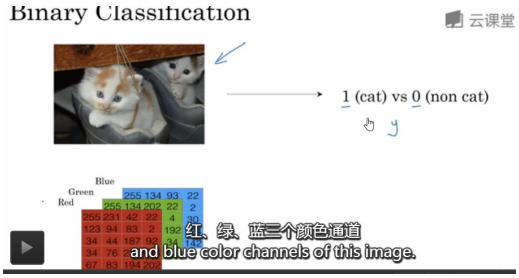
第一篇第二周

1、神经网络基础二分分类：

logistic回归是一个用于二分分类的算法。

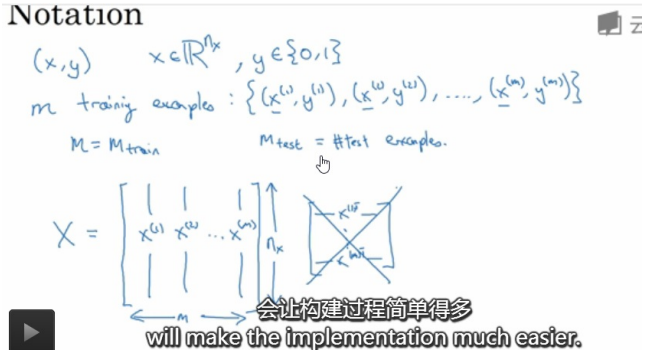


问题描述：识别此图的标签，是不是猫，是的话输出为1，不是的话输出为0,计算机要保存一张图片，要保存三个独立的矩阵，对应红绿蓝三个通道，输入图片是64*64像素的，就有三个64*64的矩阵，分别对应图片中的红绿蓝，三种像素的亮度，这里为了简单表示，用的是5*4的，像素强度用0到256表示。

所有的像素提出来，放入一个特征向量X，定义X表示这张图片，得到很长的特征向量，每个像素表示一个特征，64*64的图片，向量X的总维度是64*64*3，也就是12288个维度，也就是这张图片的特征向量维度。

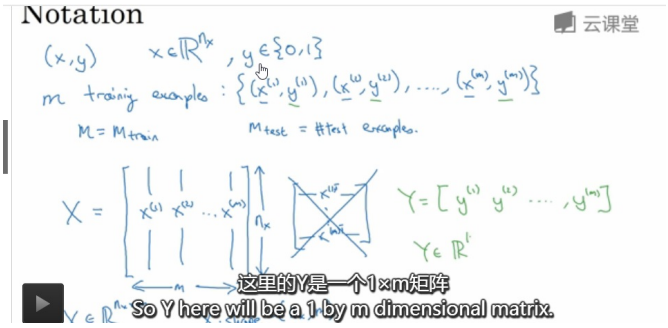
在二分类问题中，目标是训练出一个分类器，它以图片的特征向量X作为输入，预测输出的结果标签y是1还是0，也就是预测图片中是否有猫。

用(x,y)表示一个单独的样本，x是n维度的特征向量，y为标签，训练集由m个样本组成，训练集X的每一行表示每一个样本的特征向量,但是构建神经网络时，用的是每一列来表示如下图：



会让构建变得简单很多。

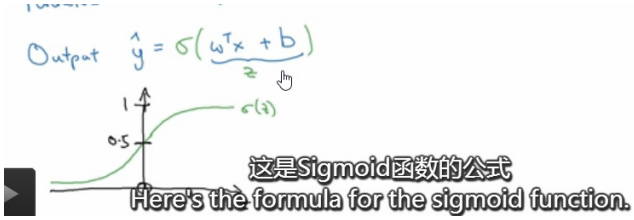
训练集X是nx*m的矩阵（nx是维度，m是样本个数），用X.shape函数就可以看到(nx,m)这样的结果，表示X是一个(nx,m)的矩阵。



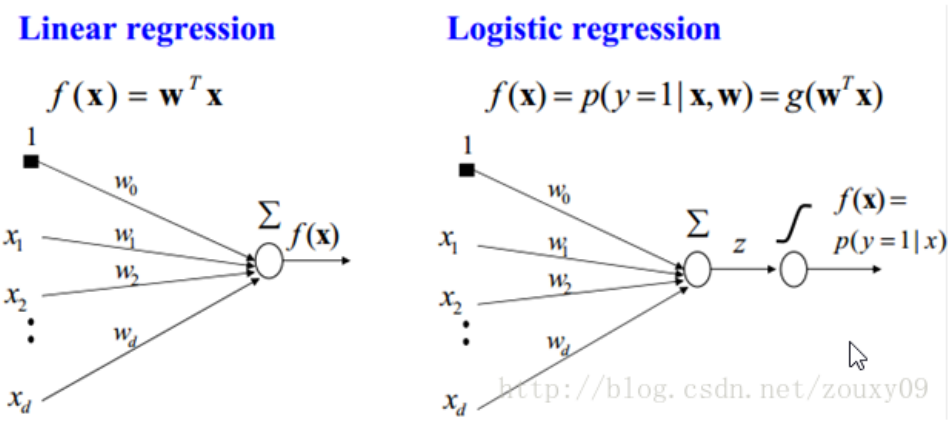
Y是一个1*m的矩阵。Y.shape等于(1,m).

2、逻辑回归算法讲解：

logistic回归算法，用在监督学习问题中。sigmoid的原型。

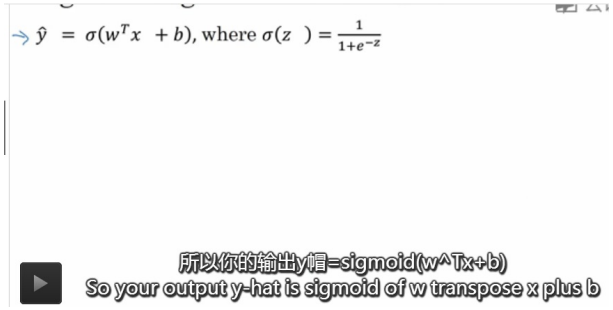


logistic回归就是一个线性分类模型，它与线性回归的不同点在于：为了将线性回归输出的很大范围的数，例如从负无穷到正无穷，压缩到0和1之间，这样的输出值表达为“可能性”才能说服广大民众。当然了，把大值压缩到这个范围还有个很好的好处，就是可以消除特别冒尖的变量的影响（不知道理解的是否正确）。而实现这个伟大的功能其实就只需要平凡一举，也就是在输出加一个logistic函数。另外，对于二分类来说，可以简单的认为：如果样本x属于正类的概率大于0.5，那么就判定它是正类，否则就是负类。实际上，SVM的类概率就是样本到边界的距离，这个活实际上就让logistic regression给干了。



所以说，LogisticRegression 就是一个被logistic方程归一化后的线性回归，仅此而已。

逻辑回归函数：



2.3、损失函数（误差函数）：衡量算法的运行情况，用来衡量预测值和真实值之间的差距有多近，有误差平方，但用这个的话，梯度下降就不那么好用，因为它不会给我们一个凸的优化问题。

损失函数，它衡量了在单个训练样本上的表现。

成本函数：它衡量参数的效果在全体训练样本上的表现。所有样本上的总损失和。基于参数的总成本，它是参数的函数。

Logistic Regression: Cost Function

To train the parameters w and b, we need to define a cost function.

Recap:

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

$x^{(i)}$ the i-th training example

Given $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$, *we want* $\hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function:

The loss function measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function computes the error for a single training example.

$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

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

- If $y^{(i)} = 1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0

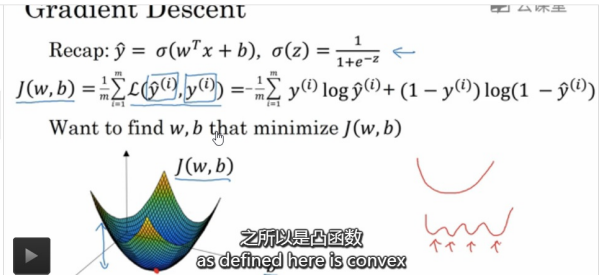
Cost function

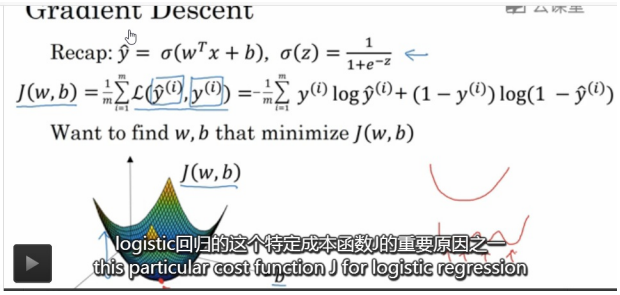
The cost function is the average of the loss function of the entire training set. We are going to find the parameters w *and* b that minimize the overall cost function.

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

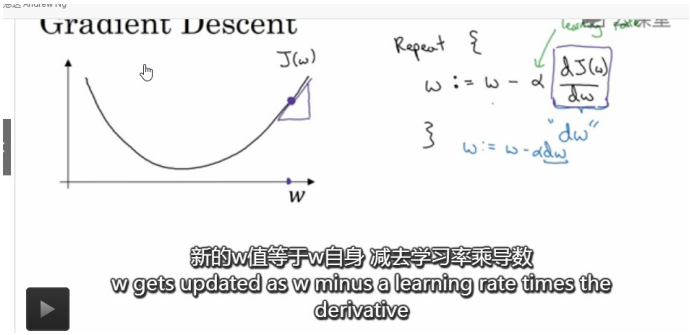
成本函数衡量了w和b在训练计算的效果，要习的合适的参数w和b，就用找到使得成本函数J（w,b)尽可能小的参数w和b，

2.4、梯度下降算法：来训练或者学习训练集上的参数。找到使成本函数J尽可能小的参数(W，b)

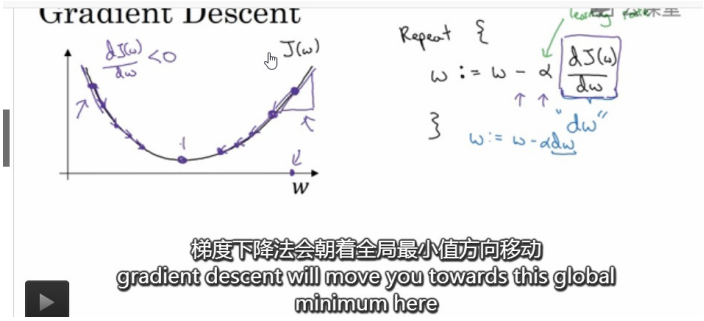




梯度下降算法简单描述就是从一个初始位置开始，朝最陡的下坡方向走一步，之后，或许在那里停下，因为它正试图沿着，最快下降的方向，往下走，或者说尽可能的往下走，这就是梯度下降的一轮迭代，直到收敛到全局最优解或者最优解附近为止（当然成本函数要是凸函数才行）

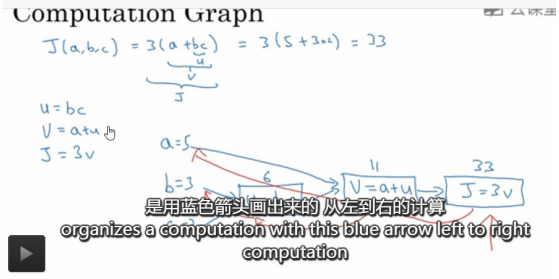


参数的更新算法是：w为w减去学习率乘以dw，学习率控制每一次迭代或者梯度下降的步长。



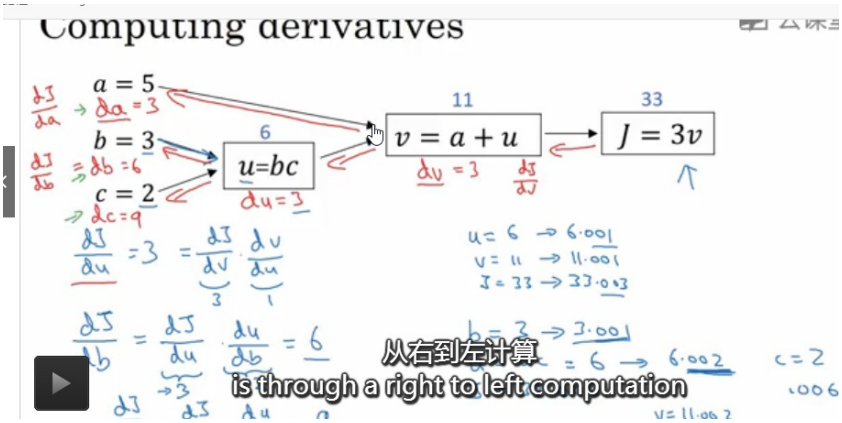
2.7、计算图谱：

前向传播：从左到右计算成本函数J,或者优化函数



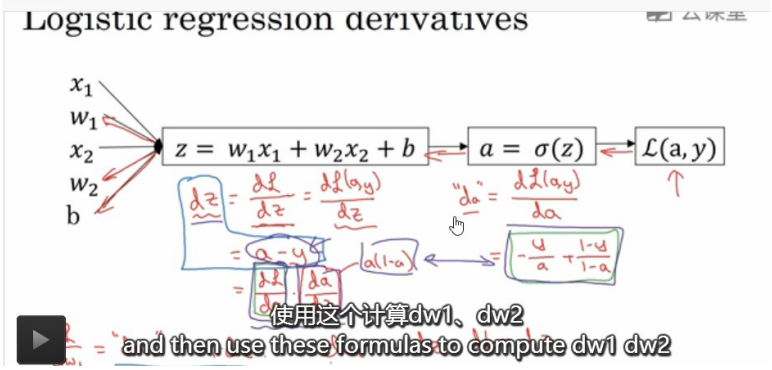
2.8、当计算导数时，从右到左是最有效的方法。跟着红色箭头走。

反向传播：从右到左计算参数的导数，求导数就是为了更新参数，因为参数给了个初始值，但不能使得成本函数最小，所以用各个参数的导数（或者中间值的导数）来更新参数，使得更新的参数让成本函数达到最小。



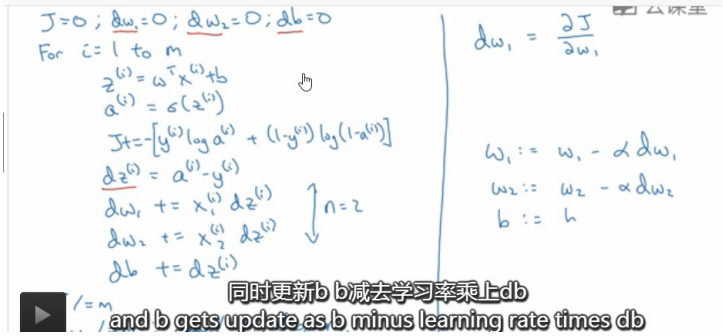
2.9、计算导数和把梯度下降应用到一个训练样本上去：

梯度下降其实就是使用反向传播算法计算dw1，dw2和db,然后更新w1为w1减去学习率乘以dw1,更新w2为w2减去学习率乘以dw2，b为b减去学习率乘以db，可以看出有两层for循环，如果特征很多的话，将会有很多for循环，这样就计算很低下，使用向量化技术，可以摆脱显示for循环



2.10、应用带m个训练样本上去：

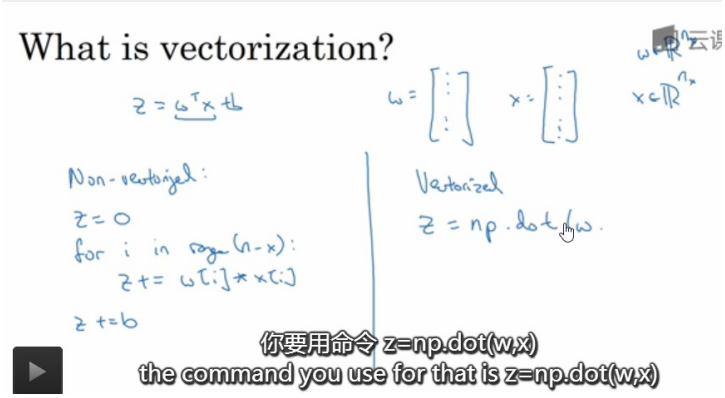
我们使用dw1,dw2,db作为累加器，所以在这些计算之后，dw1等于你的全局成本函数对w1的导数，dw2,db也是一样。然后使用梯度下降，使得w1为w1减去学习率乘以dw1，w2为w2减去学习率乘以dw2，b为b减去学习率乘以db。



2.11、向量化

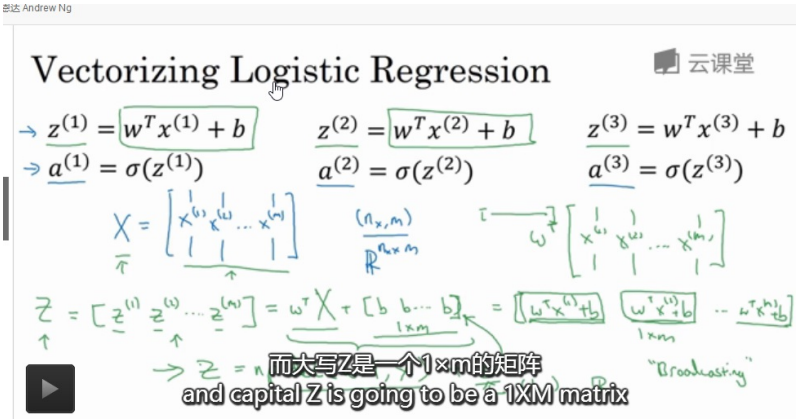
消除你的代码中显示for循环语句的艺术。

什么是向量化：将要计算的列变成矩阵相乘，这样就非常快

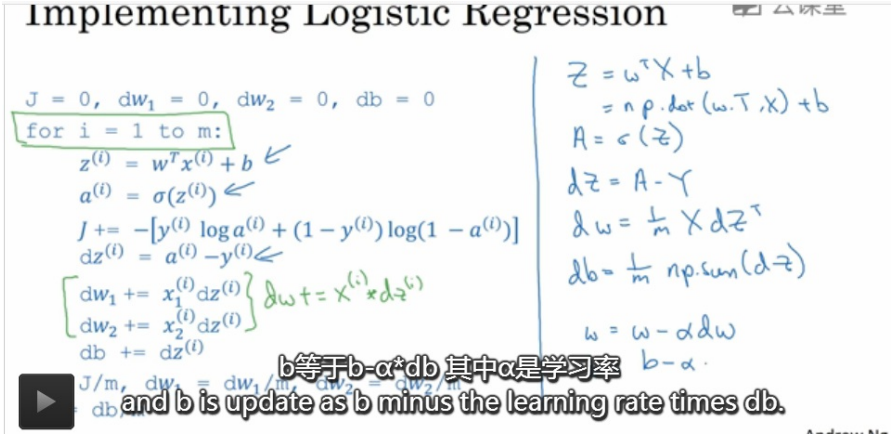


2.12、能不用for循环就尽量不用for循环，比如可以使用python中numpy库中的函数np.dot(a,b)计算

2.13、



逻辑回归中的前向传播和反向传播算法的python伪代码如下图右边：

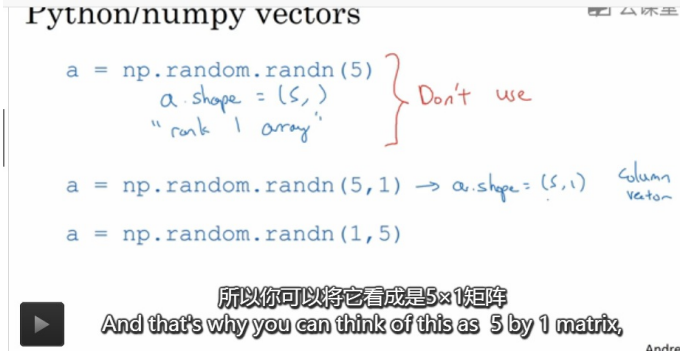


2.15、广播变量

```
import numpy as np
A=np.array([[]])
```

2.16、关于python和numpy去除bug的一些建议：

```
import numpy as np
a=np.random.randn(5)
print(a)
print(a.shape)  #输出（5，），它既不是行向量，也不是列向量,所以它的转置还是它
print(a.T)  #等于print(a)，转置跟没转置的内积是一个数，所以既不是行向量，也不是列向量
#编写神经网络的时候，不要出现这种数据结构
#其中形状是（5，）或者（n，）这种秩为1的数组。
#要用如下的代码：
a=np.random.randn(5，1)#这样就是5*1的列向量，转置就是一样行向量。
```



不太确定一个向量是多少维度时，使用assert()函数。

```
assert(a.shape==(5,1))
```

如果a是（5,）这样的秩，也可以用reshape转化成向量

```
a=a.reshape(5,1)
```