**《深度学习理论与实践》实验报告**

**姓 名：** 徐云哲  **学 号：** 21200211047

**年 级：** 20级 **专 业：** 计算机软件与理论

1. **实验名称**

**Experiment 5 Back Propagation**

1. **实验要求**

1. Use the dataset for the experiment of k-NN classifier.

2. Write the Python code of a 4-layer MLP (including 2 hidden layers) to solve the classification problem, and compare the results of this MLP and your k-NN classifier.

1. **实验过程**

**1、环境介绍：**

使用Google Colab进行实验。

**2、定义神经网络的框架：**

考虑一个神经网络，很容易可以抽象出三种操作：

（1）初始化函数：指定神经网络的层数，每一层的节点个数等，即指定神经网络的结构；

（2）训练函数：通过训练数据集优化权重；

（3）查询函数：通过测试数据集测试训练后的神经网络。

为此，给出如下神经网络的类定义（神经网络的框架）：

# neural network definition

class NeuralNetwork:

    # initialise the neural network

    def \_\_init\_\_(self):

        pass

    # train the network using training data set

    def training(self):

        pass

    # query the network using test data set

    def query(self):

        pass

**3、初始化：**

根据分析，编写初始化函数\_\_init\_\_()，指定神经网络的结构。

    # initialise the neural network

    def \_\_init\_\_(self, numInputNodes, numHidden1Nodes, numHidden2Nodes, numOutputNodes, learningRate):

        # 单隐藏层示例，设置各层的节点个数

        self.numInputNodes = numInputNodes

        self.numHidden1Nodes = numHidden1Nodes

        self.numHidden2Nodes = numHidden2Nodes

        self.numOutputNodes = numOutputNodes

        # 权重更新时的学习率

        self.learningRate = learningRate

        pass

**4、创建网络节点和链接：**

**（1）正态分布初始权重**

对于**设置链接的初始权重有一个经验规则**：在一个节点传入链接数量平方根倒数的范围内随机采样，即从均值为0、标准方差等于节点传入链接数量平方根倒数的正态分布中进行采样。

# 正态分布初始化权重

        self.weightInputHidden = numpy.random.normal(0.0, pow(self.numHidden1Nodes, -0.5),

                                                     (self.numHidden1Nodes, self.numInputNodes))

        self.weightHidden1Output = numpy.random.normal(0.0, pow(self.numHidden2Nodes, -0.5),

                                                      (self.numHidden2Nodes, self.numHidden1Nodes))

        self.weightHidden2Output = numpy.random.normal(0.0, pow(self.numOutputNodes, -0.5),

                                                      (self.numOutputNodes, self.numHidden2Nodes))

**5、编写查询函数：**

查询函数query()用于从训练好的神经网络处获取输出集进行预测。

（1）网络使用sigmoid激活函数，在SciPy中定义为expit()；

（2）在\_\_init\_\_()中定义激活函数，这样可以方便地扩展激活函数或者改变激活函数；

（3）使用numpy进行矩阵运算

        # 激活函数(lambda创建匿名函数)

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

**6、编写训练函数：**

训练函数training()完成两件事情：

（1）第一阶段，同query()根据输入得到输出

（2）第二阶段，反向传播误差更新链接权重

第一阶段，同query()函数：

        # 第一，同query()函数

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

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

        hidden1\_inputs = numpy.dot(self.weightInputHidden, inputs)

        hidden1\_outputs = self.activation\_function(hidden1\_inputs)

        hidden2\_inputs = numpy.dot(self.weightHidden1Output, hidden1\_outputs)

        hidden2\_outputs = self.activation\_function(hidden2\_inputs)

        final\_inputs = numpy.dot(self.weightHidden2Output, hidden2\_outputs)

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

第二阶段，误差反向传播并更新权重。  
首先计算各层的误差：

        # 计算误差

        output\_errors = targets - final\_outputs

        # 反向传播误差到隐藏层2

        hidden2\_errors = numpy.dot(self.weightHidden2Output.T, output\_errors)

        # 反向传播误差到隐藏层1

        hidden1\_errors = numpy.dot(self.weightHidden1Output.T, hidden2\_errors)

接着使用梯度下降的方法来更新权重：

# 更新隐藏层2和输出层之间的权重

        self.weightHidden2Output += self.learningRate \* numpy.dot((output\_errors \* final\_outputs \*

                                                                  (1.0 - final\_outputs)),

                                                                 numpy.transpose(hidden2\_outputs))

        # 更新隐藏层2和隐藏层1之间的权重

        self.weightHidden1Output += self.learningRate \* numpy.dot((hidden2\_errors \* hidden2\_outputs \*

                                                                  (1.0 - hidden2\_outputs)),

                                                                 numpy.transpose(hidden1\_outputs))

        # 更新输入层和隐藏层1之间的权重

        self.weightInputHidden += self.learningRate \* numpy.dot((hidden1\_errors \* hidden1\_outputs \*

                                                                 (1.0 - hidden1\_outputs)),

                                                                numpy.transpose(inputs))

**7、数据集介绍**

使用识别手写数字数据集MNIST做实验

**数据集下载：**<https://pjreddie.com/projects/mnist-in-csv/>

**展示数据：**选择数据集中的一条记录，将这张手写数字图片绘制出来：

import matplotlib.pyplot as plt

# 直接使用plt.imshow无法显示图片，需要导入pylab包

import pylab

# 打开并读取文件

data\_file = open("mnist\_train.csv")

data\_list = data\_file.readlines()

data\_file.close()

# 拆分绘制28\*28图形

all\_pixels = data\_list[0].split(',')

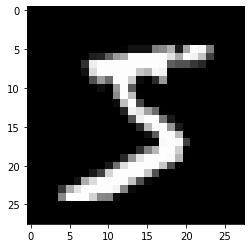
image\_array = numpy.asfarray(all\_pixels[1:]).reshape((28, 28))

plt.figure("Image")

plt.imshow(image\_array, cmap='gray', interpolation='None')

pylab.show()

我们选择了训练数据集的第一条记录，绘制结果如下：

****

**8、训练 & 测试**

import time

# 便于计算执行时间

start = time.process\_time()

# 指定神经网络的结构。隐藏层节点个数不唯一

input\_nodes, hidden1\_nodes, hidden2\_nodes, output\_nodes = 784, 256, 100, 10

# 指定权重更新的学习率

learning\_rate = 0.3

# 创建神经网络的实例

network = NeuralNetwork(input\_nodes, hidden1\_nodes, hidden2\_nodes, output\_nodes, learning\_rate)

# 读取训练数据，只读方式

training\_data\_file = open("mnist\_train.csv", 'r')

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

training\_data\_file.close()

# 训练神经网络，epochs次

epochs = 10

for e in range(epochs):

    for record in training\_data\_list:

        # 缩放输入

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

        scaled\_inputs = (numpy.asfarray(all\_pixels[1:]) / 255.0 \* 0.99) + 0.01

        # 创建目标输出

        targets = numpy.zeros(output\_nodes) + 0.01

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

        network.training(scaled\_inputs, targets)

        pass

    pass

# 读取测试数据集

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

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

test\_data\_file.close()

# 测试训练好的神经网络

# 初始化一个数据结构用于记录神经网络的表现

scorecard = []

# 遍历测试数据集

for record in test\_data\_list:

    # 打印预期输出

    all\_pixels = record.split(',')

    correct\_label = int(all\_pixels[0])

    # 查询神经网络

    inputs = (numpy.asfarray(all\_pixels[1:])/255.0 \* 0.99) + 0.01

    outputs = network.query(inputs)

    answer = numpy.argmax(outputs)

    # 更新神经网络的表现

    if answer == correct\_label:

        scorecard.append(1)

    else:

        scorecard.append(0)

        pass

    pass

# 打印得分及运行时间

print("time: ", time.process\_time()-start)

print("performance: ", sum(scorecard) / len(scorecard))

time: 1477.13756212

performance: 0.8012