AdaGrad 算法

1. 产生背景：

在 SGD（Batch Gradient Descent、Stochastic Gradient Descent、Mini-batch Gradient Descent）、Momentum、Nesterov Momentum 中，目标函数自变量的每一个元素在相同的时间步都是用相同的学习率α来实现迭代。而在 Momentum 中可以看到当x_1与x_2的梯度值由较大差别时，需要选择足够小的学习率是的自变量在梯度值较大的维度上不发散，而这样会导致自变量在梯度值较效的维度上迭代过慢。而 AdaGrad 算法可根据自变量在每个维度的梯度值大小来调整各维度上的学习率，从而避免同意学习率难以适应所有维度的问题。

2. 算法：

AdaGrad 会使用一个小批量随机梯度g_t按元素平方的累加变量s_t。在时间步 0，AdaGrad 将s_0中每个元素初始化为 0。在时间步 t，首先将小批量随机梯度g_t按元素平方后累加到s_t：

$$\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t$$

注：⊙是按元素相乘

接着，将目标函数自变量中每个元素的学习率按元素运算重新进行调整：

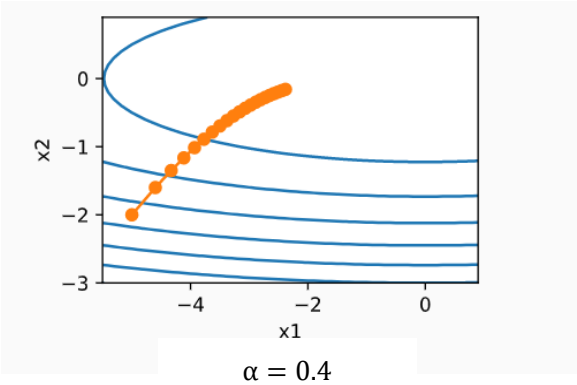
$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t$$

其中η是学习率，ϵ是维持数值稳定性而添加的常熟，如10^{-6}。注意此处的开放、除法、加法都是对每个元素进行运算。这样就可以使得目标函数自变量中每个元素都分别拥有自己的学习率。

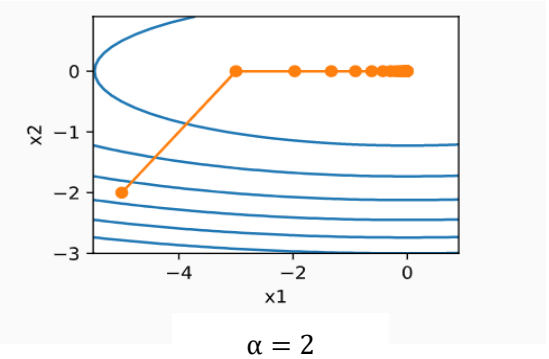
3. 特点：

小批量随机梯度按元素平方的累加变量s_t出现在学习率的分母项中。因此，如果目标函数有关自变量中某个元素的偏导数一直都较大，那么该元素的学习率将下降较快；反之，如果目标函数有关自变量中某个元素的偏导数一直都较小，那么该元素的学习率将下降较慢。然而，由于s_t一直在累加按元素平方的梯度，自变量中每个元素的学习率在迭代过程中一直在降低（或不变）。所以学习率在迭代早期降得较快而在后期可能会存在学习率较小的情况。

例如：



$\alpha = 0.4$



$\alpha = 2$

当学习率为 0.4 时，自变量的迭代轨迹较为平滑，但由于s_t的累加效果使得学习率不断衰减，自变量在迭代后期移动幅度较小。而增大学习率到 2，可以看到自变量更为迅速地逼近了最优解。