**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**

1. **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；

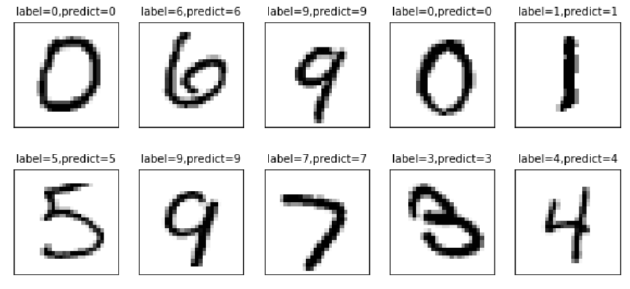
3. 掌握Keras构建全连接神经网络的方法。

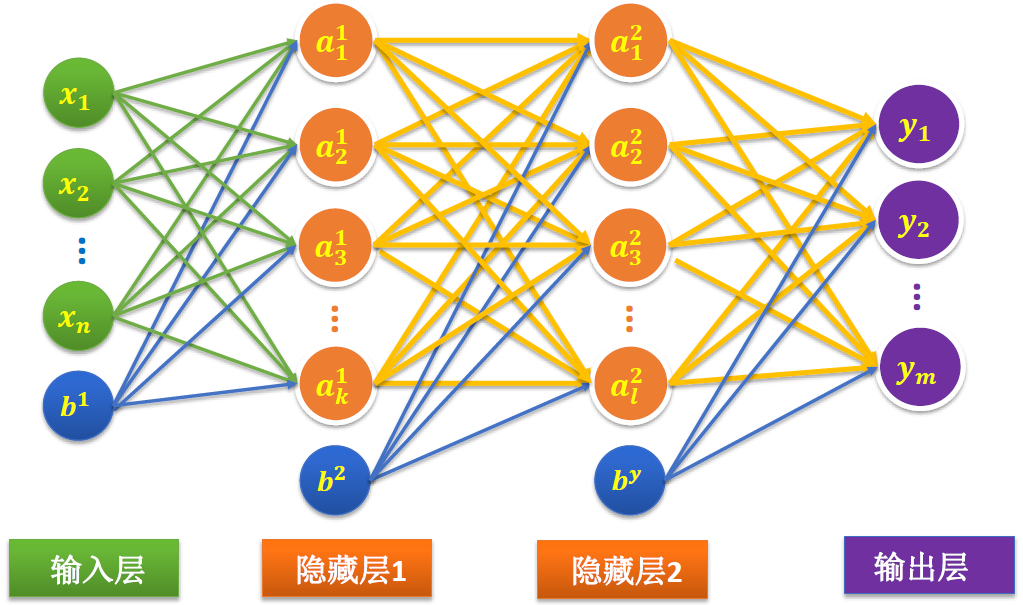
1. **实验条件：**

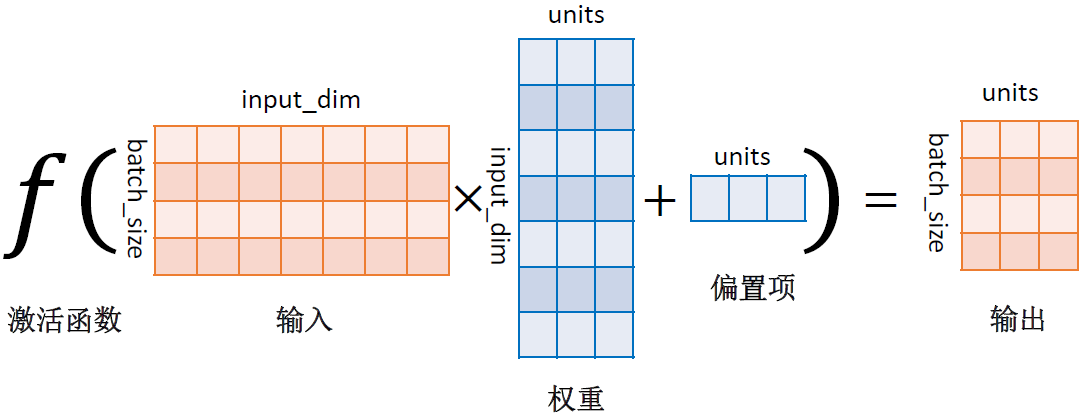
1. PC微机一台和Python+TensorFlow环境。

1. **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。







1. **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'

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))

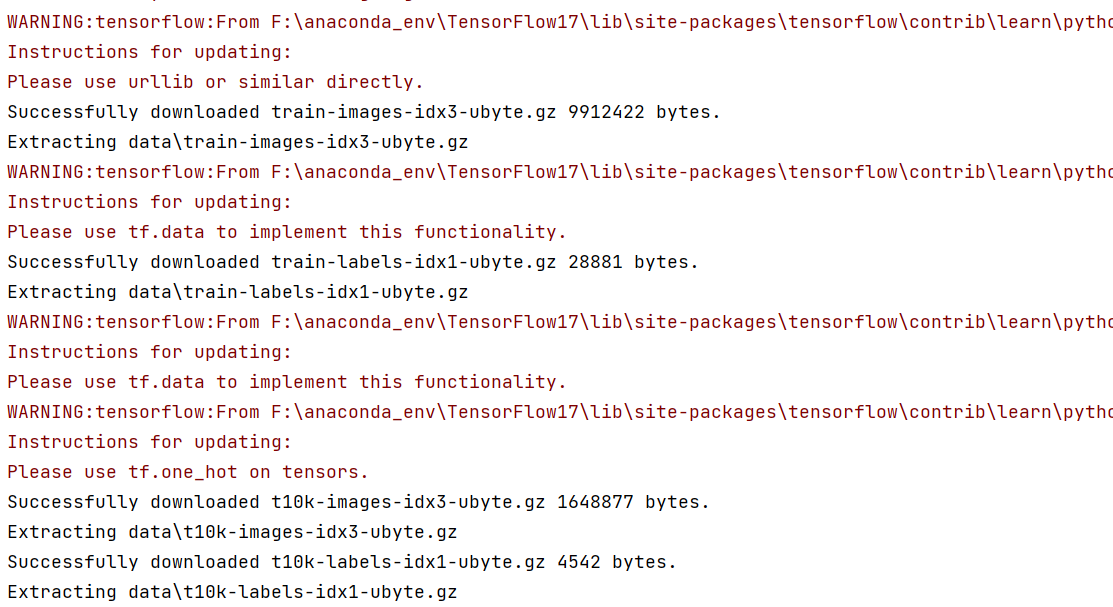
            xs, ys = mnist.train.next\_batch(batch\_size=100)

            sess.run(train\_op, feed\_dict={x: xs, y\_: ys})

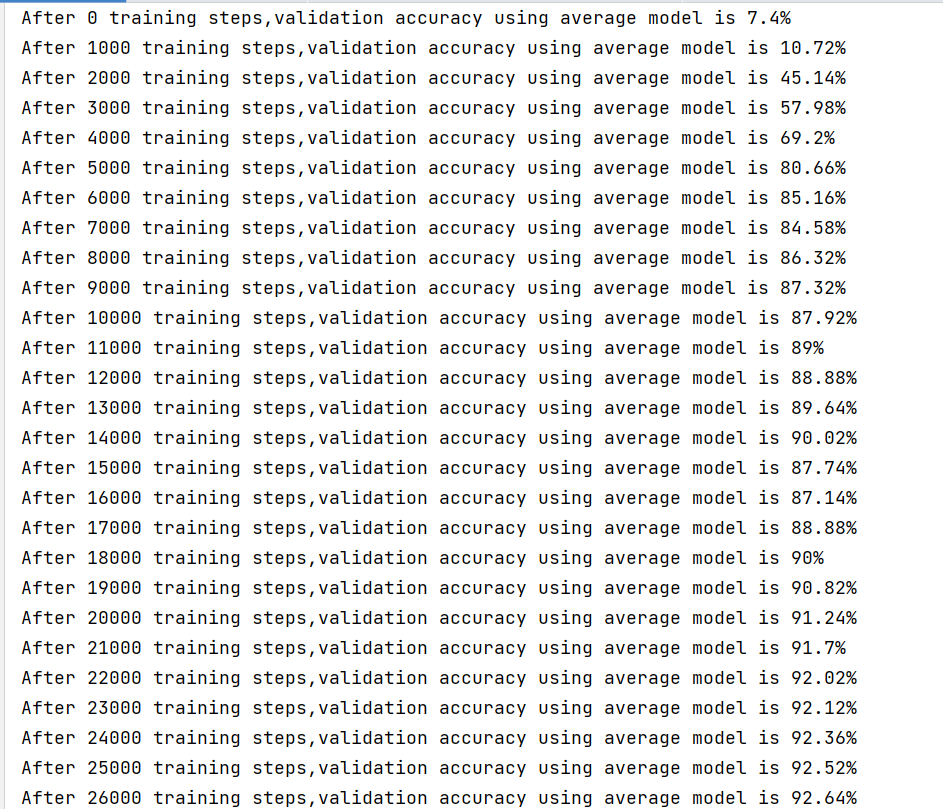
    test\_accuracy = sess.run(accuracy, feed\_dict=test\_feed)

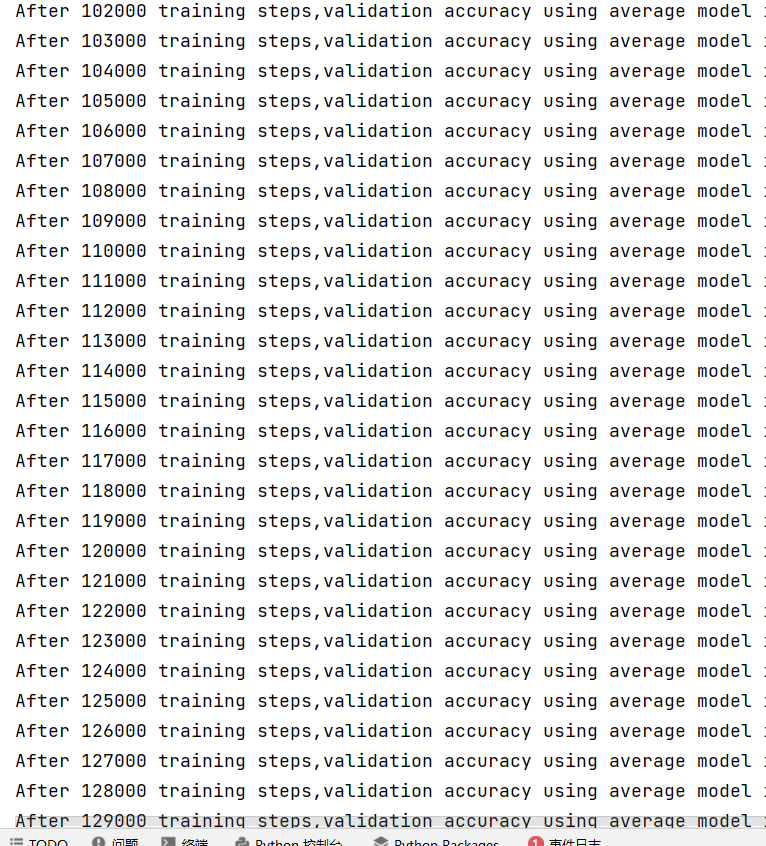
    print("After %d training steps,test accuracy using average model is %g%%" % (max\_steps, test\_accuracy \* 100))

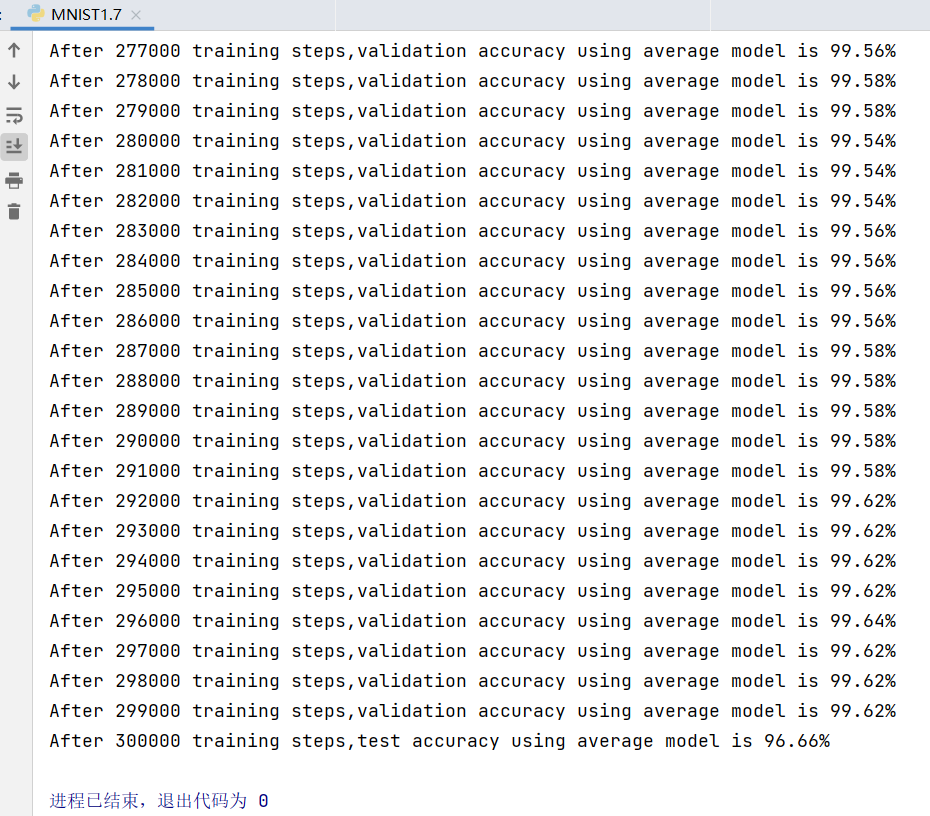
数据下载阶段：



训练阶段：







测试阶段：



**（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\_crossentropy',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

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

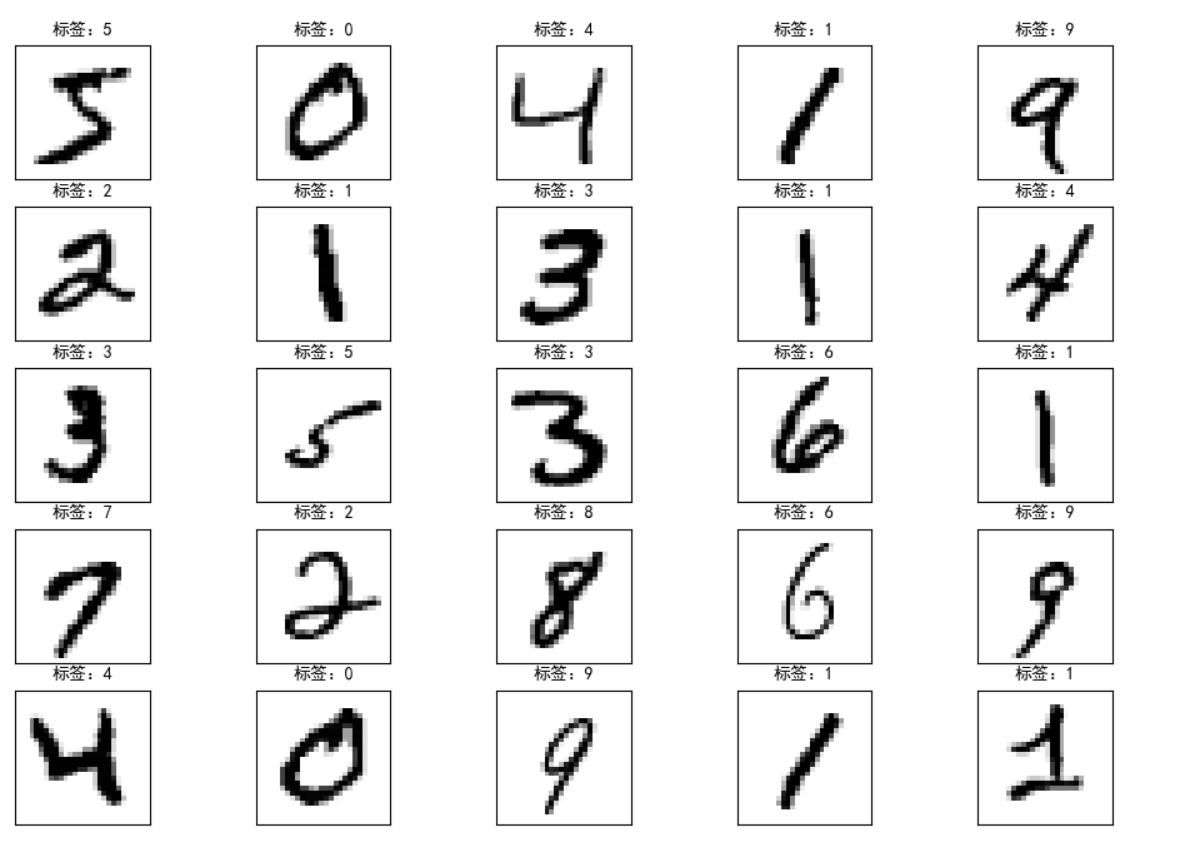
model.summary()

# 7.准确率

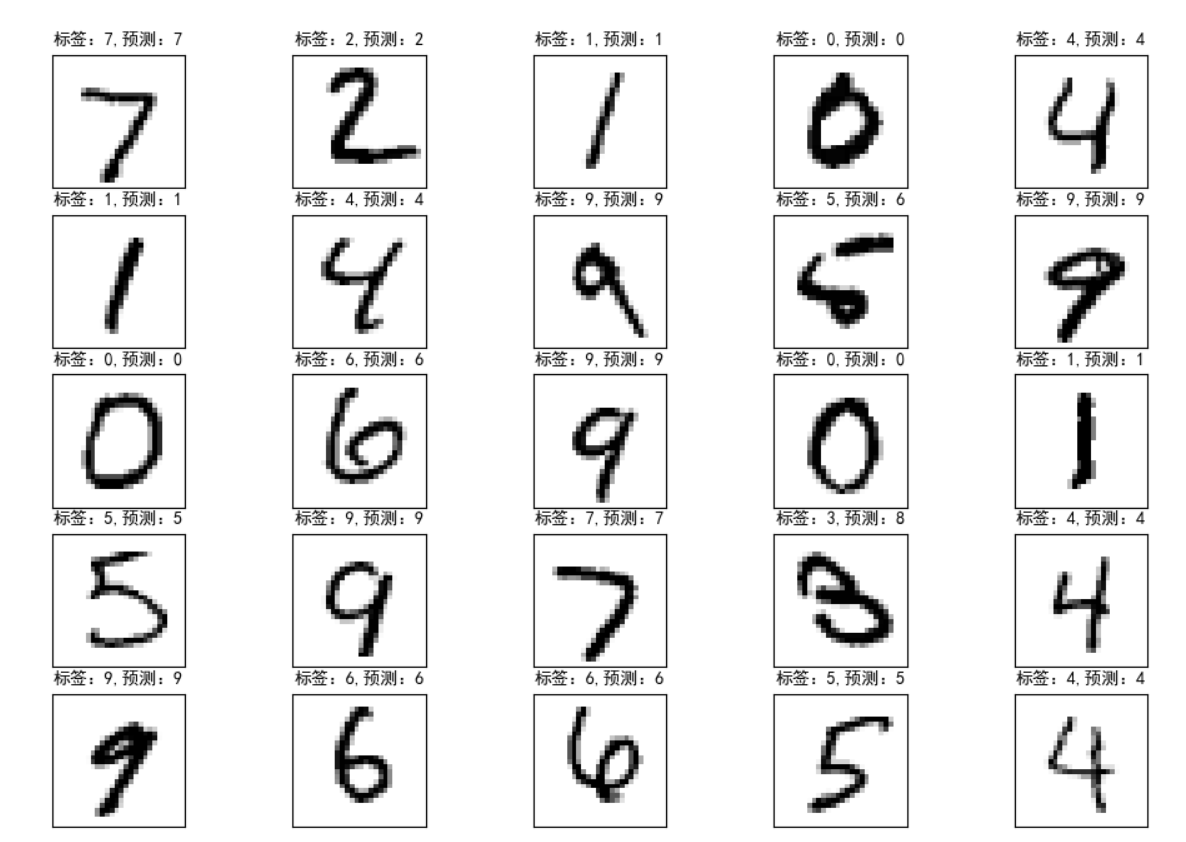
print("准确率：{:.4f}%".format(scores[1] \* 100))

plt.show()

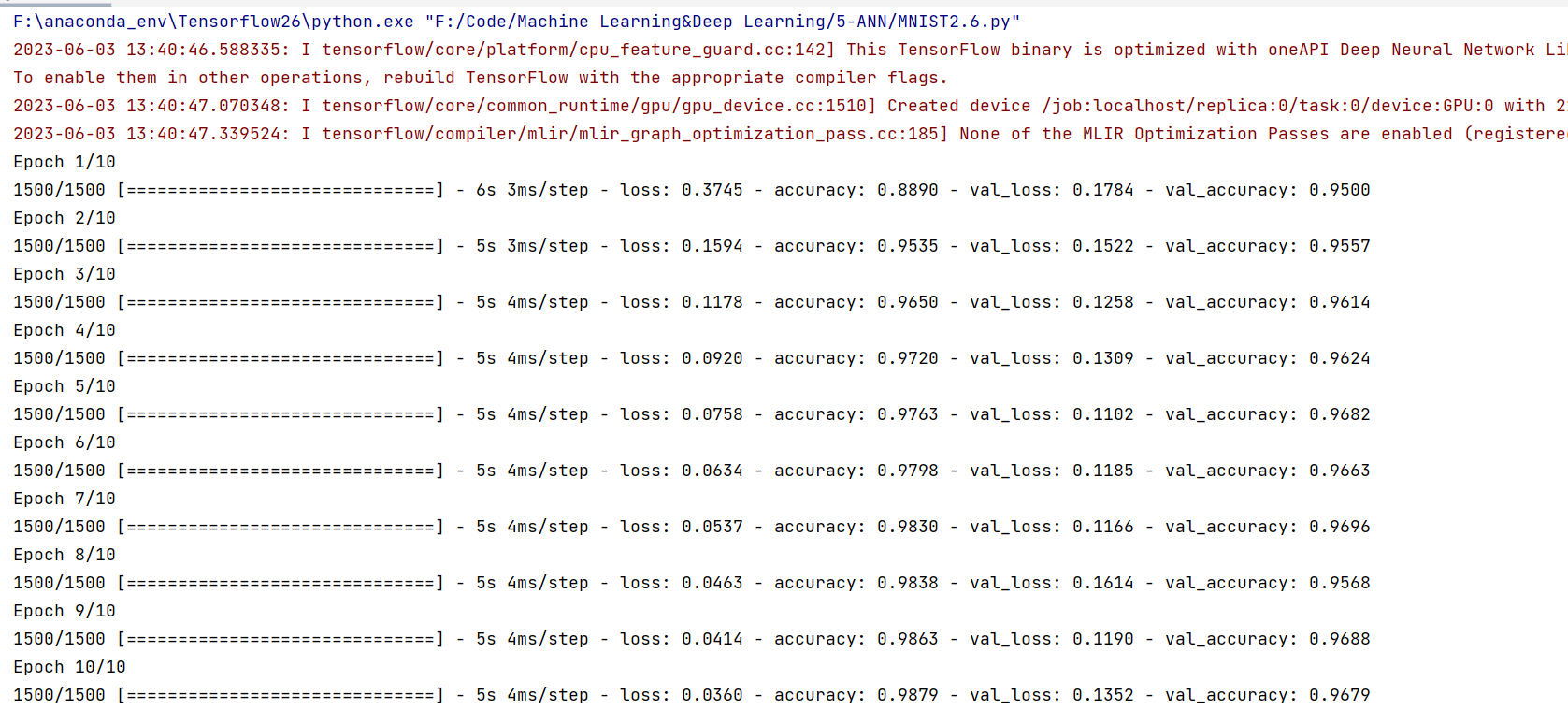
训练集的一些图像：



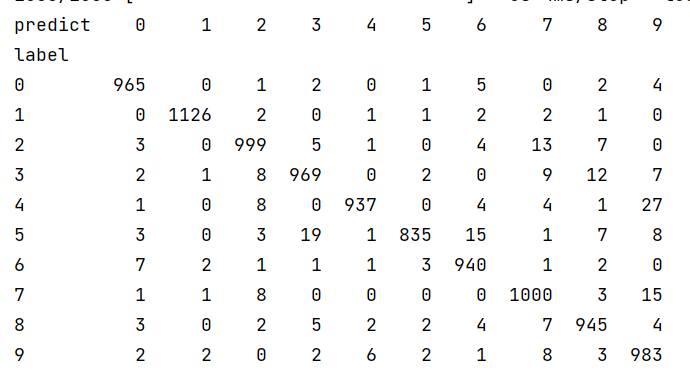
测试集测试的结果：



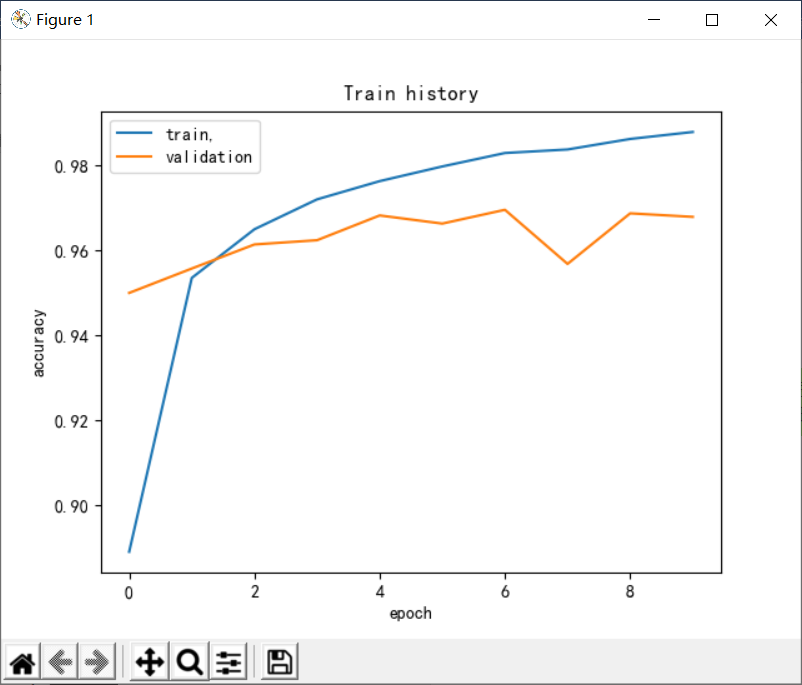
训练过程：

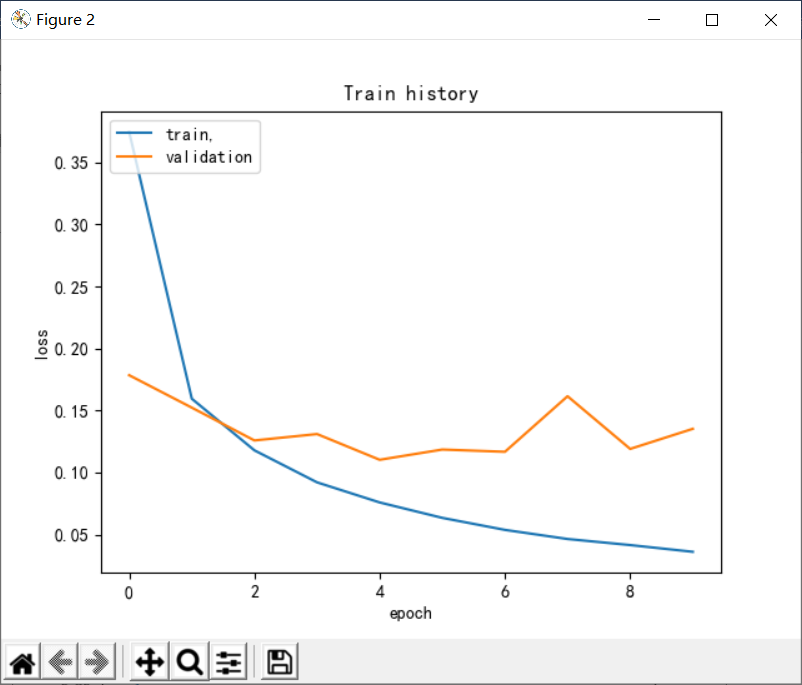


混淆矩阵：

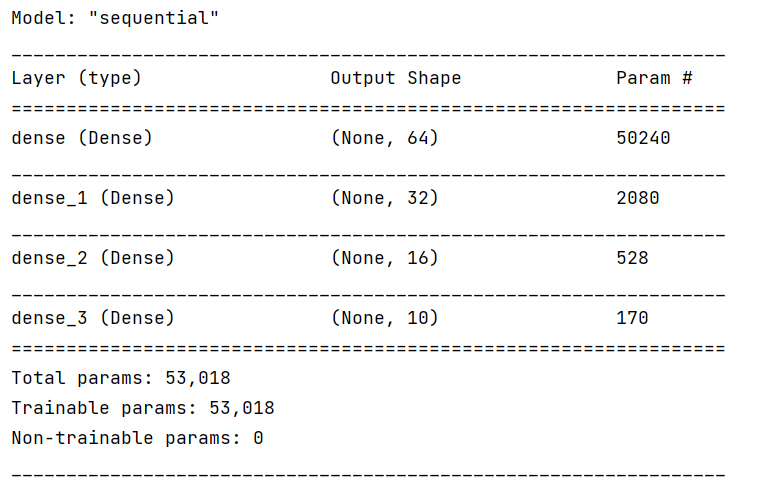


训练记录：





神经网络框架：



准确率：



**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。