**2019.1.23课程总结**

Huperparemeters(超参):leaning rate 

#iterations(迭代)

#hidden layers L

#hidden units

Choice of activation functions

Later:momentum term(动量项)，mini batch size,regularization(正则化)

**Course one 课程练习**

第二周的编程作业：搭建一个能够**【识别猫】** 的简单的神经网络

在开始之前，我们有需要引入的库：

numpy ：是用Python进行科学计算的基本软件包。

h5py：是与H5文件中存储的数据集进行交互的常用软件包。

matplotlib：是一个著名的库，用于在Python中绘制图表。

lr\_utils ：在本文的资料包里，一个加载资料包里面的数据的简单功能的库。

Lr\_utils代码：

*import* numpy *as* np  
*import* h5py  
   
*def* load\_dataset():  
 train\_dataset = h5py.File('datasets/train\_catvnoncat.h5', "r")  
 train\_set\_x\_orig = np.array(train\_dataset["train\_set\_x"][:]) # your train set features  
 train\_set\_y\_orig = np.array(train\_dataset["train\_set\_y"][:]) # your train set labels  
 test\_dataset = h5py.File('datasets/test\_catvnoncat.h5', "r")  
 test\_set\_x\_orig = np.array(test\_dataset["test\_set\_x"][:]) # your test set features  
 test\_set\_y\_orig = np.array(test\_dataset["test\_set\_y"][:]) # your test set labels  
  
 classes = np.array(test\_dataset["list\_classes"][:]) # the list of classes  
   
 train\_set\_y\_orig = train\_set\_y\_orig.reshape((1, train\_set\_y\_orig.shape[0]))  
 test\_set\_y\_orig = test\_set\_y\_orig.reshape((1, test\_set\_y\_orig.shape[0]))  
   
 *return* train\_set\_x\_orig, train\_set\_y\_orig, test\_set\_x\_orig, test\_set\_y\_orig, classes

解释以下上面的load\_dataset() 返回的值的含义：

train\_set\_x\_orig ：保存的是训练集里面的图像数据（本训练集有209张64x64的图像）。

train\_set\_y\_orig ：保存的是训练集的图像对应的分类值（【0 | 1】，0表示不是猫，1表示是猫）。

test\_set\_x\_orig ：保存的是测试集里面的图像数据（本训练集有50张64x64的图像）。

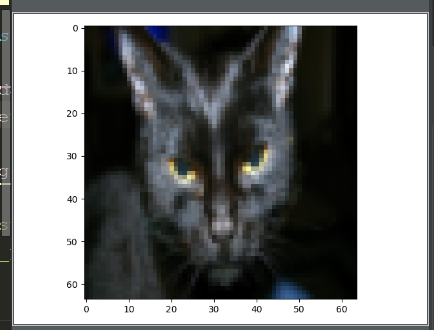
test\_set\_y\_orig ： 保存的是测试集的图像对应的分类值（【0 | 1】，0表示不是猫，1表示是猫）。

classes ： 保存的是以bytes类型保存的两个字符串数据，数据为：[b’non-cat’ b’cat’]。

index = 25  
plt.imshow(train\_set\_x\_orig[index])

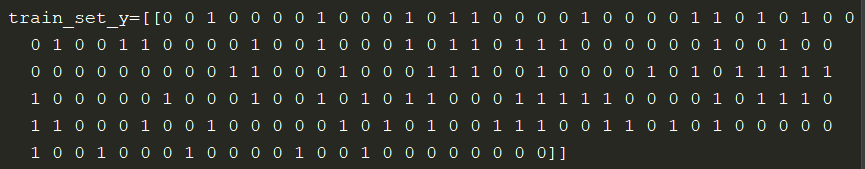
plt.show()

这个可以生成训练集中第25个图片



*print*("train\_set\_y=" + str(train\_set\_y)) #你可以看一下训练集里面的标签是什么样的

209个训练集的标签：



#打印出当前的训练标签值

#使用np.squeeze的目的是压缩维度，【未压缩】train\_set\_y[:,index]的值为[1] , 【压缩后】np.squeeze(train\_set\_y[:,index])的值为1

#print("【使用np.squeeze：" + str(np.squeeze(train\_set\_y[:,index])) + "，不使用np.squeeze： " + str(train\_set\_y[:,index]) + "】")

#只有压缩后的值才能进行解码操作

*print*("y=" + str(train\_set\_y[:,index]) + ", it's a " + classes[np.squeeze(train\_set\_y[:,index])].decode("utf-8") + "' picture")

结果



* 主代码：参数解释m\_train ：训练集里图片的数量。
* m\_test ：测试集里图片的数量。
* num\_px ： 训练、测试集里面的图片的宽度和高度（均为64x64）。

train\_set\_x\_orig 是一个维度为(m\_​​train，num\_px，num\_px，3）的数组。

m\_train = train\_set\_y.shape[1] #训练集里图片的数量。  
m\_test = test\_set\_y.shape[1] #测试集里图片的数量。  
num\_px = train\_set\_x\_orig.shape[1] #训练、测试集里面的图片的宽度和高度（均为64x64）。  
#现在看一看我们加载的东西的具体情况  
*print* ("训练集的数量: m\_train = " + str(m\_train))  
*print* ("测试集的数量 : m\_test = " + str(m\_test))  
*print* ("每张图片的宽/高 : num\_px = " + str(num\_px))  
*print* ("每张图片的大小 : (" + str(num\_px) + ", " + str(num\_px) + ", 3)")  
*print* ("训练集\_图片的维数 : " + str(train\_set\_x\_orig.shape))  
*print* ("训练集\_标签的维数 : " + str(train\_set\_y.shape))  
*print* ("测试集\_图片的维数: " + str(test\_set\_x\_orig.shape))  
*print* ("测试集\_标签的维数: " + str(test\_set\_y.shape))  
  
#将训练集的维度降低并转置。  
train\_set\_x\_flatten = train\_set\_x\_orig.reshape(train\_set\_x\_orig.shape[0],-1).T  
#将测试集的维度降低并转置。  
test\_set\_x\_flatten = test\_set\_x\_orig.reshape(test\_set\_x\_orig.shape[0], -1).T  
  
*print* ("训练集降维最后的维度： " + str(train\_set\_x\_flatten.shape))  
*print* ("训练集\_标签的维数 : " + str(train\_set\_y.shape))  
*print* ("测试集降维之后的维度: " + str(test\_set\_x\_flatten.shape))  
*print* ("测试集\_标签的维数 : " + str(test\_set\_y.shape))

训练集\_图片的维数 : (209, 64, 64, 3) 训练集降维最后的维度： (12288, 209)

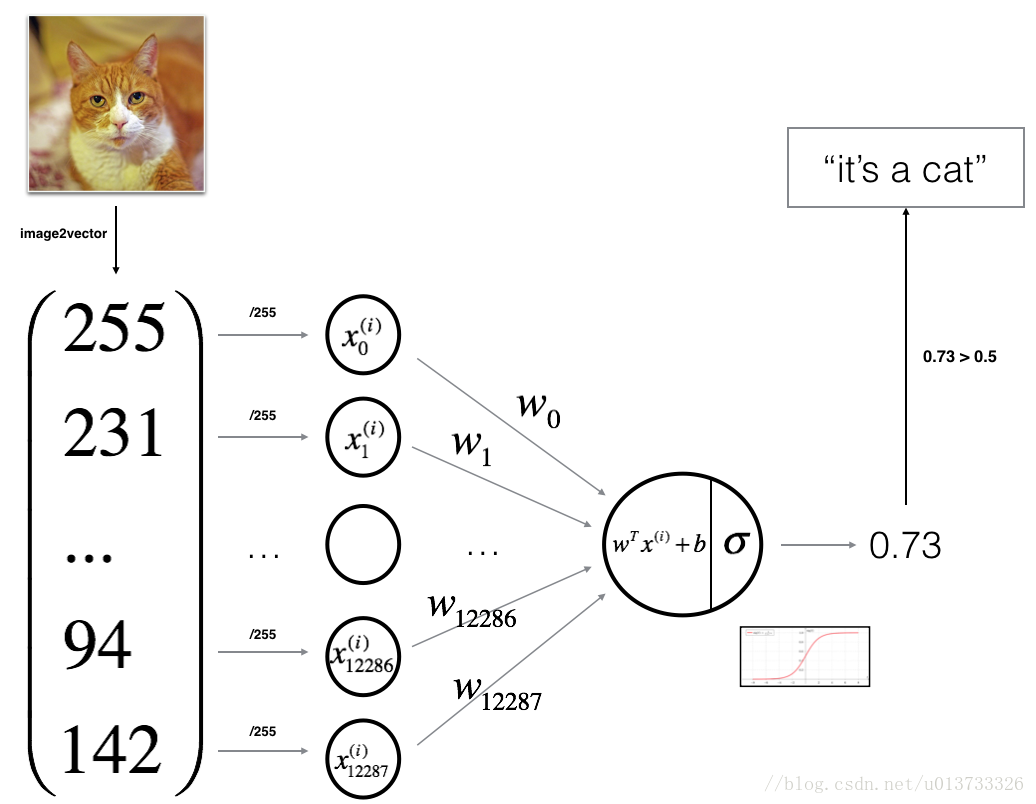
测试集\_图片的维数: (50, 64, 64, 3) 测试集降维之后的维度: (12288, 50)

为了方便，我们要把维度为（64，64，3）的numpy数组重新构造为（64 x 64 x 3，1）的数组, 在此之后，我们的训练和测试数据集是一个numpy数组，**【每列代表一个平坦的图像】** ，应该有m\_train和m\_test列。

机器学习中一个常见的预处理步骤是对**数据集进行居中和标准化：**将数据集的每一行除以255（像素通道的最大值），因为在RGB中不存在比255大的数据，所以我们可以放心的除以255，让标准化的数据位于[0,1]之间，现在标准化我们的数据集：

train\_set\_x = train\_set\_x\_flatten / 255#python会自动把常数补全为列向量

test\_set\_x = test\_set\_x\_flatten / 255



构建神经网络：数学表达式之前笔记中都有解释

对于x(i)



然后通过对所有训练样例求和来计算成本:



建立神经网络的主要步骤是：

1. 定义模型结构（例如输入特征的数量）

2. 初始化模型的参数

3. 循环：

#构建sigmoid激活函数  
*def* sigmoid(*z*):  
 s=1/(1+np.exp(-*z*))  
 *return* s

初始化(initialize)参数w,b

*def* initialize\_with\_zeros(*dim*):  
 """  
 此函数为w创建一个维度为（dim，1）的0向量，并将b初始化为0。  
 参数：  
 dim - 我们想要的w矢量的大小（或者这种情况下的参数数量）  
 返回：  
 w - 维度为（dim，1）的初始化向量。  
 b - 初始化的标量（对应于偏差）  
 """  
 w = np.zeros(shape = (*dim*,1))  
 b = 0  
 #使用断言来确保我要的数据是正确的  
 *assert*(w.shape == (*dim*, 1)) #w的维度是(dim,1)  
 *assert*(isinstance(b, float) *or* isinstance(b, int)) #b的类型是float或者是int  
  
 *return* (w , b)

 这里将w初始化为0 ，在logistics回归中没有隐层时, 可以将所有的参数初始化为0.但在深度学习中大部分情况下会导致学习失败, 得不到可用的模型.