%利用与门、非门、与非门实现异或门

import numpy as np

def AND(x1,x2):

x=np.array([x1,x2])

w= np.array([0.5,0.5])

b=-0.7

tmp=np.sum(w\*x)+b

if tmp<=0:

return 0

else:

return 1

print(AND(0,1))

# ef NAND(x1,x2):

# x =np.array([x1, x2])

# w = np.array([-0.5, -0.5])

# b = 0.7

# tmp = np.sum(w \* x) + b

# if tmp <= 0:

# return 0

# else:

# return 1

#

# def OR(x1,x2):

# x = np.array([x1, x2])

# w = np.array([0.5, 0.5])

# b = -0.2

# tmp = np.sum(w \* x) + b

# if tmp <= 0:

# return 0

# else:

# return 1

#

# def XOR(x1,x2):

# s1=NAND(x1,x2)

# S2=OR(x1,x2)

# y=AND(s1,s2)

# return y

# print(XOR(0,0))

# print(XOR(1,0))

# print(XOR(0,1))

# print(XOR(1,1))

%阶跃函数图像

import numpy as np

import matplotlib.pylab as plt

def step\_function(x):

return np.array(x>0,dtype=np.int)

x=np.arange(-5.0,5.0,0.1)

y=step\_function(x)

plt.plot(x,y)

plt.ylim(-0.1,1.1)

plt.show()

%sigmod函数图像

import numpy as np

import matplotlib.pylab as plt

def sigmod\_function(x):

return 1/(1+np.exp(-x))

x=np.arange(-5.0,5.0,0.1)

y=sigmod\_function(x)

plt.plot(x,y)

plt.ylim(-0.1,1.1)

plt.show()

%阶跃函数图像和sigmod函数图像在同一坐标系中

import numpy as np

import matplotlib.pylab as plt

def sigmod\_function(x):

return 1/(1+np.exp(-x))

def step\_function(x):

return np.array(x>0,dtype=np.int)

x=np.arange(-5.0,5.0,0.1)

y1=sigmod\_function(x)

y2=step\_function(x)

plt.ylim(-0.1,1.1)

plt.plot(x,y1)

plt.plot(x,y2,linestyle="--")

plt.show()

%阶跃函数图像和sigmod函数图像、relu函数在同一坐标系中

import numpy as np

import matplotlib.pylab as plt

def sigmod\_function(x):

return 1/(1+np.exp(-x))

def step\_function(x):

return np.array(x>0,dtype=np.int)

def relu(x):

return np.maximum(0,x)

x=np.arange(-5.0,5.0,0.1)

y1=sigmod\_function(x)

y2=step\_function(x)

y3=relu(x)

plt.ylim(-5.0,5.0)

plt.plot(x,y1)

plt.plot(x,y2,linestyle="--")

plt.plot(x,y3)

plt.show()

%NumPy多维数组的相乘（矩阵的相乘）

mport numpy as np

def sigmoid(x):

return 1/(1+np.exp(-x))

X=np.array([1.0,0.5])

W1=np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])

B1=np.array([0.1,0.2,0.3])

print(W1.shape)

print(X.shape)

print(B1.shape)

A1=np.dot(X,W1)+B1

x=A1

Z1=sigmoid(A1)

print(A1)

print(Z1)

%三层神经网络的实现

1. 分析方法代码

import numpy as np

def sigmoid(x):#定义的隐藏层的激活函数

return 1/(1+np.exp(-x))

def identity(x):#定义的输出层的激活函数，恒等函数

return x

X=np.array([1.0,0.5])#输入层，两个输入神经元

W1=np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])#从输入层到第一层的各个权重

W2=np.array([[0.1,0.4],[0.2,0.5,],[0.3,0.6]])#从第一层到第二层的各个权重

W3=np.array([[0.1,0.3],[0.2,0.4]])#从第二层到输出层的各个权重

B1=np.array([0.1,0.2,0.3])#从输入层到第一层的偏置

B2=np.array([0.1,0.2])#从第一层到第二层的偏置

B3=np.array([0.1,0.2])#从第二层到输出层的偏置

A1=np.dot(X,W1)+B1#从输入层传递到第一层的信号

Z1=sigmoid(A1)#第一层中的激活函数过程

A2=np.dot(Z1,W2)+B2#从第一层传递到第二层的信号

Z2=sigmoid(A2)#第二层中的激活函数过程

A3=np.dot(Z2,W3)+B3#从第二层传递到输出层的信号

Y=identity(A3)#第二层中的激活函数过程，也即最后的输出结果

print(W1.shape)#矩阵的形状

print(W2.shape)

print(X.shape)

print(Z1.shape)

print(B1.shape)

print(B2.shape)

print(A1)

print(Z1)

print(Z2)

print(Y)

1. 整合代码

import numpy as np

def sigmoid(x):#定义的隐藏层的激活函数

return 1/(1+np.exp(-x))

def identity(x):#定义的输出层的激活函数，恒等函数

return x

def init\_network():

network={}

network['W1']=np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])

network['b1']=np.array([0.1,0.2,0.3])

network['W2'] = np.array([[0.1, 0.4], [0.2,0.5],[0.3,0.6]])

network['b2'] = np.array([0.1, 0.2])

network['W3'] = np.array([[0.1, 0.3], [0.2, 0.4]])

network['b3'] = np.array([0.1, 0.2])

return network

def forward(network,x):

W1,W2,W3=network['W1'],network['W2'],network['W3']

b1,b2,b3=network['b1'],network['b2'],network['b3']

a1=np.dot(x,W1)+b1

z1=sigmoid(a1)

a2 = np.dot(z1, W2) + b2

z2 = sigmoid(a2)

a3 = np.dot(z2, W3) + b3

y=identity(a3)

return y

network=init\_network()

x=np.array([1.0,0.5])

y=forward(network,x)

print(y)

定义softmax函数

def softmax(a):

exp\_a=np.exp(a)

sum\_exp\_a = np.sum(exp\_a)

y = exp\_a / sum\_exp\_a

return y

实例

a=np.array([0.3,2.9,4.0])

yh=softmax(a)

print(yh)

为了防止数过大溢出定义softmax函数

def softmax(a):

c=np.max(a)

exp\_a=np.exp(a-c)

sum\_exp\_a = np.sum(exp\_a)

y = exp\_a / sum\_exp\_a

return y