李宏毅深度学习笔记-P2

Gradient Descent

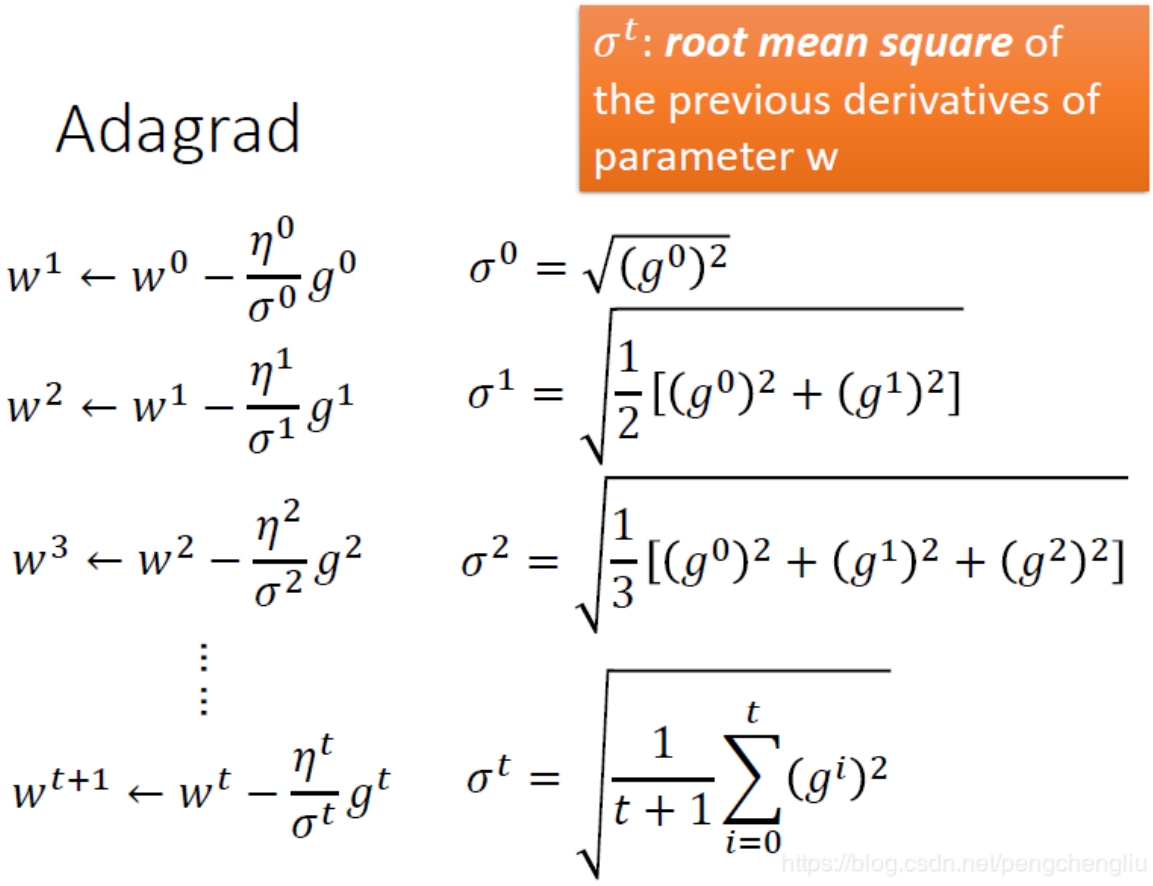
Tips 1: Tuning your learning rate

通过实验我们可以知道learning rate越大速度越快，但这会导致我们越难以逼近我们要的最小值。但learning rate过小会导致速度慢，所以有个变化的learning rate再好不过了

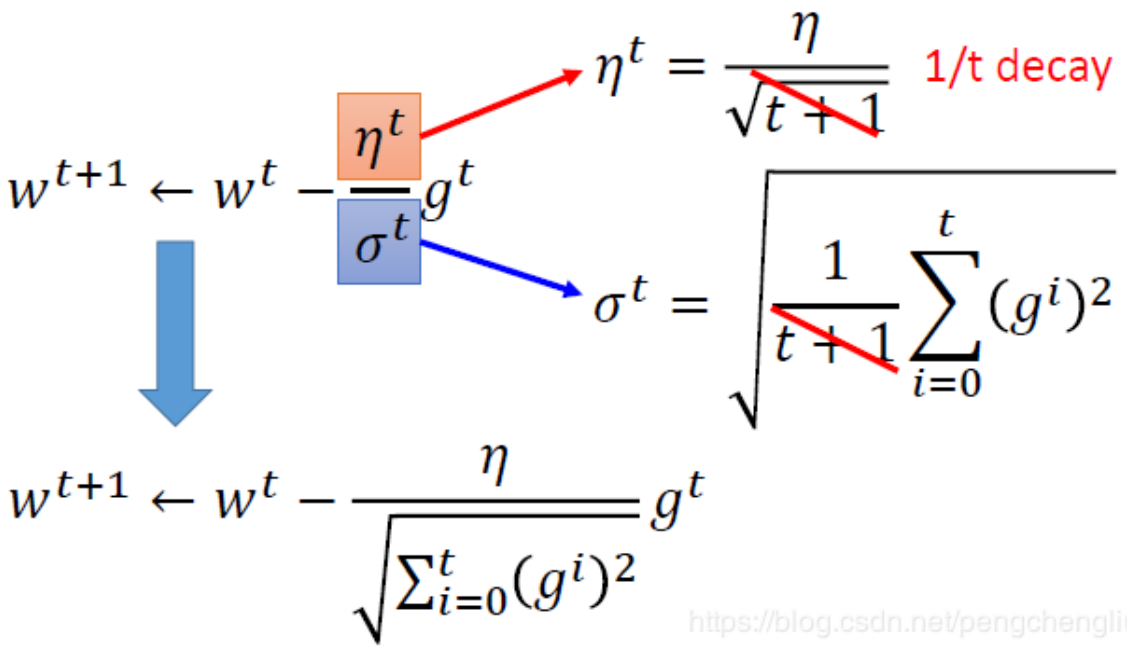
$$\eta^t = \frac{\eta}{\sqrt{t+1}}$$

有了变化的learning rate之后，我们再让每个因素有单独的learning rate效果会更好这里介绍一种最简单的方法Adagrad

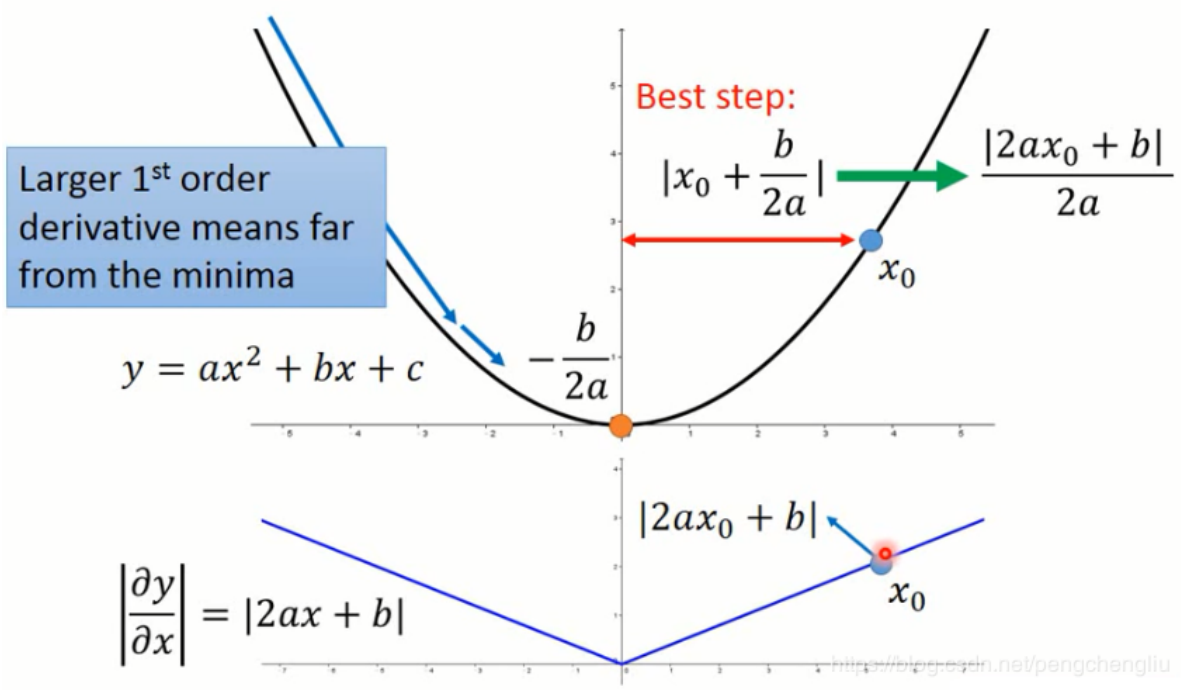
在Adagrad里面呢，你把η^t除以σ^t。σ^t是过去所有的微分的值的均方根。这个值对每一个参数而言，都是不一样。所以现在变成说，不同的参数，他的learning rate都是不一样的。这里举个实例



化简过后



从式子我们可以了解到Adagrad是会越来越慢的，也有更好的方法，只是在实现上会比较复杂。

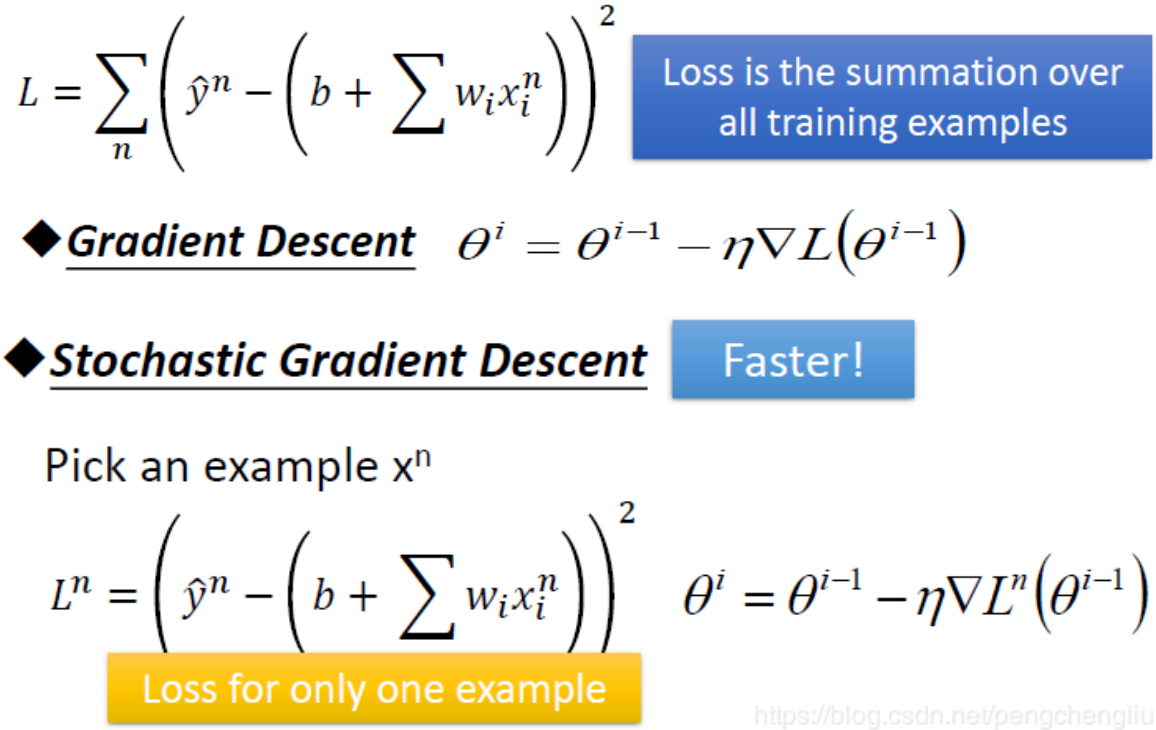


从实验中我们可以知道，最好的步伐应该正比于一次微分，反比于二次微分A。

但是二次微分的计算会让我们的计算量增大很多，特别是数据集大的情况下。而Adagrad的做法就是，我们在没有增加任何额外运算的前提之下，想办法能不能做一件事情去估一下二次微分是多少。在Adagrad里面，只需要一次微分的值，而这个本来就是要算的，所以没有多做多余的任何运算。

Tips 2: Stochastic Gradient Descent(随机梯度下降)

下图分别是两种方法的表达式



Stochastic Gradient Descent，它每次就拿一个example x_n出来（你可以按照顺序去，也可以随机取。），然后计算Loss。Loss呢，只考虑一个example。不做summation了。我们写作L上标n，表示考虑第n个example的Loss function。接下来呢，在update参数的时候，你只算L上标n的gradient。然后就很急躁的update参数了。所以在原来的gradient descent里面，要计算所有data的loss，然后再update参数。但是在随机梯度下降法里面，你看一个example就更新一下参数。

虽然说Stochastic Gradient Descent在只看一个参数时它的步伐可能是小且散乱的，但它会update很多次从而提高速度。

Tips 3: Feature Scaling(特征归一化)

举个例子，假设我们有式子$y = b + w_1 x_1 + w_2 x_2$ Feature Scaling要做的事就是让w_1和w_2的影响力基本相同从而提高我们update参数的效率。

归一化方法也有很多，这里举一个例子。对一个因素算出其平均值，然后在算出其标准差。然后对数据集中的这个因素都先减去平均值后除以标准差即可使mean(平均值)=0,variance（差额）=1。

Gradient Descent Theory

首先介绍了Taylor series,然后讲到有几个参数时也可以用

Taylor series。这也是这个的理论基础。假设我们给定中心点（a,b）

然后我们画出一个很小的圆圈在这里面将loss function泰勒展开，然后通过对向量的计算得到结果

我们做gradient descent 就是找一个初始值；然后算初始值地方的偏微分，把它排成一个向量，就是gradient；最后乘以一个learning rate ； 再把它减掉。

注意1：

我们可以用上面的方法来找一个最小的Loss（即，我们可以用gradient descent的方法来做），有一个前提就是

$$L(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b)$$

这个式子要成立。

只有当今天画出来的圈圈够小的时候，泰勒公式才成立。

你会发现说，它只考虑了泰勒公式的一次式。可不可以考虑二次式呢?

是可以的，有一些方法比如牛顿法。但是在实际上，尤其是你在做deep learning的时候，这些方法并不见得太普及，太实用。

因为你要算二次微分，甚至还会算海森矩阵，总之会多很多运算。而这些运算你在做deep learning的时候呢，你是无法承受的。用这个运算来换，update的时候比较有效率，是不划算的。所以，要做deep learning的时候，gradient descent还是比较主流的做法。

Gradient Descent的限制

首先，微分值最低的地方不一定就是local minimum,saddle point(鞍点)的微分值也是0。但真正的问题是停下来的时候并不是正好等于零，而是在微分值小于一个很小的值的时候停下来的，这会导致我们没办法确定它一定停在local minimum旁