### 3.11 随机初始化（Random Initialization）

当你训练神经网络时，权重随机初始化是很重要的。对于逻辑回归，把权重初始化为0当然也是可以的。但是对于一个神经网络，如果你把权重或者参数都初始化为0，那么梯度下降将不会起作用。

让我们看看这是为什么。有两个输入特征，，2个隐藏层单元就等于2。 因此与一个隐藏层相关的矩阵，或者说是2\*2的矩阵，假设把它初始化为0的2\*2矩阵，也等于 ，把偏置项初始化为0是合理的，但是把初始化为0就有问题了。 那这个问题如果按照这样初始化的话，你总是会发现 和 相等，这个激活单元和这个激活单元就会一样。因为两个隐含单元计算同样的函数，当你做反向传播计算时，这会导致 和 也会一样，对称这些隐含单元会初始化得一样，这样输出的权值也会一模一样，由此等于；

图片包含 文字

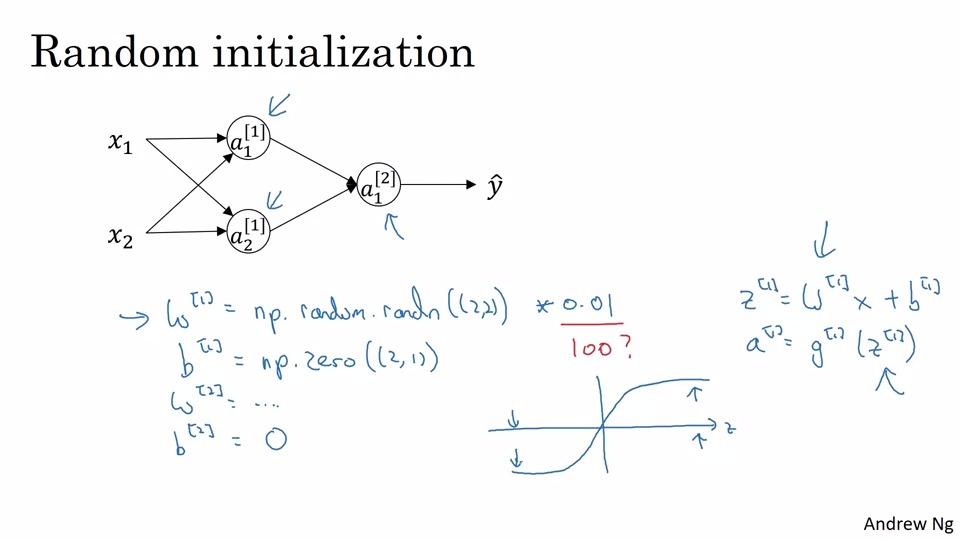
描述已自动生成

图3.11.1 但是如果你这样初始化这个神经网络，那么这两个隐含单元就会完全一样，因此他们完全对称，也就意味着计算同样的函数，并且肯定的是最终经过每次训练的迭代，这两个隐含单元仍然是同一个函数，令人困惑。会是一个这样的矩阵，每一行有同样的值因此我们做权重更新把权重每次迭代后的，第一行等于第二行。

由此可以推导，如果你把权重都初始化为0，那么由于隐含单元开始计算同一个函数，所有的隐含单元就会对输出单元有同样的影响。一次迭代后同样的表达式结果仍然是相同的，即隐含单元仍是对称的。通过推导，两次、三次、无论多少次迭代，不管你训练网络多长时间，隐含单元仍然计算的是同样的函数。因此这种情况下超过1个隐含单元也没什么意义，因为他们计算同样的东西。当然更大的网络，比如你有3个特征，还有相当多的隐含单元。

如果你要初始化成0，由于所有的隐含单元都是对称的，无论你运行梯度下降多久，他们一直计算同样的函数。这没有任何帮助，因为你想要两个不同的隐含单元计算不同的函数，这个问题的解决方法就是随机初始化参数。你应该这么做：把设为np.random.randn(2,2)(生成高斯分布)，通常再乘上一个小的数，比如0.01，这样把它初始化为很小的随机数。然后没有这个对称的问题（叫做**symmetry breaking problem**），所以可以把 初始化为0，因为只要随机初始化你就有不同的隐含单元计算不同的东西，因此不会有**symmetry breaking**问题了。相似的，对于你可以随机初始化，可以初始化为0。

你也许会疑惑，这个常数从哪里来，为什么是0.01，而不是100或者1000。我们通常倾向于初始化为很小的随机数。因为如果你用**tanh**或者**sigmoid**激活函数，或者说只在输出层有一个**Sigmoid**，如果（数值）波动太大，当你计算激活值时如果很大，就会很大。的一些值就会很大或者很小，因此这种情况下你很可能停在**tanh**/**sigmoid**函数的平坦的地方(见图3.8.2)，这些地方梯度很小也就意味着梯度下降会很慢，因此学习也就很慢。



回顾一下：如果很大，那么你很可能最终停在（甚至在训练刚刚开始的时候）很大的值，这会造成**tanh**/**Sigmoid**激活函数饱和在龟速的学习上，如果你没有**sigmoid**/**tanh**激活函数在你整个的神经网络里，就不成问题。但如果你做二分类并且你的输出单元是**Sigmoid**函数，那么你不会想让初始参数太大，因此这就是为什么乘上0.01或者其他一些小数是合理的尝试。对于一样，就是np.random.randn((1,2))，我猜会是乘以0.01。

事实上有时有比0.01更好的常数，当你训练一个只有一层隐藏层的网络时（这是相对浅的神经网络，没有太多的隐藏层），设为0.01可能也可以。但当你训练一个非常非常深的神经网络，你可能会选择一个不同于的常数而不是0.01。下一节课我们会讨论怎么并且何时去选择一个不同于0.01的常数，但是无论如何它通常都会是个相对小的数。

好了，这就是这周的视频。你现在已经知道如何建立一个一层的神经网络了，初始化参数，用前向传播预测，还有计算导数，结合反向传播用在梯度下降中。