五、 TensorFlow MNIST 手写字识别

5.1 具体实现过程

MNIST 手写字识别的训练图片是28*28寸的图片。

- 第一层隐藏层

第一步是将图片矩阵打平处理为 $[image]_{1\times784}$

第二 步是我们将第一层隐藏层将要设置的参数数量，这些数量会影响我们的训练精度，将设我们设置第一层的影藏层权重和偏置参数分别为500个，则可以设置第一个隐藏层的维数为$[\omega_1]_{784\times500}$，同样偏置的维数为$[b_1]_{1\times500}$

- 第二层隐藏层

第三步是设置第二层隐藏层的权重和偏置维数，因为我们只需要三层神经网络结构实现，所以我们这一层应该做池化实现onehot，输出应该为1×10的独热编码。所以ω_2应该是500行和10列。即$[\omega_2]_{500\times10}$，$[b_2]_{1\times10}$

- 整个流程

$\{[image]_{1\times784}\}$ -->$\{[\omega_1]_{784\times500}，[b_1]_{1\times500}\}$ --> $\{[\omega_2]_{500\times10}，[b_2]_{1\times10}\}$ --> $\{[onehot]_{1\times10}\}$

5.2 前向传播

训练的过程是，首先输入一批数据，然后对每批数据进行上个流程的计算过程，比如我一批次输入200张图片进行训练。

$\{[image]_{200\times784}\}$ -->$\{[\omega_1]_{784\times500}，[b_1]_{1\times500}\}$ =$\{[data]_{200\times500}\}$ --> $\{[\omega_2]_{500\times10}，[b_2]_{1\times10}\}$ = $\{[onehot_output]_{200\times10}\}$

前向传播算法的代码如下部分代码所示

```
import tensorflow as tf
INPUT_NODE = 784
OUTPUT_NODE = 10
LAYER1_NODE = 500

def get_weight(shape, regularizer):
    w = tf.Variables(tf.truncated_normal(shape,stddev=0.1))
    if regularizer != None: tf.add_to_collection('losses', tf.contrib.layers.l2_
    return w
def get_bias(shape):
    b = tf.Variable(tf.zeros(shape))
    return b

def forward(x, regularizer):
    # []_{784*500}
    w1 = get_weight([INPUT_NODE, LAYER1_NODE], regularizer)
    b1 = get_bias([LAYER1_NODE])
#     relu 非线性函数的，修正线性单元
    y1 = tf.nn.relu(tf.matmul(x, w1) + b1)

# 这里w= get_weight([500, 784] 因为x=[None,784]
    w2 = get_weight([LAYER1_NODE, OUTPUT_NODE], regularizer)
    b2 = get_bias([OUTPUT_NODE])
    y = tf.matmul(y1, w2) + b2
    return y
```

5.3 反向传播(最小化损失函数)

反向传播过程就是利用梯度下降算法进行最佳的各隐藏层的权值和偏置的优化过程。

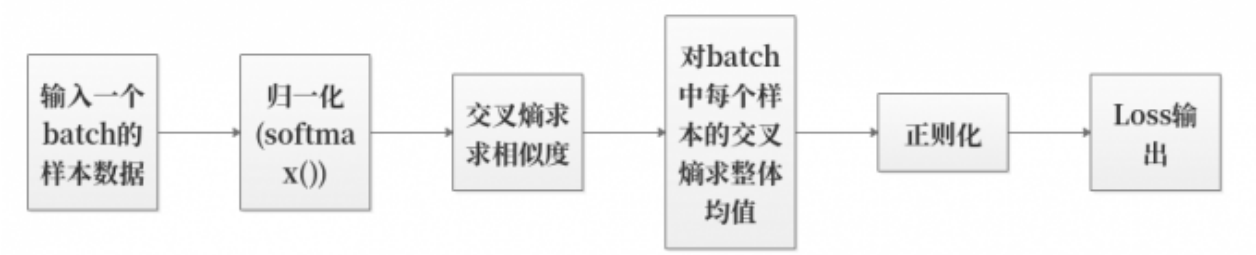
一般选择的梯度优化算法有：

- 批量梯度优化算法
- 随机梯度优化
- 自适应梯度优化
- 等等

train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)

整个流程

前向传播输出结果--> 归一化(softmax()) ---> 求取每回batch中样本的输出相似度（交叉熵求求相似度 sparse_softmax_cross_entropy_with_logits）---> 得到每个batch中每个样本中的交叉熵均值 (tf.reduce_mean()) --> 正则化输出--> loss(loss = cem + tf.add_n(tf.get_collection('losses')))



相关函数解释：

- tf.add_n(p1, p2, p3...)

实现列表中的元素相加

```
input1 = tf.constant([1.0, 2.0, 3.0])
input2 = tf.Variable(tf.random_uniform([3]))
output = tf.add_n([input1, input2]) # = input1 + input2
```

5.3.1 Softmax(归一化前向传播输出结果)

之前我们选择的代价函数为均方误差函数，表达式为

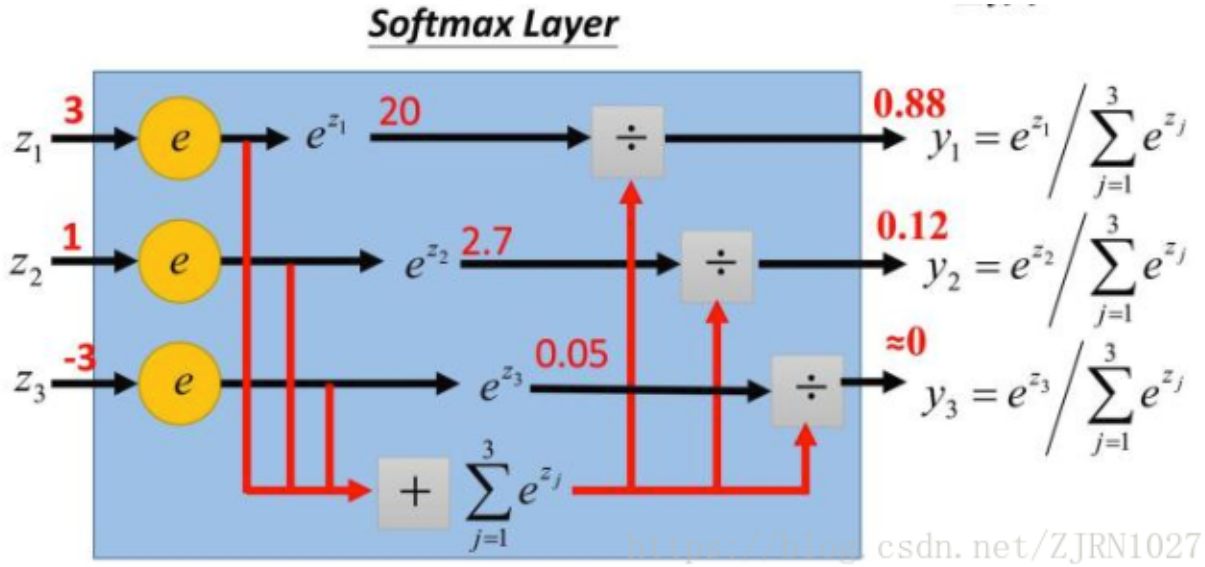
$$loss = \frac{(y - y_)^2}{n}$$

实现的代码如下：

```
loss_mse = tf.reduce_mean(tf.square(y_ - y))
train_step =
tf.train.GradientDescentOptimizer(0.001).minimize(loss_mse)
```

Softmax 归一化操作，实现输出各通道的数值编程, 概率和为1的概率表达形式

$$y_j = \frac{e^{z^j}}{\sum_i e^{z_i}}$$



如上图所示，softmax函数的作用是归一化，将所有的输出以概率和为1的方式，将输出结果用概率表示。

5.3.2 交叉熵(比较输出和样本标签的相似度)

本部分我们不简单的使用均方误差函数，而是使用交叉熵的方式实现

$$H_{y\_(y)} = -\sum_i y_i \_ log(y_i)$$

$y_$为样本的标签，而y为前向传播的输出计算结果。

例子：

$$y = [0.005899750.87560060.11849965]$$

,也就是如果batch为1(输入的训练数据，只输入1组含有一个样本的数据，一般训练一个batch会包含多个batch所以，会对本次的batch中的多个交叉熵求取均值“ tf.reduce_mean(ce)”)

$$\begin{aligned} H_{y\_(y)} &= -0 \times log(0.00589975) - 1 \times log(0.8756006) - 0 \times log(0.11849965) \\ &= 0.6355716 - 0 \times log(0.00589975) - 1 \times log(0.8756006) - 0 \times log(0.11849965) \\ &= 0.6355716 - 0 \times log(0.00589975) - 1 \times log(0.8756006) - 0 \times log(0.11849965) \\ &= 0.6355716 \end{aligned}$$

实现的代码如下

```
    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'))
```

这里的logits=y, 是前向传播中的3层神经网络的输出层结果$y_{[200\times10]}$

- tf.nn.sparse_softmax_cross_entropy_with_logits()与tf.nn.softmax_cross_entropy_with_logits的差别

  sparse_softmax_cross_entropy_with_logits(_sentinel=None, labels=None, logits=None, name=None)

唯一的区别是sparse的labels是int类型，而非sparse的labels是one-hot类型。

- tf.argmax(input,axis)根据axis取值的不同返回每行或者每列最大值的索引

axis = 0:

axis=0时比较每一列的元素，将每一列最大元素所在的索引记录下来，最后输出每一列最大元素所在的索引数组。

```
test[0] = array([1, 2, 3])
test[1] = array([2, 3, 4])
test[2] = array([5, 4, 3])
test[3] = array([8, 7, 2])
# output   :    [3, 3, 1]
```

axis = 1:

axis=1的时候，将每一行最大元素所在的索引记录下来，最后返回每一行最大元素所在的索引数组。

```
test[0] = array([1, 2, 3])  #2
test[1] = array([2, 3, 4])  #2
test[2] = array([5, 4, 3])  #0
test[3] = array([8, 7, 2])  #0
```

5.3.3 batch个样本的交叉熵后整体均值计算

cem = tf.reduce_mean(ce)

---

reduce_mean()函数的用法举例

```
import tensorflow as tf

x = [[1,2,3],
      [1,2,3]]

xx = tf.cast(x,tf.float32)

mean_all = tf.reduce_mean(xx, keep_dims=False)
mean_0 = tf.reduce_mean(xx, axis=0, keep_dims=False)
mean_1 = tf.reduce_mean(xx, axis=1, keep_dims=False)


with tf.Session() as sess:
    m_a,m_0,m_1 = sess.run([mean_all, mean_0, mean_1])

print m_a    # output: 2.0
print m_0    # output: [ 1.  2.  3.]
print m_1    #output:  [ 2.  2.]
```

5.3.4 正则化

在前向传播中将正则化对象设置成ω

L^2参数正则化

$$\Omega(\theta) = \frac{1}{2}||\omega||_2^2$$

将上面的公式整理可得到

$$\widetilde{J}(\omega; X, y) = J(\omega; X, y) + \alpha\Omega(\theta)$$
$$= J(\omega; X, y) + \frac{\alpha}{2}\omega^{\top}\omega$$

与之对应的梯度为：

$$\nabla_{\omega}\widetilde{J}(\omega; X, y) = \alpha\omega + \nabla_{\omega}J(\omega; X, y)$$

得到L^2参数正则化的参数ω参数更新的表达式为

$$\omega \leftarrow \omega - \epsilon\nabla_{\omega}\widetilde{J}(\omega; X, y)$$

正则化后的代价函数$J(\omega; X, y)$实现代码为

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

```
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.argm
    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)

    train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss,

    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')

    saver = tf.train.Saver()

    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_d
            if i % 1000 == 0:
                print("After %d training step(s), loss on training batch is %g."
```

```
                saver.save(sess, os.path.join(MODEL_SAVE_PATH, MODEL_NAME), glob
```

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

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

5.4 训练过程

5.4.1 指数衰减学习率

训练过程我们使用指数衰减学习率实现，梯度下降算法中的学习率以指数形式衰减，可以提高训练的效率。

指数衰减学习率的公式为：

$$dlr = lr * dr^{\frac{globals}{decays}}$$

各个简写分别为

简写	对应的全程	代表的含义
dlr	decayed_learning_rate	指数衰减学习率
lr	learining_rate	为学习率初始设定值
dr	decay_rate	为学习率的衰减率
globals	global_step	记录了当前训练轮数
decays	decay_steps	多少轮更新一次学习率

也可以用如下的表达式表示指数衰减学习率的表示

```
decayed_learning_rate=learining_rate*decay_rate^(global_step/decay_
steps)
```

具体的实现代码如下：

```
def exponential_decay(learning_rate,
                      global_step,
                      decay_steps,
                      decay_rate,
                      staircase=False,
                      name=None):
```

函数的的使用

```
LEARNING_RATE_BASE = 0.8
BATCH_SIZE = 200
LEARNING_RATE_DECAY = 0.99
global_step = tf.Variable(0, trainable=False)
train_step =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss,
global_step=global_step)
```

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

5.4.2 梯度下降并开始迭代训练

上面步骤确定了学习率，并且5.2 节也确定了前向传播，5.3节确定了代价函数，此时可以进行梯度下降算法的迭代训练模型参数

模型的训练，此处选择梯度下降算法，开始训练模型参数，代码如下

```
train_step =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss,
global_step=global_step)
```

5.4.3 滑动平均优化模型参数

滑动平均可以使训练的模型能在测试数据上更加健壮(好的泛化能力、鲁棒性好),其实滑动平均模型，主要是通过控制衰减率来控制参数更新前后之间的差距，从而达到减缓参数的变化值（如，参数更新前是5，更新后的值是4，通过滑动平均模型之后，参数的值会在4到5之间），如果参数更新前后的值保持不变，通过滑动平均模型之后，参数的值仍然保持不变。

函数的原型如下

```
tf.train.ExponentialMovingAverage(decay, num_updates=None,
zero_debias=False,name="ExponentialMovingAverage")
```

参数：

decay：实数类型，衰减率。

num_updates：可选，为轮数，设置这个参数之后，将会通过min(decay, (1 + num_updates) / (10 + num_updates))函数，从中选择最小值做为衰减率。

返回值：

ExponentialMovingAverage对象，通过对象调用apply方法可以通过滑动平均模型来更新参数。

计算公式：

shadow_variable = decay * shadow_variable + (1-decay) * variable

计算公式中的shadow_variable为影子变量，也就是变量在更新之前的值，variable是变量现在的值，可能这样说不是很明白，下面用TensorFlow的程序来实现滑动平均模型。

本次模型训练使用的滑动平均算法调用如下：

```
ma_decay = 0.99
ema = tf.train.ExponentialMovingAverage(decay = ma_decay,
                                        num_updates = global_step)
ema_op = ema.apply(tf.trainable_variables())
```

总结下；

滑动平均值的参数更新，比如对权重ω_1更新，开始轮数为num_updates= 0 次时，ω_1初值设置0, 第1轮更新为variable=1($\omega_1 = 1$); decay=0.99 ;此时有：

ω_1 =
$min(ma\_decay, \frac{1+0}{10+0}) * 0 + (1 - min(1 - min(decay, \frac{1+0}{10+0}) * 1)) = 0.1 * 0 + (1 - 0.9) * 1 =$

训练过程的代码如下：

```
#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_DE
        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_P
                if ckpt and ckpt.model_checkpoint_path:
                    saver.restore(sess, ckpt.model_checkpoint_path)
                    global_step = ckpt.model_checkpoint_path.split('/')[-1].spli
                    accuracy_score = sess.run(accuracy, feed_dict={x: mnist.test
                    print("After %s training step(s), test accuracy = %g" % (glo
                else:
                    print('No checkpoint file found')
                    return
            time.sleep(TEST_INTERVAL_SECS)

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

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

5.5 模型训练后的断点保存和恢复

5.5.1 保存模型

在反向传播中，为防止模型训练过程中由于硬件或者断电的情况下，使模型前期训练全部失效，我们一般会在训练了一定间隔轮数后保存当前训练后的神经网络模型。

保存模型后会产生3个文件。

- 保存当前图解钩的.meta文件
- 保存当前参数名的.index文件
- 保存当前参数的.data文件

具体的保存过程代码，如下

```
MODEL_SAVE_PATH = "./model/"
MODEL_NAME = "mnist_model"
saver = tf.train.Server()# 实例化Sever对象

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)
```

os.path.join() 是将括号中的参数都变成路径的字符形式。

比如，当前的global_step=100，则save后的文件名是“./model/mnist_model-100.meta、./model/mnist_model-100.index ”等

5.5.2 恢复模型

恢复训练中的神经网络模型

```
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)
```

恢复模型中调用的滑动平均模型参数

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

5.6 验证阶段神经网络模型准确性的评估方法

```
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y_, 1))# 返回
的都是bool型变量
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))#
将bool型转化为float型，然后求均值
```

对训练后的模型进行准确度评判

神经网络的测试验证阶段，只需要前向传播的模型，不需要对模型参数进行梯度下降的迭代优化，具体的过程入下

```
    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_DEC
```

```
AY)
        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))

        # 开始读取保存后的神经网络模型参数，恢复到我们复制Graph()中 ，然后开
始准确性判断
        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
```

5.7 训练后的模型使用

本部分的过程和测试部分的过程稍微不同的是，需要引入输入待识别的图片的图像预处理工作。

预处理工作，应该是如下步骤

- 图片的灰度化（0-255取值范围）
- 将灰度化处理后的图片像素取值进行缩放（0-1的取值范围np.multiply(nm_arr, 1.0/255.0)）
- 图像的reshape

复制神经网络计算图

tf.Graph().as_default()

然后重保存的神经网络训练参数文件中恢复模型参数

（和测试程序一样，这部分也不需要反向传播优化模型参数）

然后就是用前向传播计算的流程一样

```
        x = tf.placeholder(tf.float32, [None,
mnist_forward.INPUT_NODE])
        y = mnist_forward.forward(x, None)
        preValue = tf.argmax(y, 1)
```

具体的整体代码如下部分所示

```
#coding:utf-8
```

```
import tensorflow as tf
import numpy as np
from PIL import Image
import mnist_backward
import mnist_forward

def restore_model(testPicArr):
    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)

        variable_averages = tf.train.ExponentialMovingAverage(mnist_backward.MOV
        variables_to_restore = variable_averages.variables_to_restore()
        saver = tf.train.Saver(variables_to_restore)

        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)

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

def pre_pic(picName):
    img = Image.open(picName)
    reIm = img.resize((28,28), Image.ANTIALIAS)
    im_arr = np.array(reIm.convert('L'))
    threshold = 50
    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

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)

def main():
    application()

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