这个笔记，记录了一些在Coursera上学习Neural Networks and Deep Learning课程时出现的一些问题以及我的一些思考的标注，以供以后参考复习。

个人认为最精彩的部分就是第3周的对神经网络的理解，而不是仅仅照搬公式带入求解，却不明白为什么如此进行矩阵运算。

week 1

第一周主要是**介绍概念**，似乎这个课程不提供讲义，所以许多概念和测试的要点需要自己记录，方便以后复习查询。

本课程只涉及了2层神经网络和多层神经网络的构建，第一周提到的诸如卷积神经网络(Convolutional neural network)和递归神经网络(Recurrent neural network)等，并不在之后的作业范围内。

week 2

Loss function and cost function

The loss function computes the error for a single training example; the cost function is the average of the loss functions of the entire training set.

Loss Function 计算一个样本的误差，这是一个新出现的概念，在之前的Machine Learning中并没有提及。

$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log (1-\hat{y}))$$

Cost Function 是整个训练集上所有样本误差的平均值，也就是损失函数的平均值。

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$$

broadcasting-Python

`np.dot(a,b)` performs a matrix multiplication on a and b, whereas `a*b` performs an element-wise multiplication.

`np.dot(a,b)` 计算的是矩阵乘法，而 `a*b` 是元素的乘法，需要两个矩阵shape相同（或者其中一个是行矩阵或列矩阵，可以进行广播）。

Reshape - Numpy

这是作业中遇到的一个问题

使用`reshape`将图像变为一维向量的时候，如果按照一张图片处理的公式处理，虽然维度正确，之后的Optimization之后的部分会出错，原因就在于**向量化之后的图片"不连续"**了，所以需要所给的提示来进行调整数组形状。

For convenience, you should now reshape images of shape (num_px, num_px, 3) in a numpy-array of shape (num_px ∗ num_px ∗ 3, 1). After this, our training (and test) dataset is a numpy-array where each column represents a flattened image. There should be m_train (respectively m_test) columns.

Exercise: Reshape the training and test data sets so that images of size (num_px, num_px, 3) are flattened into single vectors of shape (num_px ∗ num_px ∗ 3, 1).

A trick when you want to flatten a matrix X of shape (a,b,c,d) to a matrix X_flatten of shape (b∗c∗d, a) is to use:

```
X_flatten = X.reshape(X.shape[0], -1).T      # X.T is the transpose of X
```

```
# Reshape the training and test examples

### START CODE HERE ### (≈ 2 lines of code)
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0], -1).T
### END CODE HERE ###
# train_set_x_flatten = train_set_x_orig.reshape(num_px * num_px * 3, m_train) #这里注意它是有m_train个样本
#上面注释的这种写法是不正确的，讨论中的热门中有说明，
#大意是上面的写法虽然与要求解的向量维度相同，但是破坏了数据的"相关性"
#类似相邻的数据不是同一幅图片的数据，导致处理的就类似一团混乱的数据
#故而，这里需要按照题目的提示来写，关于这个提示写法的意义，我还需要进一步来研究
print ("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x_flatten shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
print ("sanity check after reshaping: " + str(train_set_x_flatten[0:5,0]))
```

下面说明一下`reshape`中的这个-1参数，因为`reshape`不会改变数组的元素数量，填的-1参数，就是自动计算的参数，举个例子

```
import numpy as np
x = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print("x: \n",x)
y = x.reshape((6,-1))
print("y: \n", y)
z = x.reshape((3,4))
print("z: \n", z)
m = x.reshape((3,-1))
print("m: \n", m)
```

输出样例

```
x:
```

```
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
y:
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]]
z:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
m:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

可以注意到，其中`z`和`m`的输出效果一致，所以这个-1参数，就可以用来自动推导出确定参数之外的参数

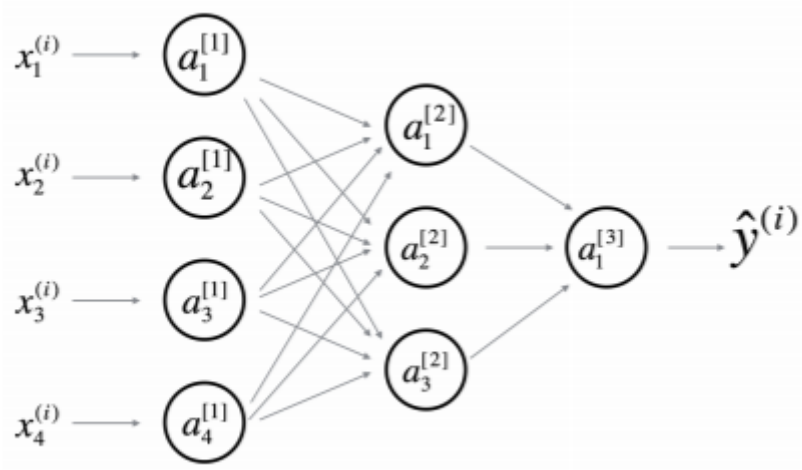
week 3

这些图片都来自课程中。

参数的含义

神经网络参数的理解，这些参数有**大小写**和**上下标**，还需要说明一下，这些参数分别放在图中的什么位置，代表什么含义。

输入层是不计算在神经网络层数中的。



1. 输入层$x, x^{(i)}, x_j, X\ x, x^{(i)}$都是**列向量**，$x$只是$x^{(i)}$中的一个特例，输入层写的$x^{(i)}$表示是第$i$个向量正在参与训练，之后的隐藏层便不再进行**圆括号的标记**，**方括号**中是表示隐藏层的序号。

   x_j是向量中的一个标量，代表着一个**特征**。

$$x=\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix};x^{(i)}=\begin{bmatrix}x_1^{(i)}\\x_2^{(i)}\\x_3^{(i)}\\x_4^{(i)}\end{bmatrix}$$

$x^{(i)},x$的维度是$n_x\times 1$，其中n_x表示列向量的**特征数**。

最终所有的输入层（每一个输入的列变量）共同组成了输入的集合X。

$$X=\begin{bmatrix}|&|&\cdots&|&\cdots&|\\x^{(1)}&x^{(2)}&\cdots&x^{(i)}&\cdots&x^{(m)}\\|&|&\cdots&|&\cdots&|\end{bmatrix}$$

这里m表示训练集中的**样本数**。显然，X的维度是$n_x\times m$。

2. 隐藏层$a^{[l]},a_j,A^{[l]}$

和x_j类似，a_j也是本层的一个表示特征的**标量**，是列向量$a^{[i]}$中的一部分。

为了方便标明特征数，这里我们以第二个隐藏层为例，下面的$l=2$，以便区分和输入层的维度关系。

$$a^{[l]}=\begin{bmatrix}a_1^{[l]}\\a_2^{[l]}\\a_3^{[l]}\end{bmatrix}$$

显然它的维度是$n_h^{[l]}\times 1$，表示第l个隐藏层中有$n_h^{[l]}$（该向量是3）个神经元的列向量。

再次强调，这里的方括号和x中的圆括号意义不同，这里表示的是**隐藏层的序号**。

$$A^{[l]}=\begin{bmatrix}|&|&\cdots&|&\cdots&|\\a^{[l](1)}&a^{[l](2)}&\cdots&a^{[l](i)}&\cdots&a^{[l](m)}\\|&|&\cdots&|&\cdots&|\end{bmatrix}$$

A的维度是$n_h^{[l]}\times m$。如上写法只是为了便于理解含义；事实上，容易想到，输入层分别输入m个训练样本，那么对于每一个输入样本x，隐藏层l都会产生一个相应的$a^{[l]}$,所以总列数是m，在图上也就是说**圆括号和圆括号是对应的。**

既然x和a的维度可能不同，则必须需要一个向量参数来进行调整，接下来则会介绍系数矩阵w,b来说明层与层之间是如何变换的。

3. 参数w,b,W,B

同样的，参数w也是**列向量。**

$$w=\begin{bmatrix}w_1\\w_2\\\vdots\\w_j\end{bmatrix};$$

而b则是标量。

$$h(x)=\sum_{i=1}^{n}w_ix_i+b$$

可以使用**内积**，**矢量化**写作，

$$h(x)=w^Tx+b$$

所以可以注意到w的维度是和x应当是相同的。

根据上图，我们可以对每个神经元分别求解。

$$z_1^{[2]} = (w_1^{[2]})^T a^{[1]} + b_1^{[2]},\ a_1^{[2]} = \sigma(z_1^{[2]})$$
$$z_2^{[2]} = (w_2^{[2]})^T a^{[1]} + b_2^{[2]},\ a_2^{[2]} = \sigma(z_2^{[2]})$$
$$z_3^{[2]} = (w_3^{[2]})^T a^{[1]} + b_3^{[2]},\ a_3^{[2]} = \sigma(z_3^{[2]})$$

值得说明的是，$a^{[1]}$是隐藏层中的第一层，$(w_1^{[2]})^T$表示的向量参数在图中是第一隐藏层和第二隐藏层之间指向第一个神经元的四个箭头，所以它的维度，可以表示为1×4，其中4就是上层隐藏层的神经元数目。

同样对于每次输入（或者是隐藏层），同时计算所有的神经元。注意，W也是为了表示为"列向量"，才有如下写法，虽然它的参数是行向量（或者说是列向量的转置）。

$$W^{[2]} = \begin{bmatrix} — (w_1^{[2]})^T — \\ — (w_2^{[2]})^T — \\ — (w_3^{[2]})^T — \end{bmatrix}$$

W的维度就是$n_h^{[l]} \times n_h^{[l-1]}$，也就是本层神经元数目×上层神经元数目。特别的，对于上述矩阵$W^{[2]}$，它的维度是3×4。

所以上述的每个神经元求解，就可以向量化为，

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]},\ a^{[2]} = \sigma(z^{[2]})$$

不过这里需要强调的是，b在这里是一个列向量了，它还涉及着**广播**的特性，我们之后开始说明这一点。

我们可以先出一个直观的例子，

$$b^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix}$$

此时，我们将$z^{[2]}$展开，具体可以看到其中的奥妙。

$$z^{[2]} = \begin{bmatrix} (w_1^{[2]})^T a^{[1]} + b_1^{[2]} \\ (w_2^{[2]})^T a^{[1]} + b_2^{[2]} \\ (w_3^{[2]})^T a^{[1]} + b_3^{[2]} \end{bmatrix}$$

所以，可以发现，$z^{[2]}$与上层的$z^{[1]}$相同，也是一个列向量。

进一步拓展，我们将输入m个列向量，所以左右两边的z, a分别扩展为Z, A，所以我们可以写作，

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]},\ A^{[2]} = \sigma(Z^{[2]})$$

$$\begin{aligned} &= \begin{bmatrix} | & | & | & \cdots & | \\ z^{[2](1)} & z^{[2](2)} & z^{[2](3)} & \cdots & z^{[2](m)} \\ | & | & | & \cdots & | \end{bmatrix} \\ &= W^{[2]} a^{[2](1)} + W^{[2]} a^{[2](2)} + \cdots + W^{[2]} a^{[2](m)} + b^{[2]} \end{aligned}$$

$$\begin{aligned} Z^{[2]} &= \begin{bmatrix} (w_1^{[2]})^T a^{[1](1)} + b_1^{[2]} \\ (w_2^{[2]})^T a^{[1](1)} + b_2^{[2]} \\ (w_3^{[2]})^T a^{[1](1)} + b_3^{[2]} \end{bmatrix} + \begin{bmatrix} (w_1^{[2]})^T a^{[1](2)} + b_1^{[2]} \\ (w_2^{[2]})^T a^{[1](2)} + b_2^{[2]} \\ (w_3^{[2]})^T a^{[1](2)} + b_3^{[2]} \end{bmatrix} + \cdots + \begin{bmatrix} (w_1^{[2]})^T a^{[1](m)} + b_1^{[2]} \\ (w_2^{[2]})^T a^{[1](m)} + b_2^{[2]} \\ (w_3^{[2]})^T a^{[1](m)} + b_3^{[2]} \end{bmatrix} \\ &= W^{[2]} (a^{[1](1)} + a^{[1](2)} + \cdots + a^{[1](m)}) + b^{[2]} \\ &= W^{[2]} A^{[1]} + b^{[2]} \end{aligned}$$

在这个过程中，可以注意到，利用了向量（行向量或者列向量）的广播特性，所以没有将b重新构成B，也是因为它的每一列都相同。

对于任意的隐藏层，我们可以得出结论，

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = \sigma(Z^{[l]})$$

至此，我们就将正向传播的具体细节刻画清楚了。

week 4

反向传播(Back propagation)，各个矩阵的维度与正向传播是相同的。

Summary of gradient descent

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]} a^{[1]^T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]} x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Andrew Ng

整个神经网络的流程图，分模块进行设计。