深度学习笔记

视频名称	【斯坦福大学】深度学习（全192讲）吴恩达
视频链接	https://www.bilibili.com/video/BV1ev4y1U7j2/
作业链接	https://blog.csdn.net/u013733326/article/details/79827273
笔记作者	刘天翼

附录

本章将对后续编程作业中常用的函数和语句做出解释

需求：查看矩阵的维度，以及某一维度的长度和数据

查看矩阵的维度，以及某一维度的长度 Python 复制代码

```
import numpy as np
# 二维矩阵
c = array([[1,1],
           [1,2],
           [1,3],
           [1,4]])
c.shape  # (4, 2)
c.shape[0]  # 4
c.shape[1]  # 2
# 一维矩阵
b =array([1,2,3,4])
b.shape  # (4,)
b.shape[0] # 4
# 也可以简写成
shape([1,2,3,4])  # (4,)
# 获取第一行数据
d = c[0][:]   # [1,1]
```

需求：按元素的矩阵运算：加法，减法，乘法，除法，指数运算，对数运算

按元素的矩阵运算 Python 复制代码

```python
import numpy as np
A = np.array([[1,1,1,1],
              [2,2,2,2],
              [3,3,3,3]
             ])
A = A-1
A = A+1
A = A/2
A = A*2
A = np.exp(A)
A = np.log(A)
A = np.array([1,2,3,4])
B = np.divide(A,4)
```

需求：矩阵的升维

矩阵的升维 Python 复制代码

```python
import numpy as np
A = np.array([1,1])
print(A.shape)           # (2,)
# 增加一个列维度
B = A.reshape(2,1)  # (2, 1)
print(B.shape)
print(B)
# 增加一个行维度
C = A.reshape(1,2)   # (1, 2)
print(C.shape)
print(C)
```

需求：矩阵的降维

矩阵的降维 Python 复制代码

```python
import numpy as np
A = np.arange(64).reshape(4,4,4)
print(A)
# 矩阵降维
B = A.reshape(-1)   # (64,)
print(B.shape)
print(B)
C = A.reshape(4,-1)  # (4, 16)
print(C.shape)
print(C)
```

需求：矩阵默认值

矩阵默认值 Python 复制代码

```python
import numpy as np
# 构造特定形状的全0矩阵
A = np.zeros((2,3))
print(A)
# 构造特定形状的全1矩阵
B = np.ones((2,3))
print(B)


```

需求：矩阵计算

矩阵计算 Python 复制代码

```python
import numpy as np
A = np.array([[1,1],
              [1,1]
             ])
B = np.array([[1,0],
              [0,1]
             ])
# 矩阵点乘
C = np.dot(A,B)
print(C)
# 矩阵求和
print(np.sum(A))    # 4
print(np.sum(B))    # 2
# 求均值
A = np.arange(1,10).reshape(3,3)
print(A)
mean = np.mean(A)
print(mean)
mean_row = np.mean(A,axis = 1)
print(mean_row)    # 2. 5. 8.
mean_col = np.mean(A,axis = 0)
print(mean_col)    # 4. 5. 6.
```

需求：矩阵索引

矩阵索引 Python 复制代码

```python
import numpy as np
# 矩阵索引  从0开始
Test = np.array([
    [1,2,3,4],
    [5,6,7,8],
    [9,10,11,12],
    [13,14,15,16]
])
# 获取第三列的数据
A = Test[:,3]
print(A)
# 获取第二行的数据
B = Test[2,:]
print(B)
```

神经网络和深度学习

深度学习引言

神经网络和监督学习

Supervised Learning

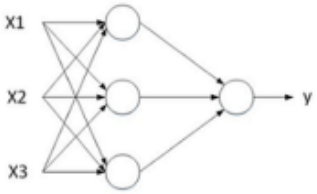
Input(x)	Output(y)	Application
Home features	Price	Real Estate
Ad,user info	Click on ad?(0/1)	Online Advertising
Image	Object(1,⋯,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image,Radar info	Position of other cars	Autonomous driving

房价预测和线上广告问题：使用标准的神经网络模型；

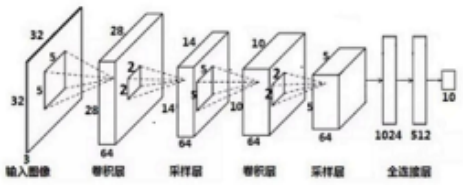
图像识别处理问题：使用**卷积神经网络（Convolution Neural Network），即CNN；**

序列信号处理问题：使用**循环神经网络（Recurrent Neural Network），即RNN；**

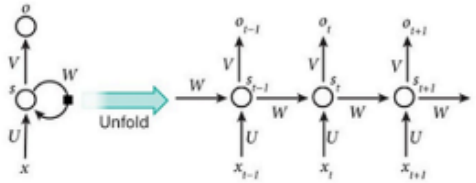
自动驾驶等复杂问题：使用**混合神经网络模型。**



Standard NN

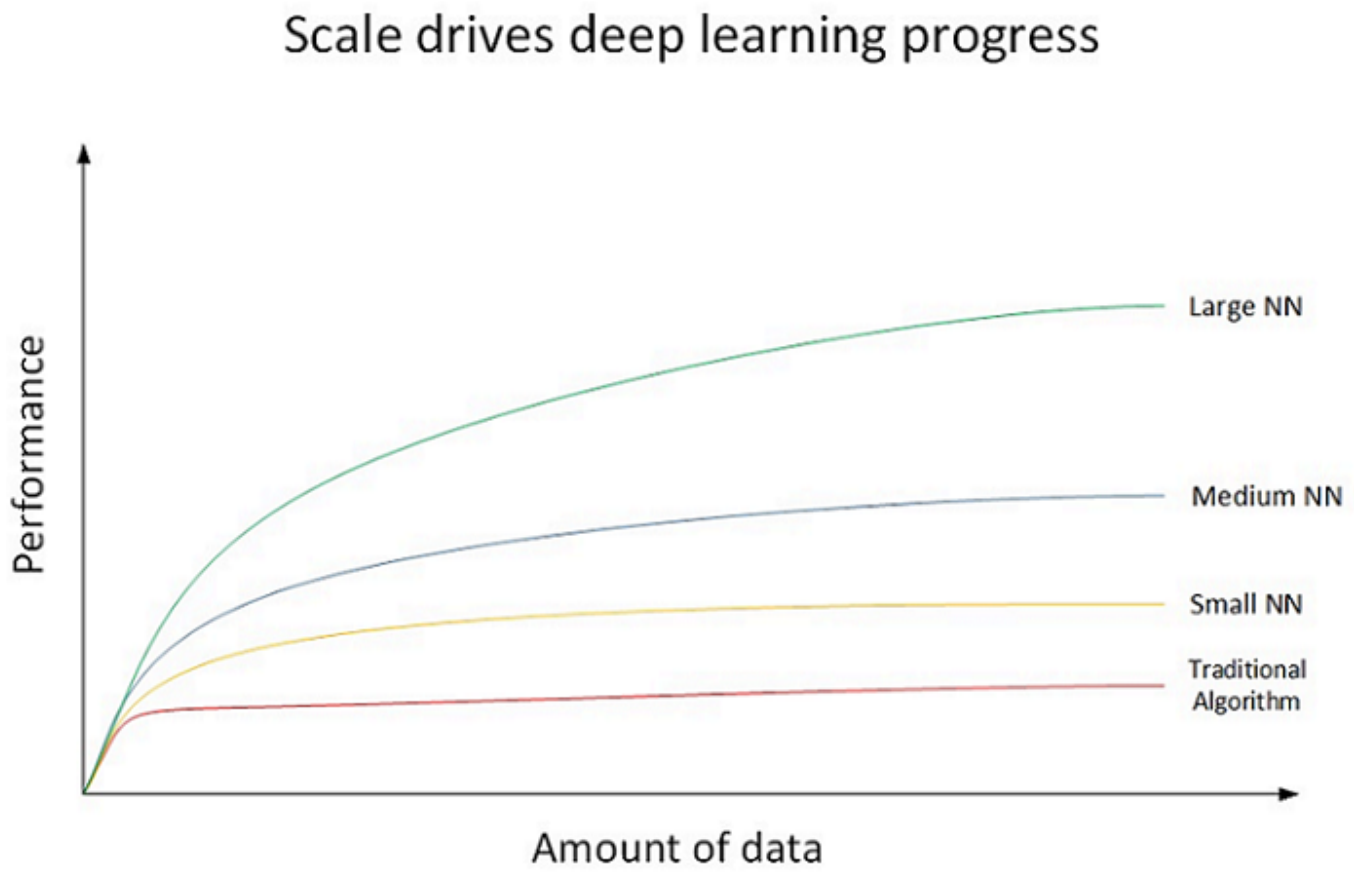


Convolutional NN



Recurrent NN

神经网络的应用场景



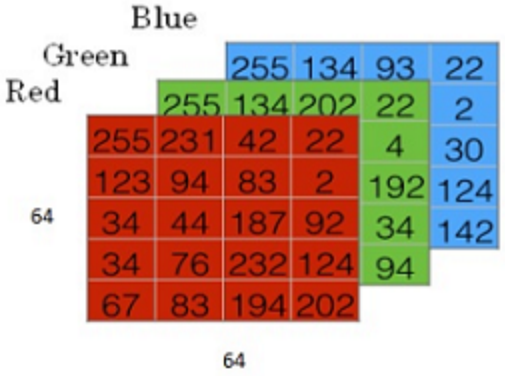
神经网络编程基础

二分类（Binary Classification）

二分类中输出值只有0和1两个离散值，通常使用逻辑回归模型解决二分类问题。

Binary Classification

$$\begin{cases} 1, & cat \\ 0, & not\ cat \end{cases}$$



X_orig = 64x64x3=12288

$$X = \begin{bmatrix} 255 \\ 231 \\ \cdots \\ 255 \\ 134 \\ \cdots \\ 132 \end{bmatrix} \qquad n = n_x = 12288$$

彩色图片包含RGB三个通道。以上述图片为例，输入值的维度是（64，64，3）。将每个通道一行一行取出，再拼接起来转化为一维的**特征向量（featurevector）**，此时输入特征向量维度为（12288，1）。通过该向量预测对应的输出是0还是1这就是一个二分类问题。

逻辑回归（Logistic Regression）

逻辑回归中，预测值用 $\hat{y} = P(y = 1 \mid x)$ 表示。

它的含义是：给定输入特征 x 时，预测标签 $y = 1$ 的概率，其取值范围在[0,1]之间。

当引入线性模型，逻辑回归的预测输出可以完整写成：

$$\hat{y} = Sigmoid(w^T x + b) = \sigma(w^T x + b)$$

Sigmoid函数的一阶导数可以用其自身表示：

$$\sigma\prime(z) = \sigma(z)(1 - \sigma(z))$$

损失函数（Loss Function）

单个样本的Cost Function用Loss Function来表示，使用**平方误差（squared error）**：

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

逻辑回归一般不使用平方误差来作为Loss Function，因为平方误差是非凸的，而非凸函数在使用梯度下降算法时，容易得到局部最小值（local minimum）即局部最优化。而最优化的目标是得到全局最优化（Global optimization），因此Loss Function应该是凸的。

构建另外一种Loss function：

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

当 $y=1$ 时， $L(\hat{y},y)=-log\hat{y}$ ；

如果 $\hat{y}$ 越接近1， $L(\hat{y},y)$ 近似为0，则表示预测效果越好；

如果 $\hat{y}$ 越接近0， $L(\hat{y},y)$ 近似为 $+\infty$ ，则表示预测效果越差；

当 $y=0$ 时， $L(\hat{y},y)=-log(1-\hat{y})$ ；

如果 $\hat{y}$ 越接近1， $L(\hat{y},y)$ 近似为 $+\infty$ ，则表示预测效果越差；

如果 $\hat{y}$ 越接近0， $L(\hat{y},y)$ 近似为0，则表示预测效果越好；

代价函数（Cost Function）

Cost function是m个样本的**Loss function**的平均值，反映了m个样本的预测输出与真实输出的平均接近程度:

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

梯度下降（Gradient Descent）

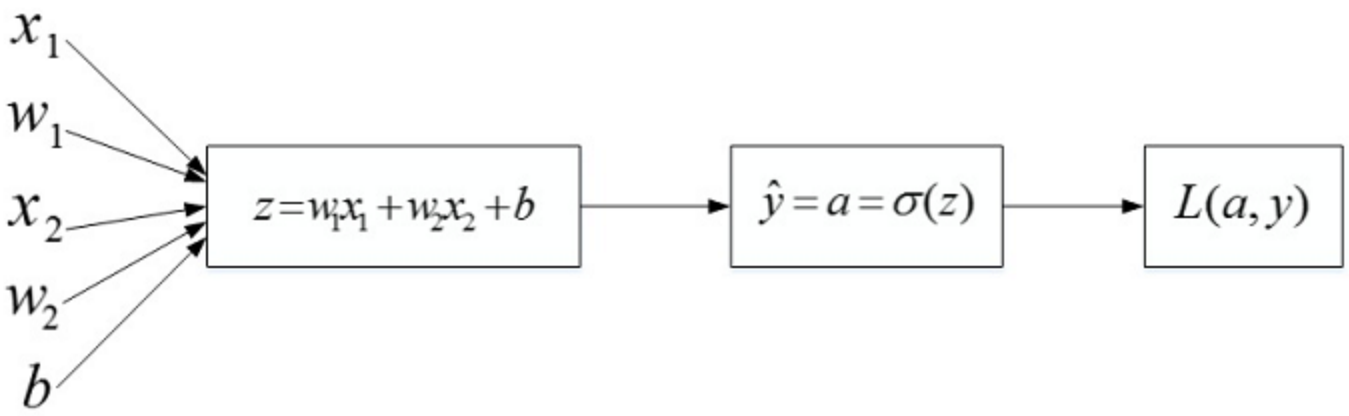
对单个样本而言，**逻辑回归Loss function**表达式如下：

$$z=w^Tx+b$$

$$\hat{y}=a=\sigma(z)$$

$$L(a,y)=-(ylog(a)+(1-y)log(1-a))$$

Logistic regression recap



计算该逻辑回归的反向传播过程:

$$da = \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = (-\frac{y}{a} + \frac{1-y}{1-a}) \cdot a(1-a) = a - y$$

$$dw_1 = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = x_1 \cdot dz = x_1(a-y)$$

$$dw_2 = \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_2} = x_2 \cdot dz = x_2(a-y)$$

$$db = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} = dz = a - y$$

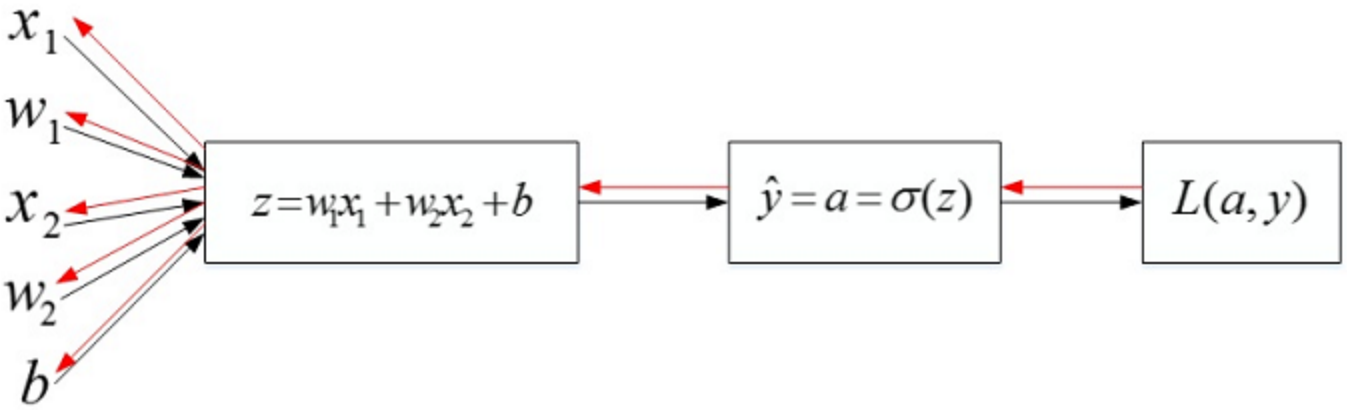
则梯度下降算法可表示为:

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

Logistic regression derivatives



对m个样本而言，**逻辑回归Cost function**表达式如下：

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$$

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

计算该逻辑回归的反向传播过程:

$$dw_m = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_m}L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m} x_m^{(i)}(a^{(i)} - y^{(i)})$$

$$db = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b}L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

编程作业1：识别猫

搭建一个能够**识别猫**的简单的神经网络

准备工作

将数据集先下载至项目目录下

下载地址：https://pan.baidu.com/s/1l4MBm7QwRGuQDp-lZc9P1Q&shfl=sharepset

密码：2u3w

Step1：加载必要的库

加载必要的库 Python 复制代码

```python
import h5py #用于读取文件
import matplotlib.pyplot as plt # 用于查看图片
import numpy as np # 矩阵操作
```

Step2：定义超参数

定义超参数 Python 复制代码

```python
EPOCHS = 2000   # 训练次数
ALPHA = 0.005   # 学习率
```

Step3：加载数据集

加载并处理数据集 Python 复制代码

```python
train_set = h5py.File("./datasets/train_catvnoncat.h5","r")
test_set = h5py.File("./datasets/test_catvnoncat.h5","r")

for key in train_set.keys():  # 查看训练集结构
    print(key)   # train_set_x  train_set_y
train_set['train_set_x'].shape # (209, 64, 64, 3) 有209张图片，图片分辨率64*64，颜色通道为3
train_set['train_set_y'].shape # (209,) 标注图片标签（是否为猫）

for key in test_set.keys():  # 查看测试集结构
    print(key)    # test_set_x   test_set_y
test_set['test_set_x'].shape # (50, 64, 64, 3) 有50张图片，图片分辨率64*64，颜色通道为3
test_set['test_set_y'].shape # (50,) 标注图片标签（是否为猫）

# 将数据集中的内容和标签分开
train_data_org = train_set['train_set_x'][:]  # 保存的是训练集里面的图像数据，即209张64x64的3通道图像
train_labels_org = train_set['train_set_y'][:] # 保存的是训练集的图像对应的分类值，其中0表示不是猫，1表示是猫
test_data_org = test_set['test_set_x'][:] # 保存的是测试集里面的图像数据，即50张64x64的3通道图像
test_labels_org = test_set['test_set_y'][:] # 保存的是测试集里面的图像对应的分类值，其中0表示不是猫，1表示是猫
```

Step4：查看图片

查看图片 Python 复制代码

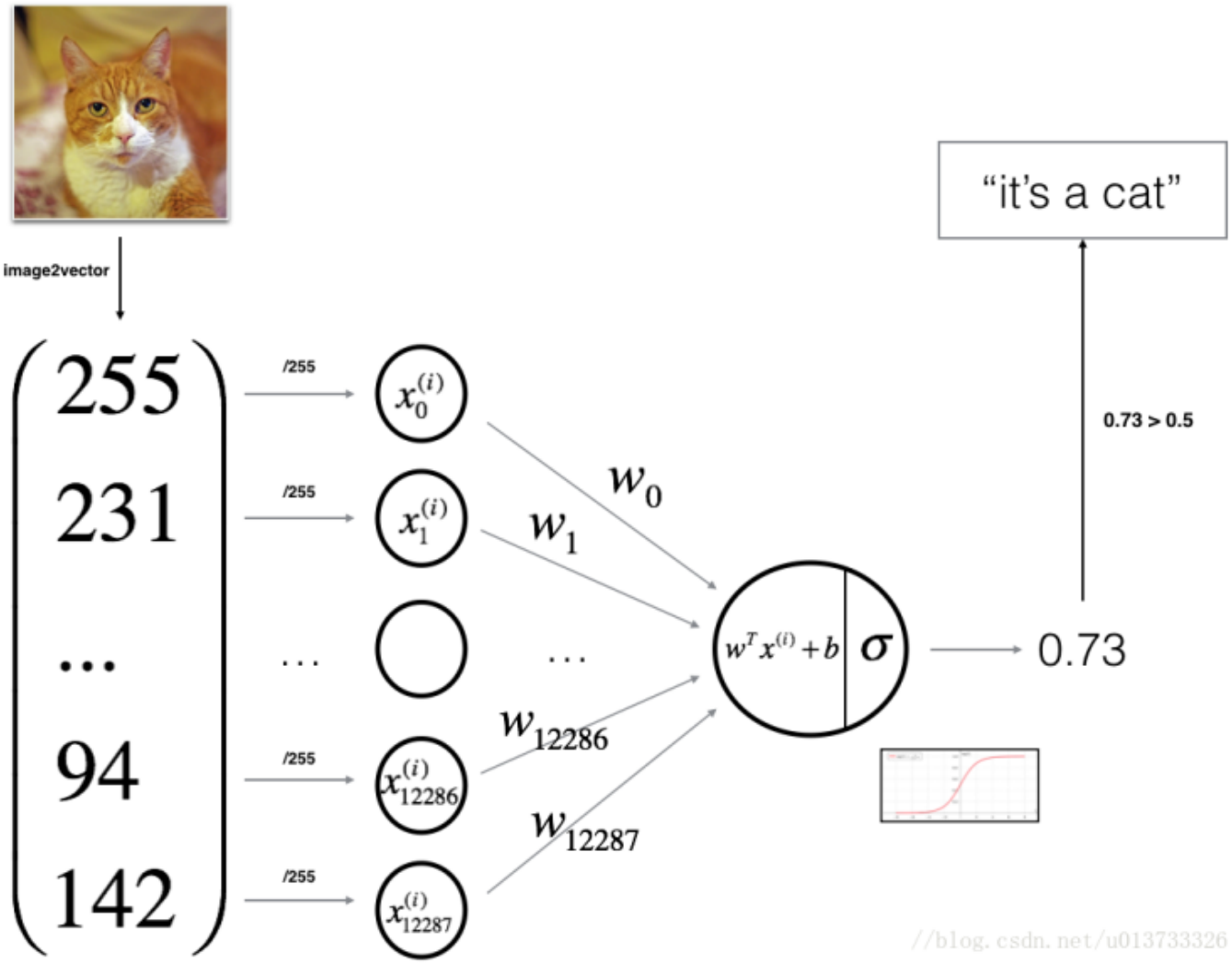
```python
%matplotlib inline
plt.imshow(train_data_org[176])
```

Step5：处理数据集

处理数据集 Python 复制代码

```python
# 维度处理
# 对于训练集样本：(209, 64, 64, 3) ----> (12288, 209)
# 对于测试集样本：(50, 64, 64, 3) ----->(12288, 50)
m_train = train_data_org.shape[0] # 训练集个数
m_test = test_data_org.shape[0] # 测试集个数
train_data_trans = train_data_org.reshape(m_train,-1).T # 训练集维度处理
test_data_trans = test_data_org.reshape(m_test,-1).T # 测试集维度处理
print(train_data_org.shape,test_data_org.shape)
print(train_data_trans.shape,test_data_trans.shape)

# 对于训练集标签：(209,) ----> (1, 209)
# 对于测试集标签：(50,) -----> (1, 50)
train_labels_trans = train_labels_org.reshape(1,m_train) # 训练集标签维度处理
test_labels_trans = test_labels_org.reshape(1,m_test) # 训练集标签维度处理
print(train_labels_org.shape,test_labels_org.shape)
print(train_labels_trans.shape,test_labels_trans.shape)

# 标准化数据
train_data_norm = train_data_trans / 255
test_data_norm = test_data_trans / 255
print(train_data_trans)
print(train_data_norm)
```

Step6：构建模型



构建模型 Python 复制代码

```python
# 前向传播
def propagate(w,b,x,y):
    # 前向传播
    Z = np.dot(w.T,x)+b
    A = 1/(1 + np.exp(-Z))

    # 代价函数
    m = x.shape[1]
    J = -1/m * np.sum(y * np.log(A) + (1-y) * np.log(1-A))

    # 计算梯度
    dw = 1/m * np.dot(x,(A-y).T)
    db = 1/m * np.sum(A-y)

    grands = {'dw':dw,'db':db}
    return grands,J

# 反向传播
def optimize(w,b,X,y):
    costs = []
    for i in range(EPOCHS):
        grands, J = propagate(w,b,X,y)
        dw = grands['dw']
        db = grands['db']
        w = w - ALPHA * dw
        b = b - ALPHA * db

        if i%100 == 0:
            costs.append(J)
            print("迭代次数为：",i,"损失是：",J)

    grands = {'dw':dw,'db':db}
    parameters = {'w':w,'b':b}
    return grands,parameters,costs
```

Step7：预测

预测 Python 复制代码

```python
def predict(w,b,X_test):
    Z = np.dot(w.T,X_test)+b
    A = 1/(1 + np.exp(-Z))

    m = X_test.shape[1]
    y_pred = np.zeros((1,m))
    for i in range(m):
        if A[:,i] > 0.5:
            y_pred[:,i] = 1
        else:
            y_pred[:,i] = 0
    return y_pred
```

Step8：模型整合并调用

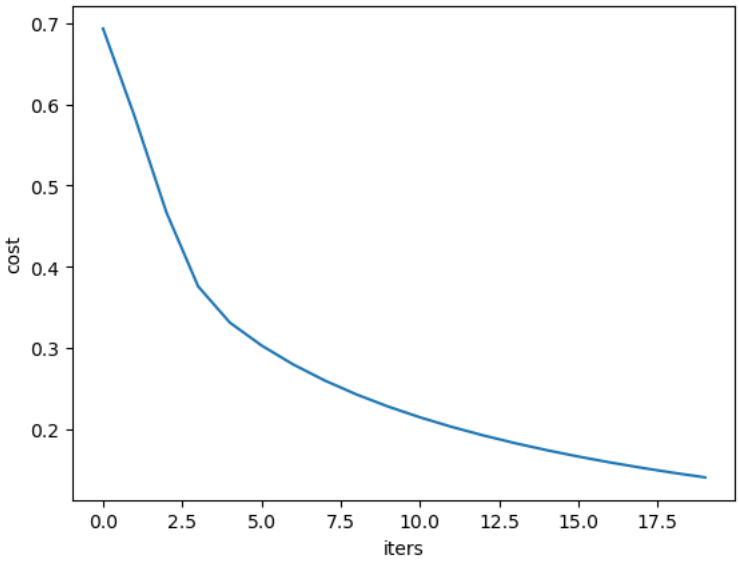
模型整合并调用 Python 复制代码

```python
def model(w,b,x_train,y_train,x_test,y_test):
    grands,params,costs = optimize(w,b,x_train,y_train)
    w = params['w']
    b = params['b']
    y_pred_train = predict(w,b,x_train)
    y_pred_test = predict(w,b,x_test)
    print("训练集上的准确率：",np.mean(y_pred_train == y_train)*100,'%')
    print("测试集上的准确率：",np.mean(y_pred_test == y_test)*100,'%')

    return w,b,y_pred_train,y_pred_test,costs

# 初始化参数
n_dim = train_data_norm.shape[0]
w = np.zeros((n_dim,1))
b = 0
result = model(w,b,train_data_norm,train_labels_trans,test_data_norm,test_
labels_trans)
```

Step9：绘图

绘图 Python 复制代码

```python
plt.plot(result[4])
plt.xlabel("iters")
plt.ylabel("cost")
```



Step10：单幅图片的测试

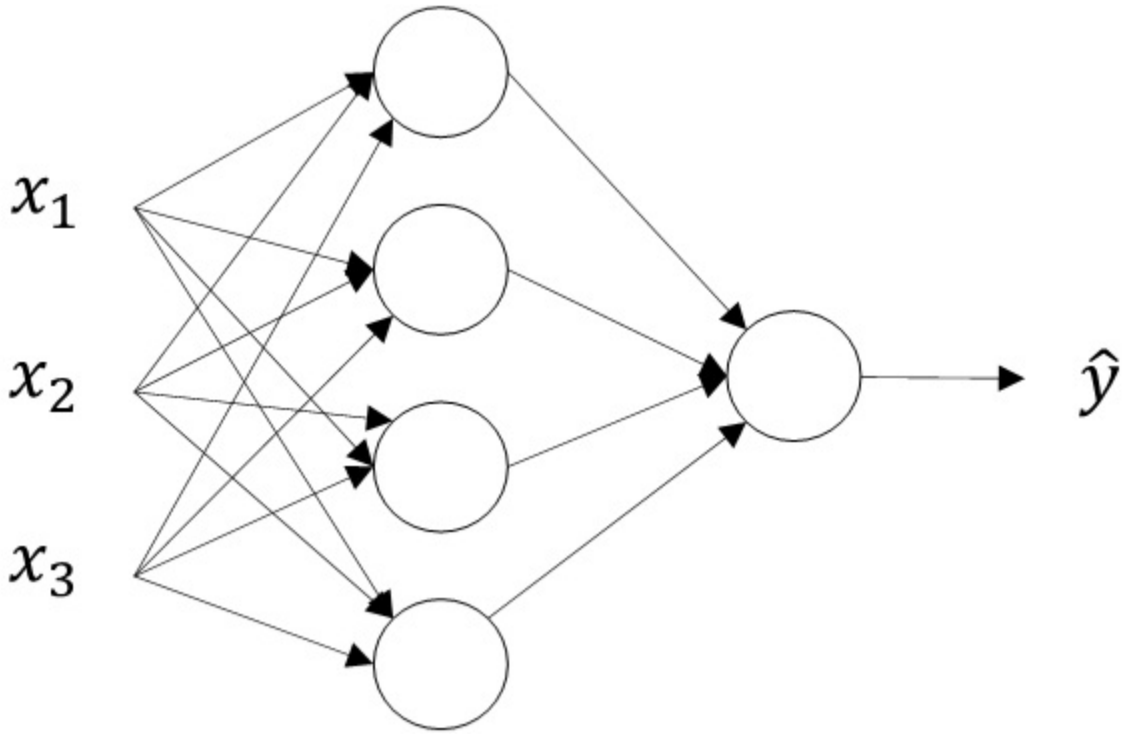
单幅图片的测试 Python 复制代码

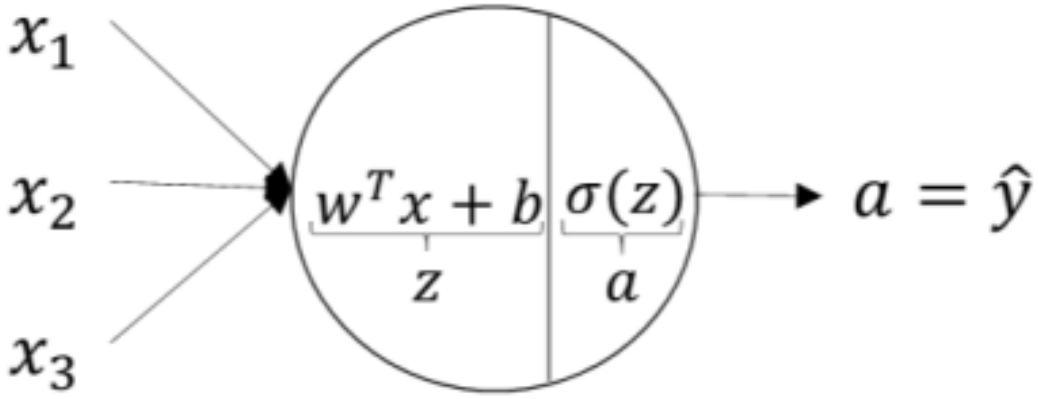
```
index = 1
print('y is',test_labels_trans[0,index])
print('y prediction is',result[3][0,index])
plt.imshow(test_data_org[index])
```

浅层神经网络

单隐藏层神经网络

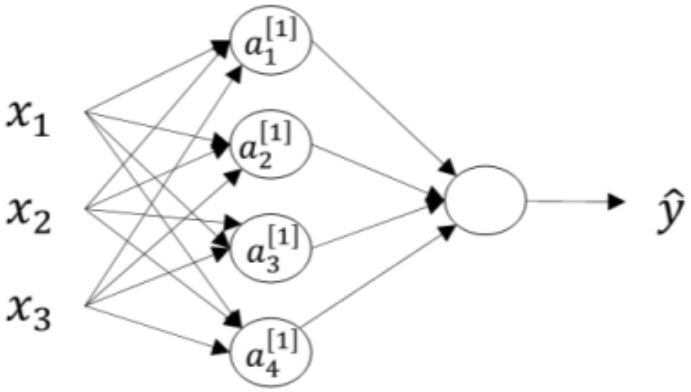
单隐藏层神经网络就是典型的**浅层（shallow）神经网络**





两层神经网络

两层神经网络可以看成是逻辑回归再重复计算一次



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

常见的非线性激活函数

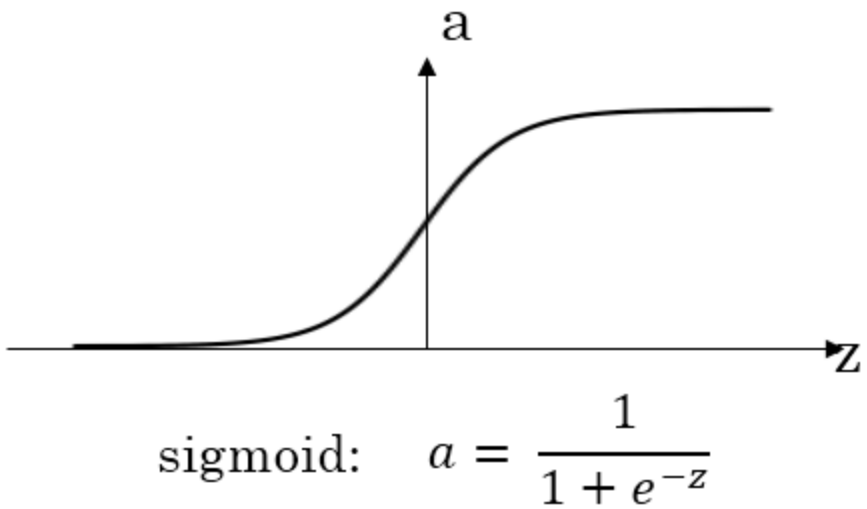
多层隐藏层的神经网络，如果使用线性函数作为激活函数，最终的输出仍然是输入的线性模型：

$$\hat{y} = W_2(W_1x + b_1) + b_2 = W_2W_1x + W_2b_1 + b_2$$

这样的话神经网络就没有任何作用了。因此，隐藏层的激活函数必须要是非线性的。

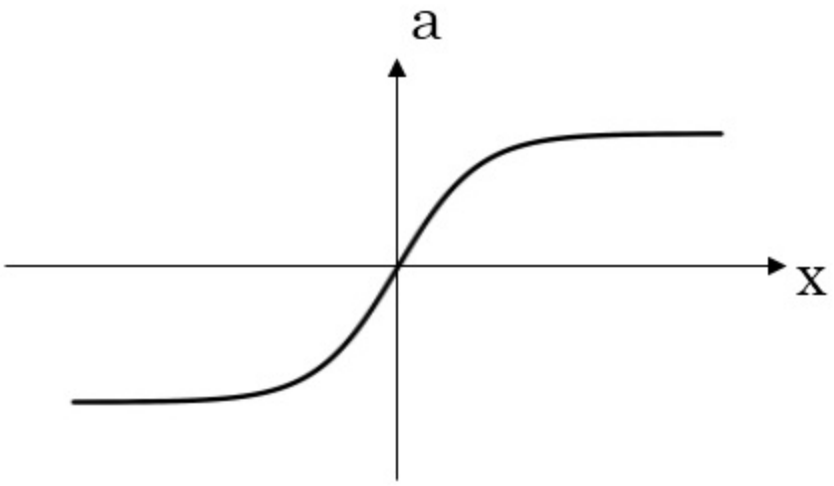
特殊的，对于预测房价等问题可以使用线性激活函数。

sigmoid函数



该函数适合作为输出层的激活函数，因为二分类问题的输出取值为 $\{0, 1\}$ 。

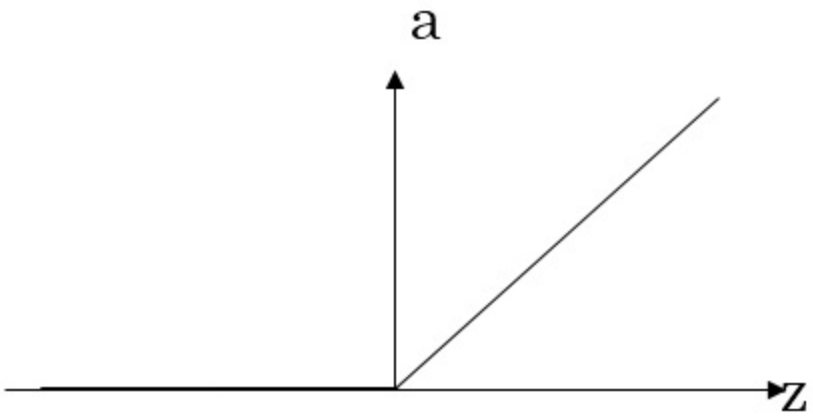
tanh函数



$$\text{tanh:} \quad a = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

该函数适合作为隐藏层的激活函数，因为该函数的取值范围在 $[-1,+1]$ 之间，隐藏层的输出被限定在 $[-1,+1]$ 之间，即有归一化的作用。

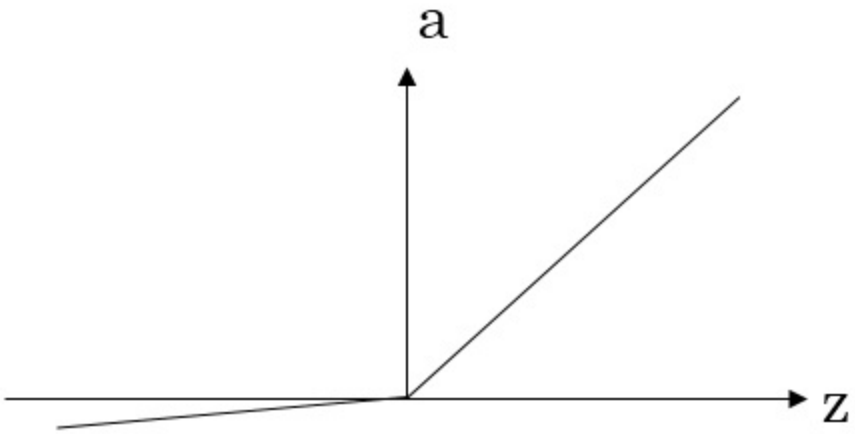
ReLU函数



$$\text{ReLU:} \quad a = \max(0, z)$$

选择该函数作为激活函数能够保证 $z > 0$ 时梯度始终为 1 ，从而提高神经网络梯度下降算法运算速度。但当 $z < 0$ 时，存在梯度为 0 的缺点。如果输出值恒为正值，可以考虑使用该函数作为激活函数。

Leaky ReLU函数

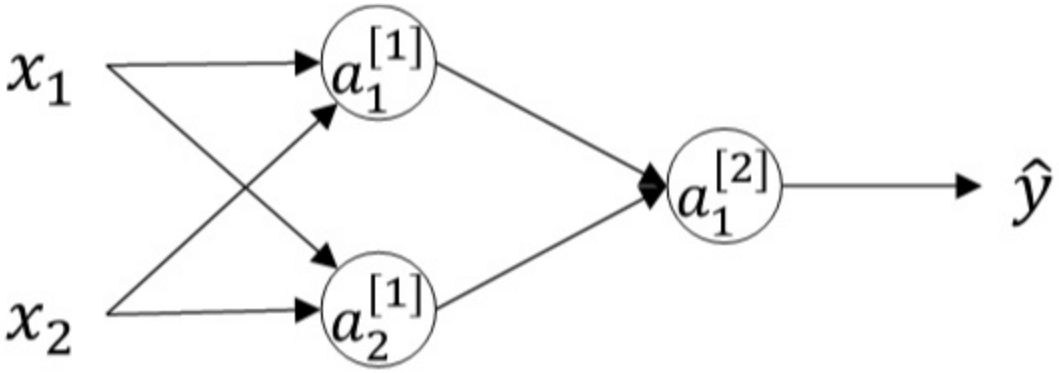


$$\text{Leaky ReLU: } a = \max(0.01z, z)$$

该函数保证了 $z < 0$ 时，梯度不为 0 。

随机初始化

神经网络模型中的参数权重 W 不能全部初始化为零



如果权重 $W^{[1]}$ 和 $W^{[2]}$ 都初始化为零，即：

$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

这样会使得隐藏层第一个神经元的输出等于第二个神经元的输出，也会使得 $dW_1^{[1]} = dW_2^{[1]}$ ，最终 $W_1^{[1]}$ 和 $W_2^{[1]}$ 每次迭代更新都会得到完全相同的结果，因此引出随机初始化：

随机初始化 Python 复制代码

```python
W_1 = np.random.randn(2,2)*0.01
b_1 = np.zero((2,1))
W_2 = np.random.randn(1,2)*0.01
b_2 = 0
```

编程作业2：带有一个隐藏层的平面数据分类

搭建一个神经网络实现平面数据分类

准备工作

先安装必要的库 conda install scikit-learn

Step1：加载必要的库

加载必要的库 Python 复制代码

```python
import numpy as np
import matplotlib.pyplot as plt
from testCases import *
from planar_utils import *
import sklearn
import sklearn.datasets
import sklearn.linear_model

%matplotlib inline
```

Step2：定义超参数

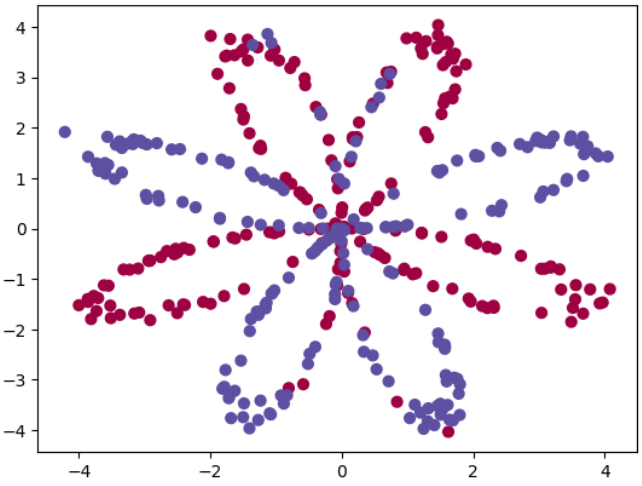
定义超参数 Python 复制代码

```python
EPOCHS = 2000
ALPHA = 0.5
```

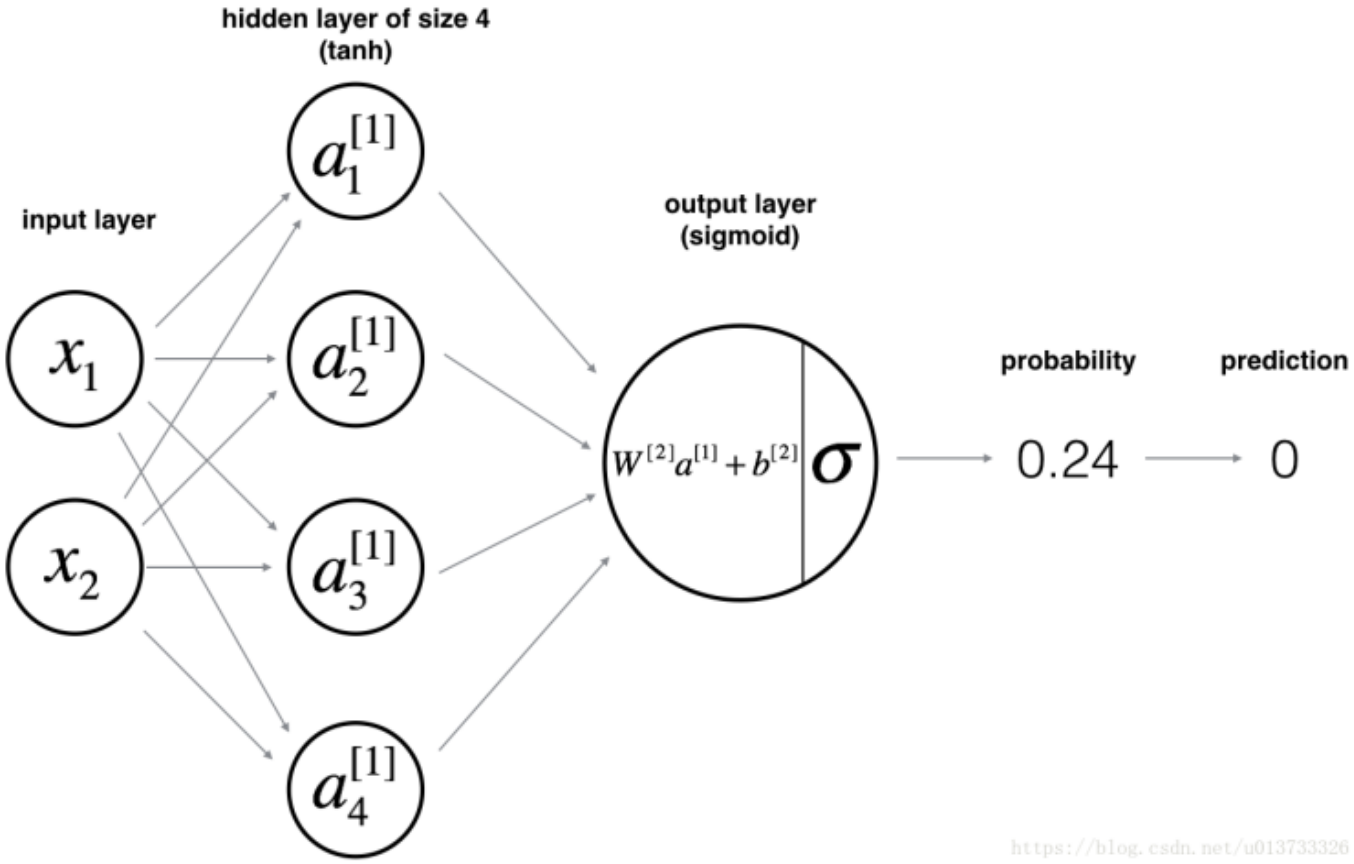
Step3：加载数据集

加载数据集 Python 复制代码

```python
X, Y = load_planar_dataset()
print("X的维度:",X.shape)   # 2*400   记录数据点位置
print("Y的维度:",Y.shape)   # 1*400   标记数据点类型：红色0，蓝色1
m = Y.shape[1]   # 400记录样本个数
print("样本个数:",m)
plt.scatter(X[0, :], X[1, :], c=Y, s=40, cmap=plt.cm.Spectral) #绘制散点图
```



Step4：构建模型



构建模型 Python 复制代码

```python
# 定义神经网络结构
def layer_size(X,Y):
    n_x = X.shape[0]     # 输入层结点数 此处是横纵坐标 取2
    n_h = 4            # 隐含层结点数 规定为4
    n_y = Y.shape[0]     # 输出层结点数 此处是标签0|1 取1
    return n_x,n_h,n_y

# 初始化参数
def init_params(n_x,n_h,n_y):
    W1=np.random.randn(n_h,n_x)*0.01    # 生成4*2的矩阵，矩阵元素服从N(0,1)
    b1=np.zeros((n_h,1))                # 生成4*1的矩阵，矩阵元素为0
    W2=np.random.randn(n_y,n_h)
    b2=np.zeros((n_y,1))
    params = { "W1" : W1,
               "W2" : W2,
               "b1" : b1,
               "b2" : b2 }
    return params

# 前向传播和反向传播
def forward_backword(X, Y, W1, W2, b1, b2):
    # 前向传播
    Z1 = np.dot(W1,X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)

    # 计算损失
    J = -1/m * np.sum(Y * np.log(A2) + (1-Y) * np.log(1-A2))

    # 反向传播
    dZ2= A2 - Y
    dW2 = (1 / m) * np.dot(dZ2, A1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    dW1 = (1 / m) * np.dot(dZ1, X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    W1 = W1 - ALPHA*dW1
    W2 = W2 - ALPHA*dW2
    b1 = b1 - ALPHA*db1
    b2 = b2 - ALPHA*db2

    return W1, W2, b1, b2,J
```

Step5: 预测

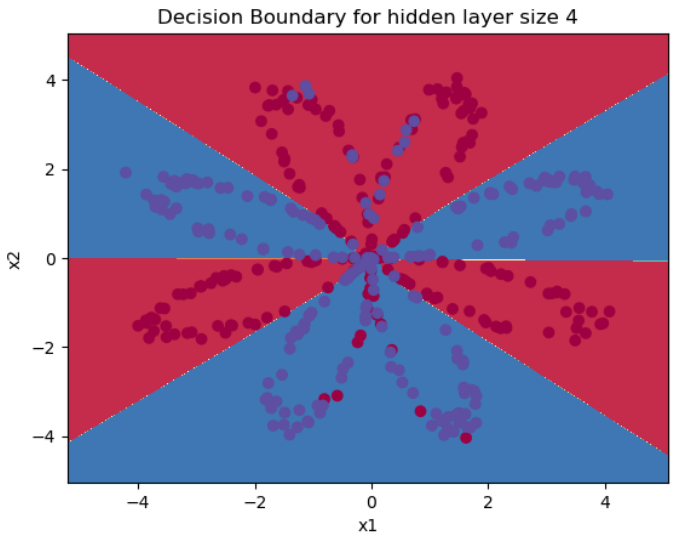
预测 Python 复制代码

```python
# 预测
def predict(parameters,X):
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]
    # 前向传播
    Z1 = np.dot(W1,X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2,A1) + b2
    A2 = sigmoid(Z2)
    prediction = np.round(A2)
    return prediction
```

Step6：模型整合并调用

模型整合并调用 Python 复制代码

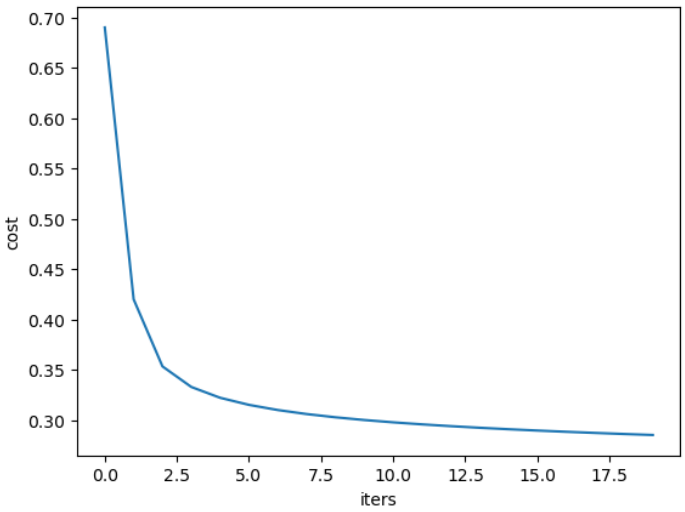
```python
# 模型整合
def nn_model(X,Y):
    n_x = layer_size(X,Y)[0]
    n_h = layer_size(X,Y)[1]
    n_y = layer_size(X,Y)[2]
    params = init_params(n_x,n_h,n_y)
    W1 = params["W1"]
    W2 = params["W2"]
    b1 = params["b1"]
    b2 = params["b2"]
    costs = []
    for i in range (EPOCHS):
        W1, W2, b1, b2, J = forward_backword(X, Y, W1, W2, b1 ,b2)
        if i%100 == 0:
            costs.append(J)
            print("迭代次数为: ",i,"损失是: ",J)
    parameters = {
        'W1':W1,
        'W2':W2,
        'b1':b1,
        'b2':b2
    }
    return costs,parameters

# 调用
costs,p = nn_model(X, Y)

#绘制边界
plot_decision_boundary(lambda x: predict(p, x.T), X, Y)
plt.title("Decision Boundary for hidden layer size " + str(4))

predictions = predict(p, X)
print ('准确率: %d' % float((np.dot(Y, predictions.T) +
        np.dot(1 - Y, 1 - predictions.T)) / float(Y.size) * 100) + '%')
```



Step7：绘图

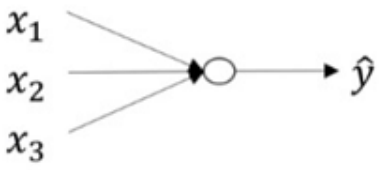
绘图 Python 复制代码

```
plt.plot(costs)
plt.xlabel("iters")
plt.ylabel("cost")
```

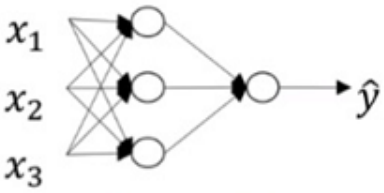


深层神经网络

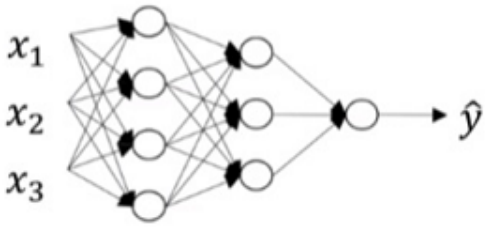
深层神经网络的结构



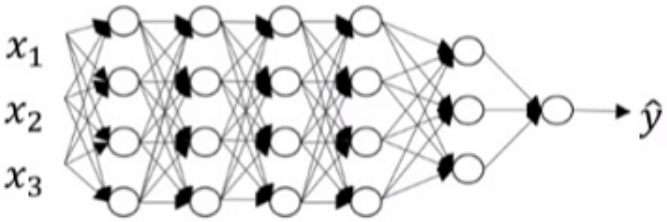
logistic regression



1 hidden layer



2 hidden layers



5 hidden layers

对于一个 L 层的神经网络，它包含了 $L-1$ 个隐藏层，最后的 L 层是输出层；

$a^{[l]}$ 和 $W^{[l]}$ 的上标 l 均从 1 开始计数 $l=1,\cdots,L$ ；

把输入 x 记为 $a^{[0]}$ ，输出层的 $\hat{y}$ 记为 $a^{[L]}$

第 L 层有： $Z^{[L]}=W^{[L]}A^{[L-1]}+b^{[L]}$ ，其中 $W^{[L]}$ 的维度是 $(n^{[L]},n^{[L-1]})$ ， $b^{[L]}$ 的维度是 $(n^{[L]},1)$

此外 $dW^{[L]}$ 和 $W^{[L]}$ 的维度一致， $db^{[L]}$ 和 $b^{[L]}$ 的维度一致.

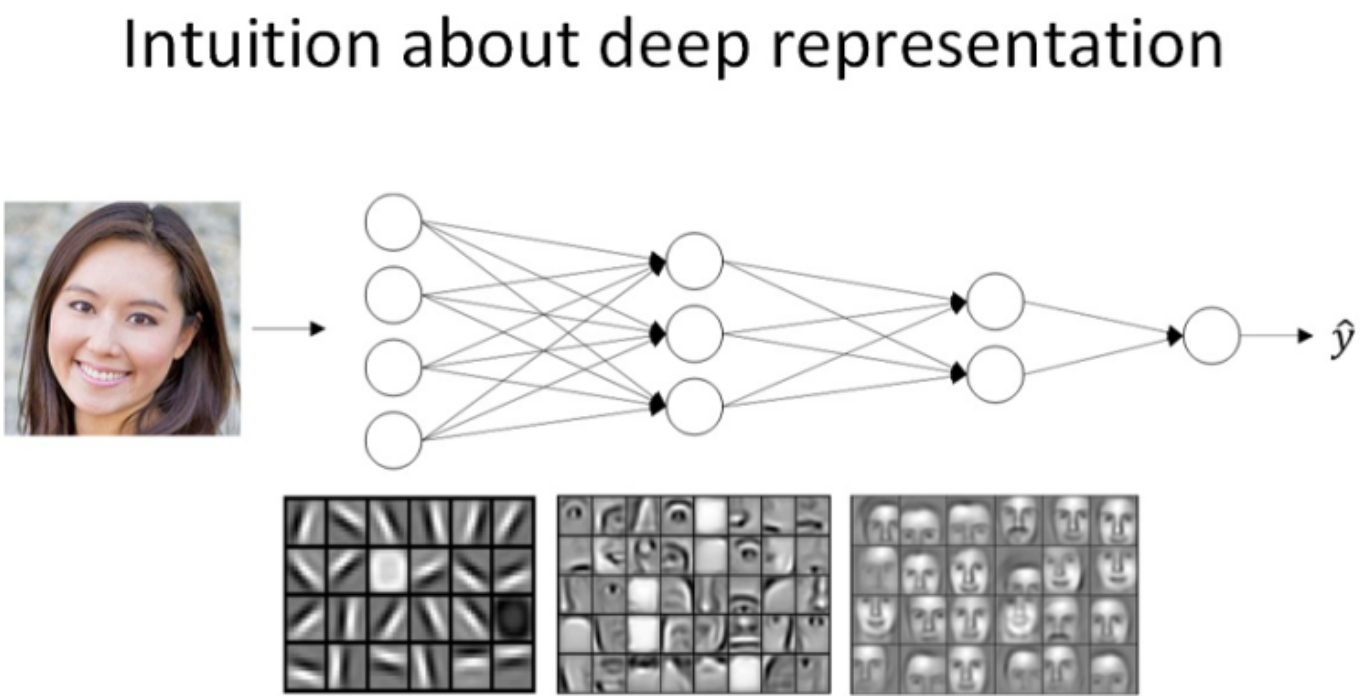
前向传播和反向传播

Summary of gradient descent

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

深层神经网络的优势

1.人脸识别



神经网络第一层所做的事就是从原始图片中提取出人脸的轮廓与边缘，即**边缘检测**。这样每个神经元得到的是一些边缘信息。神经网络第二层所做的事情就是将前一层的边缘进行组合，组合成人脸一些局部特征，比如眼睛、鼻子、嘴巴等。再往后面，就将这些局部特征组合起来，融合成人脸的模样。

随着层数由浅到深，神经网络提取的特征也是从边缘到局部特征到整体，由简单到复杂。如果隐藏层足够多，那么能够提取的特征就越丰富、越复杂，模型的准确率就会越高。

2.语音识别

浅层的神经元能够检测一些简单的音调，较深的神经元能够检测出基本的音素，更深的神经元就能够检测出单词信息。如果网络够深，还能对短语、句子进行检测。

神经网络从左到右，神经元提取的特征从简单到复杂。特征复杂度与神经网络层数成正相关。特征越来越复杂，功能也越来越强大。

3.神经元个数

处理同一逻辑问题，深层网络所需的神经元个数比浅层网络要少很多处理同一逻辑问题，深层网络所需的神经元个数比浅层网络要少很多。

参数和超参数

参数：$W^{[L]}$ 、 $b^{[L]}$

超参数：学习速率 α 、训练迭代次数、神经网络层数、各层神经元个数，激活函数等

叫做超参数的原因是它们决定了参数的值

如何设置最优的超参数：

通常的做法是选择超参数一定范围内的值，分别代入神经网络进行训练，测试**cost function**随着迭代次数增加的变化，根据结果选择**cost function**最小时对应的超参数值

编程作业3：搭建多层神经网络——2层

准备工作

将数据集先下载至项目目录下

下载地址：https://pan.baidu.com/s/1l4MBm7QwRGuQDp-lZc9P1Q&shfl=sharepset

密码：2u3w

Step1：加载必要的库

加载必要的库 Python 复制代码

```python
# 加载必要的库
import h5py
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Step2：加载数据集

加载数据集 Python 复制代码

```python
# 加载数据集
train_set = h5py.File("./datasets/train_catvnoncat.h5","r")
test_set = h5py.File("./datasets/test_catvnoncat.h5","r")

for key in train_set.keys():  # 查看训练集结构
    print(key)   # train_set_x  train_set_y
train_set['train_set_x'].shape # (209, 64, 64, 3) 有209张图片，图片分辨率64*64，颜色通道为3
train_set['train_set_y'].shape # (209,) 标注图片标签（是否为猫）

for key in test_set.keys():  # 查看测试集结构
    print(key)    # test_set_x   test_set_y
test_set['test_set_x'].shape # (50, 64, 64, 3) 有50张图片，图片分辨率64*64，颜色通道为3
test_set['test_set_y'].shape # (50,) 标注图片标签（是否为猫）

# 将数据集中的内容和标签分开
train_data_org = train_set['train_set_x'][:]  # 保存的是训练集里面的图像数据，即209张64x64的3通道图像
train_labels_org = train_set['train_set_y'][:] # 保存的是训练集的图像对应的分类值，其中0表示不是猫，1表示是猫
test_data_org = test_set['test_set_x'][:] # 保存的是测试集里面的图像数据，即50张64x64的3通道图像
test_labels_org = test_set['test_set_y'][:] # 保存的是测试集里面的图像对应的分类值，其中0表示不是猫，1表示是猫
```

Step3：处理数据集

处理数据集 Python 复制代码

```python
# 处理数据集------维度处理
# 对于训练集样本：(209, 64, 64, 3) ----> (12288, 209)
# 对于测试集样本：(50, 64, 64, 3) ----->(12288, 50)
m_train = train_data_org.shape[0] # 训练集个数
m_test = test_data_org.shape[0] # 测试集个数
train_data_trans = train_data_org.reshape(m_train,-1).T # 训练集维度处理
test_data_trans = test_data_org.reshape(m_test,-1).T # 测试集维度处理
print(train_data_org.shape,test_data_org.shape)
print(train_data_trans.shape,test_data_trans.shape)

# 对于训练集标签：(209,) ----> (1, 209)
# 对于测试集标签：(50,) -----> (1, 50)
train_labels_trans = train_labels_org.reshape(1,m_train) # 训练集标签维度处理
test_labels_trans = test_labels_org.reshape(1,m_test) # 训练集标签维度处理
print(train_labels_org.shape,test_labels_org.shape)
print(train_labels_trans.shape,test_labels_trans.shape)

# 处理数据集------标准化数据
train_data_norm = train_data_trans / 255
test_data_norm = test_data_trans / 255
print(train_data_trans)
print(train_data_norm)
```

Step4：定义超参数

定义超参数 Python 复制代码

```python
# 定义超参数
EPOCHS = 3000
ALPHA = 0.0075
layers_dims = [12288,7,1] # 神经元数量
```

Step5：初始化参数

初始化参数 Python 复制代码

```python
# 初始化参数
def init_params(layer):
    n_x = layers_dims[layer-1]
    n_h = layers_dims[layer]
    W = np.random.randn(n_h,n_x)/np.sqrt(layers_dims[layer - 1])
    b = np.zeros((n_h,1))
    return W,b
```

Step6：前向和反向传播

前向和反向传播 Python 复制代码

```python
def forward_backward(X, Y, W1, W2, b1, b2):
    # 前向传播
    Z1 = np.dot(W1,X)+b1
    assert(W1.shape == (7,12288))
    assert(b1.shape == (7,1))
    assert(Z1.shape == (7,209))
    A1 = np.maximum(0,Z1)
    assert(A1.shape == (7,209))
    Z2 = np.dot(W2,A1)+b2
    assert(W2.shape == (1,7))
    assert(b2.shape == (1,1))
    assert(Z2.shape == (1,209))
    A2 = 1/(1 + np.exp(-Z2))
    assert(A2.shape == (1,209))

    # 计算损失
    m = X.shape[1]
    J = -1/m * np.sum(Y * np.log(A2) + (1-Y) * np.log(1-A2))

    # 反向传播
    dA2 = - (np.divide(Y, A2) - np.divide(1 - Y, 1 - A2))
    dZ2 = A2 - Y
    dW2 = (1 / m) * np.dot(dZ2, A1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dA1 = np.dot(W2.T, dZ2)

    dZ1 = np.array(dA1, copy=True)
    dZ1[A1 <= 0] = 0

    dW1 = (1 / m) * np.dot(dZ1,X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    # 更新梯度
    W1 = W1 - ALPHA*dW1
    W2 = W2 - ALPHA*dW2
    b1 = b1 - ALPHA*db1
    b2 = b2 - ALPHA*db2
    return W1, W2, b1, b2,J
```

Step7：预测

预测 Python 复制代码

```python
def predict(W1,b1,W2,b2,X):
    # 前向传播
    Z1 = np.dot(W1,X)+b1
    A1 = np.maximum(0,Z1)
    Z2 = np.dot(W2,A1)+b2
    A2 = 1/(1 + np.exp(-Z2))

    m = X.shape[1]
    y_pred = np.zeros((1,m))
    for i in range(m):
        if A2[:,i] > 0.5:
            y_pred[:,i] = 1
        else:
            y_pred[:,i] = 0

    return y_pred
```

Step8：模型整合

模型整合 Python 复制代码

```python
# 模型整合
def nn_model(X,Y,test_data_norm,test_labels_trans):
    W1,b1 = init_params(1)
    W2,b2 = init_params(2)
    costs = []
    for i in range (EPOCHS):
        W1, W2, b1, b2, J = forward_backward(X, Y, W1, W2, b1 ,b2)
        if i%100 == 0:
            costs.append(J)
            print("迭代次数为：",i,"损失是：",J)
    y_pred_train = predict(W1,b1,W2,b2,X)
    y_pred_test = predict(W1,b1,W2,b2,test_data_norm)
    print("训练集上的准确率：",np.mean(y_pred_train == Y)*100,'%')
    print("测试集上的准确率：",np.mean(y_pred_test == test_labels_trans)*100,'%')
    return costs,W1,W2,b1,b2
```

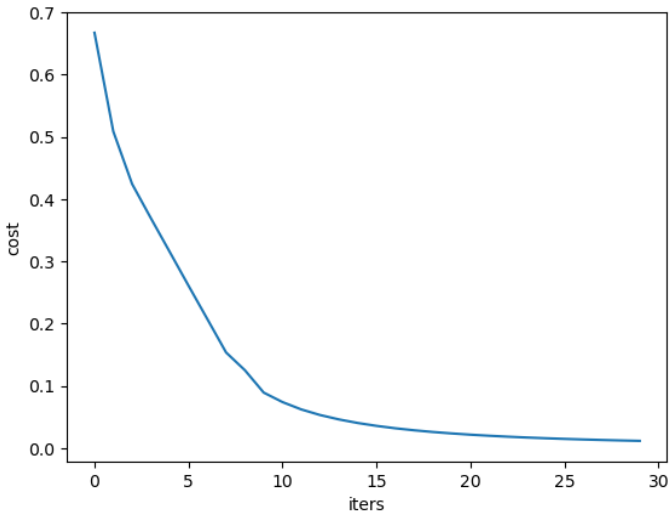
Step9：调用并绘图

调用并绘图 Python 复制代码

```
costs,W1,W2,b1,b2 = nn_model(train_data_norm,train_labels_trans,test_data_norm,test_labels_trans)
plt.plot(costs)
plt.xlabel("iters")
plt.ylabel("cost")
```

训练集上的准确率： 100.0 %

测试集上的准确率： 76.0 %



编程作业3：搭建多层神经网络——4层

准备工作

将数据集先下载至项目目录下

下载地址：https://pan.baidu.com/s/1l4MBm7QwRGuQDp-lZc9P1Q&shfl=sharepset

密码：2u3w

Step1~Step3与之前一致

Step4：定义超参数

定义超参数 Python 复制代码

```
# 定义超参数
EPOCHS = 3000
ALPHA = 0.0075
layers_dims = [12288, 20, 7, 5 ,1]
```

Step5：初始化参数

初始化参数 Python 复制代码

```python
# 初始化参数

# 注意：！！！！！！！！！！！
# 这里对权重矩阵初始化方法与之前几章的方法不同！！！！！

def init_params(layer):
    n_x = layers_dims[layer-1]
    n_h = layers_dims[layer]
    W = np.random.randn(n_h,n_x)/np.sqrt(layers_dims[layer - 1])
    b = np.zeros((n_h,1))
    return W,b
```

Step6：前向和反向传播

前向和反向传播 Python 复制代码

```python
def deep_forward_backward(X, Y, W1, W2, W3, W4, b1, b2, b3, b4):
    Z1 = np.dot(W1, X) + b1
    A1 = np.maximum(0,Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = np.maximum(0,Z2)
    Z3 = np.dot(W3, A2) + b3
    A3 = np.maximum(0,Z3)
    Z4 = np.dot(W4, A3) + b4
    A4 = 1/(1+np.exp(-Z4))

    m = X.shape[1]
    cost = -np.sum(Y*np.log(A4) + (1 - Y)*np.log(1 - A4))/m

    dZ4 = A4 - Y
    dW4 = np.dot(dZ4, A3.T)/m
    db4 = dZ4.sum(axis = 1, keepdims = True)/m
    dA3 = np.dot(W4.T, dZ4)

    dZ3 = np.array(dA3, copy=True)
    dZ3[A3 <= 0] = 0
    dW3 = np.dot(dZ3, A2.T)/m
    db3 = dZ3.sum(axis = 1, keepdims = True)/m
    dA2 = np.dot(W3.T, dZ3)

    dZ2 = np.array(dA2, copy=True)
    dZ2[A2 <= 0] = 0
    dW2 = np.dot(dZ2, A1.T)/m
    db2 = dZ2.sum(axis = 1, keepdims = True)/m
    dA1 = np.dot(W2.T, dZ2)

    dZ1 = np.array(dA1, copy=True)
    dZ1[A1 <= 0] = 0
    dW1 = np.dot(dZ1, X.T)/m
    db1 = dZ1.sum(axis = 1, keepdims = True)/m
    dX = np.dot(W1.T, dZ1)

    W1 = W1 - ALPHA*dW1
    W2 = W2 - ALPHA*dW2
    W3 = W3 - ALPHA*dW3
    W4 = W4 - ALPHA*dW4
    b1 = b1 - ALPHA*db1
    b2 = b2 - ALPHA*db2
    b3 = b3 - ALPHA*db3
    b4 = b4 - ALPHA*db4

```

```
46        return W1, W2, W3, W4, b1, b2, b3, b4,cost
```

Step7：预测

预测 Python 复制代码

```python
# 预测
def deep_predict( W1,W2,W3,W4,b1,b2,b3,b4,X):
    Z1 = np.dot(W1, X) + b1
    A1 = np.maximum(0,Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = np.maximum(0,Z2)
    Z3 = np.dot(W3, A2) + b3
    A3 = np.maximum(0,Z3)
    Z4 = np.dot(W4, A3) + b4
    A4 = 1/(1+np.exp(-Z4))

    m = X.shape[1]
    y_pred = np.zeros((1,m))

    for i in range(m):
        if A4[:,i] > 0.5:
            y_pred[:,i] = 1
        else:
            y_pred[:,i] = 0

    return y_pred
```

Step8：模型整合

模型整合 Python 复制代码

```python
# 模型整合
def deep_nn_model(X,Y,test_data_norm,test_labels_trans):
    W1,b1 = init_params(1)
    W2,b2 = init_params(2)
    W3,b3 = init_params(3)
    W4,b4 = init_params(4)
    costs = []
    for i in range (EPOCHS):
        W1, W2, W3, W4, b1, b2, b3, b4,J = deep_forward_backward(X, Y, W1, W2, W3, W4, b1, b2, b3, b4)
        if i%100 == 0:
            costs.append(J)
            print("迭代次数为：",i,"损失是：",J)
    y_pred_train = deep_predict( W1, W2, W3, W4, b1, b2, b3, b4,X)
    y_pred_test = deep_predict( W1, W2, W3, W4, b1, b2, b3, b4,test_data_norm)
    print("训练集上的准确率：",np.mean(y_pred_train == Y)*100,'%')
    print("测试集上的准确率：",np.mean(y_pred_test == test_labels_trans)*100,'%')
    return costs, W1, W2, W3, W4, b1, b2, b3, b4
```

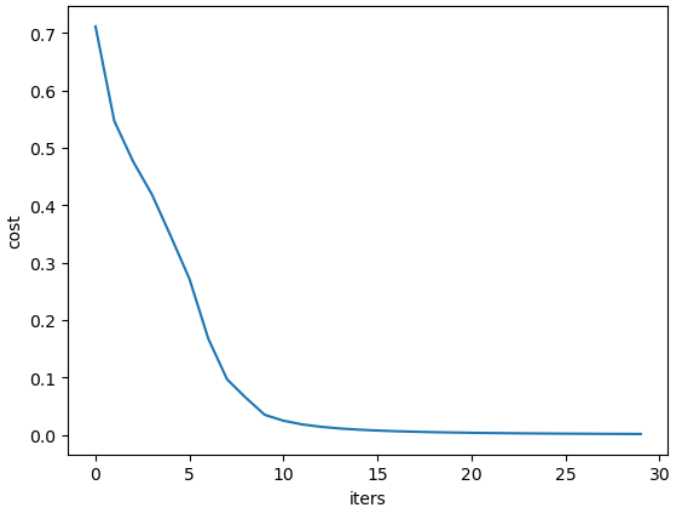
Step9：调用并绘图

调用并绘图 Python 复制代码

```python
costs, W1, W2, W3, W4, b1, b2, b3, b4 = deep_nn_model(train_data_norm,train_labels_trans,test_data_norm,test_labels_trans)
plt.plot(costs)
plt.xlabel("iters")
plt.ylabel("cost")
```

训练集上的准确率： 100.0 %

测试集上的准确率： 80.0 %



改善深层神经网络

结构化机器学习项目

卷积神经网络

自然语言处理