初始化函数——设定输入层节点、隐藏层节点和输出层节点的数量。

训练——学习给定训练集样本后，优化权重。

查询——给定输入，从输出节点给出答案。

query()函数接受神经网络的输入，返回网络的输出。

下式显示了输入层

和隐藏层之间的链接权重矩阵如何与输入矩阵相乘，给出隐藏层节点的输

入信号。

*X* hidden = *W* input\_hidden •*I*

以下代码应用了numpy代

码库，将链接权重矩阵*W* input\_hidden 点乘输入矩阵I。

hidden\_inputs = numpy.dot(self.wih, inputs)

hidden\_inputs = numpy.dot(self.wih, inputs

为了获得从隐藏层节点处出现的信号，我们简单地将S抑制函数应用

到每一个出现的信号上：

*O* hidden = sigmoid( *X* hidden )

下列代码定义了希望使用的激活

函数。

S函数

# activation function is the sigmoid function

self.activation\_function = **lambda**

x: scipy.special.expit(x)

无论何时任何人需要使

用激活函数，那么他所需做的就是调用self.activation\_function()。

我们要将激活函数应用到组合调整后，准备进入隐

藏层节点的信号。

# calculate the signals emerging from hidden layer

隐藏节点输出信号 hidden\_outputs = self.activation\_function(hidden\_inputs)

二、

计算误差

输出误差：

# error is the (target - actual) output\_errors = targets - final\_outputs

**errors** hidden = **weights** T

hidden\_output • **errors** output

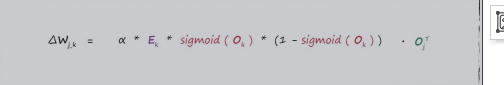
由于Python有能力使用numpy进行点乘，因此，这段代码再简单不过

了：

# hidden layer error is the output\_errors, split by weights,

recombined at hidden nodes

hidden\_errors = numpy.dot(self.who.T, output\_errors)



在Python代码中，这种转换很容易。我们首先为隐藏层和最终层之间

的权重进行编码。

# update the weights for the links between the hidden and output

layers

self.who += self.lr \* numpy.dot( ( **output\_errors \* final\_outputs \***

**(1.0 - final\_outputs)**

) , **numpy.transpose(hidden\_outputs)**

)

用于输入层和隐藏层之间权重的代码也是类似的。我们只是利用对称

性，重写代码，更换名字，这样它们指的就是神经网络的前一层了。下面

是这两个权重集的代码，代码着色了，这样你可以发现它们之间的异同

点：

# update the weights for the links between the hidden and output

layers

self. **who**

+= self.lr \* numpy.dot(( **output\_errors**

\* **final\_outputs \*(1.0 - final\_outputs)**

), **numpy.transpose(hidden\_outputs)**

)

# update the weights for the links between the input and hidden

layers

self. **wih**

+= self.lr \* numpy.dot(( **hidden\_errors**

\* **hidden\_outputs \***

**(1.0 - hidden\_outputs)**

), **numpy.transpose(inputs)**

第二

手写数据收集

测试数据

首先需要获得测试记录，这与用于获取训练数据的Python代码非常相

似。

# load the mnist test data CSV file into a list

test\_data\_file = open("mnist\_dataset/mnist\_test\_10.csv", 'r')

test\_data\_list = test\_data\_file.readlines()

test\_data\_file.close()

import numpy

import scipy.special

import matplotlib.pyplot

%matplotlib inline

class neuralNetwork :

def \_\_int\_\_(self,inputnodes,hiddennodes,outnodes,learningrate) :

self.inodes = inputnodes

self.hnodes = hiddennodes

self.onodes = outputnodes

self.wih = numpy.random.normal(0.0,pow(self.hnodes,-0.5),(self.hnodes,self.inodes))

self.who = numpy.random.normal(0.0,pow(self.onodes,-0.5),(self.ondes,self.hnodes))

self.1r = learningrate

self.activation\_function = lambda x: scipy.special.expit(x)

pass

def train(self,imputs\_list,targets\_list) :

inputs = numpy.array(inputs\_list,ndmin = 2).T

targets = numpy.array(targets\_list,ndmin = 2).T

hidden\_imputs = numpy.dot(self.wih, inputs)

hidden\_outputs = self.activation\_function(hidden\_inputs)

hidden\_inputs = numpy.dot(self.wih, inputs) #计算隐藏层中的信号

hidden\_outputs = self.activation\_function(hidden\_inputs) #计算隐藏层中出现的信号

final\_inputs = numpy.dot(self.who, hidden\_outputs) #将信号计算到最终输出层

final\_outputs = self.activation\_function(final\_inputs) #计算从最终输出层发出的信号

output\_errors = targets - final\_outputs #输出层错误是（目标-实际）

hidden\_errors = numpy.dot(self.who.T, output\_errors) #隐藏层错误是输出错误，按权重分割，在隐藏节点重新组合

#更新隐藏层和输出层之间链接的权重

self.who += self.lr \* numpy.dot((output\_errors \* final\_outputs \* (1.0 - final\_outputs)), numpy.transpose(hidden\_outputs))

#更新输入层和隐藏层之间链接的权重

self.wih += self.lr \* numpy.dot((hidden\_errors \* hidden\_outputs \* (1.0 - hidden\_outputs)), numpy.transpose(inputs))

pass

#查询神经网络

def query(self, inputs\_list):

#将输入列表转换为二维数组

inputs = numpy.array(inputs\_list, ndmin = 2).T

hidden\_inputs = numpy.dot(self.wih, inputs) #计算隐藏层中的信号

hidden\_outputs = self.activation\_function(hidden\_inputs) #计算隐藏层中出现的信号

final\_inputs = numpy.dot(self.who, hidden\_outputs) #将信号计算到最终输出层

final\_outputs = self.activation\_function(final\_inputs) #计算从最终输出层发出的信号

return final\_outputs

#输入、隐藏、输出节点数

input\_nodes = 784

hidden\_nodes = 200

output\_nodes = 10

#学习率

learning\_rate = 0.1

#创建神经网络实例

n = neuralNetwork(input\_nodes, hidden\_nodes, output\_nodes, learning\_rate)

#将文件导入列表

training\_data\_file = open("C:\Users\daxiguammm\Documents\Tencent Files\2277493107\FileRecv", 'r')

training\_data\_list = training\_data\_file.readlines()

training\_data\_file.close()

#epochs是训练数据集用于训练的次数

epochs = 5

for e in range(epochs):

#查看培训数据集中所有数据

for record in training\_data\_list:

all\_values = record.split(',') #用逗号拆分记录

inputs = (numpy.asfarray(all\_values[1:]) / 255.0 \* 0.99) + 0.01 #将输入颜色值范围进行缩放

targets = numpy.zeros(output\_nodes) + 0.01 #创建目标输出值

#所有值【0】都是此纪录的目标标签

targets[int(all\_values[0])] = 0.99

n.train(inputs, targets)

pass

pass

#将文件导入列表

test\_data\_file = open("E:\pzy的学习文档\python 神经网络\ML\_PG\test\_30", 'r')

test\_data\_list = test\_data\_file.readlines()

test\_data\_file.close()

#测试神经网络

#检查测试数据集中所有记录

for record in test\_data\_list:

all\_values = record.split(',')

correct\_label = int(all\_values[0]) #正确答案是第一个值

inputs = (numpy.asfarray(all\_values[1:]) / 255.0 \* 0.99) + 0.01 #缩放和移动输入

outputs = n.query(inputs) #查询网络

label = numpy.argmax(outputs) #最大值的索引对应标签

#追加正确或不正确的列表

if (label == correct\_label):

scorecard.append(1) #正确记分卡加1

else:

scorecard.append(0) #错误记分卡加0

pass

pass

#计算成绩分数，正确答案的分数

scorecard\_array = numpy.asarray(scorecard)

print ("performance = ", scorecard\_array.sum() / scorecard\_array.size)