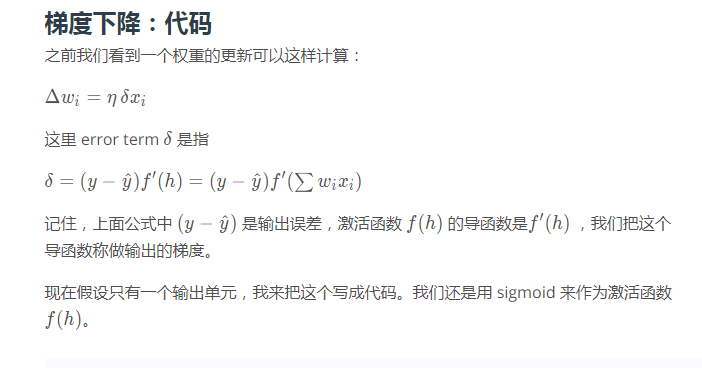
**1、标准梯度下降**

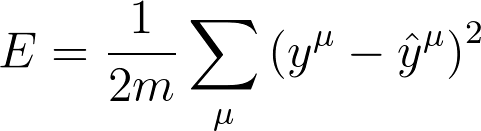


**2、数据清理**

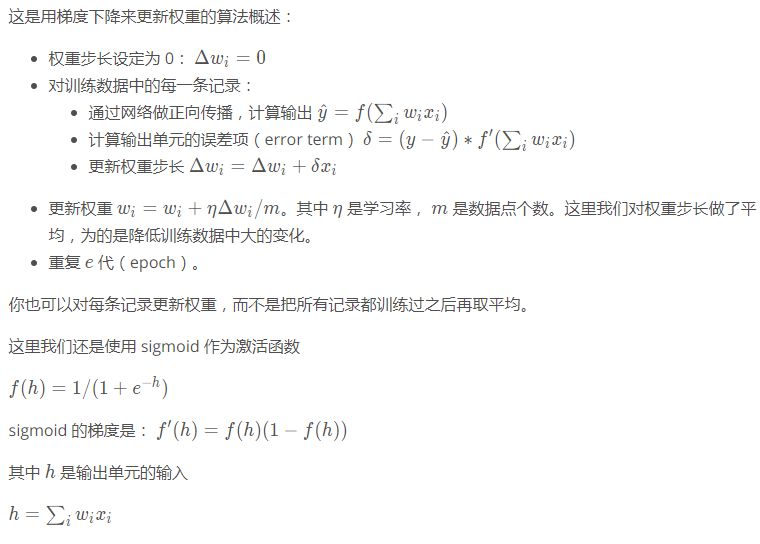
**我们还需要把输入数据标准化，也就是说使得它们的均值为 0，标准偏差为 1。因为 sigmoid 函数会挤压很大或者很小的输入，所以这一步是必要的。很大或者很小输入的梯度为 0，这意味着梯度下降的步长也会是 0**。由于可能输入的数值都相当大，我们在初始化权重的时候需要非常小心，否则梯度下降步长将会消失，网络也没法训练了。相对地，如果我们对数据做了标准化处理，就能更容易地对权重进行初始化。

### **均方差**

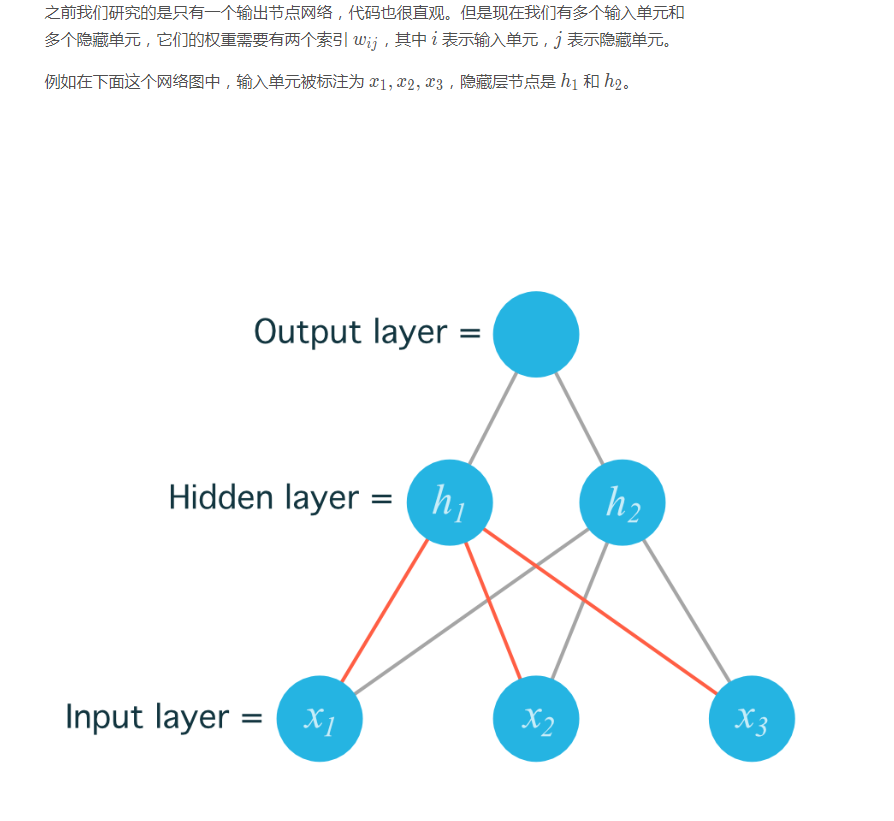
这里我们要对如何计算误差做一点小改变。我们不计算 SSE，而是用误差平方的均值（mean of the square errors，MSE）。现在我们要处理很多数据，把所有权重更新加起来会导致很大的更新，使得梯度下降无法收敛。为了避免这种情况，你需要一个很小的学习率。这里我们还可以除以数据点的数量 mm 来取平均。这样，无论我们有多少数据，我们的学习率通常会在 0.01 to 0.001 之间。我们用 MSE（下图）来计算梯度，结果跟之前一样，只是取了平均而不是取和。

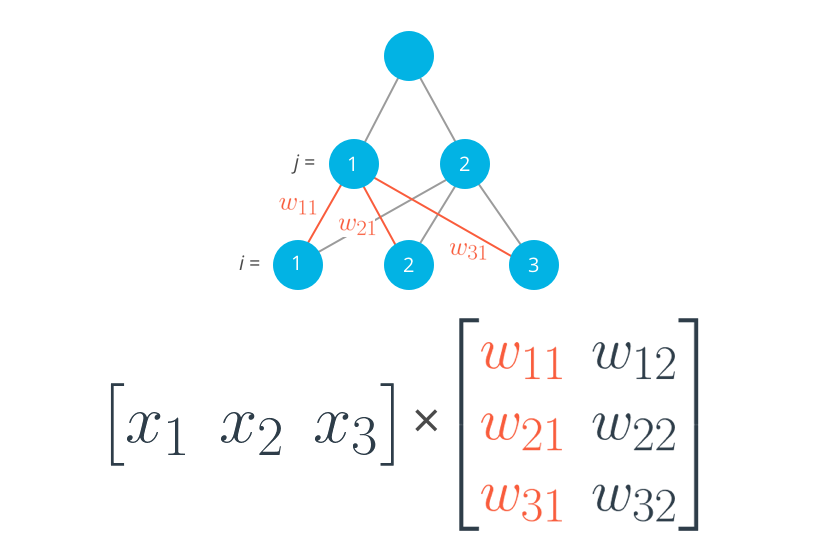
**[](https://classroom.udacity.com/nanodegrees/nd101-cn-advanced/parts/484efacf-1085-4481-9218-982074a4627c/modules/675d02b4-7881-4c86-8df8-7a6b3e12bbce/lessons/0e07fafa-e796-4fab-b119-13f47f1d5c1b/concepts/4b167ce0-9d45-45e1-bfe6-891b2c68ac94)**

这是用梯度下降来更新权重的算法概述：

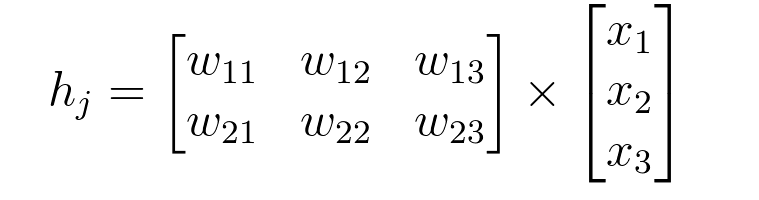


1. **多层神经网络的梯度下降法**





或者统一将每层的输入输出写成列向量



# **反向传播**

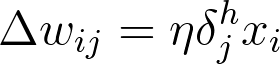
如何让多层神经网络****学习****呢？我们已了解了使用梯度下降来更新权重，反向传播算法则是它的一个延伸。以一个两层神经网络为例，可以使用****链式法则****计算输入层-隐藏层间权重的误差。

要使用梯度下降法更新隐藏层的权重，你需要知道各隐藏层节点的误差对最终输出的影响。每层的输出是由两层间的权重决定的，两层之间产生的误差，按权重缩放后在网络中向前传播。既然我们知道输出误差，便可以用权重来反向传播到隐藏层。

例如，输出层每个输出节点 k 的误差是 *δko*​ ，**隐藏节点 j的误差即为输出误差乘以输出层-隐藏层间的权重矩阵（以及梯度）。**



然后，梯度下降与之前相同，只是用新的误差：

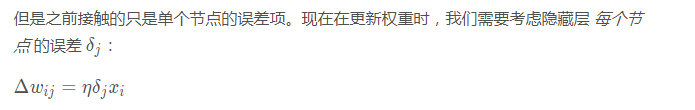


其中wij是输入和隐藏层之间的权重，xi 是输入值**。这个形式可以表示任意层数。权重更新步长等于步长乘以层输出误差再乘以该层的输入值。**

IMG_256

现在，你有了输出误差，delta output，便可以反向传播这些误差了。V\_{in} 是该层的输入，比如经过隐藏层激活函数的输出值。

sigmoid 函数导数的最大值是 0.25，因此输出层的误差被减少了至少 75%，隐藏层的误差被减少了至少 93.75%！如果你的神经网络有很多层，使用 sigmoid 激活函数会很快把靠近输入层的权重步长降为很小的值，该问题称作****梯度消失****。后面的课程中你会学到在这方面表现更好，也被广泛用于最新神经网络中的其它激活函数。

[](https://classroom.udacity.com/nanodegrees/nd101-cn-advanced/parts/484efacf-1085-4481-9218-982074a4627c/modules/675d02b4-7881-4c86-8df8-7a6b3e12bbce/lessons/0e07fafa-e796-4fab-b119-13f47f1d5c1b/concepts/87d85ff2-db15-438b-9be8-d097ea917f1e)

