1. 权重衰减（weight decay）

L2正则化的目的就是为了让权重衰减到更小的值，在一定程度上减少模型过拟合的问题，衰减也叫L2正则化。

1.1 L2正则化与权重衰减系数

L2正则化就是在代价函数后面再加上一个正则化项：

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

其中 C_0 代表原始的代价函数，后面那一项就是L2正则化项，它是这样来的：所有参数w的平方的和，除以训练集的样本大小n。λ就是正则项系数，权衡正则项与 C_0 项的比重。另外还有一个系数 $\frac{1}{2}$ ， $\frac{1}{2}$ 经常会看到，主要是为了后面求导的结果方便，后面那一项求导会产生一个2，与 $\frac{1}{2}$ 相乘刚好凑整为1。系数 λ 就是**权重衰减系数**。

1.2 为什么可以对权重进行衰减

我们对加入L2正则化后的代价函数进行推导，先求导：

$$\begin{aligned}\frac{\partial C}{\partial w} &= \frac{\partial C_0}{\partial w} + \frac{\lambda}{n}w \\ \frac{\partial C}{\partial b} &= \frac{\partial C_0}{\partial b}.\end{aligned}$$

可以发现L2正则化项对b的更新没有影响，但是对于w的更新有影响：

$$
\begin{aligned}
w &\rightarrow w-\eta \frac{\partial C_0}{\partial w}-\frac{\eta \lambda}{n} w \\
&=\left(1-\frac{\eta \lambda}{n}\right) w-\eta \frac{\partial C_0}{\partial w}.
\end{aligned}
$$

在不使用L2正则化时，求导结果中w前系数为1，现在w前面系数为 $1-\frac{\eta \lambda}{n}$ ，因为η、λ、n都是正的，所以 $1-\frac{\eta \lambda}{n}$ 小于1，它的效果是减小w，**这也就是权重衰减（weight decay）的由来。**当然考虑到后面的导数项，w最终的值可能增大也可能减小。

另外，需要提一下，对于基于mini-batch的随机梯度下降，w和b更新的公式跟上面给出的有点不同：

$$
w \rightarrow\left(1-\frac{\eta \lambda}{n}\right) w-\frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial w}
$$

$$
b \rightarrow b-\frac{\eta}{m} \sum_x \frac{\partial C_x}{\partial b}
$$

1.3 权重衰减（L2正则化）的作用

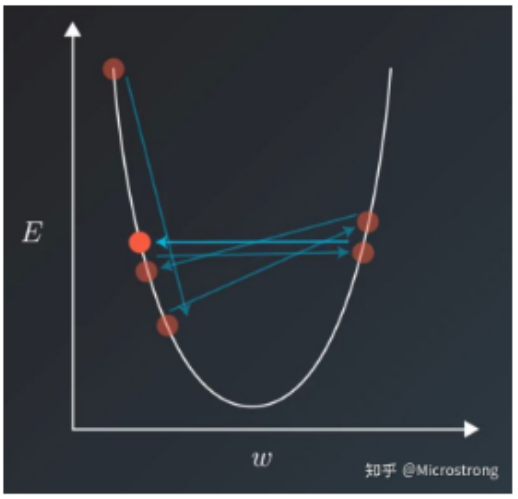
作用：权重衰减（L2正则化）可以避免模型过拟合问题。

思考：L2正则化项有让w变小的效果，但是为什么w变小可以防止过拟合呢?

原理：（1）从模型的复杂度上解释：更小的权值w，从某种意义上说，表示网络的复杂度更低，对数据的拟合更好（这个法则也叫做奥卡姆剃刀），而在实际应用中，也验证了这一点，L2正则化的效果往往好于未经正则化的效果。（2）从数学方面的解释：过拟合的时候，拟合函数的系数往往非常大，为什么？如下图所示，过拟合，就是拟合函数需要顾忌每一个点，最终形成的拟合函数波动很大。在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。而正则化是通过约束参数的范数使其不要太大，所以可以在一定程度上减少过拟合情况。

2. 学习率衰减（learning rate decay）

在训练模型的时候，通常会遇到这种情况：我们平衡模型的训练速度和损失（loss）后选择了相对合适的学习率（learning rate），但是训练集的损失下降到一定的程度后就不在下降了，比如training loss一直在0.7和0.9之间来回震荡，不能进一步下降。如下图所示：



遇到这种情况通常可以通过适当降低学习率（learning rate）来实现。但是，降低学习率又会延长训练所需的时间。

学习率衰减（learning rate decay）就是一种可以平衡这两者之间矛盾的解决方案。学习率衰减的基本思想是：学习率随着训练的进行逐渐衰减。

学习率衰减基本有两种实现方法：

1. 线性衰减。例如：每过5个epochs学习率减半。
2. 指数衰减。例如：随着迭代轮数的增加学习率自动发生衰减，每过5个epochs将学习率乘以0.9998。具体算法如下：

decayed_learning_rate=learning_rate*decay_rate^(global_step/decay_steps)

其中decayed_learning_rate为每一轮优化时使用的学习率，learning_rate为事先设定的初始学习率，decay_rate为衰减系数，decay_steps为衰减速度。