能非常好的算法，在TensorFlow其实现如下： tf.train.AdamOptimizer(learning\_rate=0.001,beta1=0.9, beta2=0.999, epsilon=1e-08)。

二

**学习速率**

前面也说过学习速率的问题，对于梯度下降算法，这应该是一个最重要的超参数。如果学习速率设置得非常大，那么训练可能不会收敛，就直接发散了；如果设置的比较小，虽然可以收敛，但是训练时间可能无法接受；如果设置的稍微高一些，训练速度会很快，但是当接近最优点会发生震荡，甚至无法稳定。不同学习速率的选择影响可能非常大，如图3所示。

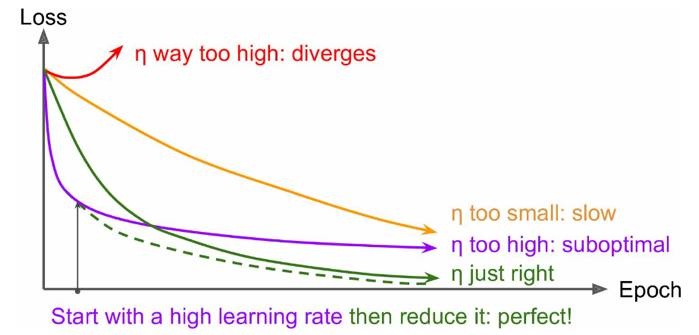
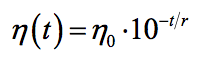


图3 不同学习速率的训练效果

理想的学习速率是：刚开始设置较大，有很快的收敛速度，然后慢慢衰减，保证稳定到达最优点。所以，前面的很多算法都是学习速率自适应的。除此之外，还可以手动实现这样一个自适应过程，如实现学习速率指数式衰减：



在TensorFlow中，你可以这样实现：

initial\_learning\_rate = 0.1  
decay\_steps = 10000  
decay\_rate = 1/10  
global\_step = tf.Variable(0, trainable=False)  
learning\_rate = tf.train.exponential\_decay(initial\_learning\_rate,                             
                            global\_step, decay\_steps, decay\_rate)  
# decayed\_learning\_rate = learning\_rate \*  
#                decay\_rate ^ (global\_step / decay\_steps)  
optimizer = tf.train.MomentumOptimizer(learning\_rate, momentum=0.9)  
training\_op = optimizer.minimize(loss, global\_step=global\_step)

三

**总结**

本文简单介绍了梯度下降算法的分类以及常用的改进算法，总结来看，优先选择学习速率自适应的算法如RMSprop和Adam算法，大部分情况下其效果是较好的。还有一定要特别注意学习速率的问题。其实还有很多方面会影响梯度下降算法，如梯度的消失与爆炸，这也是要额外注意的。最后不得不说，梯度下降算法目前无法保证全局收敛还将是一个持续性的数学难题。

四

**参考文献**

1. Anoverview of gradient descent optimization algorithms: http://sebastianruder.com/optimizing-gradient-descent/.
2. Hands-OnMachine Learning with Scikit-Learn and TensorFlow, Aurélien Géron, 2017.
3. NAG:http://proceedings.mlr.press/v28/sutskever13.pdf.
4. Adagrad:http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.
5. RMSprop:http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\_slides\_lec6.pdf.
6. Adadelta:https://arxiv.org/pdf/1212.5701v1.pdf.
7. Adam:https://arxiv.org/pdf/1412.6980.pdf.
8. 不同的算法的效果可视化：https://imgur.com/a/Hqolp.