Tensorflow 笔记：第五讲

MNIST 数据集输出手写数字识别准确率

本节目标：搭建神经网络，在 mnist 数据集上训练模型，输出手写数字识别准确率。

5.1

√mnist 数据集：包含 7 万张黑底白字手写数字图片，其中 55000 张为训练集，5000 张为验证集，10000 张为测试集。每张图片大小为 28*28 像素，图片中纯黑色像素值为 0，纯白色像素值为 1。数据集的标签是长度为 10 的一维数组，数组中每个元素索引号表示对应数字出现的概率。

在将 mnist 数据集作为输入喂入神经网络时，需先将数据集中每张图片变为长度 784 一维数组，将该数组作为神经网络输入特征喂入神经网络。

例如：

一张数字手写体图片变成长度为 784 的一维数组[0.0.0.0.0.231 0.235 0.459 ……0.219 0.0.0.0.]输入神经网络。该图片对应的标签为[0.0.0.0.0.0.1.0.0.0]，标签中索引号为 6 的元素为 1，表示是数字 6 出现的概率为 100%，则该图片对应的识别结果是 6。

√使用 input_data 模块中的 read_data_sets()函数加载 mnist 数据集：

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('./data/',one_hot=True)
```

在 read_data_sets()函数中有两个参数，第一个参数表示数据集存放路径，第二个参数表示数据集的存取形式。当第二个参数为 Ture 时，表示以独热码形式存取数据集。read_data_sets()函数运行时，会检查指定路径内是否已经有数据集，若指定路径中没有数据集，则自动下载，并将 mnist 数据集分为训练集 train、验证集 validation 和测试集 test 存放。在终端显示如下内容：

```
Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz
```

Extracting ./data/t10k-images-idx3-ubyte.gz

Extracting ./data/ t10k-labels-idx1-ubyte.gz

√返回 mnist 数据集中训练集 train、验证集 validation 和测试集 test 样本数

在 Tensorflow 中用以下函数返回子集样本数：

①返回训练集 train 样本数

print “train data size:”,mnist.train.mun_examples

输出结果：train data size:55000

②返回验证集 validation 样本数

print “validation data size:”,mnist.validation.mun_examples

输出结果：validation data size:5000

③返回测试集 test 样本数

print “test data size:”,mnist.test.mun_examples

输出结果：test data size:10000

√使用 train.labels 函数返回 mnist 数据集标签

例如：

在 mnist 数据集中，若想要查看训练集中第 0 张图片的标签，则使用如下函数

mnist.train.labels[0]

输出结果：array([0.,0.,0.,0.,0.,0.,1.,0.,0.,0])

√使用 train.images 函数返回 mnist 数据集图片像素值

例如：

在 mnist 数据集中，若想要查看训练集中第 0 张图片像素值，则使用如下函数

mnist.train.images[0]

输出结果：array([0. ,0. ,0. ,
0. ,0. ,0. ,
0. ,0. ,0. ,
… … …])

√使用 mnist.train.next_batch()函数将数据输入神经网络

例如：

BATCH_SIZE = 200

xs,ys = mnist.train.next_batch(BATCH_SIZE)

print “xs shape:”,xs.shape

print “ys shape:”,ys.shape

输出结果：xs.shape(200,784)

输出结果：ys.shape(200,10)

其中，mnist.train.next_batch()函数包含一个参数 BATCH_SIZE，表示随机从训练集中抽取 BATCH_SIZE 个样本输入神经网络，并将样本的像素值和标签分别赋给 xs 和 ys。在本例中，BATCH_SIZE 设置为 200，表示一次将 200 个样本的像素值和标签分别赋值给 xs 和 ys，故 xs 的形状为(200,784)，对应的 ys 的形状为(200,10)。

√实现“Mnist 数据集手写数字识别”的常用函数：

①tf.get_collection(“”)函数表示从 collection 集合中取出全部变量生成一个列表。

②tf.add( )函数表示将参数列表中对应元素相加。

例如：

x=tf.constant([[1,2],[1,2]])

y=tf.constant([[1,1],[1,2]])

z=tf.add(x,y)

print z

输出结果：[[2,3],[2,4]]

③tf.cast(x,dtype)函数表示将参数 x 转换为指定数据类型。

例如：

A = tf.convert_to_tensor(np.array([[1,1,2,4], [3,4,8,5]]))

print A.dtype

b = tf.cast(A, tf.float32)

print b.dtype

结果输出：

<dtype: 'int64'>

<dtype: 'float32'>

从输出结果看出，将矩阵 A 由整数型变为 32 位浮点型。

④tf.equal( )函数表示对比两个矩阵或者向量的元素。若对应元素相等，则返回 True；若对应元素不相等，则返回 False。

例如：

```
A = [[1,3,4,5,6]]
B = [[1,3,4,3,2]]
with tf.Session( ) as sess:
    print(sess.run(tf.equal(A, B)))
```

输出结果：[[True True True False False]]

在矩阵 A 和 B 中，第 1、2、3 个元素相等，第 4、5 个元素不等，故输出结果中，第 1、2、3 个元素取值为 True，第 4、5 个元素取值为 False。

⑤tf.reduce_mean(x,axis)函数表示求取矩阵或张量指定维度的平均值。若不指定第二个参数，则在所有元素中取平均值；若指定第二个参数为 0，则在第一维元素上取平均值，即每一列求平均值；若指定第二个参数为 1，则在第二维元素上取平均值，即每一行求平均值。

例如：

```
x = [[1., 1.]
     [2., 2.]]
print(tf.reduce_mean(x))
```

输出结果：1.5

```
print(tf.reduce_mean(x, 0))
```

输出结果：[1.5, 1.5]

```
print(tf.reduce_mean(x, 1))
```

输出结果：[1., 1.]

⑥tf.argmax(x,axis)函数表示返回指定维度 axis 下，参数 x 中最大值索引号。

例如：

在 tf.argmax([1,0,0],1)函数中，axis 为 1，参数 x 为[1,0,0]，表示在参数 x 的第一个维度取最大值对应的索引号，故返回 0。

⑦os.path.join()函数表示把参数字符串按照路径命名规则拼接。

例如：

```
import os
os.path.join('/hello/','good/boy/','doiido')
```

输出结果：'/hello/good/boy/doiido'

⑧字符串.split( )函数表示按照指定“拆分符”对字符串拆分，返回拆分列表。

例如：

```
'./model/mnist_model-1001'.split('/')[-1].split('-')[-1]
```

在该例子中，共进行两次拆分。第一个拆分符为‘/’，返回拆分列表，并提取列表中索引为-1 的元素即倒数第一个元素；第二个拆分符为‘-’，返回拆分列表，并提取列表中索引为-1 的元素即倒数第一个元素，故函数返回值为 1001。

⑨tf.Graph( ).as_default( )函数表示将当前图设置成为默认图，并返回一个上下文管理器。该函数一般与 with 关键字搭配使用，应用于将已经定义好的神经网络在计算图中复现。

例如：

with tf.Graph().as_default() as g，表示将在 Graph()内定义的节点加入到计算图 g 中。

√神经网络模型的保存

在反向传播过程中，一般会间隔一定轮数保存一次神经网络模型，并产生三个文件（保存当前图结构的.meta 文件、保存当前参数名的.index 文件、保存当前参数的.data 文件），在 Tensorflow 中如下表示：

```
saver = tf.train.Saver()
with tf.Session() as sess:
    for i in range(STEPS):
        if i % 轮数 == 0:
            saver.save(sess, os.path.join(MODEL_SAVE_PATH,
                       MODEL_NAME), global_step=global_step)
```

其中，tf.train.Saver()用来实例化 saver 对象。上述代码表示，神经网络每循环规定的轮数，将神经网络模型中所有的参数等信息保存到指定的路径中，并在存放网络模型的文件夹名称中注明保存模型时的训练轮数。

√神经网络模型的加载

在测试网络效果时，需要将训练好的神经网络模型加载，在 Tensorflow 中这样表示：

```
with tf.Session() as sess:
    ckpt = tf.train.get_checkpoint_state(存储路径)
    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(sess, ckpt.model_checkpoint_path)
```

在 with 结构中进行加载保存的神经网络模型，若 ckpt 和保存的模型在指定路径中存在，则将保存的神经网络模型加载到当前会话中。

√加载模型中参数的滑动平均值

在保存模型时，若模型中采用滑动平均，则参数的滑动平均值会保存在相应文件中。通过实例化 saver 对象，实现参数滑动平均值的加载，在 Tensorflow 中如下表示：

```
ema = tf.train.ExponentialMovingAverage(滑动平均基数)
ema_restore = ema.variables_to_restore()
saver = tf.train.Saver(ema_restore)
```

√神经网络模型准确率评估方法

在网络评估时，一般通过计算在一组数据上的识别准确率，评估神经网络的效果。在 Tensorflow 中这样表示：

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

在上述中，y 表示在一组数据（即 batch_size 个数据）上神经网络模型的预测结果，y 的形状为[batch_size,10]，每一行表示一张图片的识别结果。通过 tf.argmax()函数取出每张图片对应向量中最大值元素对应的索引值，组成长度为输入数据 batch_size 个的一维数组。通过 tf.equal()函数判断预测结果张量和实际标签张量的每个维度是否相等，若相等则返回 True，不相等则返回 False。通过 tf.cast()函数将得到的布尔型数值转化为实数型，再通过 tf.reduce_mean()函数求平均值，最终得到神经网络模型在本组数据上的准确率。

5.2

神经网络八股包括前向传播过程、反向传播过程、反向传播过程中用到的正则化、指数衰减学习率、滑动平均方法的设置、以及测试模块。

√前向传播过程（forward.py）

前向传播过程完成神经网络的搭建，结构如下：

```
def forward(x, regularizer):
    w=
    b=
    y=
    return y
def get_weight(shape, regularizer):
def get_bias(shape):
```

前向传播过程中，需要定义神经网络中的参数 w 和偏置 b，定义由输入到输出的网络结构。通过定义函数 get_weight()实现对参数 w 的设置，包括参数 w 的形状和是否正则化的标志。同样，通过定义函数 get_bias()实现对偏置 b 的设置。

√反向传播过程（backword.py）

反向传播过程完成网络参数的训练，结构如下：

```
def backward( mnist ):
x = tf.placeholder(dtype, shape )
y_ = tf.placeholder(dtype, shape )
#定义前向传播函数
y = forward( )
global_step =
loss =
train_step = tf.train.GradientDescentOptimizer(learning_rate).
             minimize(loss, global_step=global_step)
#实例化 saver 对象
saver = tf.train.Saver()
with tf.Session() as sess:
    #初始化所有模型参数
```

```
tf.initialize_all_variables().run()
#训练模型
for i in range(STEPS):
    sess.run(train_step, feed_dict={x:  , y_:  })
    if i % 轮数 == 0:
        print
        saver.save(  )
```

反向传播过程中，用 tf.placeholder(dtype, shape)函数实现训练样本 x 和样本标签 y_占位，函数参数 dtype 表示数据的类型，shape 表示数据的形状；y 表示定义的前向传播函数 forward；loss 表示定义的损失函数，一般为预测值与样本标签的交叉熵（或均方误差）与正则化损失之和；train_step 表示利用优化算法对模型参数进行优化，常用优化算法 GradientDescentOptimizer、AdamOptimizer、MomentumOptimizer 算法，在上述代码中使用的 GradientDescentOptimizer 优化算法。接着实例化 saver 对象，其中利用 tf.initialize_all_variables().run()函数实例化所有参数模型，利用 sess.run( )函数实现模型的训练优化过程，并每间隔一定轮数保存一次模型。

√正则化、指数衰减学习率、滑动平均方法的设置

①正则化项 regularization

当在前向传播过程中即 forward.py 文件中，设置正则化参数 regularization 为 1 时，则表明在反向传播过程中优化模型参数时，需要在损失函数中加入正则化项。

结构如下：

首先，需要在前向传播过程即 forward.py 文件中加入

```
if regularizer != None: tf.add_to_collection('losses',
tf.contrib.layers.l2_regularizer(regularizer)(w))
```

其次，需要在反向传播过程即 backword.py 文件中加入

```
ce = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y,
labels=tf.argmax(y_, 1))
cem = tf.reduce_mean(ce)
```

```
loss = cem + tf.add_n(tf.get_collection('losses'))
```

其中，tf.nn.sparse_softmax_cross_entropy_with_logits()表示 softmax()函数与交叉熵一起使用。

②指数衰减学习率

在训练模型时，使用指数衰减学习率可以使模型在训练的前期快速收敛接近较优解，又可以保证模型在训练后期不会有太大波动。

运用指数衰减学习率，需要在反向传播过程即 backword.py 文件中加入：

```
learning_rate = tf.train.exponential_decay(
LEARNING_RATE_BASE,
global_step,
LEARNING_RATE_STEP, LEARNING_RATE_DECAY,
staircase=True)
```

③滑动平均

在模型训练时引入滑动平均可以使模型在测试数据上表现的更加健壮。

需要在反向传播过程即 backword.py 文件中加入：

```
ema = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY,
      global_step)
ema_op = ema.apply(tf.trainable_variables())
with tf.control_dependencies([train_step, ema_op]):
    train_op = tf.no_op(name='train')
```

√测试过程（test.py）

当神经网络模型训练完成后，便可用于测试数据集，验证神经网络的性能。结构如下：

首先，制定模型测试函数 test()

```
def test( mnist ):
    with tf.Graph( ).as_default( ) as g:
        #给 x y_占位
        x = tf.placeholder(dtype,shape)
        y_ = tf.placeholder(dtype,shape)
```

```
        #前向传播得到预测结果 y
         y = mnist_forward.forward(x, None)  #前向传播得到 y
          #实例化可还原滑动平均的 saver
          ema = tf.train.ExponentialMovingAverage(滑动衰减率)
          ema_restore = ema.variables_to_restore()
          saver = tf.train.Saver(ema_restore)
          #计算正确率
           correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y_,
                                1))
          accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                     tf.float32))
          while True:
              with tf.Session() as sess:
                  #加载训练好的模型
                  ckpt = tf.train.get_checkpoint_state(存储路径)
                  #如果已有 ckpt 模型则恢复
                  if ckpt and ckpt.model_checkpoint_path:
                      #恢复会话
                      saver.restore(sess, ckpt.model_checkpoint_path)
                      #恢复轮数
                      global_ste = ckpt.model_checkpoint_path.split
                                   ('/')[-1].split('-')[-1]
                      #计算准确率
                      accuracy_score = sess.run(accuracy, feed_dict=
                                       {x:测试数据, y_:测试数据标签 })
                      # 打印提示
                      print("After %s training step(s), test accuracy=
                            %g" % (global_step, accuracy_score))
                  #如果没有模型
                  else:
                      print('No checkpoint file found') #模型不存在提示
                      return
```

其次，制定 main()函数

```
def main():
    #加载测试数据集
    mnist = input_data.read_data_sets("./data/", one_hot=True)
    #调用定义好的测试函数 test()
```

```
    test(mnist)
if __name__ == '__main__':
    main()
```

通过对测试数据的预测得到准确率，从而判断出训练出的神经网络模型的性能好坏。当准确率低时，可能原因有模型需要改进，或者是训练数据量太少导致过拟合。

5.3

实现手写体 mnist 数据集的识别任务，共分为三个模块文件，分别是描述网络结构的前向传播过程文件（mnist_forward.py）、描述网络参数优化方法的反向传播过程文件（mnist_backward.py）、验证模型准确率的测试过程文件（mnist_test.py）。

√前向传播过程文件（mnist_forward.py）

在前向传播过程中，需要定义网络模型输入层个数、隐藏层节点数、输出层个数，定义网络参数 w、偏置 b，定义由输入到输出的神经网络架构。

实现手写体 mnist 数据集的识别任务前向传播过程如下：

```python
import tensorflow as tf

INPUT_NODE = 784
OUTPUT_NODE = 10
LAYER1_NODE = 500

def get_weight(shape, regularizer):
    w = tf.Variable(tf.truncated_normal(shape,stddev=0.1))
    if regularizer != None: tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))
    return w


def get_bias(shape):
    b = tf.Variable(tf.zeros(shape))
    return b

def forward(x, regularizer):
    w1 = get_weight([INPUT_NODE, LAYER1_NODE], regularizer)
    b1 = get_bias([LAYER1_NODE])
    y1 = tf.nn.relu(tf.matmul(x, w1) + b1)

    w2 = get_weight([LAYER1_NODE, OUTPUT_NODE], regularizer)
    b2 = get_bias([OUTPUT_NODE])
    y = tf.matmul(y1, w2) + b2
    return y
```

由上述代码可知，在前向传播过程中，规定网络输入结点为 784 个（代表每张输入图片的像素个数），隐藏层节点 500 个，输出节点 10 个（表示输出为数字 0-9 的十分类）。由输入层到隐藏层的参数 w1 形状为[784,500]，由隐藏层到输出层的参数 w2 形状为[500,10]，参数满足截断正态分布，并使用正则化，将每个参

数的正则化损失加到总损失中。由输入层到隐藏层的偏置 b1 形状为长度为 500 的一维数组，由隐藏层到输出层的偏置 b2 形状为长度为 10 的一维数组，初始化值为全 0。前向传播结构第一层为输入 x 与参数 w1 矩阵相乘加上偏置 b1，再经过 relu 函数，得到隐藏层输出 y1。前向传播结构第二层为隐藏层输出 y1 与参数 w2 矩阵相乘加上偏置 b2，得到输出 y。由于输出 y 要经过 softmax 函数，使其符合概率分布，故输出 y 不经过 relu 函数。

√反向传播过程文件（mnist_backward.py）

反向传播过程实现利用训练数据集对神经网络模型训练，通过降低损失函数值，实现网络模型参数的优化，从而得到准确率高且泛化能力强的神经网络模型。

实现手写体 mnist 数据集的识别任务反向传播过程如下：

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import mnist_forward
import os

BATCH_SIZE = 200
LEARNING_RATE_BASE = 0.1
LEARNING_RATE_DECAY = 0.99
REGULARIZER = 0.0001
STEPS = 50000
MOVING_AVERAGE_DECAY = 0.99
MODEL_SAVE_PATH="./model/"
MODEL_NAME="mnist_model"


def backward(mnist):

    x = tf.placeholder(tf.float32, [None, mnist_forward.INPUT_NODE])
    y_ = tf.placeholder(tf.float32, [None, mnist_forward.OUTPUT_NODE])
    y = mnist_forward.forward(x, REGULARIZER)
    global_step = tf.Variable(0, trainable=False)

    ce = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, labels=tf.argmax(y_, 1))
    cem = tf.reduce_mean(ce)
    loss = cem + tf.add_n(tf.get_collection('losses'))

    learning_rate = tf.train.exponential_decay(
        LEARNING_RATE_BASE,
        global_step,
        mnist.train.num_examples / BATCH_SIZE,
        LEARNING_RATE_DECAY,
        staircase=True)

```

```
33
34      train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)
35
36      ema = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, global_step)
37      ema_op = ema.apply(tf.trainable_variables())
38      with tf.control_dependencies([train_step, ema_op]):
39          train_op = tf.no_op(name='train')
40
41      saver = tf.train.Saver()
42
43      with tf.Session() as sess:
44          init_op = tf.global_variables_initializer()
45          sess.run(init_op)
46
47          for i in range(STEPS):
48              xs, ys = mnist.train.next_batch(BATCH_SIZE)
49              _, loss_value, step = sess.run([train_op, loss, global_step], feed_dict={x: xs, y_: ys})
50              if i % 1000 == 0:
51                  print("After %d training step(s), loss on training batch is %g." % (step, loss_value))
52                  saver.save(sess, os.path.join(MODEL_SAVE_PATH, MODEL_NAME), global_step=global_step)
53
54
55  def main():
56      mnist = input_data.read_data_sets("./data/", one_hot=True)
57      backward(mnist)
58
59  if __name__ == '__main__':
60      main()
61
```

由上述代码可知，在反向传播过程中，首先引入 tensorflow、input_data、前向传播 mnist_forward 和 os 模块，定义每轮喂入神经网络的图片数、初始学习率、学习率衰减率、正则化系数、训练轮数、模型保存路径以及模型保存名称等相关信息。在反向传播函数 backword 中，首先读入 mnist，用 placeholder 给训练数据 x 和标签 y_占位，调用 mnist_forward 文件中的前向传播过程 forword()函数，并设置正则化，计算训练数据集上的预测结果 y，并给当前计算轮数计数器赋值，设定为不可训练类型。接着，调用包含所有参数正则化损失的损失函数 loss，并设定指数衰减学习率 learning_rate。然后，使用梯度衰减算法对模型优化，降低损失函数，并定义参数的滑动平均。最后，在 with 结构中，实现所有参数初始化，每次喂入 batch_size 组（即 200 组）训练数据和对应标签，循环迭代 steps 轮，并每隔 1000 轮打印出一次损失函数值信息，并将当前会话加载到指定路径。最后，通过主函数 main()，加载指定路径下的训练数据集，并调用规定的 backward()函数训练模型。

√测试过程文件（mnist_test.py）

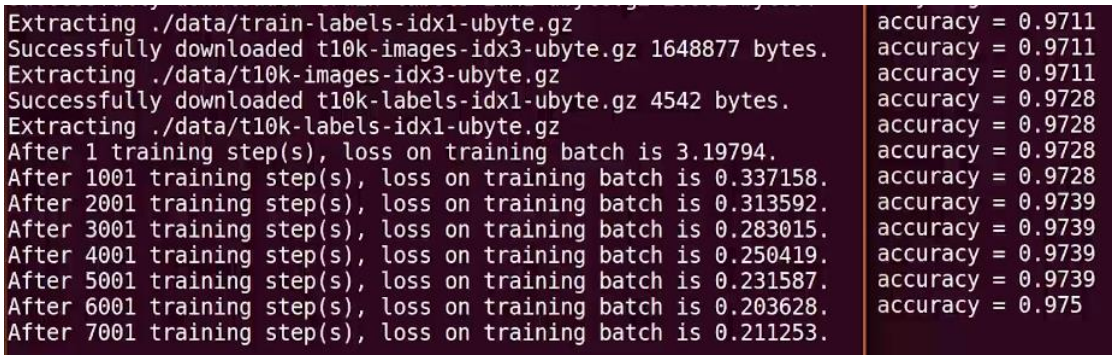
当训练完模型后，给神经网络模型输入测试集验证网络的准确性和泛化性。注意，所用的测试集和训练集是相互独立的。

实现手写体 mnist 数据集的识别任务测试传播过程如下：

```python
#coding:utf-8
import time
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import mnist_forward
import mnist_backward
TEST_INTERVAL_SECS = 5

def test(mnist):
    with tf.Graph().as_default() as g:
        x = tf.placeholder(tf.float32, [None, mnist_forward.INPUT_NODE])
        y_ = tf.placeholder(tf.float32, [None, mnist_forward.OUTPUT_NODE])
        y = mnist_forward.forward(x, None)

        ema = tf.train.ExponentialMovingAverage(mnist_backward.MOVING_AVERAGE_DECAY)
        ema_restore = ema.variables_to_restore()
        saver = tf.train.Saver(ema_restore)

        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

        while True:
            with tf.Session() as sess:
                ckpt = tf.train.get_checkpoint_state(mnist_backward.MODEL_SAVE_PATH)
                if ckpt and ckpt.model_checkpoint_path:
                    saver.restore(sess, ckpt.model_checkpoint_path)
                    global_step = ckpt.model_checkpoint_path.split('/')[-1].split('-')[-1]
                    accuracy_score = sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
                    print("After %s training step(s), test accuracy = %g" % (global_step, accuracy_score))
                else:
                    print('No checkpoint file found')
                    return
            time.sleep(TEST_INTERVAL_SECS)

def main():
    mnist = input_data.read_data_sets("./data/", one_hot=True)
    test(mnist)

if __name__ == '__main__':
    main()
```

在上述代码中，首先需要引入 time 模块、tensorflow、input_data、前向传播 mnist_forward、反向传播 mnist_backward 模块和 os 模块，并规定程序 5 秒的循环间隔时间。接着，定义测试函数 test()，读入 mnist 数据集，利用 tf.Graph() 复现之前定义的计算图，利用 placeholder 给训练数据 x 和标签 y_占位，调用 mnist_forward 文件中的前向传播过程 forword()函数，计算训练数据集上的预测结果 y。接着，实例化具有滑动平均的 saver 对象，从而在会话被加载时模型中的所有参数被赋值为各自的滑动平均值，增强模型的稳定性，然后计算模型在测试集上的准确率。在 with 结构中，加载指定路径下的 ckpt，若模型存在，则加载出模型到当前对话，在测试数据集上进行准确率验证，并打印出当前轮数下的准确率，若模型不存在，则打印出模型不存在的提示，从而 test()函数完成。通过主函数 main()，加载指定路径下的测试数据集，并调用规定的 test 函数，进行模型在测试集上的准确率验证。

运行以上三个文件，可得到手写体 mnist 数据集的识别任务的运行结果：



从终端显示的运行结果可以看出，随着训练轮数的增加，网络模型的损失函数值在不断降低，并且在测试集上的准确率在不断提升，有较好的泛化能力。