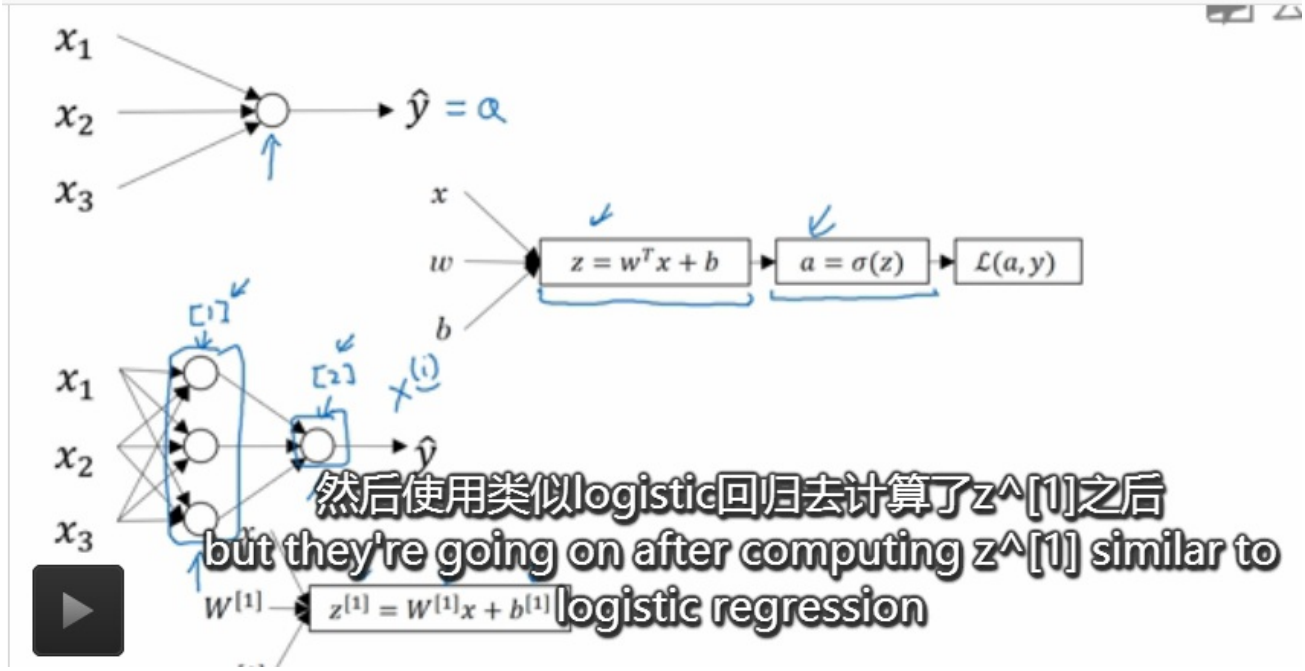
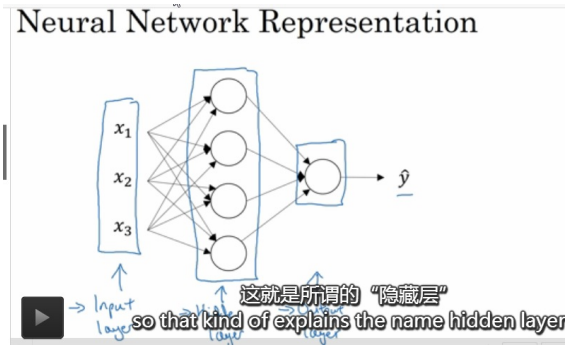
第一篇第三周

3.1、如何实现神经网络：



从上图可以看出左下角的是一个神经网络的结构，矩形框的是它的层数，有两层，x是输入，w和b是参数，用线性回归计算z[1]之后，需要使用sigmoid(z[1])计算a[1]，接着你需要用另外一个线性方程计算z[2],接着计算a[2].

3.2、神经网络的表示：



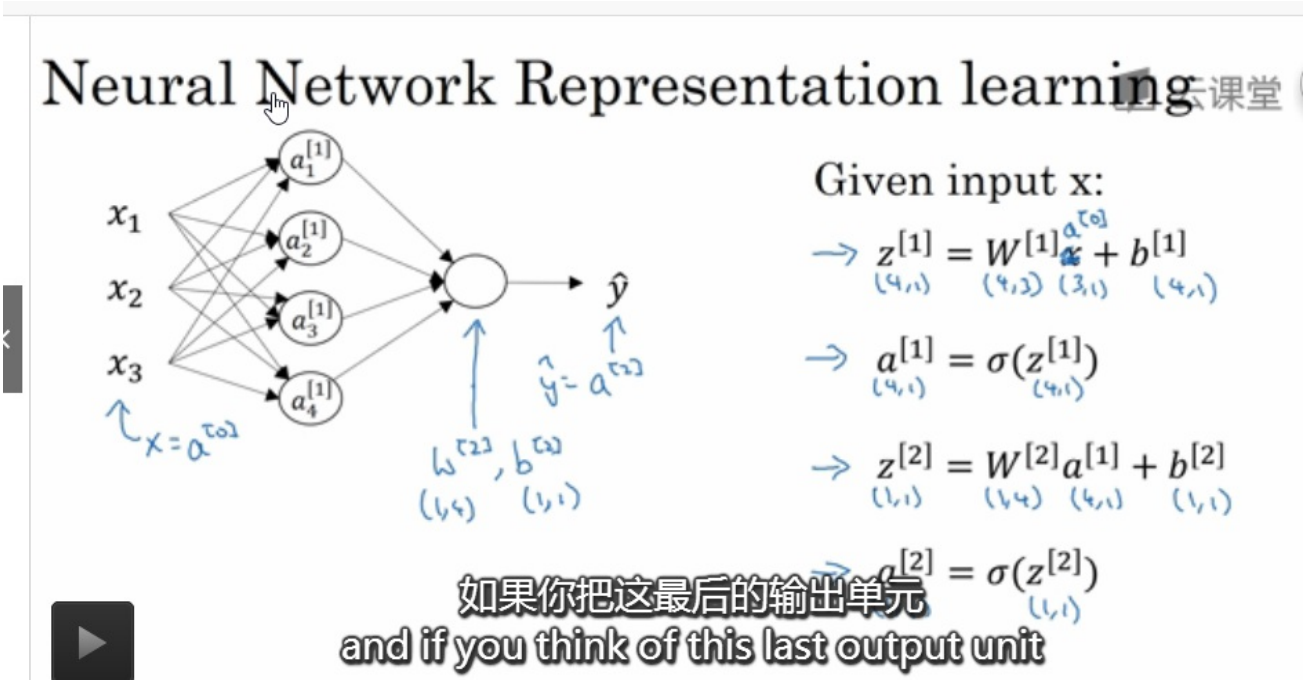
神经网络分为输入层，隐藏层和输出层。

隐藏层：在训练集中，这些中间节点的真正数值，我们是不知道的，在训练集你看不到他们的值，你只能看到输入层和输出层。

计算神经网络的层数，一般不包括输入层，所以上面的是两层神经网络。隐藏层是第一层，输出层是第二层。

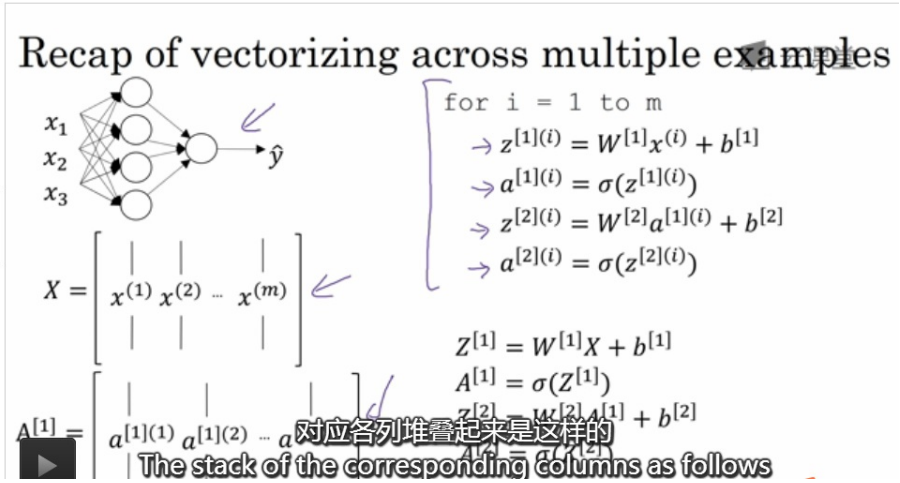
隐藏层的参数维度是3*4的矩阵，输出层的参数矩阵是4*1，与节点数有关。

3.3、神经网络的输出：

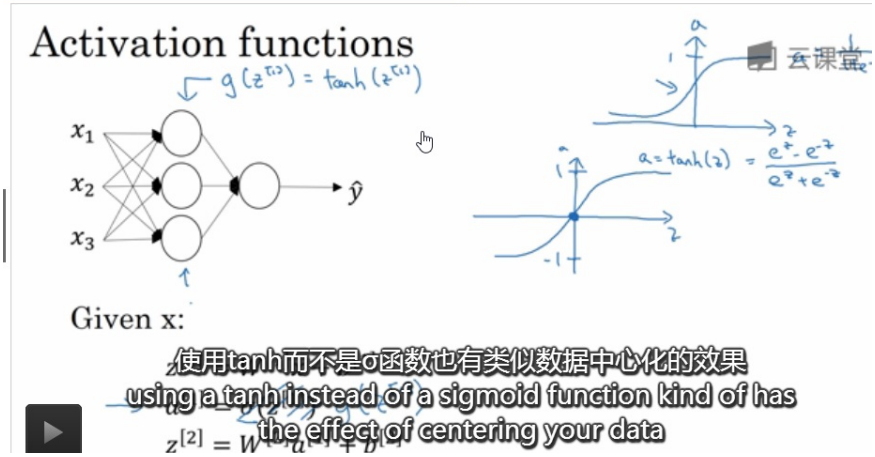


注意上图的矩阵化的维度，一定要能够进行矩阵相乘。通过上面右边的函数转化成python的4行代码，你就可以算出上面神经网络的输出。

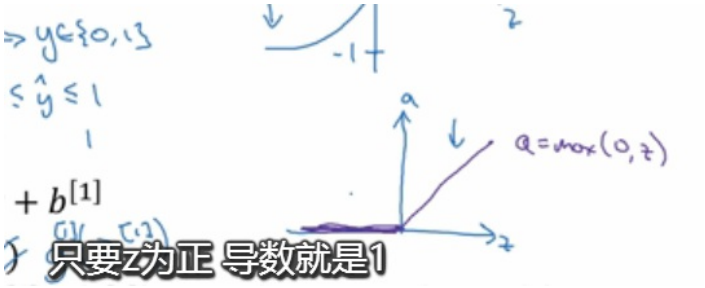
3.4、神经网络向量化：



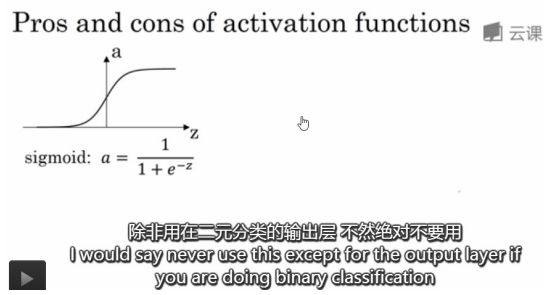
3.6、激活函数

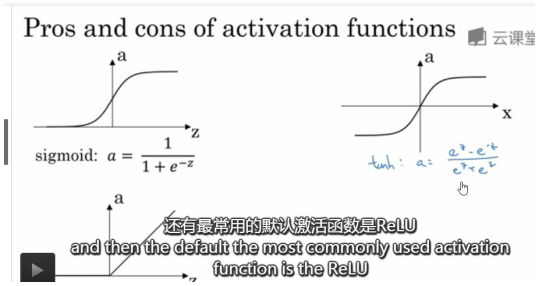


tanh的效果更好。在所有场合都更优越，这两个函数的缺点是当z很大或者很小时，使得导数接近0，这样会拖慢梯度下降算法。为了不拖慢梯度下降算法，可以使用relu函数：



经验：如果输出是0或者1，做二元分类，用第一个函数，作为输出层的激活函数，然后其他所有单元都用relu函数。relu的优点是学习速度非常快。





3.7、为什么要使用非线性的激活函数：

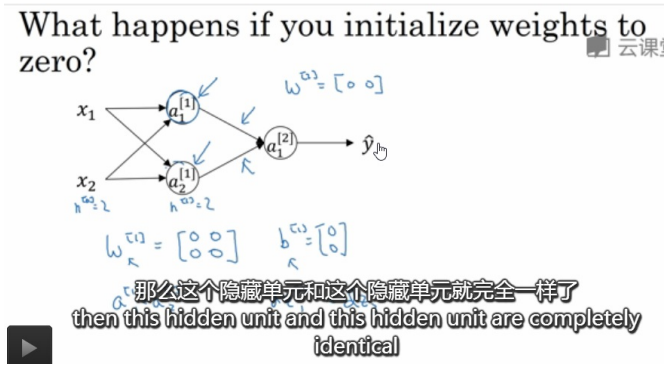
如果使用线性激活函数，那么神经网络只是把输入线性组合再输出，这样子的话无无论你的深度网络多少层，一直在做的只是线性激活函数，所以不如直接去掉隐藏层，线性隐藏层一点都没有用，因为不管多少个线性函数它们的组合仍然是线性函数。

3.8、激活函数的导数

当你对你的神经网络使用反向传播的时候，你需要计算激活函数的导数或者斜率。

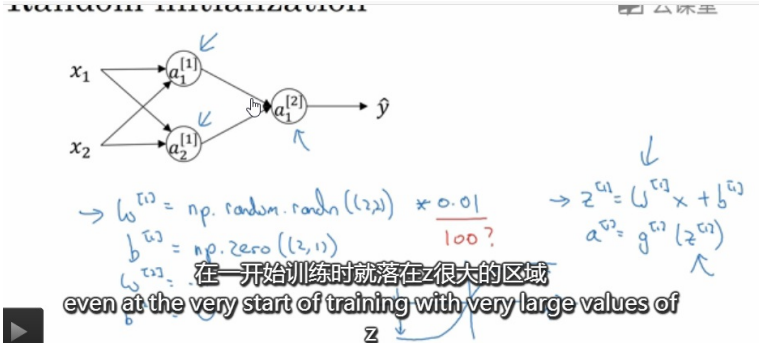
3.9、要训练参数，就要使用梯度下降，在训练神经网络时，随机初始化参数很重要，而不是初始化成全0，全部初始化为0的话，梯度下降算法就会失效。

3.11、



这就是所谓的完全对称。意味着节点计算完全一样的函数，如果你将w值所有初始化为0，那么因为两个隐藏层节点一开始就做同样的计算，两个隐藏层单元对输出单元的影响也一样大，那么在一次迭代之后，同样的对称性依旧存在，两个隐藏层单元仍然是对称的，所以无论你训练多长时间，两个隐藏层单元仍在计算完全一样的函数，所以，多个隐藏层单元真的没有意义，这个问题的解决是随机初始化所有的参数，你可以用w1=np.random.randn([2,2]),b没有这种问题，所以就可以都初始化为0，w初始化为0.0几的数据，是为了防止tanh或者sigmoid函数落在平缓部分，梯度的斜度非常小，也就是说w如果太大，在一开始训练的时候就落在Z很大的区域，导致你的tanh或者sigmoid激活函数接近饱和，从而减慢学习速

度，如果没有tanh或者sigmoid激活函数，就没那么大的问题，因为二分类的时候，有sigmoid函数，所以w不要初始化的太大。0.01是比较合理的，所以初始化参数一般都很小，但是层次很深的就要有大于0.01的初始值，下周再讲，上面结论仅仅针对单层神经网络，如下图：



吴恩达深度学习代码：https://github.com/hejujie/Deep_learning_coursera_Andrew-Ng