动量法

使用梯度下降法，每次都会朝着目标函数下降最快的方向，这也称为最速下降法。这种更新方法看似非常快，实际上存在一些问题。

梯度下降法的问题

考虑一个二维输入，$[x_1, x_2]$，输出的损失函数 $L: R^2 \rightarrow R$，下面是这个函数的等高线

可以想象成一个很扁的漏斗，这样在竖直方向上，梯度就非常大，在水平方向上，梯度就相对较小，所以我们在设置学习率的时候就不能设置太大，为了防止竖直方向上参数更新太过了，这样一个较小的学习率又导致了水平方向上参数在更新的时候太过于缓慢，所以就导致最终收敛起来非常慢。

动量法

动量法的提出就是为了应对这个问题，我们梯度下降法做一个修改如下

$$v_i = \gamma v_{i-1} + \eta \nabla L(\theta) \tag{1}$$

$$\theta_i = \theta_{i-1} - v_i \tag{2}$$

其中 v_i 是当前速度，γ 是动量参数，是一个小于 1的正数，η 是学习率

相当于每次在进行参数更新的时候，都会将之前的速度考虑进来，每个参数在各方向上的移动幅度不仅取决于当前的梯度，还取决于过去各个梯度在各个方向上是否一致，如果一个梯度一直沿着当前方向进行更新，那么每次更新的幅度就越来越大，如果一个梯度在一个方向上不断变化，那么其更新幅度就会被衰减，这样我们就可以使用一个较大的学习率，使得收敛更快，同时梯度比较大的方向就会因为动量的关系每次更新的幅度减少，如下图

比如我们的梯度每次都等于 g，而且方向都相同，那么动量法在该方向上使参数加速移动，有下面的公式：

$$v_0 = 0 \tag{3}$$

$$v_1 = \gamma v_0 + \eta g = \eta g \tag{4}$$

$$v_2 = \gamma v_1 + \eta g = (1+\gamma)\eta g \tag{5}$$

$$v_3 = \gamma v_2 + \eta g = (1+\gamma+\gamma^2)\eta g \tag{6}$$

$$\cdots \tag{7}$$

$$v_{+\infty} = (1+\gamma+\gamma^2+\gamma^3+\cdots)\eta g = \frac{1}{1-\gamma}\eta g \tag{8}$$

如果我们把 γ 定为 0.9，那么更新幅度的峰值就是原本梯度乘学习率的 10 倍。

本质上说，动量法就仿佛我们从高坡上推一个球，小球在向下滚动的过程中积累了动量，在途中也会变得越来越快，最后会达到一个峰值，对应于我们的算法中就是，动量项会沿着梯度指向方向相同的方向不断增大，对于梯度方向改变的方向逐渐减小，得到了更快的收敛速度以及更小的震荡。

这个优化算法被称为自适应学习率优化算法，之前我们讲的随机梯度下降以及动量法对所有的参数都使用的固定的学习率进行参数更新，但是不同的参数梯度可能不一样，所以需要不同的学习率才能比较好的进行训练，但是这个事情又不能很好地被人为操作，所以 Adagrad 便能够帮助我们做这件事。

Adagrad 算法

Adagrad 的想法非常简答，在每次使用一个 batch size 的数据进行参数更新的时候，我们需要计算所有参数的梯度，那么其想法就是对于每个参数，初始化一个变量 s 为 0，然后每次将该参数的梯度平方求和累加到这个变量 s 上，然后在更新这个参数的时候，学习率就变为

$$\frac{\eta}{\sqrt{s+\epsilon}} \tag{9}$$

这里的 ϵ 是为了数值稳定性而加上的，因为有可能 s 的值为 0，那么 0 出现在分母就会出现无穷大的情况，通常 ϵ 取 10^{-10}，这样不同的参数由于梯度不同，他们对应的 s 大小也就不同，所以上面的公式得到的学习率也就不同，这也就实现了自适应的学习率。

Adagrad 的核心想法就是，如果一个参数的梯度一直都非常大，那么其对应的学习率就变小一点，防止震荡，而一个参数的梯度一直都非常小，那么这个参数的学习率就变大一点，使得其能够更快地更新

Adagrad 也有一些问题，因为 s 不断累加梯度的平方，所以会越来越大，导致学习率在后期会变得较小，导致收敛乏力的情况，可能无法收敛到表较好的结果，当然后面有一个对其的改进，我们之后会讲到

RMSProp

RMSprop 是由 Geoff Hinton 在他 Coursera 课程中提出的一种适应性学习率方法，至今仍未被公开发表。前面我们提到了 Adagrad 算法有一个问题，就是学习率分母上的变量 s 不断被累加增大，最后会导致学习率除以一个比较大的数之后变得非常小，这不利于我们找到最后的最优解，所以 RMSProp 的提出就是为了解决这个问题。

RMSProp 算法

RMSProp 仍然会使用梯度的平方量，不同于 Adagrad，其会使用一个指数加权移动平均来计算这个 s，也就是

$$s_i = \alpha s_{i-1} + (1-\alpha)\, g^2 \tag{10}$$

这里 g 表示当前求出的参数梯度，然后最终更新和 Adagrad 是一样的，学习率变成了

$$\frac{\eta}{\sqrt{s+\epsilon}} \tag{11}$$

这里 α 是一个移动平均的系数，也是因为这个系数，导致了 RMSProp 和 Adagrad 不同的地方，这个系数使得 RMSProp 更新到后期累加的梯度平方较小，从而保证 s 不会太大，也就使得模型后期依然能够找到比较优的结果

实现上和 Adagrad 非常像。

Adadelta

Adadelta 算是 Adagrad 法的延伸，它跟 RMSProp 一样，都是为了解决 Adagrad 中学习率不断减小的问题，RMSProp 是通过移动加权平均的方式，而 Adadelta 也是一种方法，有趣的是，它并不需要学习率这个参数。

Adadelta 法

Adadelta 跟 RMSProp 一样，先使用移动平均来计算 s

$$s = \rho s + (1-\rho)g^2 \tag{12}$$

这里 ρ 和 RMSProp 中的 α 都是移动平均系数，g 是参数的梯度，然后我们会计算需要更新的参数的变化量

$$g' = \frac{\sqrt{\Delta\theta + \epsilon}}{\sqrt{s+\epsilon}}g \tag{13}$$

$\Delta\theta$ 初始为 0 张量，每一步做如下的指数加权移动平均更新

$$\Delta\theta = \rho\Delta\theta + (1-\rho)g'^2 \tag{14}$$

最后参数更新如下

$$\theta = \theta - g' \tag{15}$$

Adam

Adam 是一个结合了动量法和 RMSProp 的优化算法，其结合了两者的优点。

Adam 算法

Adam 算法会使用一个动量变量 v 和一个 RMSProp 中的梯度元素平方的移动指数加权平均 s，首先将他们全部初始化为 0，然后在每次迭代中，计算他们的移动加权平均进行更新

$$\begin{aligned} v &= \beta_1 v + (1-\beta_1)g \\ s &= \beta_2 s + (1-\beta_2)g^2 \end{aligned} \tag{16}$$

在 adam 算法里，为了减轻 v 和 s 被初始化为 0 的初期对计算指数加权移动平均的影响，每次 v 和 s 都做下面的修正

$$\begin{aligned} \hat{v} &= \frac{v}{1-\beta_1^t} \\ \hat{s} &= \frac{s}{1-\beta_2^t} \end{aligned} \tag{17}$$

这里 t 是迭代次数，可以看到，当 $0 \le \beta_1, \beta_2 \le 1$ 的时候，迭代到后期 t 比较大，那么 β_1^t 和 β_2^t 就几乎为 0，就不会对 v 和 s 有任何影响了，算法作者建议$\beta_1 = 0.9$, $\beta_2 = 0.999$。

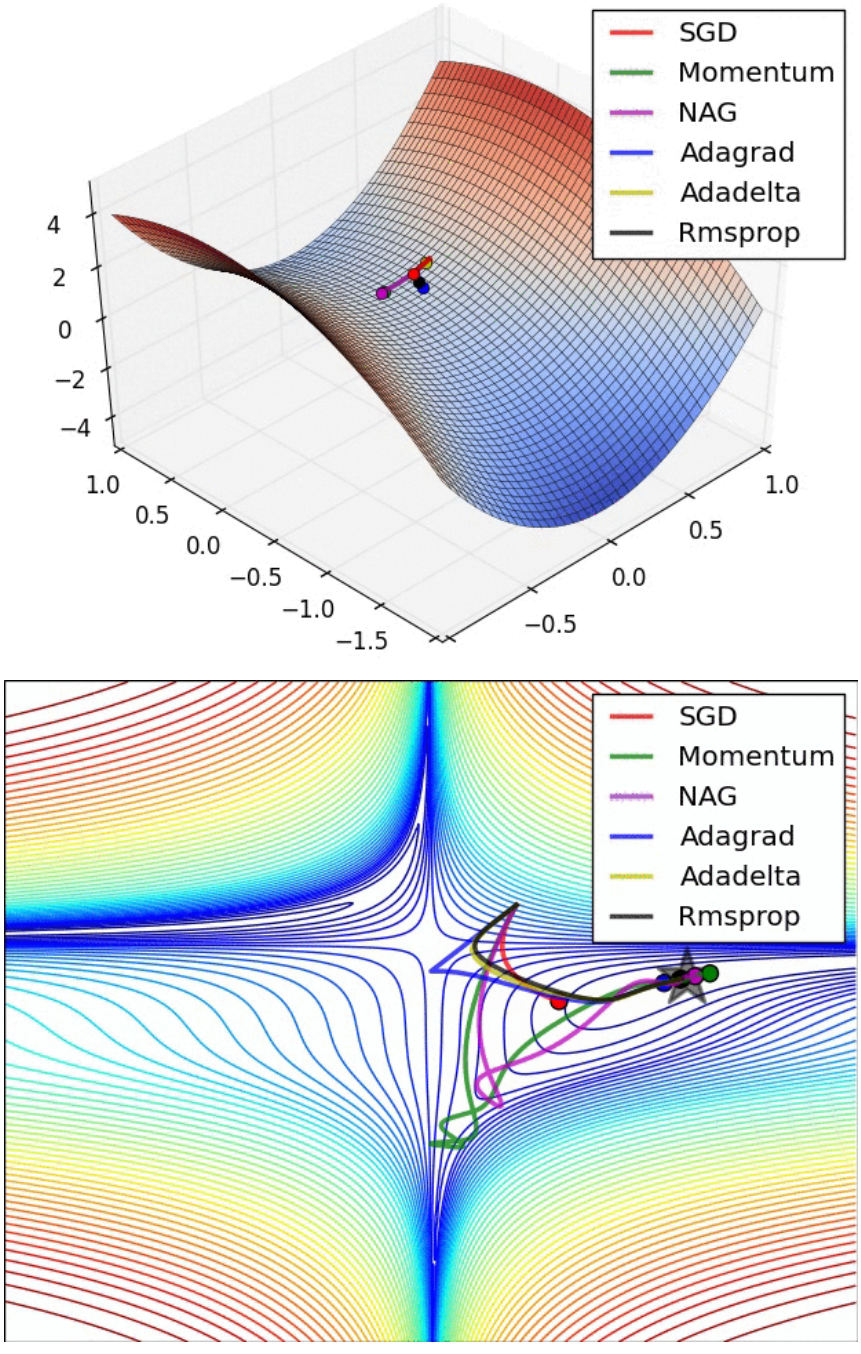
最后使用修正之后的 $\hat{v}$ 和 $\hat{s}$ 进行学习率的重新计算

$$g' = \frac{\eta \hat{v}}{\sqrt{\hat{s} + \epsilon}} \tag{18}$$

这里 η 是学习率，$epsilon$ 仍然是为了数值稳定性而添加的常数，最后参数更新有

$$\theta_i = \theta_{i-1} - g' \tag{19}$$

这是我们讲的最后一个优化算法，下面放一张各个优化算法的对比图结束这一节的内容



这两张图生动形象地展示了各种优化算法的实际效果