Gradient Descent Optimization Notes

1. **优化算法框架**

   首先定义：待优化参数 w，目标函数 f(w)，初始学习率α。

   而后，开始进行迭代优化。在每个 epoch t ：

   （1） 计算目标函数关于当前参数的梯度： $g_t=\nabla f(w_t)$

   （2） 根据历史梯度计算一阶动量和二阶动量：

   $m_t=\varphi(g_1,g_2,\ldots g_t)$, $v_t=\psi(g_1,g_2,\ldots g_t)$

   一阶动量 $m_t=\beta_1\cdot m_{t-1}+(1-\beta_1)\cdot g_t$ 是各个时刻梯度方向的指数移动平均值，约等于最近 $1/(1-\beta_1)$ 个时刻的梯度向量和的平均值。

   （3） 计算当前时刻的下降梯度：$\eta_t=\alpha\cdot m_t/\sqrt{\mathbf{v_t}}$

   （4） 根据下降梯度进行更新：$w_{t+1}=w_t-\eta_t$

2. **Gradient Descent**

   - **Batch Gradient Descent**

   $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

   在整个训练集上计算梯度

   - **Stochastic Gradient Descent**

   $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

   每个训练样本更新一次参数

   - **Mini-batch Gradient Descent**

   $$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

   每 n 个样本更新一次参数

   - **SGD 缺点**

   下降速度慢，而且可能会在沟壑的两边持续震荡，停留在一个局部最优点。

3. **Momentum**

   - **公式**

   $$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
   $$\theta = \theta - v_t$$

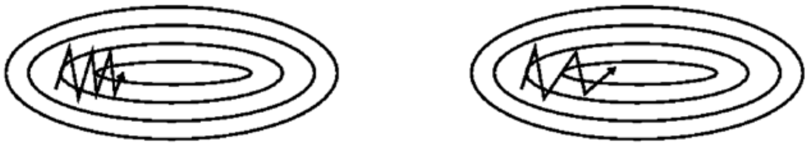
对于当前 batch 求得的梯度，其中如果某个维度梯度的方向和上次更新梯度的方向(v_{t-1})相同，则这次更新速度会更快，反之更新的会更慢，这样可以减少上面那种摆动的情况。

- **思想**

为了抑制 SGD 的震荡，SGDM 认为梯度下降过程可以加入惯性。下坡的时候，如果发现是陡坡，那就利用惯性跑的快一些。下降方向主要是此前累积的下降方向，并略微偏向当前时刻的下降方向。想象高速公路上汽车转弯，在高速向前的同时略微偏向，急转弯可是要出事的。

一般梯度下降方法更新的是位置，或者说时位移，通俗的说就是在这个点还没达到最优值，那我沿着负梯度方向迈一步试试；而 momentum update 更新的是速度，通俗说就是这个点没达到最优值，那我沿着负梯度方向走快点试试，然后再更新位移。这里超参数γ是为了保证系统最后收敛到局部值，比如我现在快到局部最小点了，因此速度更新量越来越小（梯度接近于 0），但是速度有啊，有速度就会继续走，因此加入小于 1 的γ使每次迭代后速度降下来，最后为 0 也就不走了。γ通常设置为 0.9。

- **示意图**



(a) SGD without momentum　　(b) SGD with momentum

4. **Nesterov Accelerated Gradient**

- **思想**

SGD 还有一个问题是困在局部最优的沟壑里面震荡。想象一下你走到一个盆地，四周都是略高的小山，你觉得没有下坡的方向，那就只能待在这里了。可是如果你爬上高地，就会发现外面的世界还很广阔。因此，我们不能停留在当前位置去观察未来的方向，而要向前一步、多看一步、看远一些。

- **公式**

$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

我们使用γv_{t-1}来更新参数θ，因此计算$\theta-\gamma v_{t-1}$可以大概估计出下一次要更新参数的位置，这个公式的意思是我们用上一步的梯度先走一步，然后再进行调整。

5. **Adagrad**

- **思想**

此前我们都没有用到二阶动量。二阶动量的出现，才意味着“自适应学习率”优化算法时代的到来。SGD 及其变种以同样的学习率更新每个参数，但深度神经网络往往包含大量的参数，这些参数并不是总会用得到（想想大规模的 embedding）。对于经常更新的参数，我们已经积累了大量关于它的知识，不希望被单个样本影响太大，希望学习速率慢一些；对于偶尔更新的参数，我们了解的信息太少，希望能从每个偶然出现的样本身上多学一些，即学习速率大一些。

怎么样去度量历史更新频率呢？那就是二阶动量——该维度上，迄今为止所有梯度值的平方和：$V_t = \sum_{T=1}^{t} g_T^2$，该算法在稀疏数据场景下表现非常好。

- **公式**

梯度下降：

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

Adagrad：

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

其中 G_t 是对之前所有梯度平方和的累计

- **缺点**

随着 t 的增加，分母越来越大，导致后期学习率几乎为 0。

6. **RMSprop**

- **优点**

解决 Adagrad 学习率不断变小的问题。

- **公式**

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

公式中的 $E[g^2]_t$ 是对梯度平方的指数移动平均值

7. **Adam**

- **公式**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t和 v_t初始化为 0 向量，β1=0.9， β2=0.999，前者是

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

$\varepsilon=10^{-8}$