

## Tensorflow 笔记：第六讲

### 输入手写数字输出识别结果

本节目标：1、实现断点续训  
2、输入真实图片，输出预测结果  
3、制作数据集，实现特定应用

#### 6.1

##### 一、断点续训

✓关键处理：加入 **ckpt 操作**：

```
ckpt = tf.train.get_checkpoint_state(MODEL_SAVE_PATH)
if ckpt and ckpt.model_checkpoint_path:
    saver.restore(sess, ckpt.model_checkpoint_path)
```

1、注解：

1) `tf.train.get_checkpoint_state(checkpoint_dir, latest_filename=None)`

该函数表示如果断点文件夹中包含有效断点状态文件，则返回该文件。

参数说明：checkpoint\_dir：表示存储断点文件的目录

latest\_filename=None：断点文件的可选名称，默认为“checkpoint”

2) `saver.restore(sess, ckpt.model_checkpoint_path)`

该函数表示恢复当前会话，将 ckpt 中的值赋给 w 和 b。

参数说明：sess：表示当前会话，之前保存的结果将被加载入这个会话

ckpt.model\_checkpoint\_path：表示模型存储的位置，不需要提供模型的名字，它会去查看 checkpoint 文件，看看最新的是谁，叫做什么。

2、ckpt 代码位置：

```

with tf.Session() as sess:
    init_op = tf.global_variables_initializer()
    sess.run(init_op)

    ckpt = tf.train.get_checkpoint_state(MODEL_SAVE_PATH)
    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(sess, ckpt.model_checkpoint_path)

    for i in range(STEPS):
        xs, ys = mnist.train.next_batch(BATCH_SIZE)
        _, loss_value, step = sess.run([train_op, loss, global_step], feed_dict={x: xs, y_: ys})
        if i % 1000 == 0:
            print("After %d training step(s), loss on training batch is %g." % (step, loss_value))
            saver.save(sess, os.path.join(MODEL_SAVE_PATH, MODEL_NAME), global_step=global_step)

```

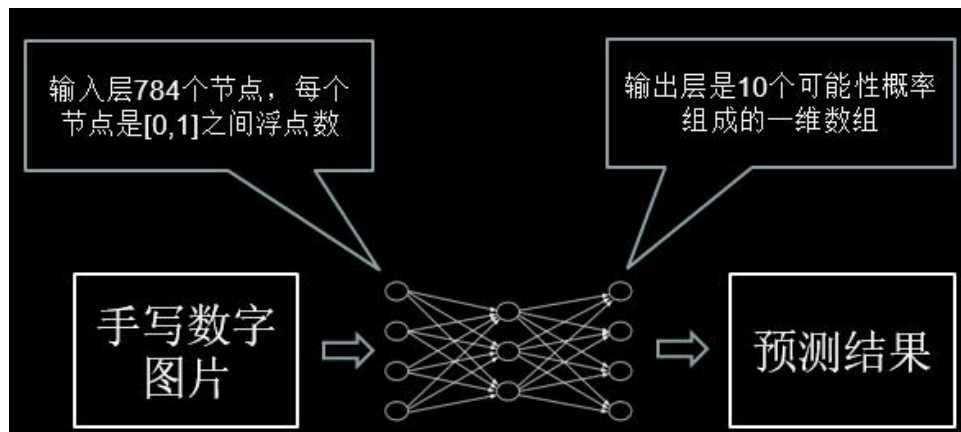
### 3、实践代码验证

```

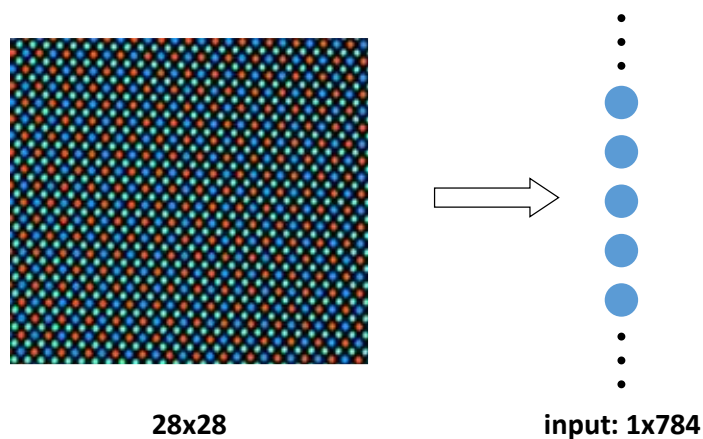
lab@aillab: ~/fc3
lab@aillab:~$ cd fc3
lab@aillab:~/fc3$ python mnist_backward.py
Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz
Extracting ./data/t10k-images-idx3-ubyte.gz
Extracting ./data/t10k-labels-idx1-ubyte.gz
After 50003 training step(s), loss on training batch is 0.127414.
After 51003 training step(s), loss on training batch is 0.132541.
After 52003 training step(s), loss on training batch is 0.128016.
After 53003 training step(s), loss on training batch is 0.119224.

```

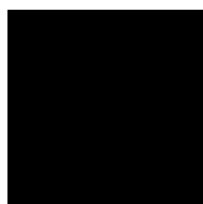
## 二、输入真实图片，输出预测结果



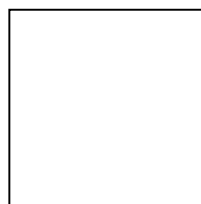
✓ 网络输入：一维数组（784 个像素点）



✓ 像素点：0-1 之间的浮点数（接近 0 越黑，接近 1 越白）



像素为0



像素为1

✓ 网络输出：一维数组（十个可能性~~概率~~），数组中最大的那个元素所对应的索引号就是预测的结果。

✓ 关键处理：

```
def application():
    testNum = input("input the number of test pictures:")
    for i in range(testNum):
        testPic = raw_input("the path of test picture:")
        testPicArr = pre_pic(testPic)
        preValue = restore_model(testPicArr)
        print "The prediction number is:", preValue
```

注解：

任务分成两个函数完成

1) testPicArr = pre\_pic(testPic) 对手写数字图片做预处理

2) preValue = restore\_model(testPicArr) 将符合神经网络输入要求的图片喂给复现的神经网络模型，输出预测值

✓ 具体代码：

```

def restore_model(testPicArr):
    # 创建一个默认图，在该图中执行以下操作（多数操作和train中一样，就不再重复解释，大家对照学习即可）
    with tf.Graph().as_default() as tg:
        x = tf.placeholder(tf.float32, [None, mnist_forward.INPUT_NODE])
        y = mnist_forward.forward(x, None)
        preValue = tf.argmax(y, 1) # 得到概率最大的预测值

        # 实现滑动平均模型，参数MOVING_AVERAGE_DECAY用于控制模型更新的速度。训练过程中会对每一个变量维护一个影子变量，这个影子变量的初始值
        # 就是相应变量的初始值，每次变量更新时，影子变量就会随之更新
        variable_averages = tf.train.ExponentialMovingAverage(mnist_backward.MOVING_AVERAGE_DECAY)
        variables_to_restore = variable_averages.variables_to_restore()
        saver = tf.train.Saver(variables_to_restore)

    with tf.Session() as sess:
        # 通过checkpoint文件定位到最新保存的模型
        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)

            preValue = sess.run(preValue, feed_dict={x:testPicArr})
            return preValue
        else:
            print("No checkpoint file found")
            return -1

```

```

# 预处理函数，包括resize、转变灰度图、二值化操作
def pre_pic(picName):
    img = Image.open(picName)
    reIm = img.resize((28,28), Image.ANTIALIAS)
    im_arr = np.array(reIm.convert('L'))
    threshold = 50 # 设定合理的阈值
    for i in range(28):
        for j in range(28):
            im_arr[i][j] = 255 - im_arr[i][j]
            if (im_arr[i][j] < threshold):
                im_arr[i][j] = 0
            else: im_arr[i][j] = 255

    nm_arr = im_arr.reshape([1, 784])
    nm_arr = nm_arr.astype(np.float32)
    img_ready = np.multiply(nm_arr, 1.0/255.0)

    return img_ready

```

## 1、注解：

1) main 函数中的 application 函数：输入要识别的几张图片（注意要给出待识别图片的**路径和名称**）。

## 2）代码处理过程：

（1）模型的要求是黑底白字，但输入的图是白底黑字，所以需要每个像素点的值改为 255 减去原值以得到互补的反色。

（2）对图片做二值化处理（这样以滤掉噪声，另外调试中可适当调节阈值）。



(3) 把图片形状拉成 1 行 784 列，并把值变为浮点型（因为要求像素点是 0-1 之间的浮点数）。

(4) 接着让现有的 RGB 图从 0-255 之间的数变为 0-1 之间的浮点数。

(5) 运行完成后返回到 main 函数。

(6) 计算求得输出 y，y 的**最大值**所对应的**列表索引号**就是预测结果。

## 2、实践代码验证

### 1) 运行 mnist\_backward.py

```
lab@aillab: ~/fc3
lab@aillab:~$ cd fc3
lab@aillab:~/fc3$ python mnist_backward.py
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
After 50003 training step(s), loss on training batch is 0.127414.
After 51003 training step(s), loss on training batch is 0.132541.
After 52003 training step(s), loss on training batch is 0.128016.
After 53003 training step(s), loss on training batch is 0.119224.
```

### 2) 运行 mnist\_test.py 来监测模型的准确率

```
lab@aillab: ~/fc3
lab@aillab:~$ cd fc3
lab@aillab:~/fc3$ python mnist_test.py
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
After 51003 training step(s), test accuracy = 0.9804
After 52003 training step(s), test accuracy = 0.9805
After 52003 training step(s), test accuracy = 0.9805
After 53003 training step(s), test accuracy = 0.9805
After 53003 training step(s), test accuracy = 0.9805
After 54003 training step(s), test accuracy = 0.9806
```

### 3) 运行 mnist\_app.py 输入 10（表示循环验证十张图片）

```
lab@ailab: ~/fc3
lab@ailab:~$ cd fc3
lab@ailab:~/fc3$ python mnist_app.py
input the number of test pictures:10
the path of test picture:pic/0.png
The prediction number is: [0]
the path of test picture:pic/1.png
The prediction number is: [1]
the path of test picture:pic/2.png
The prediction number is: [2]
the path of test picture:pic/3.png
The prediction number is: [3]
the path of test picture:pic/4.png
The prediction number is: [4]
the path of test picture:pic/5.png
The prediction number is: [5]
the path of test picture:pic/6.png
The prediction number is: [6]
the path of test picture:pic/7.png
The prediction number is: [7]
the path of test picture:pic/8.png
The prediction number is: [8]
the path of test picture:pic/9.png
The prediction number is: [9]
```

## 6.2

制作数据集，实现特定应用：

1、数据集生成读取文件（mnist\_generateds.py）

√tfrecords 文件

1) tfrecords：是一种二进制文件，可先将图片和标签制作成该格式的文件。

使用 tfrecords 进行数据读取，会提高内存利用率。

2) tf.train.Example：用来存储训练数据。训练数据的特征用键值对的形式表示。

如：‘img\_raw’：值 ‘label’：值 值是 Byteslist/FloatList/Int64List

3) SerializeToString()：把数据序列化成字符串存储。

√生成 tfrecords 文件



具体代码：

```

16
17 def write_tfRecord(tfRecordName, image_path, label_path):
18     writer = tf.python_io.TFRecordWriter(tfRecordName)
19     num_pic = 0
20     f = open(label_path, 'r')
21     contents = f.readlines()
22     f.close()
23     for content in contents:
24         value = content.split()
25         img_path = image_path + value[0]
26         img = Image.open(img_path)
27         img_raw = img.tobytes()
28         labels = [0] * 10
29         labels[int(value[1])] = 1
30
31         example = tf.train.Example(features=tf.train.Features(feature={
32             'img_raw': tf.train.Feature(bytes_list=tf.train.BytesList(value=[img_raw])),
33             'label': tf.train.Feature(int64_list=tf.train.Int64List(value=labels))
34         }))
35         writer.write(example.SerializeToString())
36         num_pic += 1
37         print("the number of picture:", num_pic)
38     writer.close()
39     print("write tfrecord successful")
40
41 def generate_tfRecord():
42     isExists = os.path.exists(data_path)
43     if not isExists:
44         os.makedirs(data_path)
45         print 'The directory was created successfully'
46     else:
47         print 'directory already exists'
48     write_tfRecord(tfRecord_train, image_train_path, label_train_path)
49     write_tfRecord(tfRecord_test, image_test_path, label_test_path)
50

```

注解:

- 1) writer = tf.python\_io.TFRecordWriter(tfRecordName) #新建一个 writer
- 2) for 循环遍历每张图和标签
- 3) example = tf.train.Example(features=tf.train.Features(feature={  
     'img\_raw':tf.train.Feature(bytes\_list=tf.train.BytesList(value=[  
img\_raw])),  
     'label':tf.train.Feature(int64\_list=tf.train.Int64List(value=labels))))) # 把每张图片 and 标签封装到 example 中
- 4) writer.write(example.SerializeToString()) # 把 example 进行序列化
- 5) writer.close() #关闭 writer

✓解析 tfrecords 文件

具体代码:

```

50
51 def read_tfRecord(tfRecord_path):
52     filename_queue = tf.train.string_input_producer([tfRecord_path])
53     reader = tf.TFRecordReader()
54     _, serialized_example = reader.read(filename_queue)
55     features = tf.parse_single_example(serialized_example,
56                                       features={
57                                           'label': tf.FixedLenFeature([10], tf.int64),
58                                           'img_raw': tf.FixedLenFeature([], tf.string)
59                                       })
60     img = tf.decode_raw(features['img_raw'], tf.uint8)
61     img.set_shape([784])
62     img = tf.cast(img, tf.float32) * (1. / 255)
63     label = tf.cast(features['label'], tf.float32)
64     return img, label
65
66 def get_tfrecord(num, isTrain=True):
67     if isTrain:
68         tfRecord_path = tfRecord_train
69     else:
70         tfRecord_path = tfRecord_test
71     img, label = read_tfRecord(tfRecord_path)
72     img_batch, label_batch = tf.train.shuffle_batch([img, label],
73                                                    batch_size = num,
74                                                    num_threads = 2,
75                                                    capacity = 1000,
76                                                    min_after_dequeue = 700)
77     return img_batch, label_batch
78
79 def main():
80     generate_tfRecord()
81
82 if __name__ == '__main__':
83     main()

```

注解:

1) `filename_queue = tf.train.string_input_producer([tfRecord_path])`

```

tf.train.string_input_producer( string_tensor,
                                num_epochs=None,
                                shuffle=True,
                                seed=None,
                                capacity=32,
                                shared_name=None,
                                name=None,
                                cancel_op=None)

```

该函数会生成一个先入先出的队列，文件阅读器会使用它来读取数据。

参数说明: `string_tensor`: 存储图像和标签信息的 TFRecord 文件名列表

`num_epochs`: 循环读取的轮数（可选）

`shuffle`: 布尔值（可选），如果为 True，则在每轮随机打乱读取顺序

`seed`: 随机读取时设置的种子（可选）

`capacity`: 设置队列容量



shared\_name: (可选) 如果设置, 该队列将在多个会话中以给定名称共享。所有具有此队列的设备都可以通过 shared\_name 访问它。在分布式设置中使用这种方法意味着每个名称只能被访问此操作的其中一个会话看到。

name: 操作的名称 (可选)

cancel\_op: 取消队列 (None)

2) reader = tf.TFRecordReader() #新建一个 reader

3) \_, serialized\_example = reader.read(filename\_queue)

```
features = tf.parse_single_example(serialized_example, features={
    'img_raw': tf.FixedLenFeature([], tf.string),
    'label': tf.FixedLenFeature([10], tf.int64)})
```

#把读出的每个样本保存在 serialized\_example 中进行解序列化, 标签和图片的键名应该和制作 tfrecords 的键名相同, 其中标签给出几分类。

```
tf.parse_single_example(serialized,
                        features,
                        name=None,
                        example_names=None)
```

该函数可以将 tf.train.Example 协议内存块(protocol buffer)解析为张量。

参数说明: serialized: 一个标量字符串张量

features: 一个字典映射功能键 FixedLenFeature 或 VarLenFeature 值, 也就是在协议内存块中储存的

name: 操作的名称 (可选)

example\_names: 标量字符串的名称 (可选)

4) img = tf.decode\_raw(features['img\_raw'], tf.uint8)

#将 img\_raw 字符串转换为 8 位无符号整型

5) img.set\_shape([784]) #将形状变为一行 784 列

6) img = tf.cast(img, tf.float32) \* (1. / 255) #变成 0 到 1 之间的浮点数

7) label = tf.cast(features['label'], tf.float32) #把标签列表变为浮点数

8) return image, label #返回图片和标签 (跳回到 get\_tfrecord)

9) tf.train.shuffle\_batch( tensors,

```

        batch_size,
        capacity,
        min_after_dequeue,
        num_threads=1,
        seed=None,
        enqueue_many=False,
        shapes=None,
        allow_smaller_final_batch=False,
        shared_name=None,
        name=None)

```

这个函数随机读取一个 batch 的数据。

参数说明：tensors：待乱序处理的列表中的样本（图像和标签）

batch\_size：从队列中提取的新批量大小

capacity：队列中元素的最大数量

min\_after\_dequeue：出队后队列中的最小数量元素，用于确保元素的混合级别

num\_threads：排列 tensors 的线程数

seed：用于队列内的随机洗牌

enqueue\_many：tensor 中的每个张量是否是一个例子

shapes：每个示例的形状

allow\_smaller\_final\_batch：（可选）布尔值。 如果为 True，则在队列中剩余数量不足时允许最终批次更小。

shared\_name：（可选）如果设置，该队列将在多个会话中以给定名称共享。

name：操作的名称（可选）

10) return img\_batch, label\_batch

#返回的图片和标签为随机抽取的 batch\_size 组

## 2、反向传播文件修改图片标签获取的接口（mnist\_backward.py）

✓关键操作：利用多线程提高图片和标签的批获取效率

方法：将批获取的操作放到线程协调器开启和关闭之间

开启线程协调器：

```
coord = tf.train.Coordinator( )  
threads = tf.train.start_queue_runners(sess=sess, coord=coord)
```

关闭线程协调器：

```
coord.request_stop( )  
coord.join(threads)
```

注解：

```
tf.train.start_queue_runners( sess=None,  
                               coord=None,  
                               daemon=True,  
                               start=True,  
                               collection=tf.GraphKeys.QUEUE_RUNNERS)
```

这个函数将会启动输入队列的线程，填充训练样本到队列中，以便出队操作可以从队列中拿到样本。这种情况下最好配合使用一个 `tf.train.Coordinator`，这样可以在发生错误的情况下正确地关闭这些线程。

参数说明：`sess`：用于运行队列操作的会话。默认为默认会话。

`coord`：可选协调器，用于协调启动的线程。

`daemon`：守护进程，线程是否应该标记为守护进程，这意味着它们不会阻止程序退出。

`start`：设置为 `False` 只创建线程，不启动它们。

`collection`：指定图集合以获取启动队列的 `GraphKey`。默认为 `GraphKeys.QUEUE_RUNNERS`。

✓ 具体对比反向传播中的 `fc4` 与 `fc3` 代码

```

1 import tensorflow as tf
2 from tensorflow.examples.tutorials.mnist import input_data
3 import mnist_forward
4 import os
5 import mnist_generateds#1
6
7 BATCH_SIZE = 200
8 LEARNING_RATE_BASE = 0.1
9 LEARNING_RATE_DECAY = 0.99
10 REGULARIZER = 0.0001
11 STEPS = 50000
12 MOVING_AVERAGE_DECAY = 0.99
13 MODEL_SAVE_PATH = './model/'
14 MODEL_NAME = 'mnist_model'
15 train_num_examples = 60000#2 mnist.train_num_examples
16
17 def backward():
18
19     x = tf.placeholder(tf.float32, [None, mnist_forward.INPUT_NODE])
20     y = tf.placeholder(tf.float32, [None, mnist_forward.OUTPUT_NODE])
21     y = mnist_forward.forward(x, REGULARIZER)
22     global_step = tf.Variable(0, trainable=False)
23
24     ce = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, labels=tf.argmax(y, 1))
25     cem = tf.reduce_mean(ce)
26     loss = cem + tf.add_n(tf.get_collection('losses'))
27
28     learning_rate = tf.train.exponential_decay(
29         LEARNING_RATE_BASE,
30         global_step,
31         train_num_examples / BATCH_SIZE,
32         LEARNING_RATE_DECAY,
33         staircase=True)
34
35     train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)
36
37
38 ema = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, global_step)
39 ema_op = ema.apply(tf.trainable_variables())
40 with tf.control_dependencies([train_step, ema_op]):
41     train_op = tf.no_op(name='train')
42
43 saver = tf.train.Saver()
44 img_batch, label_batch = mnist_generateds.get_tfrecord(BATCH_SIZE, isTrain=True)#3
45
46 with tf.Session() as sess:
47     init_op = tf.global_variables_initializer()
48     sess.run(init_op)
49
50 ckpt = tf.train.get_checkpoint_state(MODEL_SAVE_PATH)
51 if ckpt and ckpt.model_checkpoint_path:
52     saver.restore(sess, ckpt.model_checkpoint_path)
53
54 coord = tf.train.Coordinator()#4
55 threads = tf.train.start_queue_runners(sess=sess, coord=coord)#5
56
57 for i in range(STEPS):
58     xs, ys = sess.run([img_batch, label_batch]) #6 xs, ys = mnist.train.next_batch(BATCH_SIZE)
59     loss_value, step = sess.run([train_op, loss, global_step], feed_dict={x: xs, y: ys})
60     if i % 1000 == 0:
61         print('After %d training step(s), loss on training batch is %g.' % (step, loss_value))
62         saver.save(sess, os.path.join(MODEL_SAVE_PATH, MODEL_NAME), global_step=global_step)
63
64 coord.request_stop()#7
65 coord.join(threads)#8
66
67 def main():
68     backward()#9
69
70 if __name__ == '__main__':
71     main()

```

注解:

1) train\_num\_examples=60000

在梯度下降学习率中需要计算多少轮更新一次学习率，这个值是  $\frac{\text{总样本数}}{\text{batch size}}$

之前: 用 mnist.train.num\_examples 表示总样本数;

现在: 要手动给出训练的总样本数，这个数是 6 万。

2) `image_batch, label_batch=mnist_generators.get_tfrecord(BATCH_SIZE,`  
`isTrain=True)`

之前: 用 `mnist.train.next_batch` 函数读出图片和标签喂给网络;

现在: 用函数 `get_tfrecord` 替换, 一次批获取 **batch\_size** 张图片和标签。

`isTrain`: 用来区分训练阶段和测试阶段, `True` 表示训练, `False` 表示测试。

3) `xs,ys=sess.run([img_batch,label_batch])`

之前: 使用函数 `xs,ys=mnist.train.next_batch(BATCH_SIZE)`

现在: 在 `sess.run` 中执行图片和标签的批获取。

### 3、测试文件修改图片标签获取的接口 (`mnist_test.py`)

✓ 具体对比反向传播中的 `fc4` 与 `fc3` 代码 (和反向传播类似)

```
1 #coding:utf-8
2 import time
3 import tensorflow as tf
4 from tensorflow.examples.tutorials.mnist import input_data
5 import mnist_forward
6 import mnist_backward
7 import mnist_generators
8 TEST_INTERVAL_SECS = 5
9 TEST_NUM = 10000#1 mnist.test.num_examples
10
11 def test():
12     with tf.Graph().as_default() as g:
13         x = tf.placeholder(tf.float32, [None, mnist_forward.INPUT_NODE])
14         y_ = tf.placeholder(tf.float32, [None, mnist_forward.OUTPUT_NODE])
15         y = mnist_forward.forward(x, None)
16
17         ema = tf.train.ExponentialMovingAverage(mnist_backward.MOVING_AVERAGE_DECAY)
18         ema_restore = ema.variables_to_restore()
19         saver = tf.train.Saver(ema_restore)
20
21         correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
22         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
23
24     img_batch, label_batch = mnist_generators.get_tfrecord(TEST_NUM, isTrain=False)#2
25
26     while True:
27         with tf.Session() as sess:
28             ckpt = tf.train.get_checkpoint_state(mnist_backward.MODEL_SAVE_PATH)
29             if ckpt and ckpt.model_checkpoint_path:
30                 saver.restore(sess, ckpt.model_checkpoint_path)
31                 global_step = ckpt.model_checkpoint_path.split('/')[-1].split('-')[-1]
32
33                 coord = tf.train.Coordinator()#3
34                 threads = tf.train.start_queue_runners(sess=sess, coord=coord)#4
35
36                 xs, ys = sess.run([img_batch, label_batch])#5
37
38                 accuracy_score = sess.run(accuracy, feed_dict={x: xs, y_: ys})
39
40                 print("After %s training step(s), test accuracy = %g" % (global_step, accuracy_score))
41
42                 coord.request_stop()#6
43                 coord.join(threads)#7
44
45             else:
46                 print('No checkpoint file found')
47                 return
48             time.sleep(TEST_INTERVAL_SECS)
49
50 def main():
51     test()#8
52
53 if __name__ == '__main__':
54     main()
```

注解:

1) `TEST_NUM=10000`

之前: 用 `mnist.test.num_examples` 表示总样本数;



现在：要手动给出测试的总样本数，这个数是 1 万。

```
2) image_batch, label_batch=mnist_generators.get_tfrecord(TEST_NUM,  
                                                             isTrain=False)
```

之前：用 `mnist.test.next_batch` 函数读出图片和标签喂给网络；

现在：用函数 `get_tfrecord` 替换读取所有测试集 1 万张图片。

`isTrain`：用来区分训练阶段和测试阶段，True 表示训练，False 表示测试。

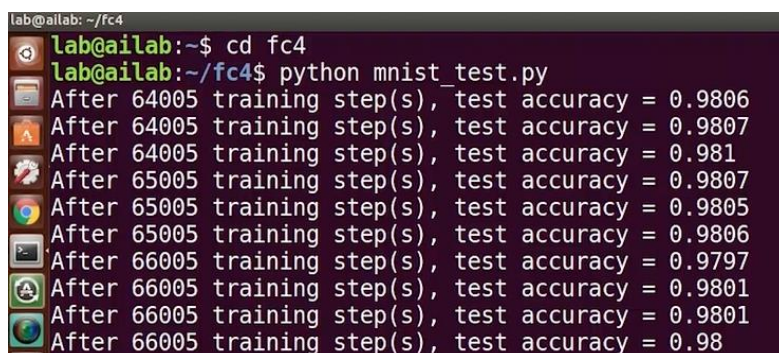
```
3) xs,ys=sess.run([img_batch,label_batch])
```

之前：使用函数 `xs,ys=mnist.test.next_batch(BATCH_SIZE)`

现在：在 `sess.run` 中执行图片和标签的批获取。

#### 4、实践代码验证

1) 运行测试代码 `mnist_test.py`



```
lab@aialab: ~/fc4  
lab@aialab:~$ cd fc4  
lab@aialab:~/fc4$ python mnist_test.py  
After 64005 training step(s), test accuracy = 0.9806  
After 64005 training step(s), test accuracy = 0.9807  
After 64005 training step(s), test accuracy = 0.981  
After 65005 training step(s), test accuracy = 0.9807  
After 65005 training step(s), test accuracy = 0.9805  
After 65005 training step(s), test accuracy = 0.9806  
After 66005 training step(s), test accuracy = 0.9797  
After 66005 training step(s), test accuracy = 0.9801  
After 66005 training step(s), test accuracy = 0.9801  
After 66005 training step(s), test accuracy = 0.98
```

2) 准确率稳定在 95%以上后运行应用程序 `mnist_app.py`



```
lab@aialab: ~/fc4  
lab@aialab:~$ cd fc4  
lab@aialab:~/fc4$ python mnist_app.py  
input the number of test pictures:10  
the path of test picture:pic/0.png  
The prediction number is: [0]  
the path of test picture:pic/1.png  
The prediction number is: [1]  
the path of test picture:pic/2.png  
The prediction number is: [2]  
the path of test picture:pic/3.png  
The prediction number is: [3]  
the path of test picture:pic/4.png  
The prediction number is: [4]  
the path of test picture:pic/5.png  
The prediction number is: [5]  
the path of test picture:pic/6.png  
The prediction number is: [6]  
the path of test picture:pic/7.png  
The prediction number is: [7]  
the path of test picture:pic/8.png  
The prediction number is: [8]  
the path of test picture:pic/9.png
```