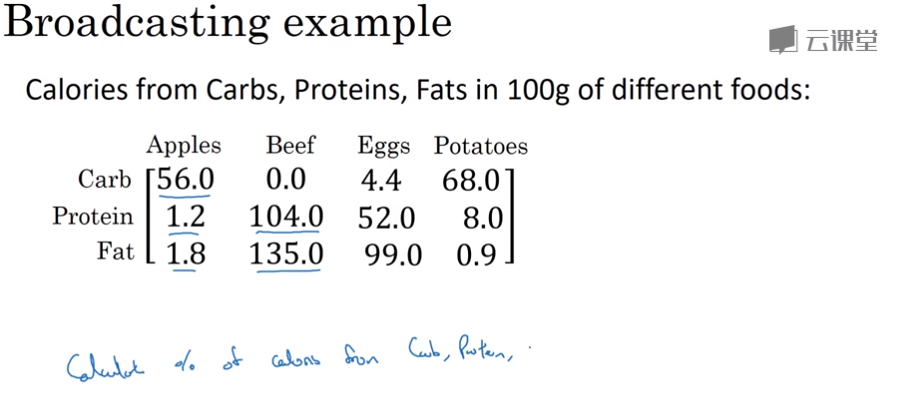
2.15节

广播：一种手段，可以让你的python代码执行的更快。例如以下例子：

以上是四种不同的100g食物中来自于碳水化合物、蛋白质、脂肪的热量，求 四种食物中，来自碳水化合物、蛋白质和脂肪热量的百分比各占多少？

import numpy as np

A = np.array([[56.0,0.0,4.4,68.0],

[1.2,104.0,52.0,8.0],

[1.8,135.0,99.0,0.9]])

print(A)

## axis=0表示矩阵竖方向求和，axis=1表示横方向

cal = A.sum(axis=0)

print(cal)

percentage = 100\*A/cal.reshape(1,4)

print(percentage)

上面这种运算的原理：

例如:一个4x1的矩阵加上一个100，则应该把100扩展为一个4x1的矩阵，即： ,最后得到

**2.16节**

排出写代码时的一些bug：

**import** numpy **as** np   
  
a = np.random.randn(5)  
print(a)  
输出：

[-0.469054 0.0535773 1.52468239 0.82707283 -0.0657255 ]

改进：

**import** numpy **as** np  
a = np.random.randn(5,1)  
print(a)

输出：[[-0.01754468]

[-0.9718692 ]

[-0.49760268]

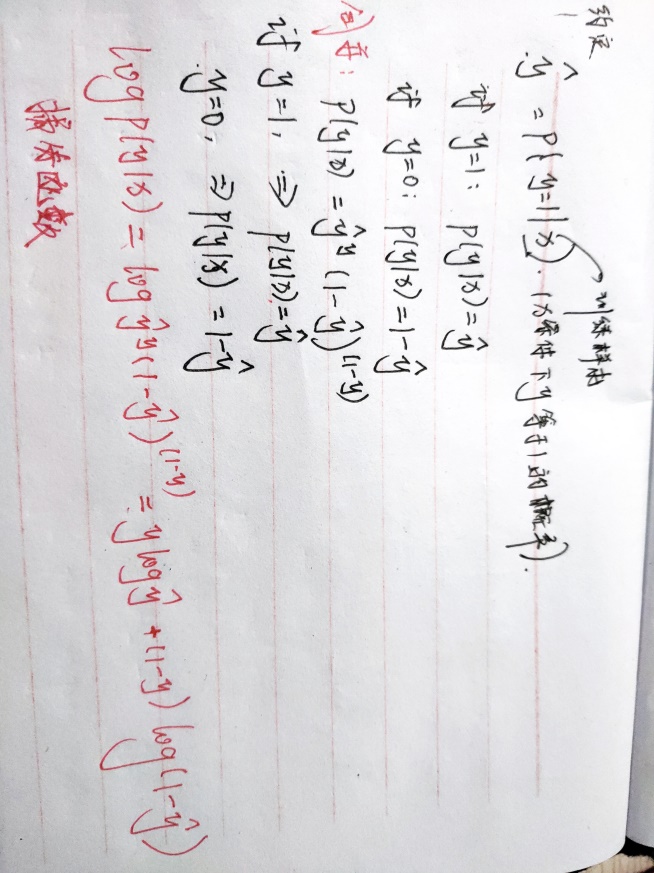
[-0.14504778]

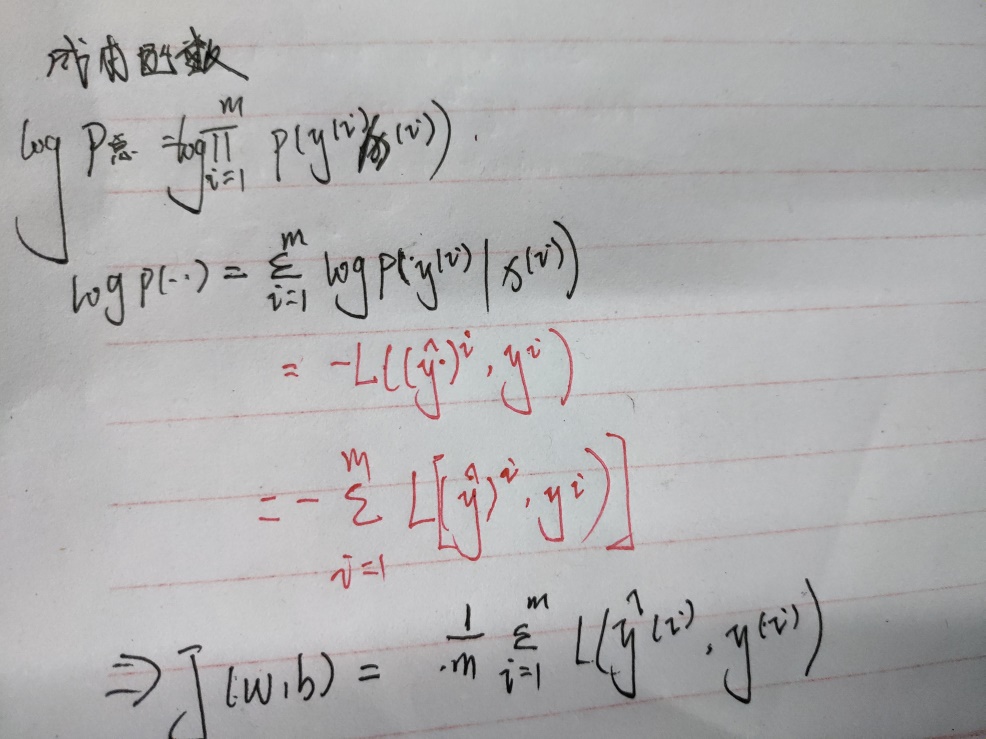
[ 2.5808667 ]]

则输出的是一个5x1的列向量。

2.17节：介绍Coursea中的ipython笔记本使用方法及一些交互命令。

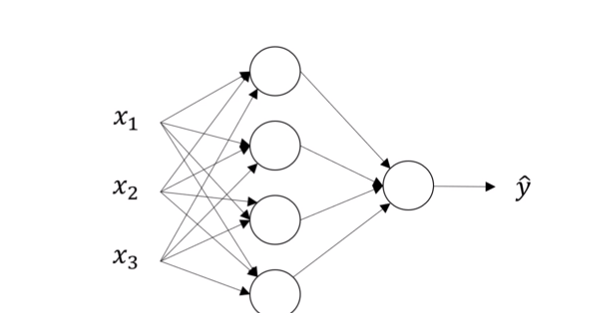
2.18 logistic损失函数的解释

è¿éåå¾çæè¿°è¿éåå¾çæè¿°



3.1 神经网络介绍

1、含一层隐藏层的神经网络

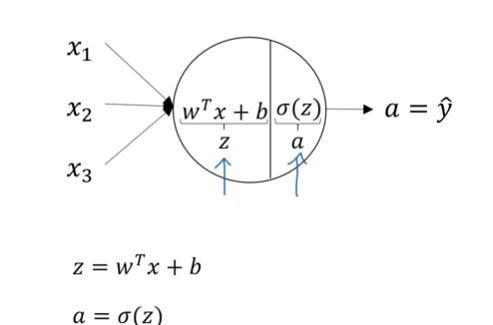


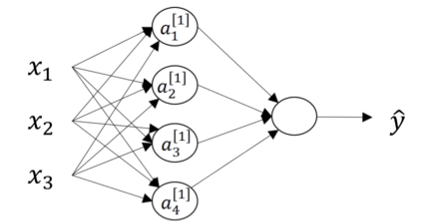
隐藏层

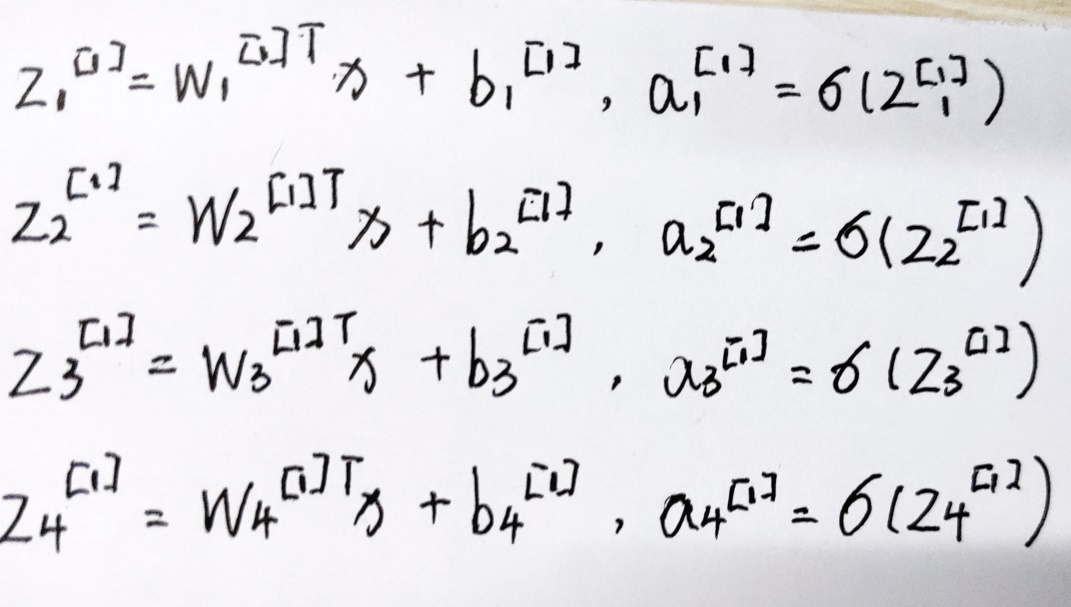
输入层

输出层

2、神经网络的输出

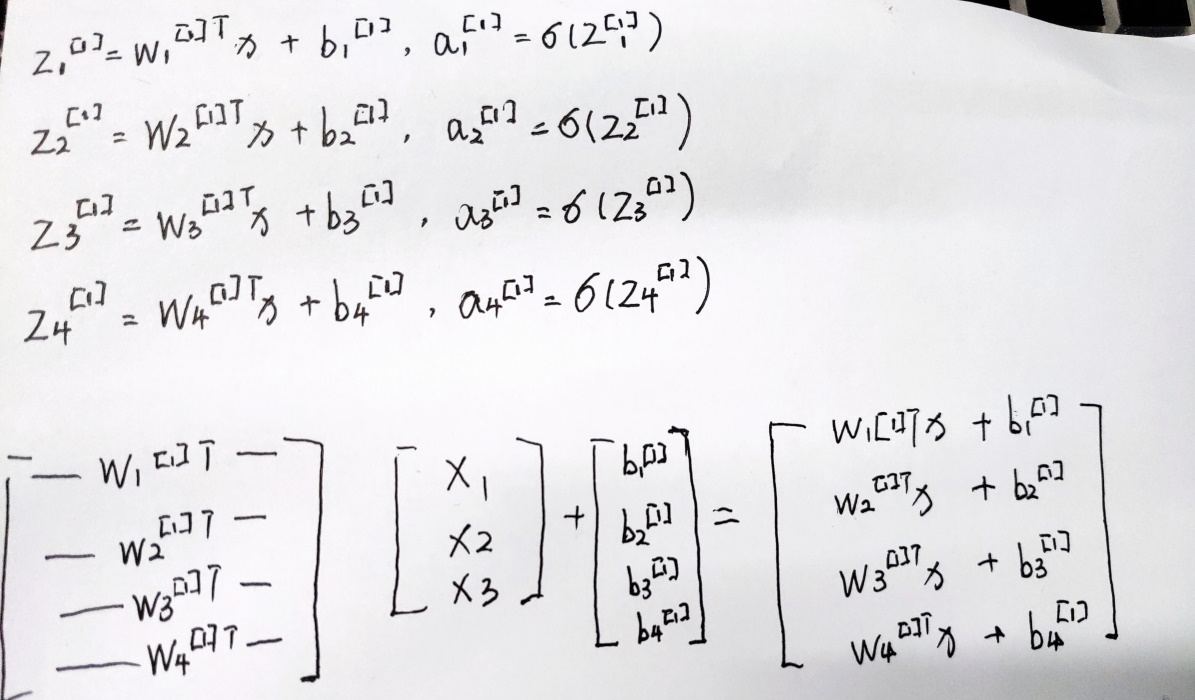




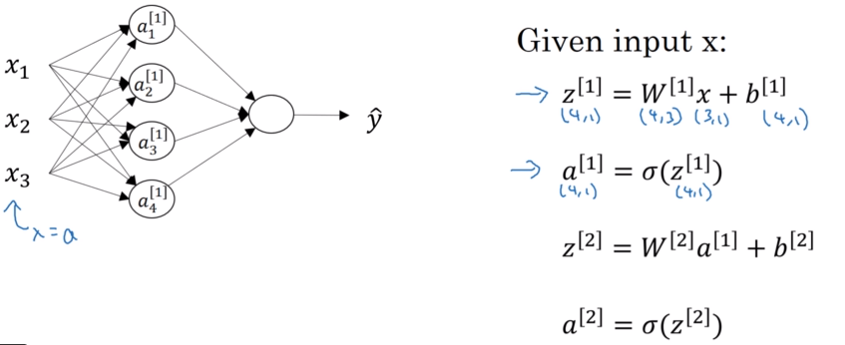


其中WT表示W的转置，接下来把这四个等式向量化，把这些w堆起来，构成一个矩阵，

则：

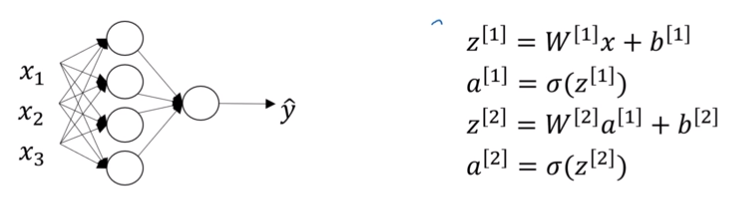
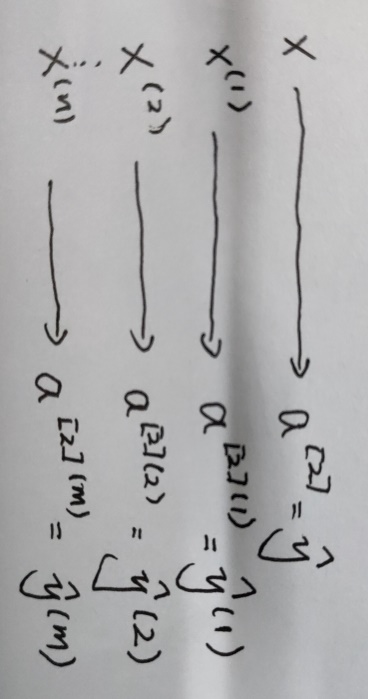


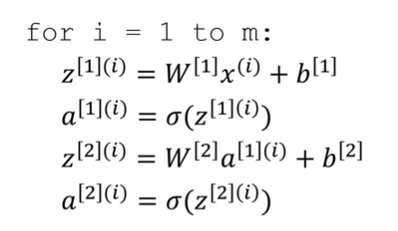
其中x可以用a[0],a[1]……表示：



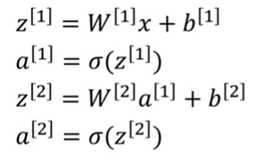
3.4节

1、如何将不同的训练样本向量化，如何将不同的样本堆叠起来，放入矩阵的各列：





怎么把上面这个for循环向量化：

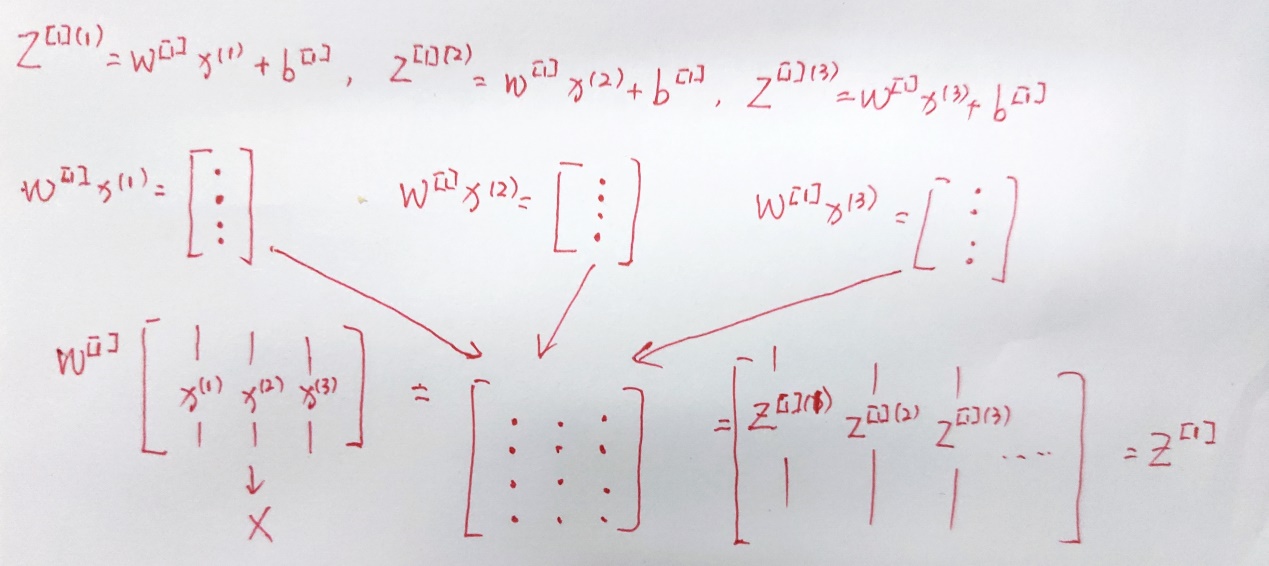


就好比把上面的小x向量化，堆叠到矩阵的各列，构成一个大X矩阵，对于z也可以这样做，比如Z[1](1),Z[1](2)等，这些列向量排到Z[1](m),同理，将a[1](1),a[1](2),也排列成一个列向量。同样对于Z(2),a(2)。

这样就很好理解，横向的话，我们对所有训练样本用指标排序，所以横向指标就对应不同的训练样本，当从左往右就扫过了整个训练集；在竖向，竖向指标就对应了神经网络里的不同节点。

3.5节

向量化实现的解释:把W[1]x[1]，看作一个向量，最后得出Z[1]，最后把b[i]单独加到向量的各列。下面是具体的步骤：

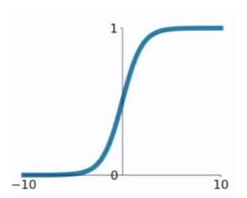


3.6节 激活函数

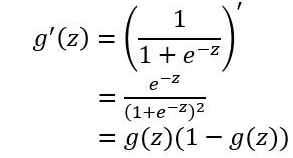
**1、sigmoid函数**

  在逻辑回归中我们介绍过sigmoid函数，该函数是将取值为 (−∞,+∞) 的数映射到 (0,1) 之间。sigmoid函数的公式以及图形如下：





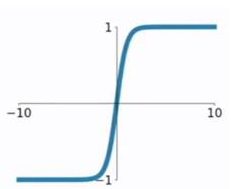
对于sigmoid函数的求导推导为：



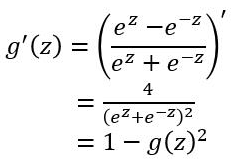
**2、tanh函数**

  tanh函数相较于sigmoid函数要常见一些，该函数是将取值为 (−∞,+∞) 的数映射到 (−1,1) 之间，其公式与图形为：



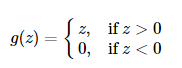


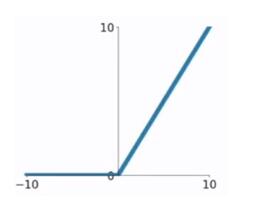
对于tanh函数的求导推导为：



**3、ReLU函数**

  ReLU函数又称为修正线性单元（Rectified Linear Unit），是一种分段线性函数，其弥补了sigmoid函数以及tanh函数的梯度消失问题。ReLU函数的公式以及图形如下：





对于ReLU函数的求导为：

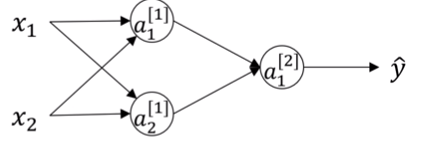


3.7节 为什么需要非线性激活函数：

如果一直使用线性的激活函数，无论神经网络有多少层，一直在做的只是计算线性激活函数，那就毫无意义，还不如去掉隐藏层。

3.11随机初始化

训练神经网络时，初始化权重很重要，对于logistic回归，可以把权重初始化为0，但是如果将神经网络的各项参数初始化为0的话，再使用梯度下降法，那将没有什么用。你给神经网络输入任何样本，无论你训练多少次，节点都是计算完全一样的函数，隐藏单元一开始就在做同样的计算。

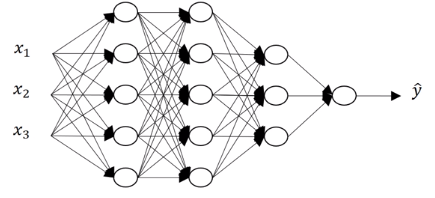


**所以你应该这样做：**

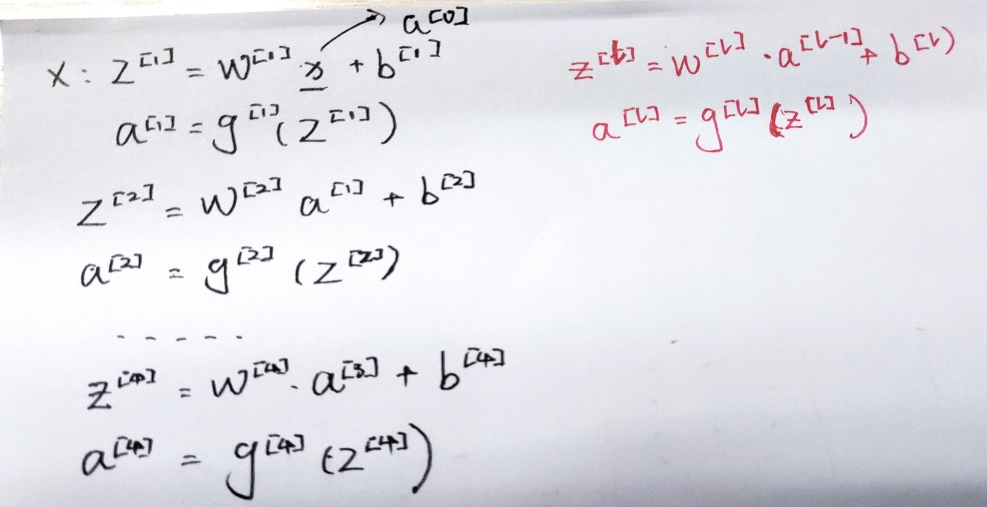
令W[1] = np.random.randn，可以产生高斯分布随机变量，然后开始后续的训练。

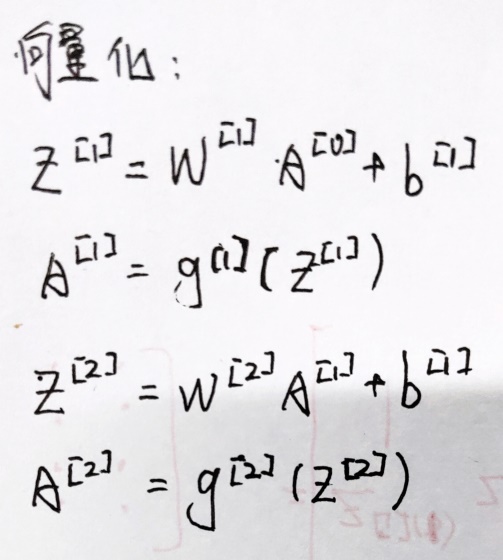
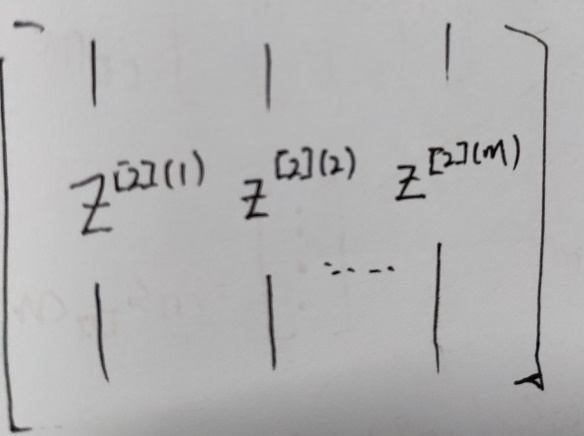
第四周：深层神经网络

**1、深层神经网络的前向传播**



一个训练样本的计算过程计算如下：

  
用向量化的方法训练整个训练集：

把所有的z和a向量叠起来，