华东师范大学计算机科学技术系上机实践报告

课程名称：人工智能	**年级**：2016 级	**上机实践成绩**：
指导教师：周爱民	**姓名**：汪春雨	**创新实践成绩**：
上机实践名称：数据分类	**学号**：10152150127	**上机实践日期**：2018/5/29
上机实践编号：No.6	**组号**：	**上机实践时间**：

一、 问题介绍

（本节介绍需要求解的问题是什么，为什么要采用我们介绍的方法求解）

1．1问题描述

MNIST 数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology (NIST)。

训练集(training set)由来自250 个不同人手写的数字构成，其中50%

是高中学生，50%来自人口普查局(the Census Bureau)的工作人员。测试集(test set)也是同样比例的手写数字数据。

MNIST数据集可在 http://yann.lecun.com/exdb/mnist/ 获取，它包含了四个部分：

• Training set images: train-images-idx3-ubyte.gz (9.9 MB，解压后47 MB，包含60,000个样本)

• Training set labels: train-labels-idx1-ubyte.gz (29 KB，解压后60 KB，包含60,000个标签)

• Test set images: t10k-images-idx3-ubyte.gz (1.6 MB，解压后7.8 MB，包含10,000个样本)

• Test set labels: t10k-labels-idx1-ubyte.gz (5KB，解压后10 KB，包含10,000个标签)

任务：通过训练bp神经网络，预测手写字符。

1.2、求解方法介绍

1.2.1 bp神经网络简介

BP（Back Propagation）神经网络分为两个过程

（1）工作信号正向传递子过程

（2）误差信号反向传递子过程

在BP神经网络中，单个样本有个输入，有个输出，在输入层和输出层之间通常还有若干个隐含层。

实际上，1989年Robert Hecht-Nielsen证明了对于任何闭区间内的一个连续函数都可以用一个隐

含层的BP网络来逼近，这就是万能逼近定理。所以一个三层的BP网络就可以完成任意的维到维的映射。即这三层分别是输入层（I），隐含层（H），输出层（O）。如下图示

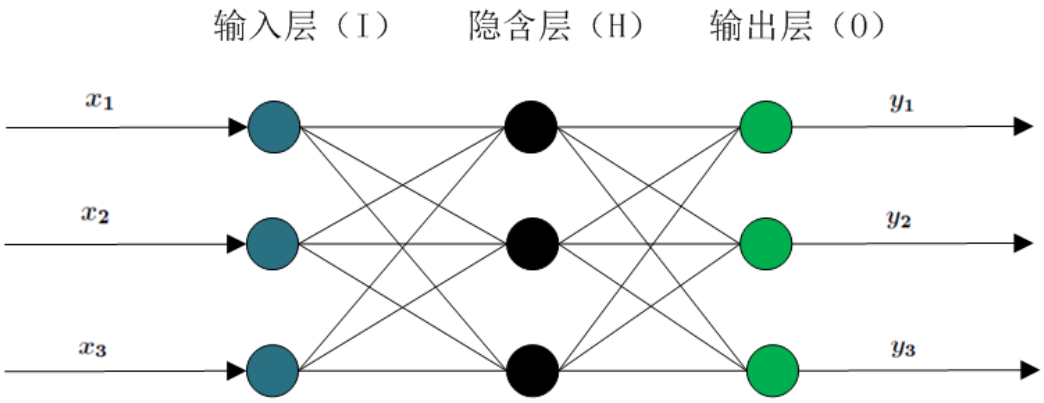

图1 bp神经网络示意图

1.2.2 隐含层的选取

在 BP 神经网络中，输入层和输出层的节点个数都是确定的，而隐含层节点个数不确定，那么应该设置为多少才合适呢？实际上，隐含层节点个数的多少对神经网络的性能是有影响的，有一个经验公式可以确定隐含层节点数目，如下

$$h = \sqrt{m+n} + a$$

其中 h 为隐含层节点数目，m 为输入层节点数目，n 为输出层节点数目，a 为 $1 \sim 10$ 之间的调节常数。H 的选取我们在实验中将具体论述。

1.2.3 正向传递子过程

现在设节点 i 和节点 j 之间的权值为 w_{ij}，节点 j 的阀值为 b_j，每个节点的输出值为 x_j，而每个节点的输出

值是根据上层所有节点的输出值、当前节点与上一层所有节点的权值和当前节点的阀值还有激活函数来实现的。具体计算方法如下

$$S_j = \sum_{i=0}^{m-1} w_{ij} x_i + b_j$$

$$x_j = f(S_j)$$

其中 f 为激活函数，一般选取 S 型函数或者线性函数。

正向传递的过程比较简单，按照上述公式计算即可。在 BP 神经网络中，输入层节点没有阀值。

1.2.4 反向传递子过程

在 BP 神经网络中，误差信号反向传递子过程比较复杂，它是基于 Widrow-Hoff 学习规则的。假设输出层的所有结果为d_j，误差函数如下

$$E(w,b)=\frac{1}{2}\sum_{j=0}^{n-1}(d_j-y_j)^2$$

而 BP 神经网络的主要目的是反复修正权值和阀值，使得误差函数值达到最小。Widrow-Hoff 学习规则是通过沿着相对误差平方和的最速下降方向，连续调整网络的权值和阀值，根据梯度下降法，权值矢量的修正正比于当前位置上 E(w,b)的梯度，对于第j个输出节点有

$$\triangle w(i,j)=-\eta\frac{\partial E(w,b)}{\partial w(i,j)}$$

假设选择激活函数为

$$f(x)=\frac{A}{1+e^{-\frac{x}{B}}}$$

对激活函数求导，得到

$$\begin{aligned}f'(x)&=\frac{Ae^{-\frac{x}{B}}}{B\left(1+e^{-\frac{x}{B}}\right)^2}\\&=\frac{1}{AB}\cdot\frac{A}{1+e^{-\frac{x}{B}}}\cdot\left(A-\frac{A}{1+e^{-\frac{x}{B}}}\right)\\&=\frac{f(x)\left[A-f(x)\right]}{AB}\end{aligned}$$

那么接下来针对w_{ij}有

$$\begin{aligned}\frac{\partial E(w,b)}{\partial w_{ij}}&=\frac{1}{\partial w_{ij}}\cdot\frac{1}{2}\sum_{j=0}^{n-1}(d_j-y_j)^2\\&=(d_j-y_j)\cdot\frac{\partial d_j}{\partial w_{ij}}\\&=(d_j-y_j)\cdot f'(S_j)\cdot\frac{\partial S_j}{\partial w_{ij}}\\&=(d_j-y_j)\cdot\frac{f(S_j)\left[A-f(S_j)\right]}{AB}\cdot\frac{\partial S_j}{\partial w_{ij}}\\&=(d_j-y_j)\cdot\frac{f(S_j)\left[A-f(S_j)\right]}{AB}\cdot x_i\\&=\delta_{ij}\cdot x_i\end{aligned}$$

其中有

$$\delta_{ij}=(d_j-y_j)\cdot\frac{f(S_j)\left[A-f(S_j)\right]}{AB}$$

同样对于b_j有

$$\frac{\partial E(w,b)}{\partial b_j}=\delta_{ij}$$

这就是著名的δ**学习规则**，通过改变神经元之间的连接权值来减少系统实际输出和期望输出的误差，这个规则又叫做 **Widrow-Hoff** 学习规则或者**纠错学习规则**。

上面是对隐含层和输出层之间的权值和输出层的阀值计算调整量，而针对输入层和隐含层和隐含层的阀值调整量的计算更为复杂。假设w_{ki}是输入层第 **k** 个节点和隐含层第 **i** 个节点之间的权值，那么有

$$\begin{aligned}\frac{\partial E(w,b)}{\partial w_{ki}}&=\frac{1}{\partial w_{ki}}\cdot\frac{1}{2}\sum_{j=0}^{n-1}(d_j-y_j)^2\\&=\sum_{j=0}^{n-1}(d_j-y_j)\cdot f'(S_j)\cdot\frac{\partial S_j}{\partial w_{ki}}\\&=\sum_{j=0}^{n-1}(d_j-y_j)\cdot f'(S_j)\cdot\frac{\partial S_j}{\partial x_i}\cdot\frac{\partial x_i}{\partial S_i}\cdot\frac{\partial S_i}{\partial w_{ki}}\\&=\sum_{j=0}^{n-1}\delta_{ij}\cdot w_{ij}\cdot\frac{f(S_i)\left[A-f(S_i)\right]}{AB}\cdot x_k\\&=x_k\cdot\sum_{j=0}^{n-1}\delta_{ij}\cdot w_{ij}\cdot\frac{f(S_i)\left[A-f(S_i)\right]}{AB}\\&=\delta_{ki}\cdot x_k\end{aligned}$$

其中有

$$\delta_{ki}=\sum_{j=0}^{n-1}\delta_{ij}\cdot w_{ij}\cdot\frac{f(S_i)\left[A-f(S_i)\right]}{AB}$$

有了上述公式，根据**梯度下降法**，那么对于隐含层和输出层之间的权值和阀值调整如下

$$w_{ij}=w_{ij}-\eta_1\cdot\frac{\partial E(w,b)}{\partial w_{ij}}=w_{ij}-\eta_1\cdot\delta_{ij}\cdot x_i$$

$$b_j=b_j-\eta_2\cdot\frac{\partial E(w,b)}{\partial b_j}=b_j-\eta_2\cdot\delta_{ij}$$

而对于输入层和隐含层之间的权值和阀值调整同样有

$$w_{ki}=w_{ki}-\eta_1\cdot\frac{\partial E(w,b)}{\partial w_{ki}}=w_{ki}-\eta_1\cdot\delta_{ki}\cdot x_k$$

$$b_i=b_i-\eta_2\cdot\frac{\partial E(w,b)}{\partial b_i}=b_i-\eta_2\cdot\delta_{ki}$$

二、 程序设计与算法分析

（本节介绍程序设计的分析过程，如数据结构定义，算法描述，算法框图等）

2.1 MNIST数据读取

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

以上是MNIST数据集文件存储方式的介绍，解析如下：

训练集是有60000个用例的，也就是说这个文件里面包含了60000个标签内容，每一个标签的值为0到9之间的一个数；回到我们的训练标签集上，按上面说的，我们先解析每一个属性的含义，**offset**代表了字节偏移量，也就是这个属性的二进制值的偏移是多少；**type**代表了这个属性的值的类型；**value**代表了这个属性的值是多少；**description**是对这个的说明；所以呢，这里对上面的进行一下说明，它的说法是“从第0个字节开始有一个32位的整数，它的值是**0x00000801**，它是一个魔数；从第4个字节开始有一个32位的整数，它的值是60000，它代表了数据集的数量；从第8个字节开始有一个unsigned byte，它的值是??，是一个标签值。

在MNIST图片集中，所有的图片都是28×28的，也就是每个图片都有28×28个像素；看回我们的上述图片，其表示，我们的**train-images-idx3-ubyte**文件中偏移量为0字节处有一个4字节的数为0000 0803表示魔数；接下来是0000 ea60值为60000代表容量，接下来从第8个字节开始有一个4字节数，值为28也就是0000 001c，表示每个图片的行数；从第12个字节开始有一个4字节数，值也为28,也就是0000 001c

表示每个图片的列数；从第16个字节开始才是我们的像素值。而且每784个字节代表一幅图片

我们可以打开文件的二进制内容进行验证查看

在图示中我们可以看到有一个MSB first，其全称是”**Most Significant Bit first**”,相对称的是一个LSB first，“**Least Significant Bit**”; MSB first是指最高有效位优先，也就是我们的大端存储，而LSB对应小端存储。

我们通过python库struct逐字节读取训练集、测试集图片文件，返回每个元素都是（28*28）的列表

同样的读取类标，返回每个元素为0-9数字的列表

读取结果展示如下：

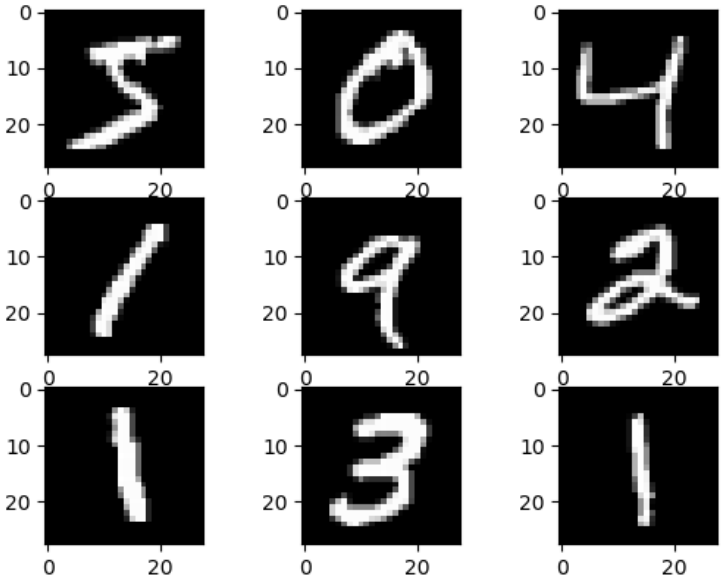

图2 训练集可视化

2.2 BP神经网络主程序算法

输入：图片数据集samples，类标labels，隐藏层节点数hiden_num(h)

1） 初始化神经网络配置：输入层节点数input_num(i) = 28*28

输出层节点数output_num(o) = 10

输入层权重矩阵$\mathbf{W1_{ixh}}$初始化

隐藏层层权重矩阵$\mathbf{W2_{hxo}}$初始化

隐藏层偏置向量**hiden_offset**置零

输出层偏置向量**output_offset**置零

输入层权值学习率input_lrate=0.3

隐藏层权值学习率hiden_lrate=0.3

2） for sample,lable in samples,lables:

a)输出真实值：real_out[lable]=1

b)前向过程：

b1)隐藏层激活值：**hiden_act** = sigmoid(np.dot(**samples[index],w1**) + **hiden_offset**)

b2)输出层激活值：**out_act** = sigmoid(np.dot(**hiden_act, w2**) + **output_offset**)

c)后向过程：

c1)**e** = **real_out** – **out_act**

out_delta = **e*out_act*(1-out_act)**

hiden_delta = **hiden_act*(1-hiden_act)***np.dot**(w2, out_delta)**

c2) 更新隐藏层到输出层权值

w2+= **hiden_lrate***out_delta***hiden_act**

c3) 更新输入层到隐藏层权值

w1+= **input_lrate***hiden_delta***sample**

3） 保存模型

2.3预测算法

1)加载模型…

2) for index,**sample** in test:

a)**hiden_value** = np.dot(**sample**, **w1**) + **hiden_offset**

b)**hiden_act** = sigmoid(**hiden_value**)

c)**out_value** = np.dot(**hiden_act**, **w2**) + **output_offset**

d)**out_act** = sigmoid(**out_value**)

e)**result_labels**[index] = np.argmax(**out_act**)

3）输出result_labels[]

三、 实验结果

（本节列出实验的结果，必要时加入一些自己的分析）

3.1隐藏层节点数选择

前文讨论到，隐藏层节点数的选取规则：

$$h = \sqrt{m+n} + a$$

约为28。

接下来测试了节点数对于正确率的影响，由于设备性能受限，最高为100

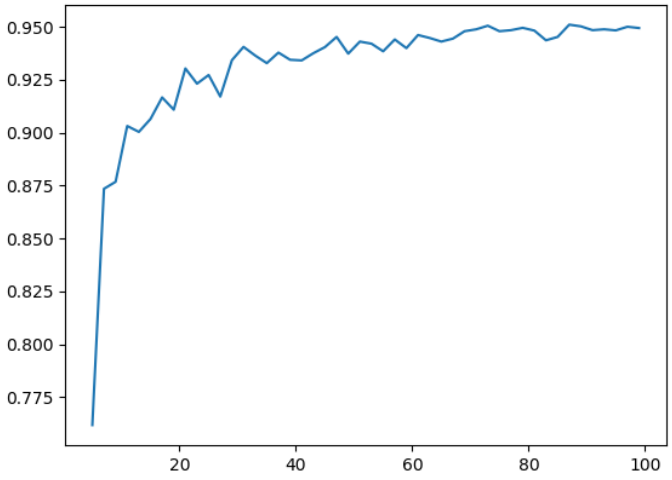

图4 节点数测试

其中横坐标为隐藏层节点数，纵坐标为正确率。

可以从中发现，节点数h与正确率acc之间呈自然对数关系，但是在节点数继续增多时，对性能提升不明显，最终我选择了一个可接受的节点数：100

3.2预测结果性能评估

用100个隐藏层节点对测试集进行预测得到的结果为：

	precision	recall	f1-score	support
0	0.95	0.99	0.97	980
1	0.96	0.99	0.98	1135
2	0.97	0.93	0.95	1032
3	0.93	0.96	0.95	1010
4	0.95	0.95	0.95	982
5	0.96	0.93	0.94	892
6	0.96	0.97	0.96	958
7	0.98	0.91	0.94	1028
8	0.95	0.93	0.94	974
9	0.90	0.96	0.93	1009
avg / total	0.95	0.95	0.95	10000

以上分别列出了对于每个数字预测的准确率，召回率，f1 score，以及出现次数。

最后一行为总计，**accuracy=95%**

我们可以发现，对于数字2,5,6,7数字的预测准确率较高，对于数字9预测准确率较低。

原因为手写体的数字9很容易被识别为数字1或7，而2,5,6,7等数字相对来说不容易被识别错。

四、 附件

（本节非必须的，可以列出源代码等，但是要把格式组织好）

Pycharm工程文件

- ·data_handle.py 加载数据与数据预处理
- ·BPNN.py 神经网络训练与预测主程序
- ·BPNetwork.model 保存的模型(json)
- ·BPNetwork.midel.report 预测数据结果评估
- ·data 存放数据文件夹
  - ·data_gz 数据压缩包
  - ·extract_gz 解压后mnist数据集