实验报告

－－－－BP 神经网络

姓名：彭靖寒　　学号：16340178　　日期：2018/12/25

摘要：（简要介绍要解决的问题，所使用的方法步骤，取得的结果或结论。）

本次实验通过构造一个三层的 BP 神经网络，完成对手写 0-9 数字的识别，通过采用 MNIST 手写数字的训练集进行模型训练，并用 MNSIT 的手写数字的测试集作为测试数据进行识别率检测，最终测试集识别率达到 97%以上。最后设计了一个简单的画图工具程序，通过鼠标进行数字书写，并对书写的单个数字进行图像识别，并输出识别结果进行对比，数字的书写和识别均可以重复进行。

1．导言

（要解决的问题描述，问题背景介绍；拟使用的方法，方法的背景介绍；）

本次实验的目标和要求是构造一个三层的 BP 神经网络，完成手写 0-9 数字的识别。

如上述实验要求所言，本次实验拟构造一个三层的 BP 神经网络（包括一层输入层、一层隐含层以及一层输出层），通过 MNIST 手写数字数据集中的训练集的训练进行错误反响传播进行学习，最终得到训练模型，最后使用测试集进行验证最终识别率。

BP（Backpropagation 的缩写）是“误差反向传播”的简称，是一种与最优化方法（如梯度下降法）结合使用的，用来训练人工神经网络的常见方法。该方法对网络中所有权重计算损失函数的梯度。这个梯度会反馈给最优化方法，用来更新权值以最小化损失函数。

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。

2．实验过程

（所用的具体的算法思想流程；

实现算法的程序主要流程，功能说明；）

2.1 实验设计

实验的流程步骤如下：

1.设计网络的结构：神经网络层数为三层，第一层输入层的神经元个数 784，第二层隐含层的神经元个数为 50，第三层输出层的神经元个数为 10；

2.根据数字识别的任务，设计网络的输入和输出：输入为元素个数为 784 个的一维数组，输出为元素个数为 10 个的一维数组；

3.实现 BP 网络的错误反传算法，完成神经网络的训练和测试，最终识别率达到 97%左右；

4.设计鼠标画板输入程序，进行鼠标手写数字的实时识别。

实验流程的前三个部分是比较常规的 BP 神经网络的实现，第四步是在 BP 神经网络构建完成的基础上进行的一个实验小拓展，使构造的 BP 神经网络能够实际应用。

三层 BP 神经网络包含了一层输入层、一层隐含层以及一层输出层，其中输入层有 784 个神经元，输出层有 10 个神经元，隐含层最终确定为 50 个神经元，整个网络进行全连接，隐含层激活函数为 ReLU 函数，输出层激活函数为 Softmax 函数，训练过程中损失函数为交叉熵函数（Cross Entropy Function）。具体三层 BP 神经网络结构参看下图：

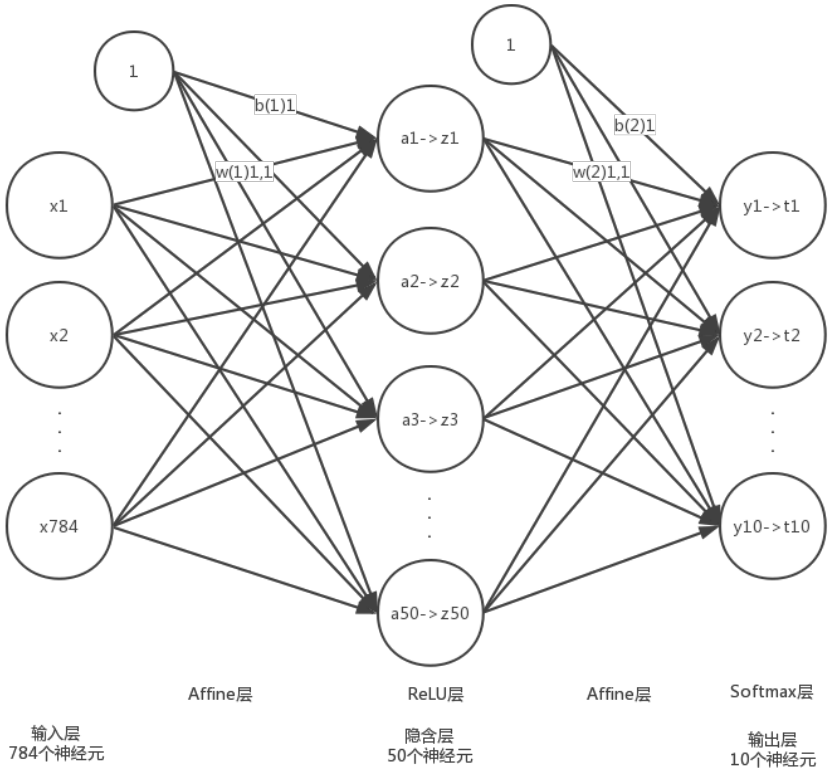


图 1 BP 神经网络结构

上图中，x1-x784 表示输入层 784 个输入数据，a1-a50 表示由输入值 x1-x784 和偏置 b 经过 Affine 计算后的隐含层神经元的值，z1-z50 表示由 a1-a50 经过 ReLU 激活函数计算后的值，y1-y10 表示由隐含层输出值 z1-z50 和偏置 b 经过 Affine 计算后得到的值，t1-t10 表示由 y1-y10 经过 Softmax 计算后的值。

b(1)1 表示对应于隐含层第一个神经元的偏置值，b(2)1 表示对应于输出层的第一个神经元的偏置值；w(1)1,1 表示输入层的第一个神经元对应于隐含层的的第一个神经元的权重值，w(2)1,1 表示隐含层的第一个神经元对应于输出层的第一个神经元的权重值。

第一个 Affine 层表示仿射变换，目的是实现矩阵的乘积与偏置和的运算：

$$A = X \cdot W1 + B1 \qquad (1)$$

其中 X 为输入，为一维数组(x1, x2, … , x784)，W 表示权重，为二维数组((w(1)1,1, w(1)2,1, … , w(1)784,1), …, (w(1)1,50, w(1)2,50, … , w(1)784,50))，B 表示偏置，为一维数组(b(1)1, b(1)2, … , b(1)50)，A 表示隐含层值，为一维数组(a1, a2, … , a50)，计算后讲 A 作为输出传递到隐含层。

以上为 Affine 层正向传播时的计算流程，而进行误差反向传播时，基于下述公式：

$$\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \cdot \boldsymbol{W}^{\mathrm{T}}$$
$$\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \boldsymbol{Y}} \qquad (2)$$

其中∂L/∂Y 为下一层 ReLU 层反向传播过来的值，计算权重 W 的梯度值 dW、偏执 B 的梯度值 dB，以及 X 的梯度值 dX。其中 W 的梯度值 dW = X 的转置点乘正向传播时的输出 A，dB = 正向传播时的输出 A 在列方向上的和组成的一维数组，而 dX = 正向传播时的输出值 A 点乘权重 W 的转置。

ReLU 层，基于激活函数 ReLU（Rectified Linear Unit）：

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leqslant 0) \end{cases} \qquad (3)$$

对于以一维数组 A=(a1, a2, … , a50)作为输入，及一维数组 Z=(z1, z2, … , z50)为输出: 若 ai > 0，则 zi = ai，否则 zi = 0，其中 i 的取值范围为 1-50。

以上为正向传播时的计算过程，当 ReLu 层进行误差反向传播时，基于以下公式 y 关于 x 的导数：

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leqslant 0) \end{cases} \qquad (4)$$

当正向传播时的值大于 0 时，则反向传播的值为 1，否则反向传播的值为 0。最后将反向传播的一维数组(以 0 或 1 作为元素)输出到第一层 Affine 层。

第二个 Affine 层表示仿射变换，目的是实现矩阵的乘积与偏置和的运算：

$$Y = Z \cdot W2 + B2 \qquad (5)$$

其中 Z 表示隐含层输出，为一维数组(z1, z2, … , z50)，W 表示权重，为二维数组((w(2)1,1, w(2)2,1, … , w(2)784,1), …, (w(2)1,50, w(2)2,50, … , w(2)784,50))，B 表示偏置，为一维数组(b(1)1, b(1)2, … , b(1)50)，Y 表示输出层值，为一维数组(y 1, y2, … , y10)。

以上为 Affine 层正向传播时的计算流程，而进行误差反向传播时，可参考上述第一层 Affine 层的误差反向传播。计算得到了第二层 Affine 层的权重的梯度值和偏执的梯度值，以及 X 的梯度值 dX，并将 dX 作为输出传递到 ReLU 层。

Softmax 层，基于激活函数 Softmax 函数：

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} \tag{6}$$

对于以一维数组 Y=(y1, y2, … , y10)为输入，及以一维数组 T(t1, t2, … , t10)为输出，使用上述 Softmax 函数进行计算。

以上为 Softmax 层的正向传播过程，当进行误差反向传播时，需要将计算损失值作为输出传递到第二层 Affine 层。

损失值的计算基于交叉熵函数，其计算公式如下：

$$E = -\sum_{k} t_k \log y_k \tag{7}$$

综上所述，在神经网络训练过程中，正向传播过程中的计算流程为，首先将输入值 X 进行第一层 Affine 层计算，然后将输出值传递到 ReLU 层计算，然后将输出值传递到第二层 Affine 层计算，然后将输出值传递到 Softmax 层计算，然后通过交叉熵函数计算损失值。反向传播时，将损失值传递到第二层 Affine 层进行反向计算，用于计算第二层中的权重的梯度值、偏执的梯度值和输入的梯度值，并将输入的梯度值作为输出传递到 ReLU 层进行反向计算，ReLU 层反向计算后到值传递到第一层 Affine 层进行反向计算，计算后得到权重的梯度值和偏置的梯度值。

首先初始化随机给两个 Affine 层的权重和偏执进行赋值，然后在训练过程中，选定训练样本作为输入，通过正向传播和反向传播计算后，得到了两个 Affine 层的权重和偏置的梯度值，然后对于设定的学习率，使权重和偏置减去梯度值与学习率的乘积，将减去后得到的新权重值和偏执值更新，即为一次训练学习过程。然后设定迭代次数，进行多次训练，使得最后，梯度值非常小，整个模型趋于稳定，则训练完成。

在神经网络测试过程，不需要进行误差反向传播计算，仅仅在正向传播计算后，计算得到的包含 10 个元素的一维数组中，值最大的元素对应的下标，即为最终的识别结果。

2.2 实验数据

本次实验的训练集和测试集数据为 MNIST 手写数字数据集，MNIST 数据来源为：http://yann.lecun.com/exdb/mnist/，它包含了 4 个部分：

1. 训练集图片（Training set images）：train-images-idx3-ubyte.gz（9.9 MB，解压后 47MB，包含 60,000 个样本）

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

3. 测试集图片（Test set images）：t10k-images-idx3-ubyte.gz（1.6 MB，解压后 7.8 MB，包含 10,000 个样本）

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

每个样本为 28*28 像素点的单通道灰度图，每个样本对应的标签为数字 0-9。具体 MNIST 样本图片参看下图 1：

图 2 选自 MNIST 测试集的部分样本，从左到右，从上到下依次为 0-9

2.3 程序流程

整个项目的文件结构如下：

```
.
├─── dataset                          //数据集及模型
│    ├─── mnist.pkl
│    ├─── t10k-images-idx3-ubyte.gz
│    ├─── t10k-labels-idx1-ubyte.gz
│    ├─── train-images-idx3-ubyte.gz
│    ├─── train-labels-idx1-ubyte.gz
│    └─── weights.pkl
├─── output                           //测试集的识别输出
│    ├─── negative
│    └─── positive
├─── src                              //源文件
│    ├─── BPNetwork.py
│    ├─── draw_board.py
│    ├─── layers.py
│    ├─── load_mnist.py
│    ├─── mnist_output.py
│    ├─── predict_number.py
│    ├─── test_mnist.py
│    └─── train_mnist.py
└─── testdata                         //画版鼠标手写数字输入输出数据
     ├─── dst.png
     └─── src.png
```

源代码的结构如下图所示：

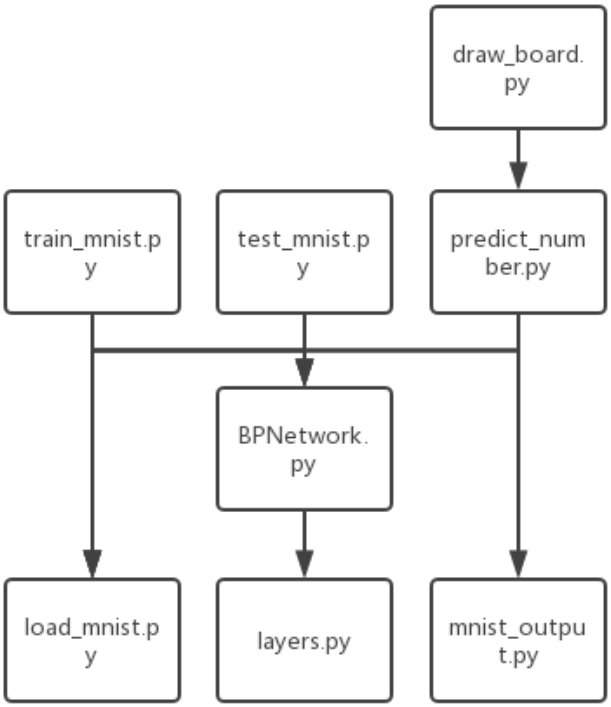


图 3 源代码程序文件调用结构

由上图，load_mnist.py 处于最底层，其功能为读取目录 dataset 下的 4 个 MNIST 数据集，并将其转化为 numpy array 类型，最后存入 dataset 目录下的 mnist.pkl 文件中，并提供一个 load_mnist 函数给上层调用，每次加载数据只需读取 mnist.pkl 文件即可，以供上层文件加载 MNIST 的数据。load_mnist 方法有三个参数，分别为 normalize，flatten，one_hot_label，其中 normalize=True 时表示需要将图像像素灰度值正规化为 0.0-1.0，否则不需要正规化，缺省值为 True，即灰度值取值为 0-255；flatten 表示是否需要将图片展开为一维数组，若 flatten=True，则输出的图片为一个一维数组，否则输出一个三维数组，缺省值为 True；on_hot_label 表示是否将标签 1 输出为[0,1,0,0,0,0,0,0,0,0]这样的数组，因为后者便于训练时进行损失值的计算，而前者便于比较测试时的结果的比较。

```python
def load_mnist(normalize=True, flatten=True, one_hot_label=False):
    """读入MNIST数据集

    Parameters
    ----------
    normalize : 将图像的像素值正规化为0.0~1.0
    one_hot_label :
        one_hot_label为True的情况下，标签作为one-hot数组返回
        one-hot数组是指[0,0,1,0,0,0,0,0,0,0]这样的数组
    flatten : 是否将图像展开为一维数组

    Returns
    -------
    (训练图像, 训练标签), (测试图像, 测试标签)
    """

    with open(save_file, 'rb') as f:
        dataset = pickle.load(f)

    if normalize: # 将图像的像素值正规化为0.0~1.0
        for key in ('train_img', 'test_img'):
            dataset[key] = dataset[key].astype(np.float32)
            dataset[key] /= 255.0

    if one_hot_label: # 标签作为one-hot(eg:[0,0,1,0,0,0,0,0,0,0])数组返回
        dataset['train_label'] = _change_one_hot_label(dataset['train_label'])
        dataset['test_label'] = _change_one_hot_label(dataset['test_label'])

    if not flatten: # 将图像恢复为三维数组，默认输出为一维数组
         for key in ('train_img', 'test_img'):
            dataset[key] = dataset[key].reshape(-1, 1, 28, 28)

    return (dataset['train_img'], dataset['train_label']), (dataset['test_img'], dataset['test_label'])
```

layers.py 同样位于底层，其实现了 softmax 函数，cross_entorpy_error 函数，ReLU 类，Affine 类以及 SoftmaxWithLoss 类（包括正向及反向传播），并提供了这三个类给上层即 BPNetwork.py 进行类的实例化调用。

Softmax 函数和交叉熵函数的实现：

```python
# softmax函数
def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x, axis=0)
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T

    x = x - np.max(x) # 溢出对策
    return np.exp(x) / np.sum(np.exp(x))

# 交叉商函数
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 监督数据是one-hot-vector的情况下，转换为正确解标签的索引
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size
```

ReLU 层的实现：

```python
# ReLu层
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

Affine 层的实现：

```python
# Affine层
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 权重和偏置参数的导数
        self.dW = None
        self.db = None

    def forward(self, x): # 前向计算
        # 对应张量
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):  # 反向梯度计算
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape)  #
        return dx
```

SoftmaxWithLoss 层的实现：

```python
# Softmax层
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None # softmax的输出
        self.t = None # 监督数据

    def forward(self, x, t): # 前向计算
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss


    def backward(self, dout=1): # 反向梯度计算
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size: # 监督数据是one-hot-vector的情况
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

mnist_output.py 的功能实现很简单，只是提供了一个 img_show 函数用于显示 MNIST 数据样本的图片，但这一方法只在调试程序中用到过，另一个函数是 img_save，这个函数通过接收的参数值，将单个 MNIST 样本数字图片以 png 格式保存到 output 目录下，若该样本为识别正确的样本，则保存到目录 output/positive 中，若为识别错误的样本，则保存到目录 output/negative 下。图片命名格式为“样本序列号_label_样本的标签_predict_识别结果.png”。

```
# 图片保存
def img_save(isPos, img, label, pre, seq):
    img = img.reshape(28, 28)
    pil_img = Image.fromarray(np.uint8(img))
    if isPos == True:
        path = os.path.abspath(os.path.join(os.getcwd(), ".."))+'/output/positive/'
    else:
        path = os.path.abspath(os.path.join(os.getcwd(), ".."))+'/output/negative/'
    file_name = path+str(seq)+'-label-'+str(label)+"-predict-"+str(pre)+'.png'
    pil_img.save(file_name, 'PNG')
```

BPNetwork.py 位于中底层，该文件创建了一个 BPNetwork 类提供给上层调用，并设置了输入层神经元数量 784，隐含层神经元数量 50，输出层神经元数量 10，并通过调用 layers.py 中的类实例化创建了 Affine1 层，Relu1 层，Affine2 层以及 SoftmaxWithLoss 层（即将 Softmax 与损失函数相结合，便于计算）。并提供了 predict 方法用于识别样本数据，loss 方法用于计算样本的损失值，accuracy 方法用于计算样本的识别精确率，gradient 方法用于计算训练样本的梯度值。

```
class BPNetwork:
    def __init__(self):
        # 初始化权重
        input_size = 784 # 输入层神经元数量
        hidden_size = 50 # 隐含层神经元数量
        output_size = 10 # 输出层神经元数量
        weight_init_std = 0.1

        self.params = {} # 参数字典
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

        # 生成层
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()
```

train_mnist.py 位于最高层，这一文件通过调用 load_mnist 函数，加载 MNIST 训练集数据，并实例化一个 BPNetwork 类，设置了训练次数为 10000，批处理的大小为 100，学习率为 0.1，然后用 MNIST 训练集作为输入迭代 10000 次进行模型的训练。每一次训练时，采用随机梯度下降法，即从训练集 60000 个样本中随

机抽取 100 个样本，并将这 100 个样本作为输入给 BPNetwork 进行计算权重和偏置的梯度，然后将得到的梯度与学习率相乘用于更新权重和偏置的值。这就算是一次学习，最后不断训练 10000 次，最后收敛稳定。并最终将训练好的模型保存到目录 dataset 下的 weights.pkl 文件中，用于 test_mnist.py 和 predict_number.py 中进行模型加载。

```
# 超参数
iters_num = 10000 # 训练次数
train_size = x_train.shape[0] # 训练size
batch_size = 100 # 批处理的size，每次训练从训练集中随机选出batch_size数量的样本进行训练
learning_rate = 0.1 # 学习率

train_acc_list = []
test_acc_list = []

print("训练开始..")
start_time = time.time()
for i in range(iters_num): # 训练次数为10000次
    # 获取mini-batch，重复随机梯度下降法
    batch_mask = np.random.choice(train_size, batch_size) # 从train_size个数据中随机选择batch_size个
    x_batch = x_train[batch_mask] # [100][784]
    t_batch = t_train[batch_mask] # [100][10]

    # 计算梯度
    grad = network.gradient(x_batch, t_batch)

    # 更新参数
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 记录每次训练后的精度变化
    '''train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)'''

    # 计算每个epch的识别精度
    if i % 1000 == 0 or i == 9999:
        train_acc = network.accuracy(x_train, t_train)
        print("\t训练", i+1, "次，训练数据集识别精确率：", train_acc)

end_time = time.time()
print("训练结束！训练耗时：", int(end_time-start_time) , "s \n")
```

test_mnist.py 与 train_mnist.py 位于同一层，这一层中，调用 load_mnist 函数读取 MNIST 测试集数据，并实例化一个 BPNetwork 类，并加载 dataset 目录中已经训练好的文件 weights.pkl 中模型数据，最后用测试集中的 10000 个样本进行识别，计算最后的识别精确率，并调用 img_save 函数，将识别正确的样本和识别错误的样本分别以 png 格式保存到 output/positive 和 output/negative 目录中。

```python
# 获取数据
(x_train, t_train), (x_test, t_test) = load_mnist(
    normalize=True, flatten=True, one_hot_label=False)

# 权重参数数据文件
dataset_dir = os.path.abspath(os.path.join(os.getcwd(), "..")) + "/dataset"
save_file = dataset_dir + "/weights.pkl"

network = BPNetwork()  # BP神经网络实例
network.load_weights(save_file) # 加载训练后的权重参数

test_acc = network.accuracy(x_test, t_test)
print("\n测试数据集样本数为 10000，识别精确率: ", test_acc, "\n")

test = network.predict(x_test) # 返回识别结果
(x_train, t_train), (x_test, t_test) = load_mnist(
    normalize=False, flatten=True, one_hot_label=False)
```

predict_number.py 的基本功能与 test_mnist.py 类似，也是通过已经训练好的模型来识别图片数字，不过此文件中只用于识别单个图片样本的数字。文件提供了 read_img 函数和 predict 函数。read_img 函数接收一个图片文件路径作为输入，读取该图片，并转化为灰度图，然后降维为一维数组，并进行正规化。predict 函数同样接受一个图片文件目录，调用 read_img 函数读取该图片并处理，然后创建实例化 BPNetwork 类并加载训练好的模型，最后输出识别结果，即一个整数。

```python
def read_img(file_name):
    with Image.open(file_name) as image:
        # img -> numpy
        img = np.array(image) # 转化为numpy
        gray_img = np.zeros(shape=(28, 28))
        # rgb转化为单通道
        for i in range(28):
            for j in range(28):
                r, g, b = img[i][j][0], img[i][j][1], img[i][j][2],
                gray_img[i][j] = r * 0.2126 + g * 0.7152 + b * 0.0722

        gray_img = gray_img.flatten()  # 降维
        # 正规化
        gray_img = gray_img.astype(np.float32)
        # gray_img /= 255.0
        gray_img[gray_img != 0] = 1
        print(gray_img)

    return np.array([gray_img])


def predict(file_name):
    img = read_img(file_name)

    # 权重参数数据文件
    dataset_dir = os.path.abspath(os.path.join(os.getcwd(), "..")) + "/dataset"
    save_file = dataset_dir + "/weights.pkl"

    network = BPNetwork()  # BP神经网络实例
    network.load_weights(save_file)  # 加载训练后的权重参数

    y = network.predict(img)  # 返回识别结果
    num = np.argmax(y)
    return num
```

draw_board.py 参考了网上大牛对于 python 实现画版的方法，通过第三方库 opencv，创建一个画版，并用鼠标可以在画版上进行书写，并提供简单的 ui 操作，进行书写结果的识别以及清空重新书写。识别过程是调用了 predict 函数进

行单个图片的识别。

```python
def drawcircle(event, x, y, flags, param):
    global drawing, mode
    color = (255, 255, 255)

    # 当按下左键时，返回起始的位置坐标
    if event == cv2.cv2.EVENT_LBUTTONDOWN:
        drawing = True
    # 当鼠标左键按下并移动则是绘画圆形，event可以查看移动，flag查看是否按下
    elif event == cv2.cv2.EVENT_MOUSEMOVE and flags == cv2.cv2.EVENT_FLAG_LBUTTON:
        if drawing == True:
            if mode == True:
                cv2.cv2.rectangle(img, (x, y), (x+1, y+1), color, -1)
            else:
                # 绘制圆圈，小圆点连接在一起成为线，1代表了比划的粗细
                cv2.cv2.circle(img, (x, y), 1, color, -1)
        elif event == cv2.cv2.EVENT_LBUTTONUP:
            drawing = False
```

简单的 ui：

```python
if __name__ == "__main__":
    img = cv2.cv2.imread('../testdata/src.png')
    cv2.cv2.namedWindow('image', cv2.cv2.WINDOW_NORMAL)
    cv2.cv2.setMouseCallback('image', drawcircle)

    print("Create draw board successfully")
    while True:
        cv2.cv2.imshow('image', img)
        key = cv2.cv2.waitKey(10) & 0xFFF
        if key == ord('m'): # 模式转化
            print("Mode changing..")
            mode = not mode
            print("Done")
        elif key == 27 or key == ord('q'): # 退出程序
            print("Quit")
            break
        elif key == ord('p'): # 进行图像识别
            print("Start to predict the number..")
            save_file = "../testdata/dst.png"
            cv2.cv2.imwrite(save_file, img)
            num = predict(save_file)
            print("Done")
            print("The number is:", num)
            # ui
            cv2.cv2.namedWindow('number', cv2.cv2.WINDOW_NORMAL)
            # number = cv2.cv2.imread('../testdata/src.png')
            number = np.zeros((600, 600, 3), np.uint8)
            cv2.cv2.putText(number, "The number is", (55, 100),
                            cv2.cv2.FONT_HERSHEY_TRIPLEX, 2, (255, 255, 255), 4)
            cv2.cv2.putText(number, str(num), (160, 500),
                        cv2.cv2.FONT_HERSHEY_TRIPLEX, 14, (255, 255, 255), 18)
            cv2.cv2.imshow('number', number)
        elif key == ord('c'): # 清空画板，重新写字
            cv2.cv2.destroyWindow('number')

            print("Clearing the board..")
            img = cv2.cv2.imread('../testdata/src.png')
            print("Done")

    cv2.cv2.destroyAllWindows()
```

3．结果分析

（交代实验环境，算法设计设计的参数说明；

结果（图或表格），比如在若干次运行后所得的最好解，最差解，平均值，标准差。

分析算法的性能，包括解的精度，算法的速度，或者与其他算法的对比分析。

算法的优缺点；本实验的不足之处，进一步改进的设想。）

3.1 实验环境

操作系统：macOS Mojave；
编程语言：Python3；
编辑器：VS Code；

3.2 算法设计的参数

BP 神经网络输入层神经元个数	784
BP 神经网络隐含层神经元个数	50
BP 神经网络输出层神经元个数	10
训练迭代次数	10000
训练批处理大小	100
学习率	0.1

3.3 实验结果

训练时间	36s
训练集识别精度	97.765%
测试集识别精度	96.94%

MNIST 数据集训练模型：
运行：在 src 目录下命令后执行命令：python train_mnist.py；

```
appledeAir:src apple$ python3 train_mnist.py
训练开始..
        训练 1 次，训练数据集识别精确率： 0.11258333333333333
        训练 1001 次，训练数据集识别精确率： 0.9229666666666667
        训练 2001 次，训练数据集识别精确率： 0.9385166666666667
        训练 3001 次，训练数据集识别精确率： 0.9516666666666667
        训练 4001 次，训练数据集识别精确率： 0.9585666666666667
        训练 5001 次，训练数据集识别精确率： 0.9635333333333334
        训练 6001 次，训练数据集识别精确率： 0.9699333333333333
        训练 7001 次，训练数据集识别精确率： 0.9720666666666666
        训练 8001 次，训练数据集识别精确率： 0.9743
        训练 9001 次，训练数据集识别精确率： 0.9752333333333333
        训练 10000 次，训练数据集识别精确率： 0.97765
训练结束！训练耗时： 36 s

训练后的权重参数数据保存到文件： /Users/apple/Desktop/src/dataset/weights.pkl
```

由上图可知，训练 10000 次后，以训练集作为测试集进行识别计算的精确度达到 97.765%。训练耗时为 36 秒。
训练过程的训练集和测试集的识别精度曲线图如下所示：
MNIST 训练集识别精度与训练次数关系：

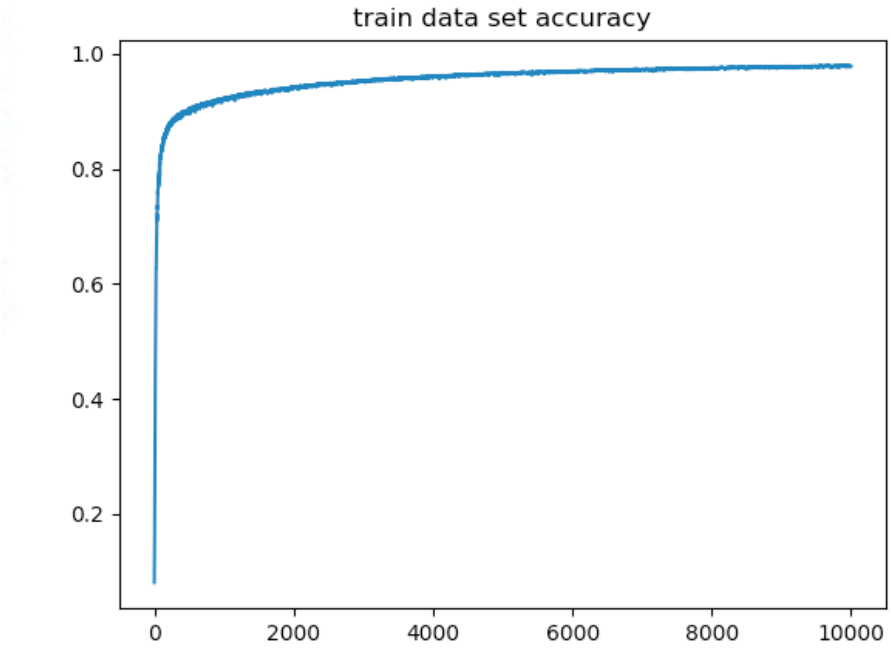
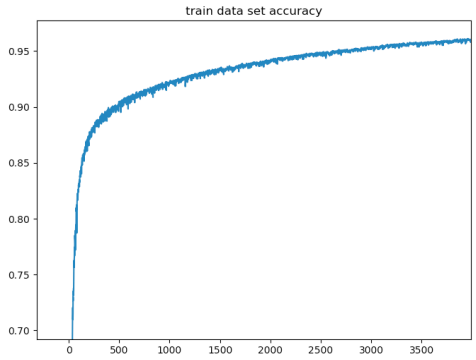
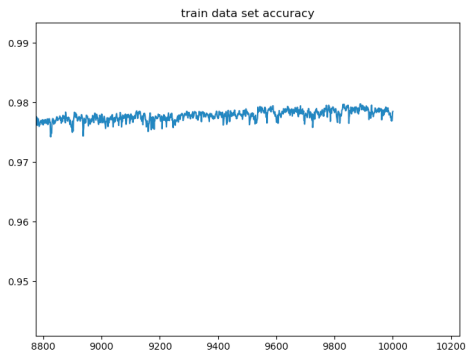


图 4 训练集识别率与训练次数关系曲线图

通过上图可以看出，大概训练前 100 次左右，训练集的识别率就已经高达 90%以上，之后 9900 次训练都只是缓慢的提高识别率。通过以上局部放大图可以观察到，当识别率达到 90%左右开始，识别率会出现振荡，这里猜测可能是由于学习率偏高（0.1），所以导致在迭代过程中识别率会出现小范围的振荡，不过振荡幅度很小，而且整体是上升趋势，所以这里也没有更改识别率，若将学习率降低为 0.01，振荡现象会减少，且振荡幅度也会减小。





MNIST 测试集识别精度与训练次数关系：

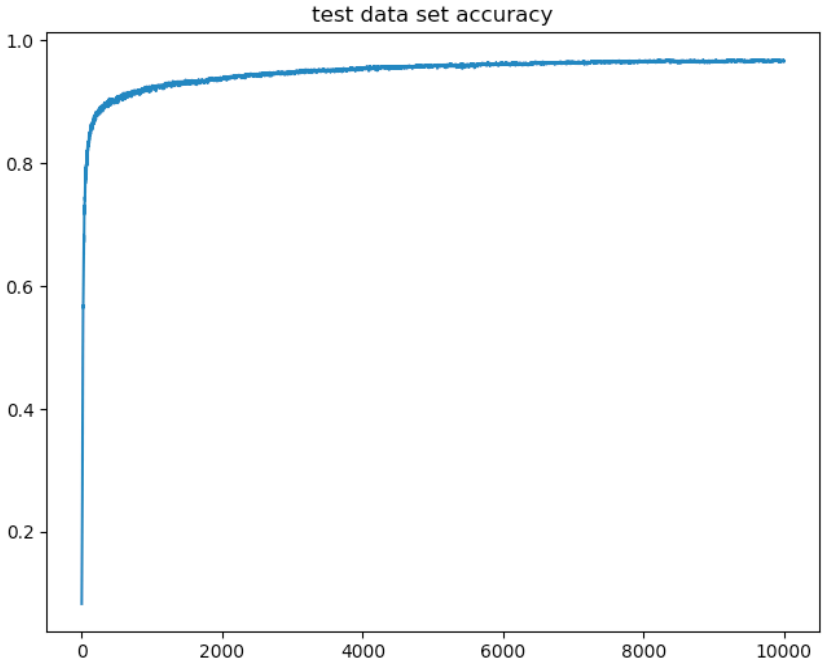
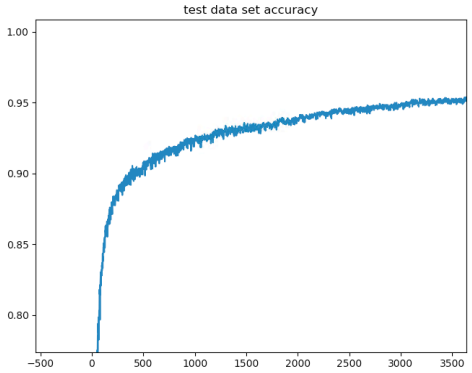
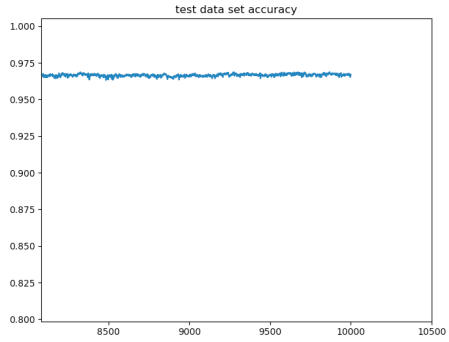


图 5 测试集识别率与训练次数关系曲线图

这里，测试集的曲线和和训练集的曲线很相似，观察如下局部放大图可以观察到，测试集也是训练了 100 次左右，识别率就达到了 90%左右，且这之后识别率的提高变得很缓慢，而且也会出现振荡现象，但是振荡幅度远远小于训练集的曲线的振荡幅度。





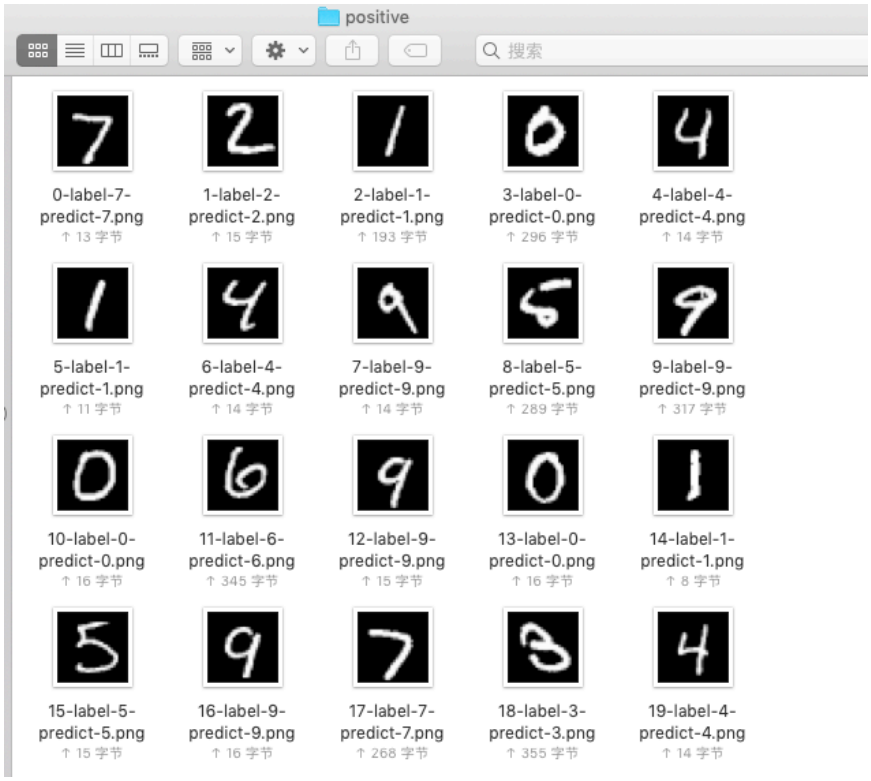
MNIST 测试集识别测试：

运行：在 src 目录下命令后执行命令：python test_mnist.py；

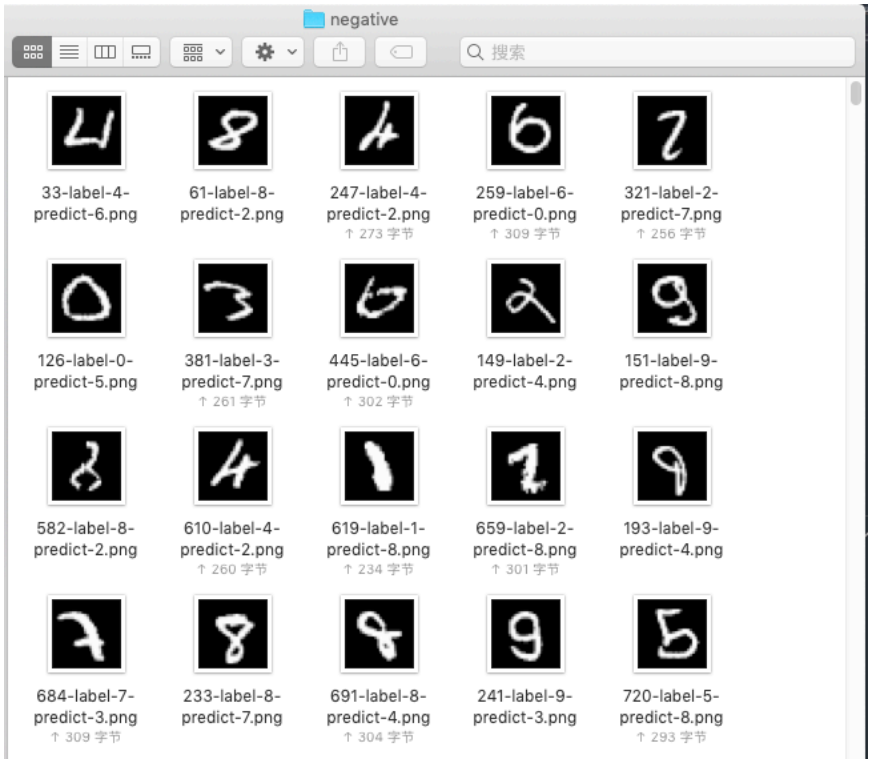
```
appledeAir:src apple$ python3 test_mnist.py
测试数据集样本数为 10000, 识别精确率： 0.9694
```

用 MNIST 测试集数据作为测试集进行识别计算的精确度达到 96.94%，接近 97%。表示 10000 张测试集样本中，有 9694 张图片识别正确，306 张图片识别错误。

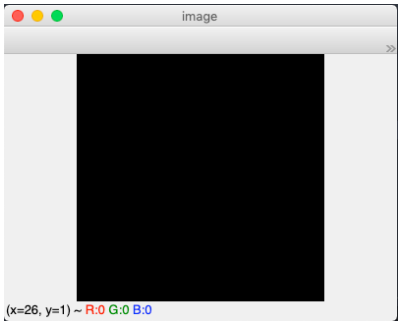
且识别后的测试集样本都保存到 output 目录下了。
output/positive：



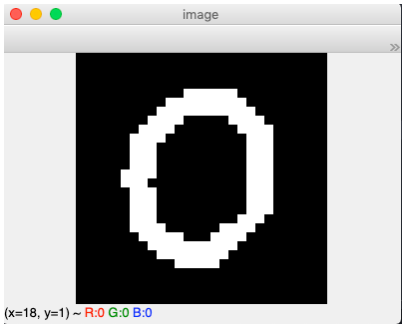
output/negative：



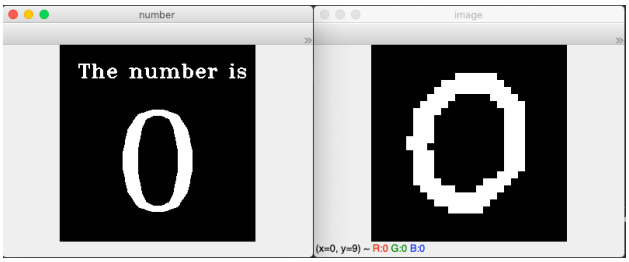
画版鼠标手写数字识别程序：
运行：在 src 目录下命令后执行命令：python draw_board.py；
会出现如下图所示画图窗口：



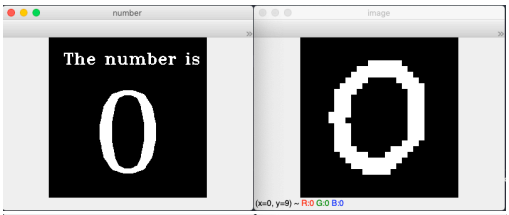
然后只需要如同在 windows 的画图里那样，按住鼠标左键在黑色画版区域中书写数字，例如书写数字 0:

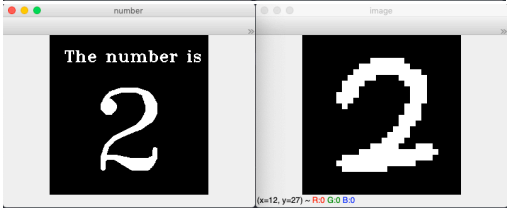


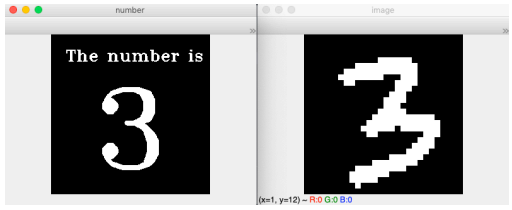
然后单击键盘上的字母“p”，则会输出识别结果，结果如下：

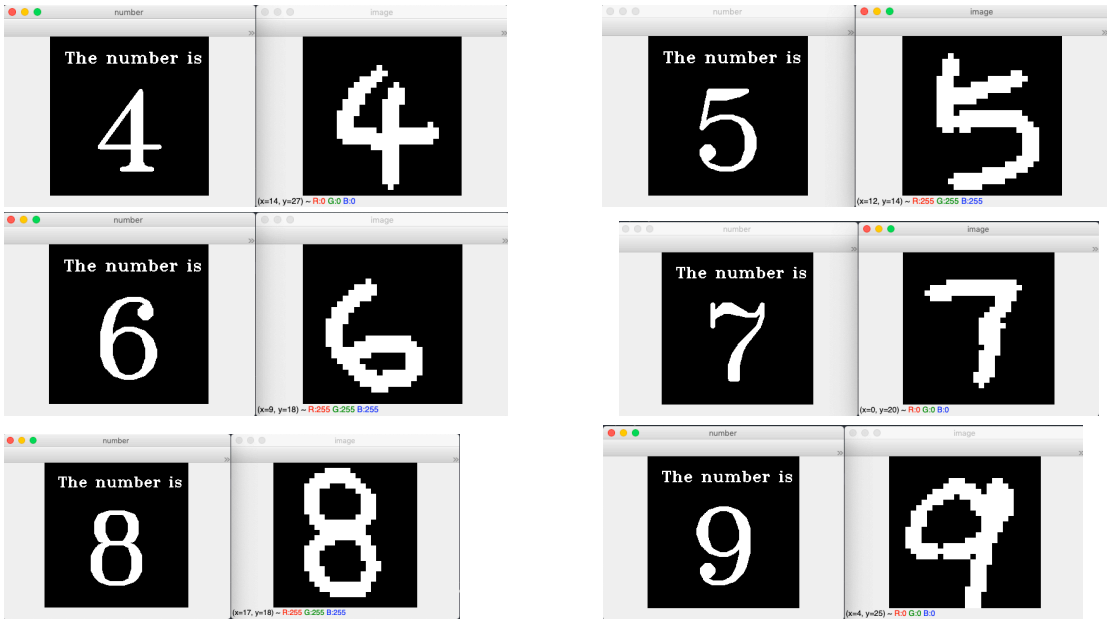


左边为按“p”后弹出的新窗口，其说明了书写的数字的识别结果为 0；
再次单击键盘上的字母“c”，则可以清空输入，重新书写数字。最后按 esc 或者字母“q”则可以关闭程序。
0-9 识别正确时的结果如下：（左边为鼠标书写，右边为识别结果）









3.4 分析

本次实验的实验结果还算比较理想，第一是训练时间短，最开始的训练选用了全训练方法，即将所有 MNIST 样本作为训练集输入进行训练，但训练时间很长，且容易出现过拟合等情况，查询资料时发现了重复随机梯度下降法和批处理的方法，修改算法后，训练时间大大减少，且识别率并未降低。

通过分析测试集中识别错误的样本发现，大部分样本确实很难识别，因为即使是人类去判断也会出现错误或困惑，因此机器存在识别错误也是可以理解的，例如测试集中 33 号样本图片如下：

这个图片的标签是 4，但是这个书写真的很不规范，神经网络的识别结果为 6；又如 1181 号样本：

这个图片的标签为 6，但是图片的圈很细小，看起来像是 1，神经网络的识别结果为 1。同样的例子很多，详情可以参看目录 output/negative 下的图片。

最后是对鼠标输入的数字进行识别，上面给出了正确的识别结果，但是，需要非常规范的书写数字，才能正确的识别，如果稍微潦草或者不规范，则无法正确的识别，会出现错误的识别结果，例如在书写 6 时可能会被识别为 5，在书写 9 时会被识别为 8 等情况发生。分析其原因，我认为是因为在 MNIST 数据集中，图片样本中等数字的边缘会出现渐变的过度层，而这里的画板书写的数字，数字的灰度值都是一样的，数字的边缘没有渐变过渡，所以这一特征可能没有办法识别；其次是鼠标的书写和人的手书写的数字风格存在差异，所以在识别时可能无法识别到。

4．结论

（简要结论或者体会。）

本次实验的主要目的是构造三层 BP 神经网络，通过这次试验，我以点到面更深入的了解了神经网络多层感知机的工作原理，也通过这次试验学习了很多神经网络的相关知识，例如输入层、隐含层、输出层、神经元、权重和偏置、激活函数 softmax、sigmoid、ReLU 等、重复随机梯度下降法、误差反向传播、损失函数 cross entropy function，批处理方法等等。真正的把书本上的理论知识转变为实际的程序应用。

由于构建 BP 神经网络的经验不多，实验过程中也参看了许多的文献、网络上大牛的博客以及论坛的问答帖子等，才慢慢梳理出了 BP 神经网络的结构和具体细节实现，并且不断的修改和改进，最终才做出一个可以具有简单数字识别功能的程序。

主要参考文献(三五个即可)

[1]沈花玉,王兆霞,高成耀,秦娟,姚福彬,徐巍.BP 神经网络隐含层单元数的确定[J].天津理工大学学报,2008(05):13-15.

[2]刘天舒. BP 神经网络的改进研究及应用[D].东北农业大学,2011.

[3]戚德虎,康继昌.BP 神经网络的设计[J].计算机工程与设计,1998(02):47-49.