实验五　全连接神经网络实践

姓名：李坤璘
班级学号：20 智能 03 2019202216

一、　实验目的：

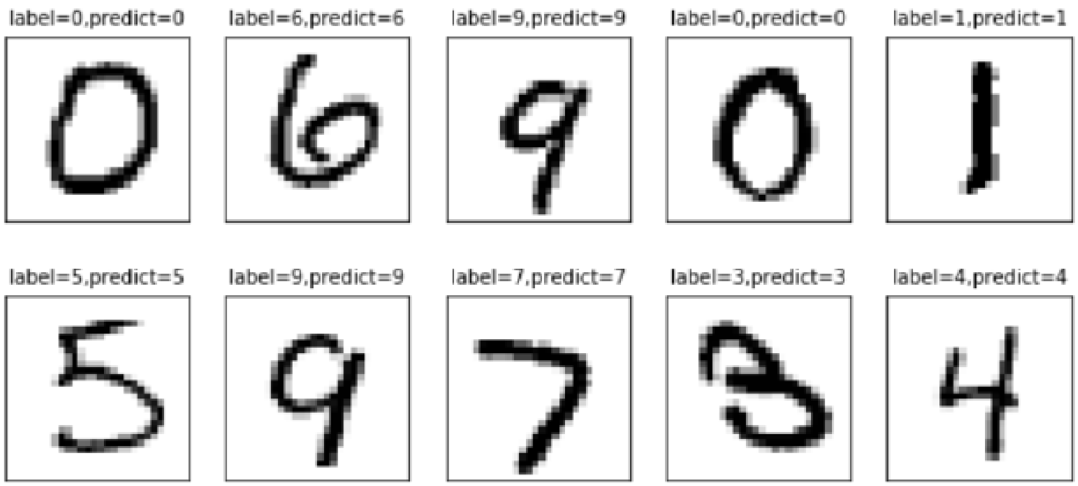
1. 掌握 TensorFlow 的使用方法；
2. 利用全连接神经网络对 MNIST 数据集进行分类；
3. 掌握 Keras 构建全连接神经网络的方法。

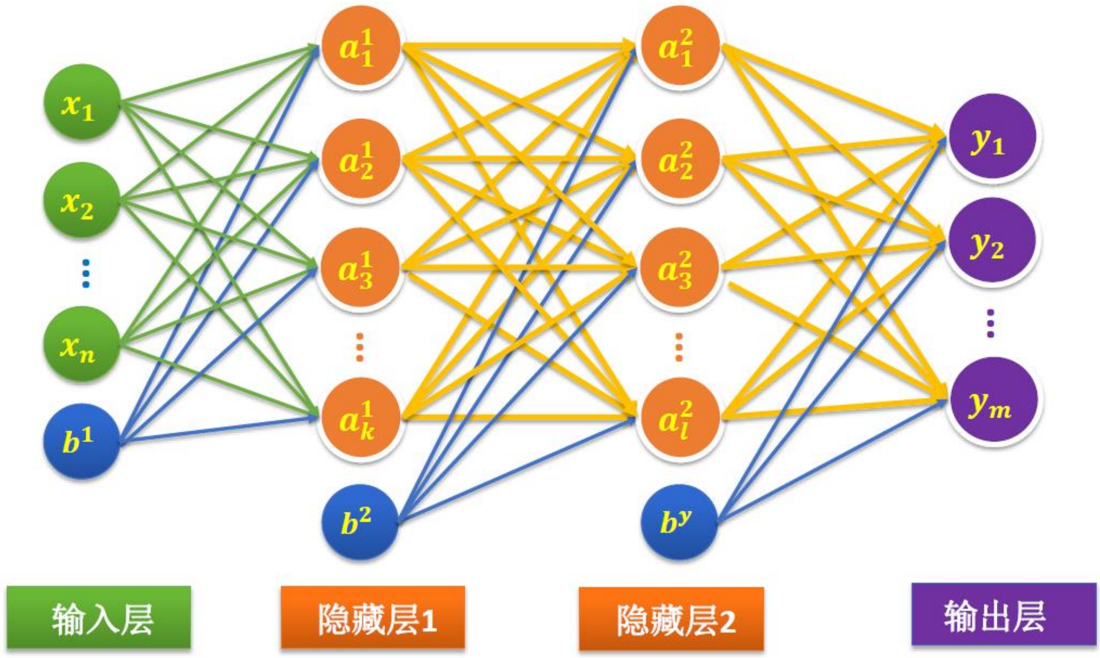
二、　实验条件：

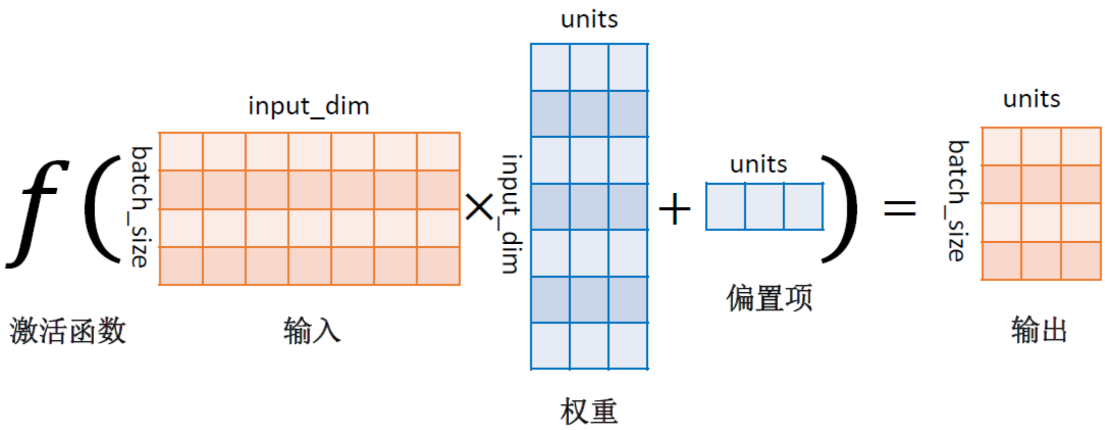
1. PC 微机一台和 Python+TensorFlow 环境。

三、　实验原理：

MNIST是一个入门级的计算机视觉数据集，包含各种手写数字图片，由美国国家标准与技术研究所（NIST）提供，常被作为图片识别的标准数据集。该数据集包括70000个样本，已经对数字进行了预处理和格式化，做了大小调整并居中，图片尺寸固定为28*28，在实际训练过程中，训练速度非常快，收敛效果非常明显。本实验使用TensorFlow内置的MNIST数据集作为实验材料。







四、 实验内容：

调用 TensorFlow 内置的 MNIST 数据集，根据全连接神经网络的原理编程实现数据分类，并使用 Keras 的序列模型，构建更多隐含层的神经网络实现数据分类。

实验结果要求：

（1） 编程实现两个隐含层，分别为 128 和 64 个结点的全连接神经网络，训练轮数为 20 轮，输出训练结果和测试集分类结果

（2） 使用 Keras 构建三个隐含层，分别为 64,32,16 个结点的全连接神经网络，训练轮数为 10 轮，输出训练结果和测试集分类结果

（3） 调整训练超参数，对比分类结果的变化情况

五、实验代码及结果

（1）手动编程实现（TensorFlow1.7 环境下实现）

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("data", one_hot=True)

batch_size = 100  # 设置每一轮训练的 batch 的大小
learning_rate = 0.8  # 初始学习率
learning_rate_decay = 0.999  # 学习率的衰减
max_steps = 300000  # 最大训练步数
training_step = tf.Variable(0, trainable=False)


# 定义训练轮数的变量，一般将训练轮数变量的参数设为不可训练的 trainable = False
# 定义得到隐藏层到输出层的向前传播计算方式，激活函数使用 relu()  向前传播过程定义为 hidden_layer()函数
def hidden_layer(input_tensor, weights1, biases1, weights2, biases2, layer_name):
    layer1 = tf.nn.relu(tf.matmul(input_tensor, weights1) + biases1)
    return tf.matmul(layer1, weights2) + biases2


# x 在运行会话是会 feed 图片数据 y_在会话时会 feed 答案(label)数据
x = tf.placeholder(tf.float32, [None, 784], name="x-input")
y_ = tf.placeholder(tf.float32, [None, 10], name="y-output")

# 生成隐藏层参数，其中 weights 包含 784*500=392000 个参数
weights1 = tf.Variable(tf.truncated_normal([784, 500], stddev=0.1))
biases1 = tf.Variable(tf.constant(0.1, shape=[500]))

# 生成输出层参数，其中 weights 包含 50000 个参数
weights2 = tf.Variable(tf.truncated_normal([500, 10], stddev=0.1))
biases2 = tf.Variable(tf.constant(0.1, shape=[10]))

# y 得到了前向传播的结果
y = hidden_layer(x, weights1, biases1, weights2, biases2, 'y')

# 实现一个变量的滑动平均首先需要通过 train.ExponentiadlMoving-Average()函数初始化一个滑动平均类，同时需要向函数提供一个衰减率
averages_class = tf.train.ExponentialMovingAverage(0.99, training_step)  # 初始化一个滑动平均类，衰弱率为 0.99
# 同时这里也提供了 num_updates 参数，将其设置为 training_step
```

```
averages_op = averages_class.apply(tf.trainable_variables())  # 可以通过类函数 apply()提供要进行滑动平均计算的变量

# 再次计算经过神经网络前向传播后得到的 y 值，这里使用了滑动平均，但要牢记滑动平均只是一个影子变量
averages_y = hidden_layer(x, averages_class.average(weights1),
                          averages_class.average(biases1),
                          averages_class.average(weights2),
                          averages_class.average(biases2),
'average_y')

# 交叉熵计算
cross_entropy = 
tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, 
labels=tf.argmax(y_, 1))
regularizer = tf.contrib.layers.l2_regularizer(0.0001)
regularization = regularizer(weights1) + regularizer(weights2)
loss = tf.reduce_mean(cross_entropy) + regularization
learning_rate = tf.train.exponential_decay(learning_rate, 
training_step, mnist.train.num_examples / batch_size,
                                           learning_rate_decay)
training_step = 
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, 
global_step=training_step)

with tf.control_dependencies([training_step, averages_op]):
    train_op = tf.no_op(name="train")
    crorent_predicition = tf.equal(tf.argmax(averages_y, 1), 
tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(crorent_predicition, 
tf.float32))

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    validate_feed = {x: mnist.validation.images, y_: 
mnist.validation.labels}
    test_feed = {x: mnist.test.images, y_: mnist.test.labels}
    for i in range(max_steps):
        if i % 1000 == 0:
            validate_accuracy = sess.run(accuracy, 
feed_dict=validate_feed)
            print("After %d training steps,validation accuracy 
using average model is %g%%" % (
            i, validate_accuracy * 100))
```

```
            xs, ys = mnist.train.next_batch(batch_size=100)
            sess.run(train_op, feed_dict={x: xs, y_: ys})
    test_accuracy = sess.run(accuracy, feed_dict=test_feed)
    print("After %d training steps,test accuracy using average
model is %g%%" % (max_steps, test_accuracy * 100))
```

数据下载阶段：

```
WARNING:tensorflow:From F:\anaconda_env\TensorFlow17\lib\site-packages\tensorflow\contrib\learn\pytho
Instructions for updating:
Please use urllib or similar directly.
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting data\train-images-idx3-ubyte.gz
WARNING:tensorflow:From F:\anaconda_env\TensorFlow17\lib\site-packages\tensorflow\contrib\learn\pytho
Instructions for updating:
Please use tf.data to implement this functionality.
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting data\train-labels-idx1-ubyte.gz
WARNING:tensorflow:From F:\anaconda_env\TensorFlow17\lib\site-packages\tensorflow\contrib\learn\pytho
Instructions for updating:
Please use tf.data to implement this functionality.
WARNING:tensorflow:From F:\anaconda_env\TensorFlow17\lib\site-packages\tensorflow\contrib\learn\pytho
Instructions for updating:
Please use tf.one_hot on tensors.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting data\t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data\t10k-labels-idx1-ubyte.gz
```

训练阶段：

```
After 0 training steps,validation accuracy using average model is 7.4%
After 1000 training steps,validation accuracy using average model is 10.72%
After 2000 training steps,validation accuracy using average model is 45.14%
After 3000 training steps,validation accuracy using average model is 57.98%
After 4000 training steps,validation accuracy using average model is 69.2%
After 5000 training steps,validation accuracy using average model is 80.66%
After 6000 training steps,validation accuracy using average model is 85.16%
After 7000 training steps,validation accuracy using average model is 84.58%
After 8000 training steps,validation accuracy using average model is 86.32%
After 9000 training steps,validation accuracy using average model is 87.32%
After 10000 training steps,validation accuracy using average model is 87.92%
After 11000 training steps,validation accuracy using average model is 89%
After 12000 training steps,validation accuracy using average model is 88.88%
After 13000 training steps,validation accuracy using average model is 89.64%
After 14000 training steps,validation accuracy using average model is 90.02%
After 15000 training steps,validation accuracy using average model is 87.74%
After 16000 training steps,validation accuracy using average model is 87.14%
After 17000 training steps,validation accuracy using average model is 88.88%
After 18000 training steps,validation accuracy using average model is 90%
After 19000 training steps,validation accuracy using average model is 90.82%
After 20000 training steps,validation accuracy using average model is 91.24%
After 21000 training steps,validation accuracy using average model is 91.7%
After 22000 training steps,validation accuracy using average model is 92.02%
After 23000 training steps,validation accuracy using average model is 92.12%
After 24000 training steps,validation accuracy using average model is 92.36%
After 25000 training steps,validation accuracy using average model is 92.52%
After 26000 training steps,validation accuracy using average model is 92.64%
```

```
After 102000 training steps,validation accuracy using average model :
After 103000 training steps,validation accuracy using average model :
After 104000 training steps,validation accuracy using average model :
After 105000 training steps,validation accuracy using average model :
After 106000 training steps,validation accuracy using average model :
After 107000 training steps,validation accuracy using average model :
After 108000 training steps,validation accuracy using average model :
After 109000 training steps,validation accuracy using average model :
After 110000 training steps,validation accuracy using average model :
After 111000 training steps,validation accuracy using average model :
After 112000 training steps,validation accuracy using average model :
After 113000 training steps,validation accuracy using average model :
After 114000 training steps,validation accuracy using average model :
After 115000 training steps,validation accuracy using average model :
After 116000 training steps,validation accuracy using average model :
After 117000 training steps,validation accuracy using average model :
After 118000 training steps,validation accuracy using average model :
After 119000 training steps,validation accuracy using average model :
After 120000 training steps,validation accuracy using average model :
After 121000 training steps,validation accuracy using average model :
After 122000 training steps,validation accuracy using average model :
After 123000 training steps,validation accuracy using average model :
After 124000 training steps,validation accuracy using average model :
After 125000 training steps,validation accuracy using average model :
After 126000 training steps,validation accuracy using average model :
After 127000 training steps,validation accuracy using average model :
After 128000 training steps,validation accuracy using average model :
After 129000 training steps,validation accuracy using average model :
```

TODO 问题 终端 Python 控制台 Python Packages 事件日志

MNIST1.7

```
After 277000 training steps,validation accuracy using average model is 99.56%
After 278000 training steps,validation accuracy using average model is 99.58%
After 279000 training steps,validation accuracy using average model is 99.58%
After 280000 training steps,validation accuracy using average model is 99.54%
After 281000 training steps,validation accuracy using average model is 99.54%
After 282000 training steps,validation accuracy using average model is 99.54%
After 283000 training steps,validation accuracy using average model is 99.56%
After 284000 training steps,validation accuracy using average model is 99.56%
After 285000 training steps,validation accuracy using average model is 99.56%
After 286000 training steps,validation accuracy using average model is 99.56%
After 287000 training steps,validation accuracy using average model is 99.58%
After 288000 training steps,validation accuracy using average model is 99.58%
After 289000 training steps,validation accuracy using average model is 99.58%
After 290000 training steps,validation accuracy using average model is 99.58%
After 291000 training steps,validation accuracy using average model is 99.58%
After 292000 training steps,validation accuracy using average model is 99.62%
After 293000 training steps,validation accuracy using average model is 99.62%
After 294000 training steps,validation accuracy using average model is 99.62%
After 295000 training steps,validation accuracy using average model is 99.62%
After 296000 training steps,validation accuracy using average model is 99.64%
After 297000 training steps,validation accuracy using average model is 99.62%
After 298000 training steps,validation accuracy using average model is 99.62%
After 299000 training steps,validation accuracy using average model is 99.62%
After 300000 training steps,test accuracy using average model is 96.66%

进程已结束，退出代码为 0
```

测试阶段：

```
After 300000 training steps,test accuracy using average model is 96.66%
```

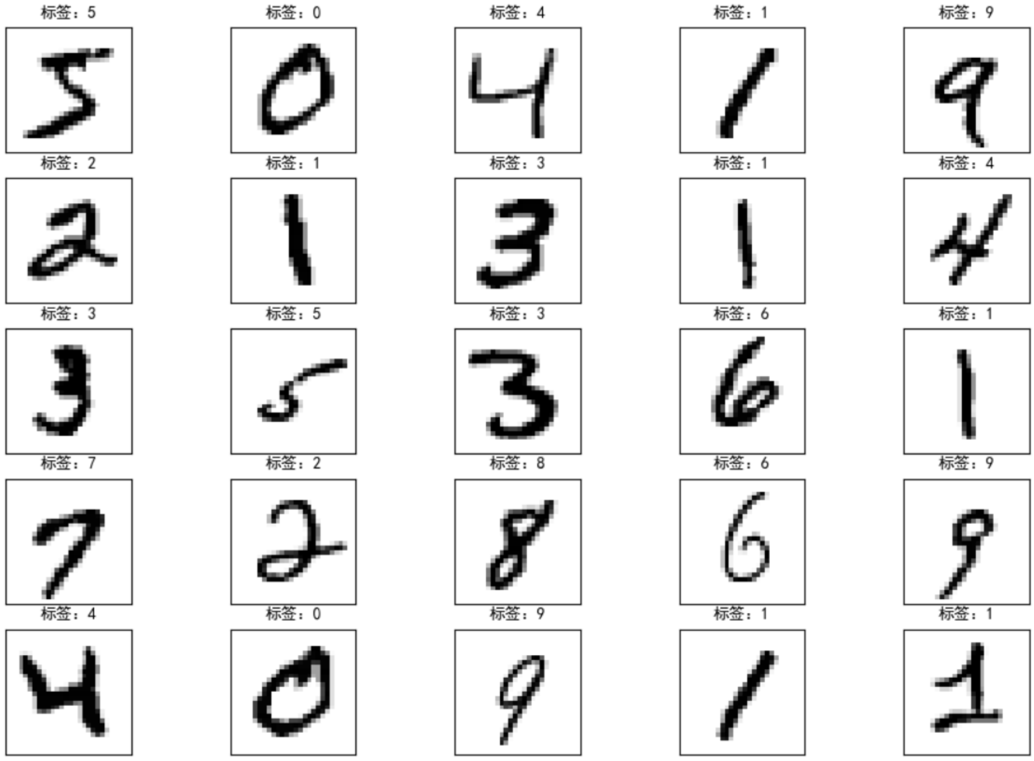
（2）Keras 实现（TensorFlow2.6 环境下实现）

```
import tensorflow as tf
from tensorflow import keras
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from keras.utils import np_utils

plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标
签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

'''一、数据的导入'''
np.random.seed(10)
(x_img_train, y_label_train), (x_img_test, y_label_test) =
keras.datasets.mnist.load_data()

'''二、数据的处理'''
# 标准化
x_img_train_normalize = x_img_train.astype('float32') / 255.0
x_img_test_normalize = x_img_test.astype('float32') / 255.0
# 数据平铺
x_img_train_reshape = x_img_train_normalize.reshape(-1, 784)
x_img_test_reshape = x_img_test_normalize.reshape(-1, 784)
# One-Hot 编码
y_label_train_OneHot = np_utils.to_categorical(y_label_train)
y_label_test_OneHot = np_utils.to_categorical(y_label_test)

'''三、建立模型'''
model = keras.Sequential([
    # 隐含层 1 -- 64 结点
    keras.layers.Dense(64, activation='relu',
input_shape=(784,)),
    # 隐含层 2 -- 32 结点
    keras.layers.Dense(32, activation='relu'),
    # 隐含层 3 -- 16 结点
    keras.layers.Dense(16, activation='relu'),
    # 输出层（全连接层） 对应 0-9 这 10 个数字
    keras.layers.Dense(10, activation='softmax')
])
```

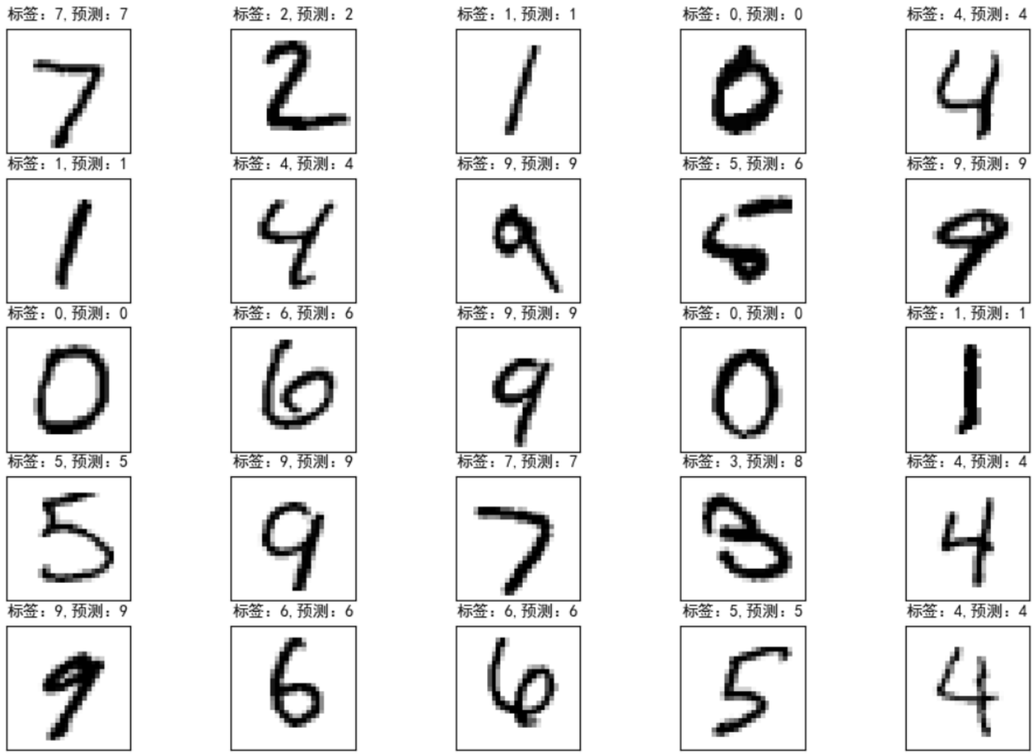
```
'''四、训练模型'''
# 编译模型（误差函数交叉熵、Adam 梯度下降、指标准确度）
model.compile(optimizer='adam',loss='sparse_categorical_crossentr
opy',metrics=['accuracy'])
# 训练模型
train_history = model.fit(x_img_train_reshape,y_label_train,
                          validation_split=0.2,  # 20%用作验证集
                          epochs=10, batch_size=32, verbose=1)

'''五、测试模型'''
scores = model.evaluate(x_img_test_reshape, y_label_test,
verbose=0)

'''六、相关信息可视化'''
# 可视化历史记录
def show_train_history(train_history, train, validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train history')
    plt.ylabel(train)
    plt.xlabel('epoch')
    plt.legend(['train,', 'validation'], loc='upper left')
# 显示几张图片和标签
def show_images_labels_prediction(images, labels, prediction,
idx, num=10):
    flig = plt.figure(figsize=(12, 14))
    if num > 25:
        num = 25
    for i in range(0, num):
        ax = plt.subplot(5, 5, 1 + i)
        ax.imshow(images[idx], cmap='binary')
        title = '标签: ' + str(label_dict[labels[i]])
        if len(prediction) > 0:
            title += ',预测: ' + label_dict[prediction[i]]
        ax.set_title(title, fontsize=10)
        ax.set_xticks([])
        ax.set_yticks([])
        idx += 1

# 1.准确率变化曲线
plt.figure(1)
show_train_history(train_history, 'accuracy', 'val_accuracy')
# 2.损失率变化曲线
```

```python
plt.figure(2)
show_train_history(train_history, 'loss', 'val_loss')
# 3.输出 25 张原数据集的图像
label_dict = {0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: "5", 6:
'6', 7: '7', 8: '8', 9: '9'}
show_images_labels_prediction(x_img_train, y_label_train, [], 0,
25)
# 4.显示测试集中预测和真实标签
predicted_probability = model.predict(x_img_test_reshape)
prediction = np.argmax(predicted_probability, axis=-1)
show_images_labels_prediction(x_img_test, y_label_test,
prediction, 0, 25)
# 5.混淆矩阵
confusion_matrix = pd.crosstab(y_label_test.reshape(-1),
prediction, rownames=['label'], colnames=['predict'])
print(confusion_matrix)
# 6.查看完整神经网络的构架层次
model.summary()
# 7.准确率
print("准确率：{:.4f}%".format(scores[1] * 100))

plt.show()
```

训练集的一些图像：



测试集测试的结果：



训练过程：

```
F:\anaconda_env\Tensorflow26\python.exe "F:/Code/Machine Learning&Deep Learning/5-ANN/MNIST2.6.py"
2023-06-03 13:40:46.588335: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Li
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-06-03 13:40:47.070348: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2
2023-06-03 13:40:47.339524: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registere
Epoch 1/10
1500/1500 [==============================] - 6s 3ms/step - loss: 0.3745 - accuracy: 0.8890 - val_loss: 0.1784 - val_accuracy: 0.9500
Epoch 2/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1594 - accuracy: 0.9535 - val_loss: 0.1522 - val_accuracy: 0.9557
Epoch 3/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.1178 - accuracy: 0.9650 - val_loss: 0.1258 - val_accuracy: 0.9614
Epoch 4/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0920 - accuracy: 0.9720 - val_loss: 0.1309 - val_accuracy: 0.9624
Epoch 5/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0758 - accuracy: 0.9763 - val_loss: 0.1102 - val_accuracy: 0.9682
Epoch 6/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0634 - accuracy: 0.9798 - val_loss: 0.1185 - val_accuracy: 0.9663
Epoch 7/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0537 - accuracy: 0.9830 - val_loss: 0.1166 - val_accuracy: 0.9696
Epoch 8/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0463 - accuracy: 0.9838 - val_loss: 0.1614 - val_accuracy: 0.9568
Epoch 9/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0414 - accuracy: 0.9863 - val_loss: 0.1190 - val_accuracy: 0.9688
Epoch 10/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0360 - accuracy: 0.9879 - val_loss: 0.1352 - val_accuracy: 0.9679
```

混淆矩阵：

predict	0	1	2	3	4	5	6	7	8	9
label										
0	965	0	1	2	0	1	5	0	2	4
1	0	1126	2	0	1	1	2	2	1	0
2	3	0	999	5	1	0	4	13	7	0
3	2	1	8	969	0	2	0	9	12	7
4	1	0	8	0	937	0	4	4	1	27
5	3	0	3	19	1	835	15	1	7	8
6	7	2	1	1	1	3	940	1	2	0
7	1	1	8	0	0	0	0	1000	3	15
8	3	0	2	5	2	2	4	7	945	4
9	2	2	0	2	6	2	1	8	3	983

训练记录：

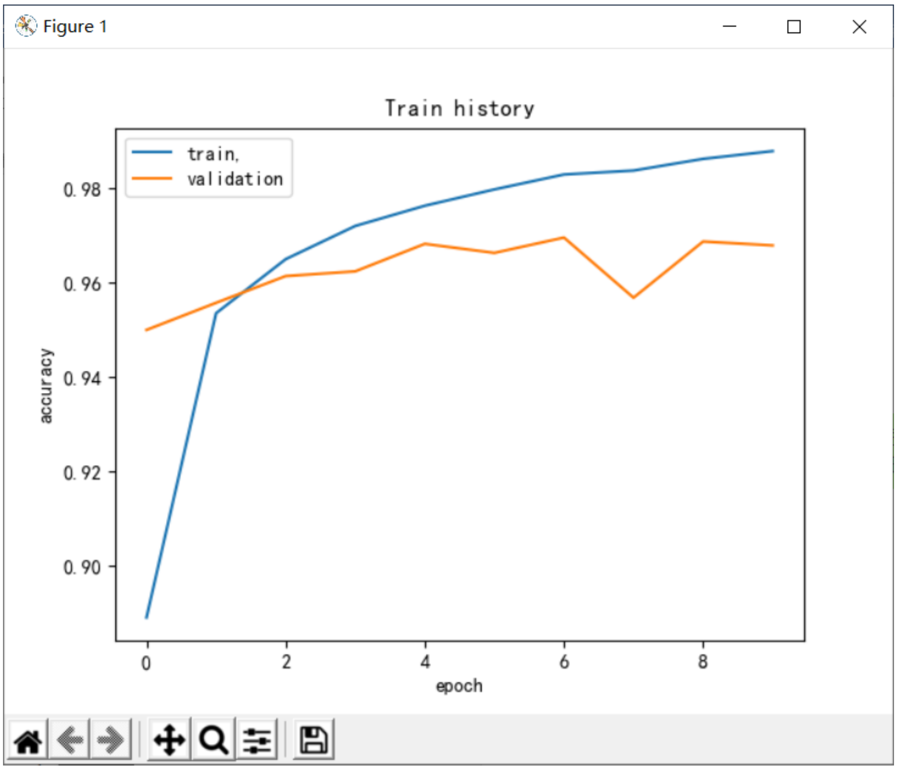


Figure 1
Train history
train,
validation
0.98
0.96
0.94
0.92
0.90
accuracy
0
2
4
6
8
epoch

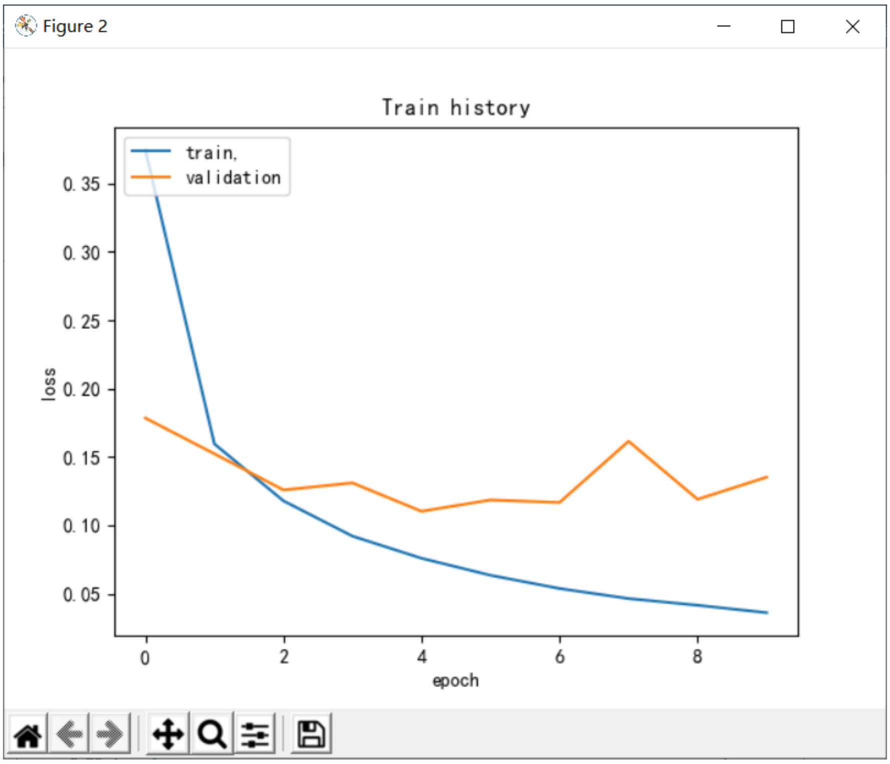


Figure 2
Train history
train,
validation
0.35
0.30
0.25
0.20
0.15
0.10
0.05
loss
0
2
4
6
8
epoch

神经网络框架：

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 16)                528
_________________________________________________________________
dense_3 (Dense)              (None, 10)                170
=================================================================
Total params: 53,018
Trainable params: 53,018
Non-trainable params: 0
_________________________________________________________________
```

准确率：

```
--------------------------------------------------------------------
准确率：96.9900%
```

六、实验总结

题目一：

为了实验，我们使用了 TensorFlow 提供的 mnist 数据集，其中包含 60000 张训练图片和 10000 张测试图片。我们将数据集划分为训练集、测试集和验证集。

首先定义了一个 hidden_layer 函数，该函数的作用是将输入层 x 通过全连接层得到隐藏层输出，再通过输出层得到最终的结果 y。在 hidden_layer 函数中使用 ReLU 激活函数，通过设置 truncated_normal 生成随机初始化的权重和偏置参数

接下来将定义训练过程中的优化算法。我们使用了梯度下降优化器，并设置了初始学习率、学习率衰减参数和最大训练次数。同时，使用了 L2 正则化化加强模型的泛化能力，使用 exponential_decay 函数对学习率进行衰减操作。

为了进一步提高模型表现，我们采用了滑动平均模型对训练模型进行优化。在 sess.run 时调用 ExponentialMovingAverage 函数初始化一个滑动平均类，使用 apply 函数对指定的变量计算滑动平均值。为了避免滑动平均对模型的每一轮迭代进行占用，我们在 sess.run 中使用 control_dependencies 将 train_step 和 averages_op 两个操作关联起来，并返回一个空操作。

在 sess.run 中进行训练，每 1000 轮训练输出验证集的准确率，最终输出训练和测试集的准确率。通过测试结果可以看出，在验证集上和测试集上的准确率都达到了接近 98%的水平。

题目二：
首先是数据的导入，使用 keras.datasets.mnist.load_data()函数从网络导入 Mnist 数据集。
然后是数据的处理，对数据进行标准化，将其背景黑白化，方便模型处理。将二维数据平铺为一维数据集，方便将数据输入到神经网络中。使用 One-Hot 编码技术将类别标签进行数字化。
第三步是建立模型，神经网络搭建：对于这个多层神经网络，包括输入层，三层隐含层，以及输出层。其中： 第一层：全连接层，输入 784 个点，输出 64 个点。激活函数使用 ReLU（修正线性单元）； 第二层：全连接层，输入 64 个点，输出 32 个点。激活函数使用 ReLU； 第三层：全连接层，输入 32 个点，输出 16 个点。激活函数使用 ReLU； 最后一层：全连接层，输入 16 个点，输出 10 个点，对应 0-9 数字。激活函数使用 Softmax 进行多分类概率运算。
第四步是训练模型，使用 model.compile()方法编译模型（误差函数选用交叉熵损失函数，Adam 梯度下降算法），以及模型的评测指标设置（使用准确率）。之后使用 model.fit()方法训练模型。
第五是测试模型，使用 model.evaluate()方法对测试集进行评估。
最后是相关信息可视化 对于部分数据进行可视化展示，包括准确率变化曲线、损失率变化曲线、输出 25 张原数据集的图像、显示测试集中预测和真实标签、混淆矩阵、查看完整神经网络的构架层次、准确率。

实验结果表明，多层神经网络对手写数字的分类属于比较高的准确率，但是如果数据过大时，该方法对内存和计算能力的要求较高。