華東理工大學

EAST CHINA UNIVERSITY OF SCIENCE AND TECHNOLOGY

《 人工智能 》

实验报告本

班　级：　计 203

学　号：　20002462

姓　名：　刘子言

指导教师：　陈志华

信息科学与工程学院

2022 年 12 月

《人工智能》实 验 报 告 四

实验 4 名称：基于神经网络的模式识别实验	**实验地点**：线上

所使用的工具软件及环境：

装有 Windows 操作系统、Python 版本：Python 3.7

一、实验目的

理解 BP 神经网络的机构和原理，掌握反向传播学习算法对神经元的训练过程，了解反向传播公式。通过构建 BP 网络识别实例，熟悉前馈网络和反馈网络的原理及结构。

二、实验内容

编译器不限,使用 tensorflow 框架,以 MNIST 数据集为数据,实现对 MNIST 数据集分类的操作，其中 MNIST 数据集由 10 类，分别为手写数字 0-9.

三、实验要求

1.用 MINSI 数据集训练编写的网络，要求记录每次迭代生成器和判别器的损失函数值。

2.改变卷积神经网络的参数，观察分类准确率。思考网络层数或参数对分类准确率的影响。

四、实验步骤

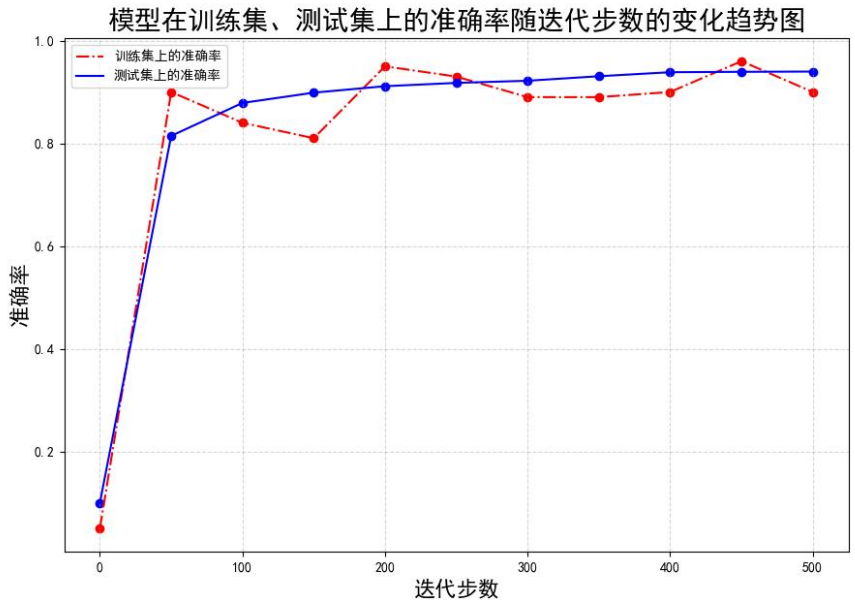
编写基于卷积神经网络的模式识别的 python 代码,用 MINSI 数据集训练编写的网络，并分别求出分步迭代的训练集准确率和测试集准确率，并在此基础上对代码进行进一步的改进及运行。

1、在基本代码的基础上添加“准确率”关于迭代步数的折线图，可视化展示变化趋势：

```
# 绘制准确率随迭代数增加的折线图
def plot_broken_line(X, Y1, Y2):
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 在图中可以显示中文
    plt.figure(figsize=(20, 10), dpi=100)
    plt.plot(X, Y1, c='red', linestyle='-.', label="训练集上的准确率")
    plt.plot(X, Y2, c='blue', label="测试集上的准确率")
    plt.scatter(X, Y1, c='red')
    plt.scatter(X, Y2, c='blue')
    plt.legend(loc='best')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xlabel("迭代步数", fontdict={'size': 16})
    plt.ylabel("准确率", fontdict={'size': 16})
    plt.title("模型在训练集、测试集上的准确率随迭代步数的变化趋势图", fontdict={'size': 20})
    plt.show()
```

修改后的运行结果如下：

```
运行:  patternRecognition_NN
2022-12-20 11:37:16.125816: I tensorflow/core/platform/cpu_feature_gu
step 0, training accuracy 0.05, test accuracy 0.0998
step 50, training accuracy 0.9, test accuracy 0.8145
step 100, training accuracy 0.84, test accuracy 0.8787
step 150, training accuracy 0.81, test accuracy 0.8987
step 200, training accuracy 0.95, test accuracy 0.9113
step 250, training accuracy 0.93, test accuracy 0.9177
step 300, training accuracy 0.89, test accuracy 0.9219
step 350, training accuracy 0.89, test accuracy 0.9307
step 400, training accuracy 0.9, test accuracy 0.9385
step 450, training accuracy 0.96, test accuracy 0.9393
step 500, training accuracy 0.9, test accuracy 0.9398
test accuracy 0.9443
学号：20002462
```



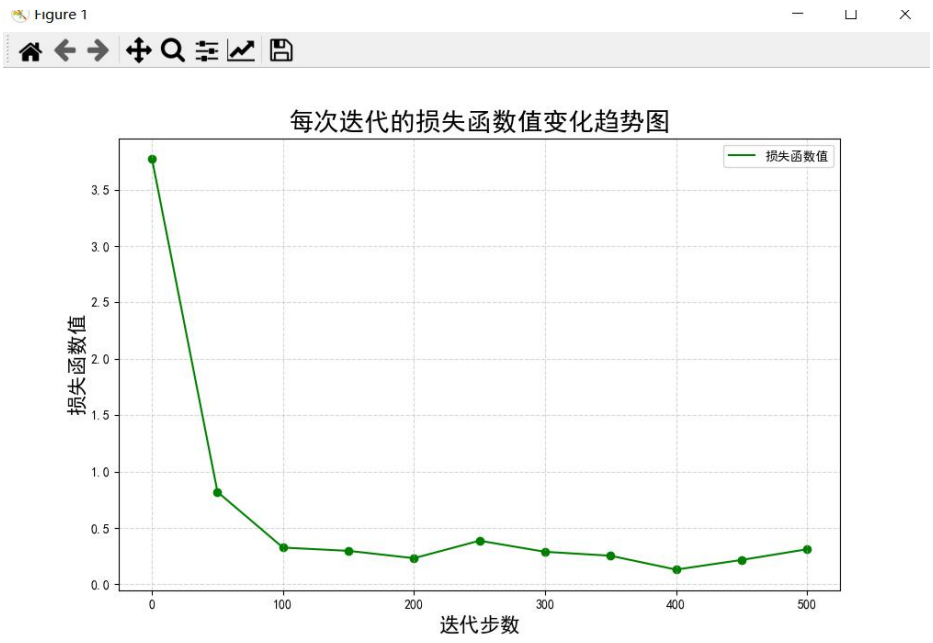
2、添加代码，计算、输出并记录每次迭代的损失函数值，并绘制“损失函数值”关于迭代步数的折线图：

```python
with tf.Session() as sess:
    sess.run(init_op)
    for i in range(550):  # 训练样本为55000，分成550批，每批为100个样本
        batch = mnist.train.next_batch(100)
        if i % 50 == 0:  # 每过50批，显示其在训练集上的准确率和在测试集上的准确率
            train_accuracy = accuracy.eval(feed_dict={x: batch[0], y: batch[1], keep_prob: 1.0})
            test_accuracy = accuracy.eval(feed_dict={x: mnist.test.images, y: mnist.test.labels})
            loss = sess.run(cross_entropy, feed_dict={x: batch[0], y: batch[1], keep_prob: 0.5})  # 计算每一步的损失函数值;
            all_train_accuracy.append(train_accuracy)  # 用于绘图的数据
            all_test_accuracy.append(test_accuracy)
            all_step.append(i)
            all_loss.append(float(loss))
            print('step %d, training accuracy %g, test accuracy %g, loss %g' % (
                i, train_accuracy, test_accuracy, loss))
        # 每一步迭代，都会加载100个训练样本，然后执行一次train_step，并通过feed_dict，用训练数据替代x和y张量占位符。
        sess.run(train_step, feed_dict={x: batch[0], y: batch[1], keep_prob: 0.5})
    # 显示最终在测试集上的准确率
    print(
        'test accuracy %g' % accuracy.eval(feed_dict={x: mnist.test.images, y: mnist.test.labels, keep_prob: 1.0}))
print("学号：20002462")
# plot_broken_line(all_step, all_train_accuracy, all_test_accuracy)  # 准确率折线图
plot_broken_line2(all_step, all_loss)  # 损失函数值折线图
```

```python
# 绘制损失函数值随迭代数增加的折线图
def plot_broken_line2(X, Y):
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 在图中可以显示中文
    plt.figure(figsize=(20, 10), dpi=100)
    plt.plot(X, Y, c='green', label="损失函数值")
    plt.scatter(X, Y, c='green')
    plt.legend(loc='best')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xlabel("迭代步数", fontdict={'size': 16})
    plt.ylabel("损失函数值", fontdict={'size': 16})
    plt.title("每次迭代的损失函数值变化趋势图", fontdict={'size': 20})
    plt.show()
```

修改后的运行结果如下（其中输出项增加了'loss'值，表示损失函数值）：

```
运行:  patternRecognition_NN
2022-12-20 14:43:19.525663: I tensorflow/core/platform/cpu_feature_guard.cc:1
step 0, training accuracy 0.16, test accuracy 0.1114, loss 3.77384
step 50, training accuracy 0.75, test accuracy 0.8126, loss 0.819603
step 100, training accuracy 0.91, test accuracy 0.8748, loss 0.326211
step 150, training accuracy 0.9, test accuracy 0.8964, loss 0.297169
step 200, training accuracy 0.91, test accuracy 0.9085, loss 0.232005
step 250, training accuracy 0.89, test accuracy 0.9198, loss 0.387245
step 300, training accuracy 0.91, test accuracy 0.9298, loss 0.288845
step 350, training accuracy 0.94, test accuracy 0.9316, loss 0.253922
step 400, training accuracy 0.95, test accuracy 0.9364, loss 0.130918
step 450, training accuracy 0.92, test accuracy 0.9394, loss 0.216545
step 500, training accuracy 0.92, test accuracy 0.9425, loss 0.31121
test accuracy 0.9421
学号：20002462
```



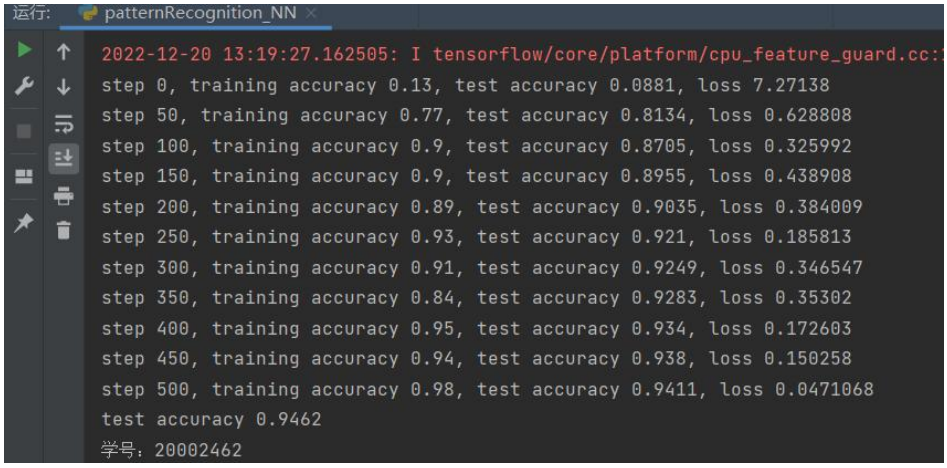
3、增加神经网络的隐藏层数，观察网络层数对分类准确率的影响（原先设置的是 2 层隐藏层，共 4 层神经网络，test accuracy=0.9421）。

（以下将各隐藏层神经元的个数都修改为了 2 的幂次数）

• 设置 3 层隐藏层，共 5 层神经网络

```
# 创建神经网络第2层，隐藏层，激活函数为relu
W_layer2 = weight_variable([512, 1024])
b_layer2 = bias_variable([1024])
h2 = tf.add(tf.matmul(h1, W_layer2), b_layer2)  # W * h1 + b，h1为第1层的输出
h2 = tf.nn.relu(h2)
# 创建神经网络第3层，隐藏层，激活函数为relu
W_layer3 = weight_variable([1024, 1024])
b_layer3 = bias_variable([1024])
h3 = tf.add(tf.matmul(h2, W_layer3), b_layer3)  # W * h2 + b，h2为第2层的输出
h3 = tf.nn.relu(h3)
# 新增神经网络第4层，隐藏层，激活函数为relu
W_layer4 = weight_variable([1024, 256])
b_layer4 = bias_variable([256])
h4 = tf.add(tf.matmul(h3, W_layer4), b_layer4)  # W * h3 + b，h3为第3层的输出
h4 = tf.nn.relu(h4)
```

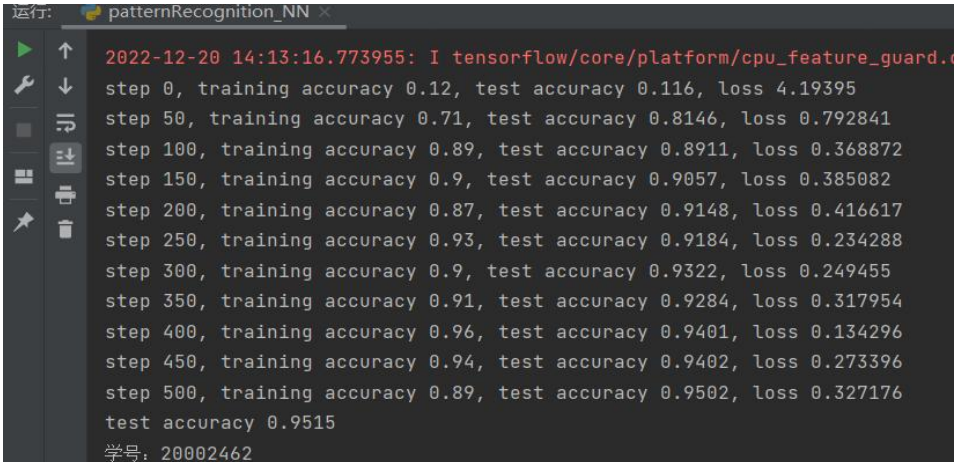
修改后得到的结果如下：



• 设置 4 层隐藏层，共 6 层神经网络

```
58      # 创建神经网络第2层，隐藏层，激活函数为relu
59      W_layer2 = weight_variable([512, 1024])
60      b_layer2 = bias_variable([1024])
61      h2 = tf.add(tf.matmul(h1, W_layer2), b_layer2)  # W * h1 + b，h1为第1层的输出
62      h2 = tf.nn.relu(h2)
63      # 创建神经网络第3层，隐藏层，激活函数为relu
64      W_layer3 = weight_variable([1024, 512])
65      b_layer3 = bias_variable([512])
66      h3 = tf.add(tf.matmul(h2, W_layer3), b_layer3)  # W * h2 + b，h2为第2层的输出
67      h3 = tf.nn.relu(h3)
68      # 新增神经网络第4层，隐藏层，激活函数为relu
69      W_layer4 = weight_variable([512, 256])
70      b_layer4 = bias_variable([256])
71      h4 = tf.add(tf.matmul(h3, W_layer4), b_layer4)  # W * h3 + b，h3为第3层的输出
72      h4 = tf.nn.relu(h4)
73      # 新增神经网络第5层，隐藏层，激活函数为relu
74      W_layer5 = weight_variable([256, 128])
75      b_layer5 = bias_variable([128])
76      h5 = tf.add(tf.matmul(h4, W_layer5), b_layer5)  # W * h4 + b，h4为第4层的输出
77      h5 = tf.nn.relu(h5)
```

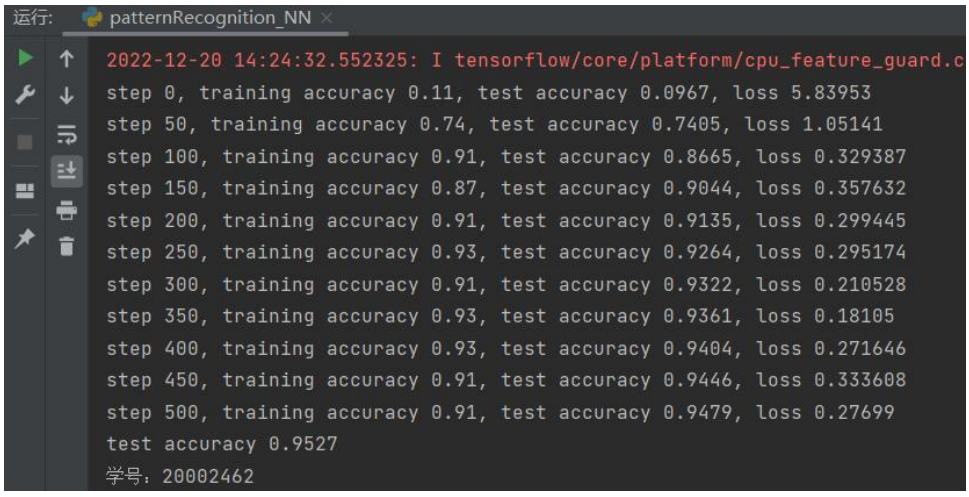
修改后得到的结果如下：



- 设置 5 层隐藏层，共 7 层神经网络

```
    # 创建神经网络第2层，隐藏层，激活函数为relu
    W_layer2 = weight_variable([512, 1024])
    b_layer2 = bias_variable([1024])
    h2 = tf.add(tf.matmul(h1, W_layer2), b_layer2)  # W * h1 + b，h1为第1层的输出
    h2 = tf.nn.relu(h2)
    # 创建神经网络第3层，隐藏层，激活函数为relu
    W_layer3 = weight_variable([1024, 512])
    b_layer3 = bias_variable([512])
    h3 = tf.add(tf.matmul(h2, W_layer3), b_layer3)  # W * h2 + b，h2为第2层的输出
    h3 = tf.nn.relu(h3)
    # 新增神经网络第4层，隐藏层，激活函数为relu
    W_layer4 = weight_variable([512, 256])
    b_layer4 = bias_variable([256])
    h4 = tf.add(tf.matmul(h3, W_layer4), b_layer4)  # W * h3 + b，h3为第3层的输出
    h4 = tf.nn.relu(h4)
    # 新增神经网络第5层，隐藏层，激活函数为relu
    W_layer5 = weight_variable([256, 128])
    b_layer5 = bias_variable([128])
    h5 = tf.add(tf.matmul(h4, W_layer5), b_layer5)  # W * h4 + b，h4为第4层的输出
    h5 = tf.nn.relu(h5)
    # 新增神经网络第6层，隐藏层，激活函数为relu
    W_layer6 = weight_variable([128, 64])
    b_layer6 = bias_variable([64])
    h6 = tf.add(tf.matmul(h5, W_layer6), b_layer6)  # W * h5 + b，h5为第5层的输出
    h6 = tf.nn.relu(h6)
```

修改后得到的结果如下：



综上测试，整理得到如下表格：

隐藏层层数	神经网络层数	Final test accuracy
2	4	0.9421
3	5	0.9462
4	6	0.9515
5	7	0.9527

由上述对隐藏层代码的修改及测试结果可见，随着神经网络层数的增加，分类的准确率也有了一定程度的提高，隐藏层的层数越多，网络输出的准确度和精度也随之提高，且个别权因子的损坏对网络输出产生的影响也越来越小。

五、 程序设计的核心代码

1、BP神经网络的核心代码：

```
# 加载MNIST数据集，通过设置 one_hot=True 来使用独热编码标签
# 独热编码：对于每个图片的标签 y，10 位中仅有一位的值为 1，其余的为 0。
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

# mnist = tf.keras.datasets.mnist
# 权重正态分布初始化函数
def weight_variable(shape):
    # 生成截断正态分布随机数,shape表示生成张量的维度，mean是均值(默认=0.0)，
        stddev是标准差。
    # 取值范围为 [ mean - 2 * stddev, mean + 2 * stddev ]，这里为[-0.2, 0.2]
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

# 偏置量初始化函数
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)  # value=0.1, shape是张量的维度
    return tf.Variable(initial)

if __name__ == "__main__":
    print(int(mnist.train.num_examples))  # 55000
    print(int(mnist.test.num_examples))  # 10000

    # 为训练数据集的输入 x 和标签 y 创建占位符
    x = tf.placeholder(tf.float32, [None, 784])  # None用以指代batch的大小，意即输入图片的数量不定，一张图28*28=784
    y = tf.placeholder(tf.float32, [None, 10])
    # 意思是每个元素被保留的概率，keep_prob=1即所有元素全部保留。大量数据训练时，为了防止过拟合，添加Dropout层，设置一个0~1之间的小数
    keep_prob = tf.placeholder(tf.float32)
```

```
    # 创建神经网络第1层，输入层，激活函数为relu
    W_layer1 = weight_variable([784, 512])
    b_layer1 = bias_variable([512])
    h1 = tf.add(tf.matmul(x, W_layer1), b_layer1)  # W * x + b
    h1 = tf.nn.relu(h1)

    # 创建神经网络第2层，隐藏层，激活函数为relu
    W_layer2 = weight_variable([512, 1024])
    b_layer2 = bias_variable([1024])
    h2 = tf.add(tf.matmul(h1, W_layer2), b_layer2)  # W * h1 + b，h1为第1层的输出
    h2 = tf.nn.relu(h2)
    # 创建神经网络第3层，隐藏层，激活函数为relu
    W_layer3 = weight_variable([1024, 512])
    b_layer3 = bias_variable([512])
    h3 = tf.add(tf.matmul(h2, W_layer3), b_layer3)  # W * h2 + b，h2为第2层的输出
    h3 = tf.nn.relu(h3)
    # 新增神经网络第4层，隐藏层，激活函数为relu
    W_layer4 = weight_variable([512, 256])
    b_layer4 = bias_variable([256])
    h4 = tf.add(tf.matmul(h3, W_layer4), b_layer4)  # W * h3 + b，h3为第3层的输出
    h4 = tf.nn.relu(h4)
    # 新增神经网络第5层，隐藏层，激活函数为relu
    W_layer5 = weight_variable([256, 128])
    b_layer5 = bias_variable([128])
    h5 = tf.add(tf.matmul(h4, W_layer5), b_layer5)  # W * h4 + b，h4为第4层的输出
    h5 = tf.nn.relu(h5)
    # 新增神经网络第6层，隐藏层，激活函数为relu
    W_layer6 = weight_variable([128, 64])
    b_layer6 = bias_variable([64])
    h6 = tf.add(tf.matmul(h5, W_layer6), b_layer6)  # W * h5 + b，h5为第5层的输出
    h6 = tf.nn.relu(h6)

    # 创建神经网络第7层，输出层，激活函数为softmax
    W_layer7 = weight_variable([64, 10])
    b_layer7 = bias_variable([10])
    predict = tf.add(tf.matmul(h6, W_layer7), b_layer7)  # W * h6 + b，h6为第6层的输出
    y_conv = tf.nn.softmax(tf.matmul(h6, W_layer7) + b_layer7)

    # 计算交叉熵代价函数
    cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=predict,
labels=y))
    # 使用Adam下降算法优化交叉熵代价函数
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

```
    # 预测是否准确的结果存放在一个布尔型的列表中
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y, 1))  # argmax返回的矩阵行中的最大值的索引号
    # 求预测准确率
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))  # cast将布尔型的数据转换成float型的数据；reduce_mean求平均值

    # 初始化
    init_op = tf.global_variables_initializer()

    all_train_accuracy = []  # 用于存储绘图所需的数据
    all_test_accuracy = []
    all_step = []
    all_loss = []

    with tf.Session() as sess:
        sess.run(init_op)
        for i in range(550):  # 训练样本为55000，分成550批，每批为100个样本
            batch = mnist.train.next_batch(100)
            if i % 50 == 0:
            # 每过50批，显示其在训练集上的准确率和在测试集上的准确率
                train_accuracy = accuracy.eval(feed_dict={x: batch[0], y: batch[1],
                                    keep_prob: 1.0})
                test_accuracy = accuracy.eval(feed_dict={x: mnist.test.images, y:
                                    mnist.test.labels})
                loss = sess.run(cross_entropy, feed_dict={x: batch[0], y: batch[1],
                                    keep_prob: 0.5})  # 计算每一步的损失函数值;
                all_train_accuracy.append(train_accuracy)  # 存储用于绘图的数据
                all_test_accuracy.append(test_accuracy)
                all_step.append(i)
                all_loss.append(float(loss))
                print('step %d, training accuracy %g, test accuracy %g, loss %g' % (
                    i, train_accuracy, test_accuracy, loss))
            # 每一步迭代，都会加载100个训练样本，然后执行一次train_step，
              并通过feed_dict，用训练数据替代x和y张量占位符。
            sess.run(train_step, feed_dict={x: batch[0], y: batch[1], keep_prob: 0.5})

        # 显示最终在测试集上的准确率
        print('test accuracy %g' % accuracy.eval(feed_dict={x: mnist.test.images, y:
            mnist.test.labels, keep_prob: 1.0}))

    print("学号：20002462")

    plot_broken_line(all_step, all_train_accuracy, all_test_accuracy)  # '准确率'折线图
```

```
    plot_broken_line2(all_step, all_loss)    # ‘损失函数值’折线图
```

2、几个趋势图的绘制函数：

```
# 绘制‘准确率’随迭代数增加的折线图
def plot_broken_line(X, Y1, Y2):
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['SimHei']    # 在图中可以显示中文
    plt.figure(figsize=(20, 10), dpi=100)
    plt.plot(X, Y1, c='red', linestyle='-.', label="训练集上的准确率")
    plt.plot(X, Y2, c='blue', label="测试集上的准确率")
    plt.scatter(X, Y1, c='red')
    plt.scatter(X, Y2, c='blue')
    plt.legend(loc='best')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xlabel("迭代步数", fontdict={'size': 16})
    plt.ylabel("准确率", fontdict={'size': 16})
    plt.title("模型在训练集、测试集上的准确率随迭代步数的变化趋势图",
              fontdict={'size': 20})
    plt.show()


# 绘制‘损失函数值’随迭代数增加的折线图
def plot_broken_line2(X, Y):
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['SimHei']    # 在图中可以显示中文
    plt.figure(figsize=(20, 10), dpi=100)
    plt.plot(X, Y, c='green', label="损失函数值")
    plt.scatter(X, Y, c='green')
    plt.legend(loc='best')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xlabel("迭代步数", fontdict={'size': 16})
    plt.ylabel("损失函数值", fontdict={'size': 16})
    plt.title("每次迭代的损失函数值变化趋势图", fontdict={'size': 20})
    plt.show()
```

六、实验体会

通过本次实验，我对 BP 神经网络的基本思想、原理和一般实现步骤有了更加深入的了解。在熟悉神经网络的基本知识、BP 学习算法的基本内容的基础之上，我掌握了如何利用 Python 设计具体的算法步骤，并且最终通过编程实现了基于 BP 神经网络的模式识别系统，记录了分步迭代的训练集准确率和测试集准确率，并在此基础上对代码进行了改进及运行，收获颇丰。

在本次实验过程中，我利用 MINSI 数据集对编写的网络进行训练，记录每次迭代的损失函数值，分别可视化展示了准确率、损失函数值随迭代步数增长的变化趋势，并且通过增加隐藏层层数观察到网络层数对分类准确率的影响，总结了以下的思考和结论：

BP 网络具有以下的特点：输入和输出是并行的模拟量，网络的输入输出关系是由各层连接的权因子决定的；权因子通过学习信号调节，学习越多，网络越聪明；隐含层越多，网络输出的准确率和精度越高，且个别权因子的损坏对网络输出的影响也越来越小。

同时 BP 网络也有一些不足之处，比如收敛速度较慢，存在局部极值，难以确定隐层和隐节点的个数等；关于隐层和隐节点个数的选择，也可以根据实际情况来决定。

BP 神经网络由两部分组成：信号的正向传播以及误差的反向传播。算法大致的原理如下：在正向传播中，输入信号从输入层经隐含层，逐层计算传向输出层，每一层神经元的状态只影响下一层神经元的状态。如果在输出层未得到期望的输出，则计算输出层的误差变化值，然后转到反向传播，通过网络将误差信号沿原来的连接通路反传回来，修改各层神经元的权值直至达到期望目标。

七、教师评语

该学生_____完成了实验任务，算法设计_____，实验结果_____，实验体会_____。

因此总体评价为_____。

教师签字：

年　月　日