**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（logisitic 回归）

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）。

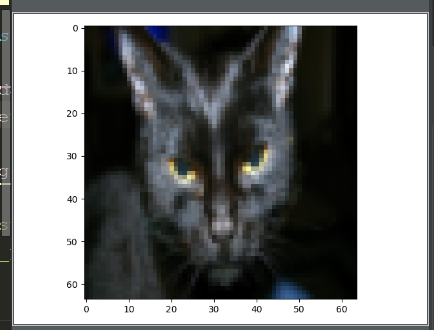
test\_set\_y\_orig ： 保存的是测试集的图像对应的分类值（【0 | 1】，0表示不是猫，1表示是猫）。

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])

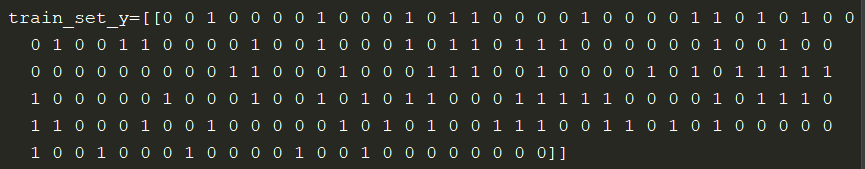
plt.show()

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



* **主程序**：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。

numpy中的ravel()、flatten()、squeeze()都有将多维数组转换为一维数组的功能，区别：   
ravel()：如果没有必要，不会产生源数据的副本   
flatten()：返回源数据的副本   
squeeze()：只能对维数为1的维度降维

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))

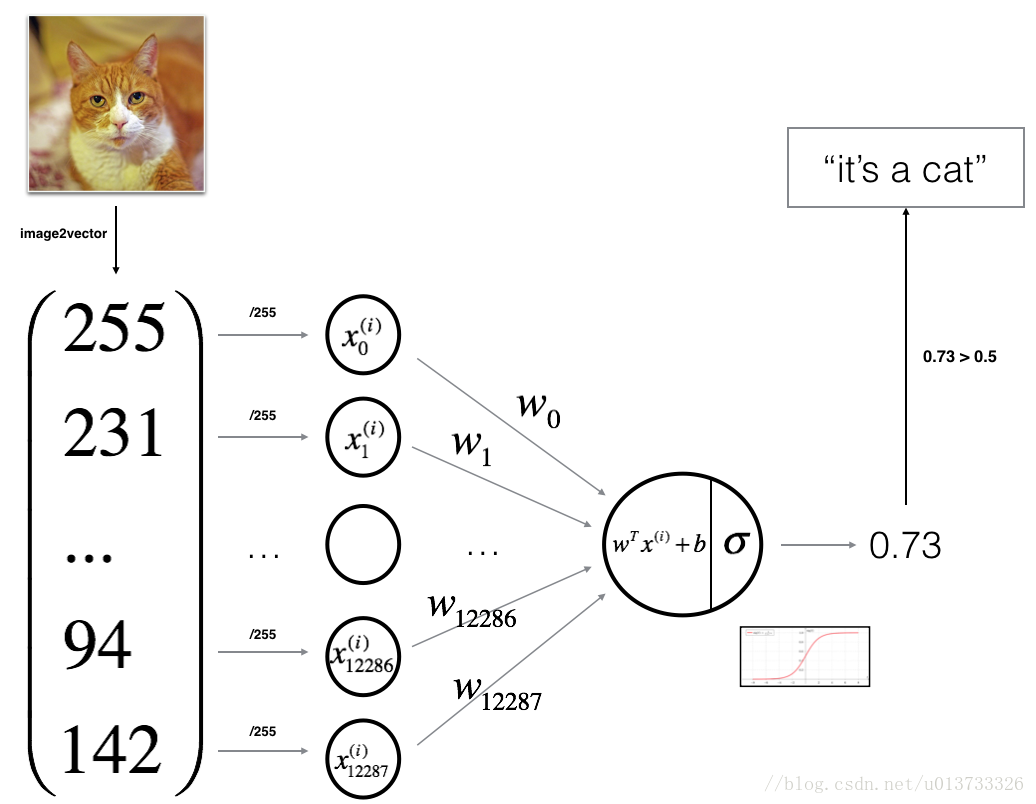
训练集\_图片的维数 : (209, 64, 64, 3) 训练集降维最后的维度： (12288, 209)

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：正向传播，反向传播，梯度下降

#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

 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.

*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)

**2019.1.24总结**

前向后向传播函数

*def* propagate(*w*, *b*, *X*, *Y*):  
 """  
 实现前向和后向传播的成本函数及其梯度。  
 参数：  
 w - 权重，大小不等的数组（num\_px \* num\_px \* 3，1）  
 b - 偏差，一个标量  
 X - 矩阵类型为（num\_px \* num\_px \* 3，训练数量）  
 Y - 真正的“标签”矢量（如果非猫则为0，如果是猫则为1），矩阵维度为(1,训练数据数量)  
 返回：  
 cost- 逻辑回归的负对数似然成本  
 dw - 相对于w的损失梯度，因此与w相同的形状  
 db - 相对于b的损失梯度，因此与b的形状相同  
 """  
 m = *X*.shape[1]  
  
 #正向传播  
 A = sigmoid(np.dot(*w*.T,*X*) + *b*) #计算激活值，请参考公式2。  
 cost = (- 1 / m) \* np.sum(*Y* \* np.log(A) + (1 - *Y*) \* (np.log(1 - A))) #计算成本，请参考公式3和4。  
  
 #反向传播  
 dw = (1 / m) \* np.dot(*X*, (A - *Y*).T) #请参考视频中的偏导公式。  
 db = (1 / m) \* np.sum(A - *Y*) #请参考视频中的偏导公式。  
  
 #使用断言确保我的数据是正确的  
 *assert*(dw.shape == *w*.shape)  
 *assert*(db.dtype == float)  
 cost = np.squeeze(cost)  
 *assert*(cost.shape == ())  
  
 #创建一个字典，把dw和db保存起来。  
 grads = {  
 "dw": dw,  
 "db": db  
 }  
 *return* (grads , cost)

优化函数，通过反向传播优化w,b

参数: w - 权重，大小不等的数组（num\_px \* num\_px \* 3，1）

b - 偏差，一个标量

X - 维度为（num\_px \* num\_px \* 3，训练数据的数量）的数组。

Y - 真正的“标签”矢量（如果非猫则为0，如果是猫则为1），矩阵维度为(1,训练数据的数量)

num\_iterations - 优化循环的迭代次数

learning\_rate - 梯度下降更新规则的学习率

print\_cost - 每100步打印一次损失值

返回：

params - 包含权重w和偏差b的字典

grads - 包含权重和偏差相对于成本函数的梯度的字典

成本 - 优化期间计算的所有成本列表，将用于绘制学习曲线。

提示：

我们需要写下两个步骤并遍历它们：

1）计算当前参数的成本和梯度，使用propagate（）。

2）使用w和b的梯度下降法则更新参数。

*def* optimize(*w* , *b* , *X* , *Y* , *num\_iterations* , *learning\_rate* , *print\_cost* = False):

costs = []  
  
*for* i *in* range(*num\_iterations*):  
  
 grads, cost = propagate(*w*, *b*, *X*, *Y*)  
  
 dw = grads["dw"]  
 db = grads["db"]  
  
 w = *w* - *learning\_rate* \* dw  
 b = *b* - *learning\_rate* \* db  
  
 #记录成本  
 *if* i % 100 == 0:  
 costs.append(cost)  
 #打印成本数据  
 *if* (*print\_cost*) *and* (i % 100 == 0):  
 *print*("迭代的次数: %i ， 误差值： %f" % (i,cost))  
  
params = {  
 "w" : *w*,  
 "b" : *b* }  
grads = {  
 "dw": dw,  
 "db": db }  
*return* (params , grads , costs)

1. **预测数据集X的标签**：计算 Y^=A=σ(wTX+b)Y^=A=σ(wTX+b)
2. 将a的值变为0（如果激活值<= 0.5）或者为1（如果激活值> 0.5），然后将预测值存储在向量Y\_prediction中。

*def* predict(*w*, *b*, *X*):  
 """ 使用学习逻辑回归参数logistic （w，b）预测标签是0还是1，  
 参数： w - 权重，大小不等的数组（num\_px \* num\_px \* 3，1）  
 b - 偏差，一个标量  
 X - 维度为（num\_px \* num\_px \* 3，训练数据的数量）的数据  
 返回： Y\_prediction - 包含X中所有图片的所有预测【0 | 1】的一个numpy数组（向量）"""  
 m = *X*.shape[1] # 图片的数量  
 Y\_prediction = np.zeros((1, m))  
 w = *w*.reshape(*X*.shape[0], 1)  
 # 计预测猫在图片中出现的概率  
 A = sigmoid(np.dot(*w*.T, *X*) + *b*)  
 *for* i *in* range(A.shape[1]):  
 # 将概率a [0，i]转换为实际预测p [0，i]  
 Y\_prediction[0, i] = 1 *if* A[0, i] > 0.5 *else* 0  
 # 使用断言  
 *assert* (Y\_prediction.shape == (1, m))  
 *return* Y\_prediction

再创建一个model函数，在其中调用其它函数

*def* model(*X\_train* , *Y\_train* , *X\_test* , *Y\_test* , *num\_iterations* = 2000 , *learning\_rate* = 0.5 , *print\_cost* = False):  
 """  
 通过调用之前实现的函数来构建逻辑回归模型  
 参数：  
 X\_train - numpy的数组,维度为（num\_px \* num\_px \* 3，m\_train）的训练集  
 Y\_train - numpy的数组,维度为（1，m\_train）（矢量）的训练标签集  
 X\_test - numpy的数组,维度为（num\_px \* num\_px \* 3，m\_test）的测试集  
 Y\_test - numpy的数组,维度为（1，m\_test）的（向量）的测试标签集  
 num\_iterations - 表示用于优化参数的迭代次数的超参数  
 learning\_rate - 表示optimize（）更新规则中使用的学习速率的超参数  
 print\_cost - 设置为true以每100次迭代打印成本  
 返回：  
 d - 包含有关模型信息的字典。  
 """  
 w , b = initialize\_with\_zeros(*X\_train*.shape[0])  
 parameters , grads , costs = optimize(w , b , *X\_train* , *Y\_train*,*num\_iterations* , *learning\_rate* , *print\_cost*)  
  
 #从字典“参数”中检索参数w和b  
 w , b = parameters["w"] , parameters["b"]  
 #预测测试/训练集的例子  
 Y\_prediction\_test = predict(w , b, *X\_test*)  
 Y\_prediction\_train = predict(w , b, *X\_train*)  
  
 #打印训练后的准确性  
 *print*("训练集准确性：" , format(100 - np.mean(np.abs(Y\_prediction\_train - *Y\_train*)) \* 100) ,"%")  
 *print*("测试集准确性：" , format(100 - np.mean(np.abs(Y\_prediction\_test - *Y\_test*)) \* 100) ,"%")  
  
 d = {  
 "costs" : costs,  
 "Y\_prediction\_test" : Y\_prediction\_test,  
 "Y\_prediciton\_train" : Y\_prediction\_train,  
 "w" : w,  
 "b" : b,  
 "learning\_rate" : *learning\_rate*,  
 "num\_iterations" : *num\_iterations* }  
 *return* d  
d = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations = 2000, learning\_rate = 0.005, print\_cost = True)

加入图表函数，输出不同学习率对应训练结果和测试结果：

#绘制图  
learning\_rates = [0.01, 0.005,0.001, 0.0001]  
models = {}  
*for* i *in* learning\_rates:  
 *print* ("learning rate is: " + str(i))  
 models[str(i)] = model(train\_set\_x, train\_set\_y, test\_set\_x, test\_set\_y, num\_iterations = 1500, learning\_rate = i, print\_cost = False)  
 *print* ('\n' + "-------------------------------------------------------" + '\n')  
  
*for* i *in* learning\_rates:  
 plt.plot(np.squeeze(models[str(i)]["costs"]), label= str(models[str(i)]["learning\_rate"]))  
  
plt.ylabel('cost')  
plt.xlabel('iterations')  
  
legend = plt.legend(loc='upper center', shadow=True)  
frame = legend.get\_frame()  
frame.set\_facecolor('0.90')  
plt.show()

