目录

1 Optimization

神经网络的优化方法有很多，以下是常用的优化方法。

1.1 GD: Gradient Descending

GD 方法是梯度下降法，假设神经网络 $f(x;W)$ 接受输入 x ，产生输出 y ，损失函数 $l(x)$ 计算样本的 loss，则计算完损失函数值之后，我们使用梯度来更新神经网络的参数 W ，即：

$$\nabla_W l(x) = \frac{\partial l}{\partial W} \tag{1}$$

其中，损失函数对各可学习参数的导数由链式法则与反向传播算法确定，则，我们可以更新神经网络的权重：

$$W_t \leftarrow W_{t-1} - \eta \nabla_W l(x)$$

其中，η 为超参数学习率。

1.2 SGD: Stochastic Gradient Descending

现实中，训练的神经网络往往需要接受大量的数据，如果先计算所有的样本再计算输入数据对权重的梯度，其时间复杂度为 $O(n)$ 会随着输入数据的规模 n 呈线性增长，计算代价往往比较大，这时候可以使用随机梯度下降，每次随机选取一个样本。我们将第 i 个样本的损失函数值表示为：$l_i(x)$，那么对应于这个样本的权重梯度可以写为： $\nabla_W l_i(x)$，该样本对神经网络参数更新的贡献为：

$$W_t \leftarrow W_{t-1} - \eta \nabla_W l_i(x)$$

由于总的梯度改变量为：$E_i[\nabla_W l_i(x)] = \frac{1}{N}\sum_i \nabla_W l_i(x) = \nabla_W l(x)$ 故随机梯度下降梯度估计方式可以很好地估计梯度下降的参数。

1.3 Mini-Batch Gradient Descending

实际上，我们通常随机选取一个小批量，利用该小批量的损失函数值更新神经网络的权重参数，若我们记小批量为 B_t，则该小批量的权重梯度可以写为：$\sum_{B_t} \nabla_W l_i(x)$，对应的，该小批量对神经网络的参数更新的贡献为：

$$W_t \leftarrow W_{t-1} - \eta \frac{1}{|B_t|} \sum_{B_t} \nabla_W l_i(x)$$

其中，η 为超参数学习率，$|B_t|$ 为小批量的大小，当 $|B_t| = 1$ 时，等价于随机梯度下降，当 $|B_t|$ 与输入样本数相等时，等价于梯度下降。当批量较小时，每次迭代中使用的样本少，这会导致并行处理和内

存使用效率变低。这使得在计算同样数目样本的情况下比使用更大批量时所花时间更多。当批量较大时，每个小批量梯度里可能含有更多的冗余信息。为了得到较好的解，批量较大时比批量较小时需要计算的样本数目可能更多，例如增大迭代周期数。除此之外，SGD / Mini-Batch SGD 还存在着优化速度慢的问题，如图 1 所示，当被优化函数在某一点处比较“陡峭”时，由于沿着“陡峭”的方向梯度大，那么更新步长就相应会变大，容易“过度优化”，越过最低点，导致优化缓慢。

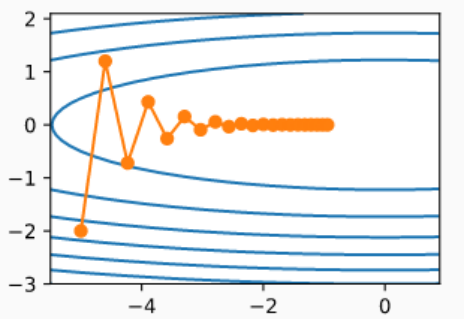

图 1: SGD 算法可能会“过度优化”

1.4 SGD With Momentum

该方法吸收了物理学上动量的思想，由于我们可以将随机梯度下降视为小球沿着山谷下降，而真实的物理过程中小球的下降是会遵循相关的物理规律的，吸收其中的动量的思想，可以帮助我们更快地优化目标函数，使之达到足够满意的局部最优点。

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta g_t \\ W_t &= W_{t-1} - v_t \end{aligned} \tag{2}$$

式中，v_t 为第 t 次更新参数时的动量值，初始时设为0，g_t 为参数的梯度，η 为学习率（超参数） W_t 为第 t 次更新后的权重值。动量法可以被视为引入了指数平均移动均值，即，将 v_t 改写为

$$v_t = \gamma v_{t-1} + (1-\gamma)\frac{\eta}{(1-\gamma)} g_t$$

，即，自变量在各个方向的移动幅度还与过去 $\frac{1}{1-\gamma}$ 项梯度有关，其方向会影响梯度下降的方向。

然而到此为止的优化算法均为不同的维度提供相同的学习率，实际中被优化函数不同维度可能需要不同的学习率，以下的自适应优化方法便被提出来，试图解决这一问题。

1.5 AdaGrad

AdaGrad 能够根据不同元素的当前梯度调整不同元素的学习率，该方法可以看成基于动量法的随机梯度下降的修改。

$$\begin{aligned} s_t &= s_{t-1} + g \odot g \\ W_t &= W_{t-1} - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g \end{aligned} \tag{3}$$

式中，ϵ 是一个用于维持数值稳定性的很小的常数，$\odot$ 表示按元素相乘。利用按元素的运算，我们可以把 $\frac{\eta}{\sqrt{s_t+\epsilon}}$ 视为新的学习率，这样不同的变量就具有不同的学习率。由于式中 s_t 随着更新过程的不断进行，

其每一个元素的梯度会逐渐变大，因此当我们把 $\frac{\eta}{\sqrt{s_t+\epsilon}}$ 视为新的学习率的时候，由于 s_t 不断累积，我们可以发现其梯度会不断减小，且对于梯度越大的参数，其学习率减小越快。虽然它能够自动调整学习率，但是由于其累计效应，当迭代早期下降较快且当前仍然解依然不佳时，可能无法找到很好的解。鉴于此，改进版的自适应优化方法提出只使用最近几项梯度用于自适应调整学习率，如 RMSProp, AdaDelta。

1.6 RMSProp

RMSProp 算法是对 AdaGrad 算法的改进，其中变量 s_t 的更新方法改为指数加权移动平均，即：

$$\begin{aligned} s_t &= \gamma s_{t-1} + (1-\gamma) g \odot g \\ W_t &= W_{t-1} - \frac{\eta}{\sqrt{s_t+\epsilon}} \odot g \end{aligned} \tag{4}$$

由于增加了指数加权移动平均，可以认为是最近 $\frac{1}{(1-\gamma)}$ 个小批量的随机梯度的平方项的加权平均，如此一来，自变量每一个元素就不再始终降低（或不变）。RMSProp 算法跟 AdaGrad 算法相比，可以更快地逼近最优解。

1.7 AdaDelta

AdaDelta 算法与 RMSProp 算法基本一致，不同的是，AdaDelta 算法将学习率 η 替换为 $\Delta W_{t-1}+\epsilon$，其中，$\Delta W_t = \rho \Delta W_{t-1} + (1-\rho) g' \odot g', g' = \sqrt{\frac{\Delta W_{t-1}+\epsilon}{s_t+\epsilon}} \odot g_t$，因此，该算法实际上没有学习率这一个参数，它通过使用有关自变量更新量平方的指数加权移动平均的项来替代RMSProp算法中的学习率。