**机器学习实验报告**

**（四）**

****

姓名：崔玉峰

学号：201600301079

班级：2016级4班

1. **实验目的**

通过BP神经网络对Mnist数据集进行分类，并用两种不同的BP模型做性能对比

1. **实验思路**

BP神经网络是使用反向传播算法进行训练的一种神经网络，在本次实验中建立的BP神经网络模型为单隐层激活函数为Sigmoid的神经网络，下面开始本实验模型的相关公式的推导：

1. **一个神经元的输出可以表示为：**



其中w为权值，x为输入，y为输出，()为sigmoid函数

**训练算法：**

1. **在本次实验中假设输入层的神经单元数为d，隐层神经单元数为q，输出层的神经单元数为l，那么对模型进行训练主要为了优化输入层到隐层d\*q个权值；隐层到输出层q\*l个权值。**

BP神经网络基于梯度下降策略：

1. 设当前神经网络的输出为，期望输出为

则此网络的均方误差为：



1. 根据梯度下降策略学习率为：



为隐层神经元输出

 为

因此 ***(1式)***

**其中**



因为sigmoid的导数为：

 ***(2式)***

1. 反向传播计算：

同理可知：输入层到隐层的权值梯度项为

 ***(3式)***

**其中**需要逆向学习算法：



因为为隐层的输出sigmoid的导数为：

 ***(4式)***

1. **通过上述的公式就可以写出训练算法的伪代码**

输入样本训练集Mnist= 学习率

过程：

随机初始网络中所有连接权的值

repeat

For all do:

求出当前神经网络输出。

通过***(2式)***求出输出层神经元梯度项

通过***(4式)***求出输入层神经元梯度项

通过***(1式)***和***(3式)***更新权值

End for

Until 均方误差趋近收敛或超出最大循环数

1. **参数准备：**

因为Mnist数据集是28\*28 =784的像素图，所以输入神经元应该设为784个，输出是0~9的数字所以设置10个输出神经元。对于隐层神经单元数需要以后调优确定。

权值初始化，对于所有的权值不能设置为0否则训练无法开始，对于并且对于每个权值初始值不能过大或过小，应该保证最后每个神经元的权值和在（-1，1），因此每个权值的初始值应该在范围内随机初始化:



1. **具体实现**
2. **实验环境：**
3. 编程语言：Python
4. 软件环境 ：Jupyter Notebook
5. 硬件环境 ： PC
6. **实验准备：**
7. 准备Mnist数据集，共有60000张训练训练集，和10000张测试数据集



1. 准备Numpy库：方便进行矩阵的运算操作。
2. **代码实现：**
3. **数据加载：**

将二进制文件的Mnist数据集读进来，并选择几个样例进行展示：

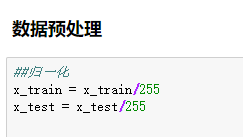




1. **数据预处理：**
2. 因为手写数字的图片x每个像素的取值范围在0~255 ，所以首先要对数据x进行归一化处理。此处有两种方案：

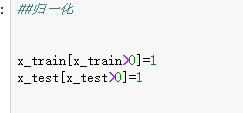
**第一种采用最值归一化：**

对于每个像素的取值除以255，得到每个像素的取值在0~1之间：



**第二种采用二值化：**

因为手写数据集不需要确定颜色，所以将像素>0点设置为1，无像素的点设置为0即可。



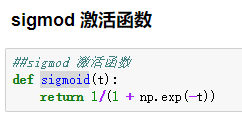
1. 其次要对y进行独热处理，：

比如5 化成 [0,0,0,0,0,1,0,0,0,0]

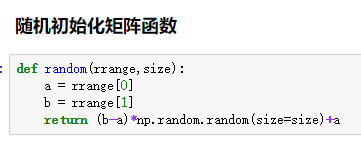
因为每一个每个神经元的输出都是0~1，并且y的取值是0~9，所以要创建10个输出神经元。

1. **相关函数的编写：**

Sigmoid激活函数编写



生成取值范围随机矩阵函数：



根据rrange范围，和size大小，随机初始化一个取值在rrange之间，大小为size的矩阵。用于初始化权值数组。

1. **变量初始化**

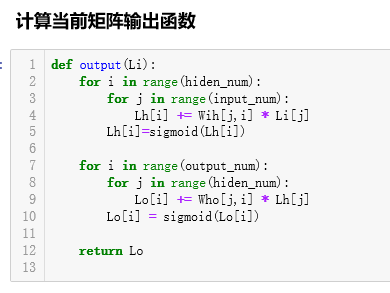
初始化输入神经元个数784，输出神经元个数10，隐身神经元个数以25个为例，在（-1/28，1/28）范围内随机初始化输入层到隐层的权值；在（-1/5，1/5）范围随机初始化隐层到输出层的权值,对于隐层，输出层神经元的阈值初始化。以及记录输入层输出，隐层输出，和输出层输出的变量。



1. **计算当前矩阵输出函数：**

**根据神经网络的输入，计算当前神经网络的输出的函数**

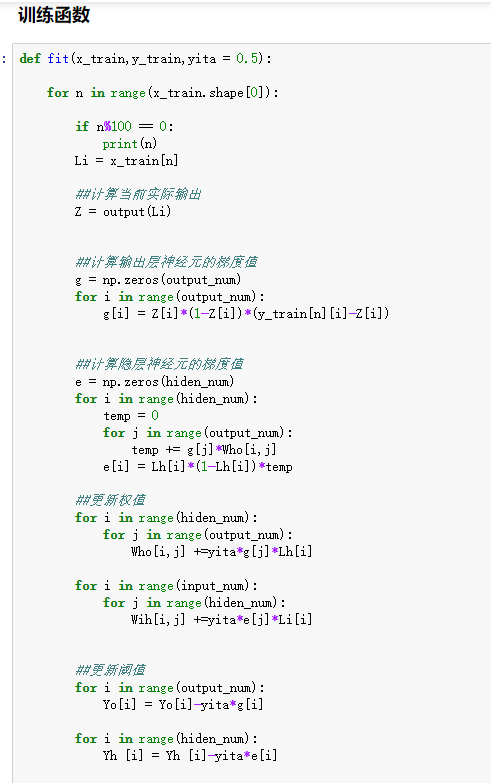




1. **训练函数：**

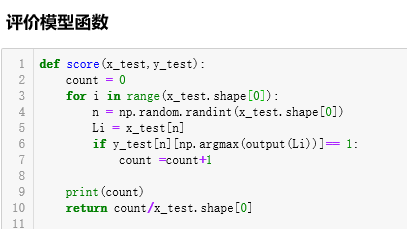
**按照之前推出的公式伪代码实现BP神经网络的训练算法：**

对于每个样本作为输入，计算当前神经网络输出，然后计算输出层神经网络梯度值，计算输入层网络梯度值，根据梯度值和学习率更新响应的权值大小。每次迭代一次全部的训练数据。



1. **评价模型函数：**

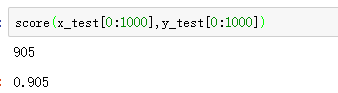
训练出的模型需要输入测试数据集进行评价准确度，根据输入的测试数据集计算神经网络的输出，输出的十个神经元中最大的作为当前神经网络预测的结果，与测试数据集的标签进行对比，输出模型在测试数据集上的准确率。

.

1. **训练模型并对模型进行评价：**

先测试一下代码编写是否正确，选择隐层神经元数25，学习率为0.5，不设置阈值，迭代一次全部训练样本（60000）。查看用1000个测试用例测试分类效果：





可以看出进行一次迭代后，正确率达到了90.5%，这个正确率还是非常令人满意的。下面会进行完整参数优化，以训练出最优模型

1. **参数调优：**

编写完代码就可以进行模型的调优过程了，这个过程是漫长且无聊的，因为训练样本集的样本数量有60000个，即使隐层只设置25个神经元，经过训练算法进行一次迭代(每次迭代训练所有的样本数据集)的时间也需要20分钟。如果设置更多的隐层神经元则训练时间是指数级的增长。所以在训练时会减少每次迭代的训练样本集的数量（2000个），尽可能减少训练时间。

根据本次代码的实际情况和本人能力，选择需要进行调优的参数有：

1. 隐层神经元数量
2. 是否设置阈值
3. 迭代次数

|  |  |  |
| --- | --- | --- |
| 隐层神经元数  是否设置阈值 | 否 | 是 |
| 25 | 0.784\0.83\0.848 | 0.772\0.821\0.85 |
| 49 | 0.802\0.839\0.857 | 0.793\0.828\0.86 |
| 100 | 0.800\0.845\0.851 | 0.805\0.849\0.853 |
| 400 | 0.466\0.815\0.842 |  |

每次设置同样的随机种子，相同的训练样本，测试样本。分别迭代一次，两次，三次的正确率：

根据上述的测试模型分析得出一下结论：

1. 越复杂的模型对于数据的需求越高，需要更多次的迭代和更多的训练样本才能训练出更好的模型。
2. 并不是模型越复杂训练的结果越好，隐层神经元100模型并没有优于隐层神经元为49的模型
3. 设置阈值的模型在多次训练之后往往会优于不设置阈值的模型。
4. 学习率可以影响模型的训练的速度

**因此根据测试结果，选择隐神经元数为49，设置阈值，并且学习率为0.5的三层BP神经网络对手写数据集进行训练。**

**fit(x\_train,y\_train)**

**score(x\_test,y\_test)**

**对60000个训练样本集迭代一次训练后，在10000个测试数据集上的准确度可以达到92.73% ; 第二次迭代准确度可以达到93.36% ;**

1. **实验总结：**

实验的代码，会以ipynb文件的形式上传可以随时进行运行和查看。

通过本次实验进行了BP神经网络的Python实现，并且对于Mnist手写数据集进行了识别。并通过参数优化选择了一个相对最优的模型进行训练和学习，最后达到了非常不错的识别准确率。

因为神经网络实在是太过于复杂，执行的时间太过于缓慢，调参的过程相当的漫长且无聊，只能降低训练样本的数量来更快的看到效果，所以综合自身能力和计算机运算能力的情况下下简单的进行了参数的优化，还是有很大的收获的。

BP神经网络的推导和理解还是有一定难度的，尤其是神经网络中参数变量众多，但是经过学习和理解，确定输入输出变量之后，编码的过程就很快了，真实实现了神经网络对于神经网络知识和理解有很大的帮助。

1. **实验代码：**

import numpy as np

import struct

import matplotlib.pyplot as plt

def load\_mnist(kind='train'):

labels\_path = kind+'-labels.idx1-ubyte'

images\_path =kind+'-images.idx3-ubyte'

with open(labels\_path, 'rb') as lbpath:

magic, n = struct.unpack('>II',lbpath.read(8))

labels = np.fromfile(lbpath,dtype=np.uint8)

with open(images\_path, 'rb') as imgpath:

magic, num, rows, cols = struct.unpack('>IIII',imgpath.read(16))

images = np.fromfile(imgpath,dtype=np.uint8).reshape(len(labels), 784)

return images, labels

x\_train,y\_train= load\_mnist()

x\_test,y\_test = load\_mnist('t10k')

fig, ax = plt.subplots(nrows=2,ncols=5,sharex=True,sharey=True, )

ax = ax.flatten()

for i in range(10):

img = x\_test[y\_test == i][0].reshape(28, 28)

ax[i].imshow(img, cmap='Greys', interpolation='nearest')

ax[0].set\_xticks([])

ax[0].set\_yticks([])

plt.tight\_layout()

plt.show()

##归一化

x\_train[x\_train>0]=1

x\_test[x\_test>0]=1

##独热处理

Y = []

for i in y\_test:

y = np.zeros(10)

y[i] = 1

Y.append(y)

y\_test = np.array(Y)

Y = []

for i in y\_train:

y = np.zeros(10)

y[i] = 1

Y.append(y)

y\_train = np.array(Y)

x\_train.shape

##sigmod 激活函数

def sigmoid(t):

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

def random(rrange,size):

a = rrange[0]

b = rrange[1]

np.random.seed(666)

return (b-a)\*np.random.random(size=size)+a

input\_num = 784 ##输入神经元数

hiden\_num = 49 ##隐神经元数

output\_num = 10 ##输出神经元数

##输入层到隐层的权值 input\_num\*hiden\_num

Wih = random((-1/28,1/28),(input\_num,hiden\_num))

##隐层到输出层的权值 hiden\_num\*output\_num

Who =random((-1/7,1/7),(hiden\_num,output\_num))

##隐层初始阈值

Yh = np.random.random(size=(hiden\_num,1))

##输出层初始阈值

Yo = np.random.random(size =(output\_num,1))

##输入层

Li = np.zeros((input\_num,1))

##隐层

Lh = np.zeros((hiden\_num,1))

##输出层

Lo = np.zeros((output\_num,1))

def output(Li):

for i in range(hiden\_num):

for j in range(input\_num):

Lh[i] += Wih[j,i] \* Li[j]

Lh[i]=sigmoid(Lh[i]-Yh[i])

for i in range(output\_num):

for j in range(hiden\_num):

Lo[i] += Who[j,i] \* Lh[j]

Lo[i] = sigmoid(Lo[i]-Yo[i])

return Lo

def fit(x\_train,y\_train,yita = 0.5):

for n in range(x\_train.shape[0]):

if n%100 == 0:

print(n)

Li = x\_train[n]

##计算当前实际输出

Z = output(Li)

##计算输出层神经元的梯度值

g = np.zeros(output\_num)

for i in range(output\_num):

g[i] = Z[i]\*(1-Z[i])\*(y\_train[n][i]-Z[i])

##计算隐层神经元的梯度值

e = np.zeros(hiden\_num)

for i in range(hiden\_num):

temp = 0

for j in range(output\_num):

temp += g[j]\*Who[i,j]

e[i] = Lh[i]\*(1-Lh[i])\*temp

##更新权值

for i in range(hiden\_num):

for j in range(output\_num):

Who[i,j] +=yita\*g[j]\*Lh[i]

for i in range(input\_num):

for j in range(hiden\_num):

Wih[i,j] +=yita\*e[j]\*Li[i]

##更新阈值

for i in range(output\_num):

Yo[i] = Yo[i]-yita\*g[i]

for i in range(hiden\_num):

Yh [i] = Yh [i]-yita\*e[i]

def score(x\_test,y\_test):

count = 0

for i in range(x\_test.shape[0]):

Li = x\_test[i]

if y\_test[i][np.argmax(output(Li))]== 1:

count =count+1

print(count)

return count/x\_test.shape[0]

fit(x\_train,y\_train)

score(x\_test,y\_test)

1. **参考文献**

*[美]RichardO.Duda PeterE.Hart DavidG.Stork 著 模式分类 第二版*

*[中]周志华著 机器学习*