人工智能导论实验报告

张钰晖
2015011372, yuhui-zh15@mails.tsinghua.edu.cn, 185-3888-2881

目录

1 问题描述

日常生活中，我们经常需要在填写信件、银行开卡填写表单的时候手写大量的数字，如身份证号、手机号、邮编等等。事实上，专业人员在过去录入这些信息非常费时费力，而且还容易出现错误。现在请你设计一个手写体数字照片智能分类程序，省去人工识别 0-9 数字的麻烦。

在 Kaggle 竞赛平台上有 Digit Recognizer https://www.kaggle.com/c/digit-recognizer 的经典任务，并且提供了上万张手写体数字照片的灰度数据（可以借助 Matlab 还原照片）。目前，该任务已经被许多科研人员设计的人工智能程序完美解决，达到 100% 测试精度。请大家在熟悉 Kaggle 平台的使用之后，直接通过该平台完成 Digit Recognizer 的竞赛任务，与一线科学家一较高下；并将你在从事竞赛过程中的心得体会、实验流程，以及最佳测评结果截图写入实验报告，同时在附录中递交你的代码。

2 模型选择

根据 Tensorflow 官方教程中 MNIST FOR EXPERT 中的 CNN 部分，和 Lenet-5 的结构，以及相关论文文献，最终选择了模型。

一般卷积神经网络均为 2 层卷积层，然后连接 2-3 层全连接层，中间层用 Relu 函数激活，输出层用 Softmax 函数激活。

实验分别以两大类方案进行实验，并对每类方案进行了参数调节，两大类方案如下，参数调节见后文。

2.1 模型 1（MNIST FOR EXPERT）

输入层：长度为 784 的向量，展平为 28x28。(28x28)

第一层：卷积层一。用 32 个卷积核进行 Padding=SAME 类型卷积，进行 2*2 最大池化，Relu 函数激活。(32@14x14)

第二层：卷积层二。用 64 个卷积核进行 Padding=SAME 类型卷积，进行 2*2 最大池化，Relu 函数激活。(64@7x7)

第三层：全连接层一。用 1024 个神经元与上一层进行全连接，Relu 函数激活。(1024)

输出层：全连接层二。用 10 个神经元与上一层进行全连接，Softmax 函数激活。(10)

2.2 模型 2（LeNet-5）

输入层：长度为 784 的向量，展平为 28x28。(28x28)

第一层：卷积层一。用 6 个卷积核进行 Padding=VALID 类型卷积，进行 2*2 最大池化，Relu 函数激活。(6@12x12)

第二层：卷积层二。用 16 个卷积核进行 Padding=VALID 类型卷积，进行 2*2 最大池化，Relu 函数激活。(16@4x4)

第三层：全连接层一。用 120 个神经元与上一层进行全连接，Relu 函数激活。(120)

第四层：全连接层二。用 84 个神经元与上一层进行全连接，Relu 函数激活。(84)

输出层：全连接层三。用 10 个神经元与上一层进行全连接，Softmax 函数激活。(10)

下图展示了图片大小为 32x32 时的 LaNet-5 神经网络

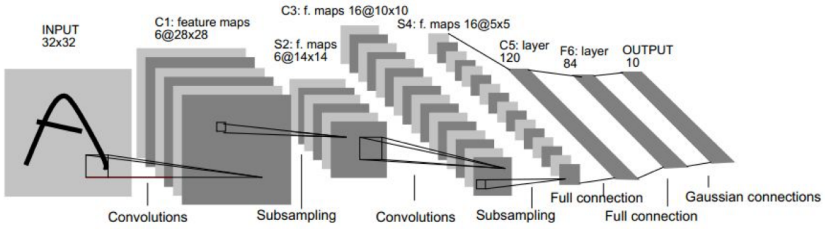

图 1: LaNet-5 神经网络

3 细节实现

作为开始，我们读进来所需要的数据。

train.csv 为训练集文件，包含 42000 行和 785 列。每一行代表一张手写的数字图片，第一列为对应数字的标签，剩下列为 784 个像素灰度值 (0255)，对应 28*28 的图片大小。

test.csv 为测试集文件，包含 18000 行和 784 列。每一行为 784 个像素灰度值 (0255)，对应 28*28 的图片大小。

将训练集样本部分抽取出来作为验证集，以检测神经网络训练结果。

至此我们得到了训练集、验证集和测试集。

下图展示了其中一个训练样本。

图 2: 一个训练样本

建立神经网络结构，进行卷积、池化。下图展示了卷积和池化的原理。

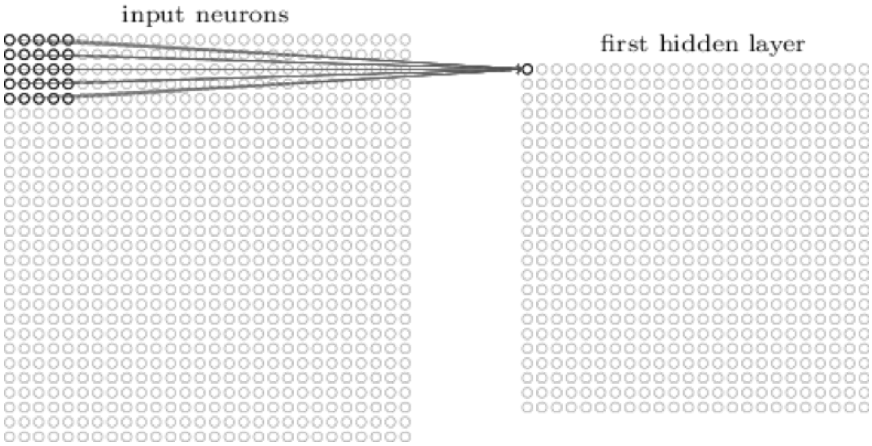

图 3: 卷积

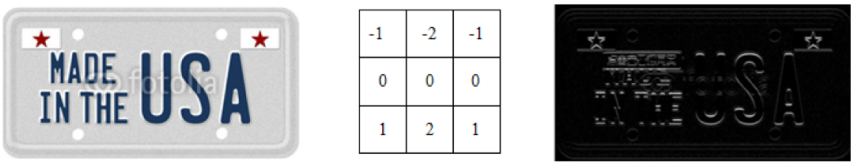

图 4: 卷积核

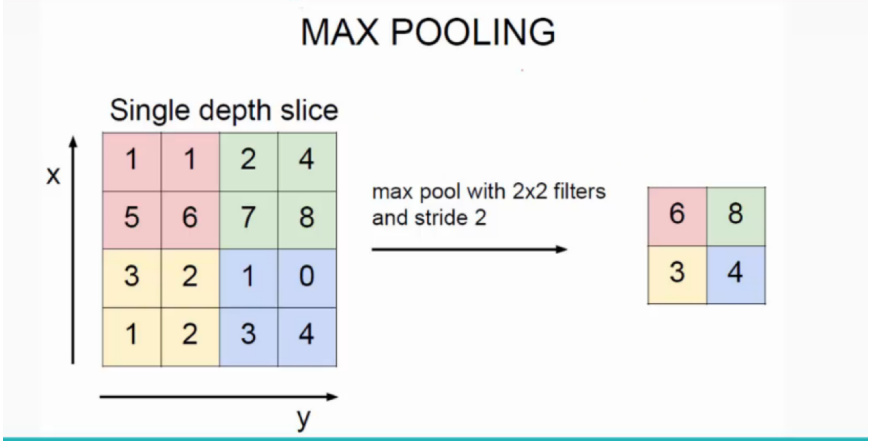

图 5: 池化

定义交叉熵（Cross Entropy）后，用改进的梯度下降法（Adam Optimizer）去降低交叉熵，训练 50000 次，每批次（Batch）投入 50 个训练数据。

因为所用神经网络神经元非常多，通过训练曲线可以看出，50000 次保证不至于欠拟合。同时为了防止过拟合，每 5000 次用验证集检验一次结果，如果结果优于上次结果，则保存本次结果，否则弃掉本次结果。

下图展示了某一次训练时的训练准确率曲线。

训练次数	准确率
0	0.0629771
500	0.935115
1000	0.958015
1500	0.96374
2000	0.975191
3000	0.977099
4000	0.984733
5000	0.98855
6000	0.990458
7000	0.992366
10500	0.992366
11000	0.994275
13500	0.996183
45000	0.998091

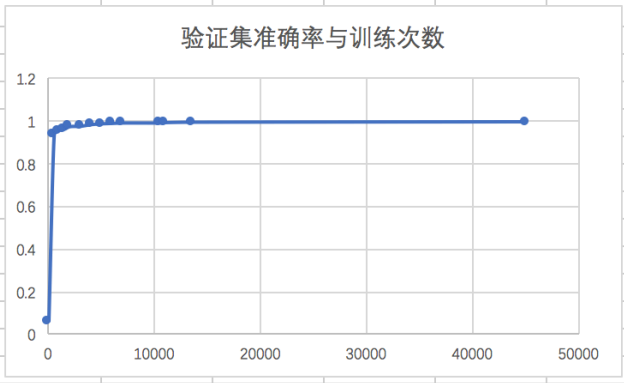

图 6: 训练准确率曲线

由训练曲线可以看出，该神经网络收敛非常快，在训练 500 次时验证集准确率便已经达到 93.5%，之后收敛逐渐变慢，最后趋于稳定。

最终可以看出，在训练集上，准确率为 99.809%，和实际提交结果 99.143% 相似，略低的原因可能是测试集噪声比较大。

为了提高准确率，我们可以加入更多的训练元素，比如著名的 MNIST 样本集结构和本次任务图像结果基本完全类似，引入 MNIST 样本集进行训练，同样训练 50000 次，准确率为 99.871%，截止写实验报告时，排名第 52。

4 实验结果

4.1 模型的选择

下文将对比模型 1 和模型 2 的结果，训练次数为 50000 次。

(1) 模型 1 (MNIST FOR EXPERT):

训练曲线如下：

训练次数	准确率
0	0.0629771
500	0.935115
1000	0.958015
1500	0.96374
2000	0.975191
3000	0.977099
4000	0.984733
5000	0.98855
6000	0.990458
7000	0.992366
10500	0.992366
11000	0.994275
13500	0.996183
45000	0.998091

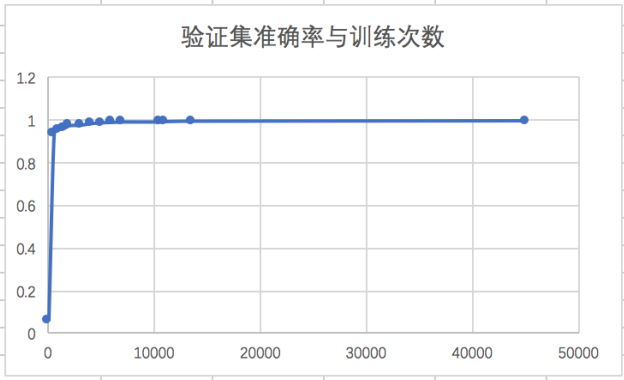

图 7: 模型 1（MNIST FOR EXPERT）训练准确率曲线

最终准确率：99.143%

（2） 模型 2（LeNet-5）:

训练曲线如下：

训练次数	准确率
0	0.0990991
500	0.735736
1000	0.858859
1500	0.896396
2000	0.917418
3000	0.930931
4000	0.944445
5000	0.947447
7000	0.965466
9000	0.966967
12000	0.968468
15000	0.971471
20000	0.972973
25000	0.977477
30000	0.980481
40000	0.983484
49000	0.987988

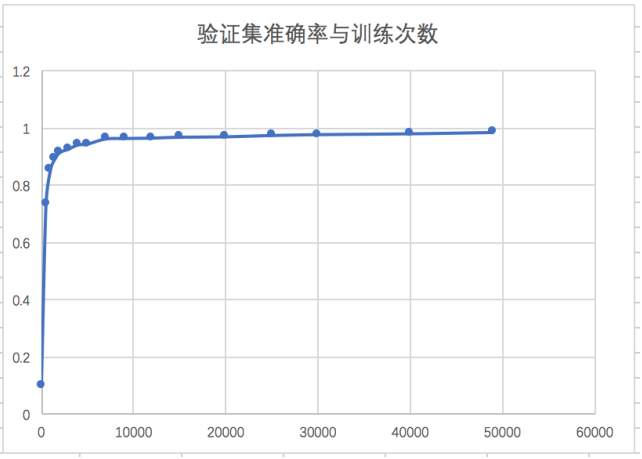

图 8: 模型 2（LeNet-5）训练准确率曲线

最终准确率：98.198%

由训练准确率曲线可以清楚地看出，模型 1 不仅收敛更快，而且最终准确率更高，实际测试中，模型 1 训练时间也较短，故以下各测试均基于模型 1 进行测试。

4.2 卷积层层数的选择

以下测试均基于模型 1（MNIST FOR EXPERT），仅改变卷积层层数，训练次数为 50000 次。

（1） 卷积层层数为 1 时，训练 10000 步时准确率为 98.4733%，最终准确率 99.0458%。

（2） 卷积层层数为 2 时，即为原始模型，训练 10000 步时准确率为 99.2366%，最终准确率 99.8091%。

（3） 卷积层层数为 3 时，训练 10000 步时准确率为 99.0458%，最终准确率 99.4275%。

三者准确率随训练次数收敛速度近乎相等，但是训练速度 1 层快于 2 层快于 3 层，消耗资源 1 层少于 2 层少于 3 层。

可见，卷积层层数为 2 为最合适的参数。层数减小训练速度加快、消耗资源减少，但准确率不高；层数增加训练速度减慢，消耗资源增加，而且准确率也并没有提高。

全连接层一般也选择 2-3 层，减少或增加层数可预测结果应该类似，由于时间有限，不再进行实验。

4.3 训练集的选择

以下测试均基于模型 1（MNIST FOR EXPERT），卷积层层数为 2，仅改变训练集，训练次数为 50000 次。

（1） 采用原始训练集时，验证集准确率为 99.8071%，Kaggle 测试最终准确率为 99.143%，排名约为 350 名。

（2） 采用 MNIST 训练集，验证集准确率为 99.6183%，Kaggle 测试最终准确率为 99.871%，排名为 52 名。

4.4 最终模型与竞赛成绩

最终采用模型 1（MNIST FOR EXPERT），卷积层层数为 2，训练次数为 50000 次，两种训练集结果如图所示：

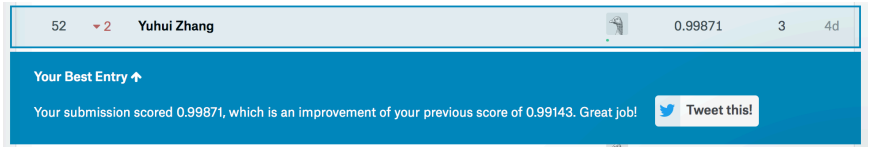

图 9: 排名和提交次数

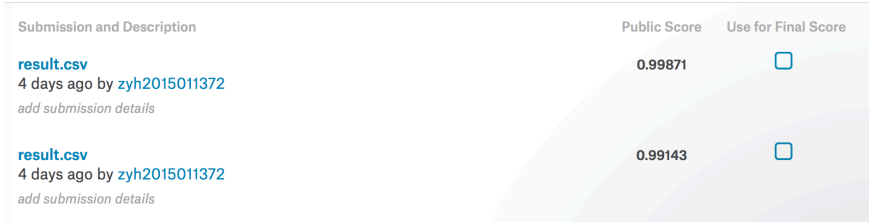

图 10: 不同数据集训练下的准确率

准确率最高时提交次数：3 次

最终提交次数：4 次

5 实验心得

通过本次实验，我大大加深了对卷积神经网络的理解，初步掌握了神经网络的参数选择技巧，也初步学习了 Tensorflow 框架的使用方法，进入了高级机器学习的新世界。同时也学习了 LaTex 排版技巧。

在这次实验中，我充分感受到了神经网络的神奇之处，通过这样的连接，识别准确率竟然可以达到 99.871%，接近 100%，而且是在噪声很高的数据集上进行的测试，实际表现应该会更好。

但是这只是一个入门，神经网络的高级技巧和参数设置技巧我还没有完全掌握，还需要进一步的练习。

值得思考的是，LeNet-5 是上课讲的专门针对手写字母识别的卷积神经网络，我本以为效果会很好，结果却并不够理想，可能是卷积核和神经元太少，没有充分提取图像的特征。

另外遗憾的是，由于期末复习时间太紧张，没有时间学习其余的神经网络，也没有时间对比训练结果，比如 LSTM，希望暑假有机会可以进一步尝试。

6 后期工作

我认为可以通过以下方式进行改进，留给以后的工作。

（1）对训练集进行处理，例如适当旋转一定角度，适当放大或缩小，从而获得更多的训练数据，使得训练更为充分。

（2）对测试集进行处理，例如去噪声，比如有的测试数据有一个黑点，可以采用深度优先搜索和设置阈值的方法得以实现。

（3）采用 RNN、LSTM 等不同的神经网络进行训练与测试，对比结果。

7 源代码

运行方式：

将文件 train.csv 和 test.cs 拷贝至 Python 程序所在目录, 在命令行中输入 python main.py 即可。

输出结果为文件 result.csv，同时会保存训练出的模型，可以直接使用 Tensorflow 中的模块读取训练好的模型。

```
#coding=utf-8
import numpy
import csv
import tensorflow as tf

train_pixel = [] # 训练集
train_label = [] # 训练标签
test_pixel = [] # 测试集
train_count = -2 # 训练集大小
validation_count = -2 # 验证集大小
test_count = -2 # 测试集大小
now = 0 # 全局计数器

# 加载训练集
def load_train_data():
   global train_pixel, train_label, train_count, validation_count
   # 从train.csv文件读取数据，train.csv每行为785个整数，第1个为标签，后面784个为对应像素灰度（0-255）
   csvfile = file('train.csv', 'rb')
   reader = csv.reader(csvfile)
   for line in reader:
      train_count = train_count + 1
      if (train_count == -1):
         continue
      # 将标签转换为对应的10维向量
      train_label.append(numpy.zeros(10))
      train_label[train_count][int(line[0])] = 1.0
      train_pixel.append([])
      for i in xrange(1, len(line)):
```

```
            train_pixel[train_count].append(float(line[i]) / 255) # 灰度归一化
    csvfile.close()
    # 从训练集中抽取部分作为验证集
    validation_count = int(0.0125 * train_count);
    train_count -= validation_count;

# 加载测试集
def load_test_data():
    global test_pixel, test_count
    # 从test.csv文件读取数据，test.csv每行为784个整数，为对应像素灰度(0-255)
    csvfile = file('test.csv', 'rb')
    reader = csv.reader(csvfile)
    for line in reader:
        test_count = test_count + 1
        if (test_count == -1):
            continue
        test_pixel.append([])
        for i in xrange(0, len(line)):
            test_pixel[test_count].append(float(line[i]) / 255) # 灰度归一化
    csvfile.close()

# 保存结果
def save_result(result):
    # 保存至result.csv文件
    csvfile = file('result.csv', 'wb')
    writer = csv.writer(csvfile)
    # csv文件表头
    writer.writerow(['ImageId', 'Label'])
    # 每行保存序号(从1开始)和对应结果
    for i in xrange(0, len(result)):
        writer.writerow([i + 1, result[i]])
    csvfile.close()

# 获取下一批训练集，大小为num
def next_batch(num):
    global now
    label = numpy.array(train_label[now: now + num])
    pixel = numpy.array(train_pixel[now: now + num])
    now = now + num
    # 如果训练集全部用完，重新开始用
    if (now > train_count): now = now - train_count
    return (pixel, label)

# 获取验证集
def validation_set():
    label = numpy.array(train_label[train_count: train_count + validation_count])
    pixel = numpy.array(train_pixel[train_count: train_count + validation_count])
```

```
    return (pixel, label)

# 随机初始化大小为shape的神经元
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev = 0.1)
    return tf.Variable(initial)

# 生成大小为shape的偏置变量
def bias_variable(shape):
    initial = tf.constant(0.1, shape = shape)
    return tf.Variable(initial)

# 卷积核
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides = [1, 1, 1, 1], padding = 'SAME')

# 2x2最大池化
def max_pool(x):
    return tf.nn.max_pool(x, ksize = [1, 2, 2, 1], strides = [1, 2, 2, 1], padding = 'SAME')

# 训练神经网络
def train_nn():
    sess = tf.InteractiveSession() # 建立session

    x = tf.placeholder(tf.float32, shape = [None, 784]) # 神经网络输入
    y_ = tf.placeholder(tf.float32, shape = [None, 10]) # 神经网络输出
    x_image = tf.reshape(x, [-1, 28, 28, 1]) # 将784像素重置为28x28像素

    W_conv1 = weight_variable([5, 5, 1, 32]) # 第一层卷积层
    b_conv1 = bias_variable([32]) # 第一层偏置
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1) # 第一层卷积，relu激活
    h_pool1 = max_pool(h_conv1) # 第一层2x2池化

    W_conv2 = weight_variable([5, 5, 32, 64]) # 第二层卷积层
    b_conv2 = bias_variable([64]) # 第二层偏置
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2) # 第二层卷积，relu激活
    h_pool2 = max_pool(h_conv2) # 第二层2x2池化

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64]) # 将第二层重置shape

    W_fc1 = weight_variable([7 * 7 * 64, 1024]) # 第三层全连接层
    b_fc1 = bias_variable([1024]) # 第三层偏置
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1) # 第三层计算，relu激活

    keep_prob = tf.placeholder(tf.float32) # 保持概率
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob) # 适度丢弃
```

```
    W_fc2 = weight_variable([1024, 10]) # 第四层全连接层
    b_fc2 = bias_variable([10]) # 第四层偏置
    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2 # 第四层计算，保留原始计算结果

'''
    # LaNet-5模型，如果换用模型直接把上述代码替换为下段代码即可

    # 第一卷积层
    W_conv1 = weight_variable([5, 5, 1, 6])
    b_conv1 = bias_variable([6])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
    h_pool1 = max_pool_2x2(h_conv1)

    # 第二卷积层
    W_conv2 = weight_variable([5, 5, 6, 16])
    b_conv2 = bias_variable([16])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
    h_pool2 = max_pool_2x2(h_conv2)

    # 第一全连接层
    W_fc1 = weight_variable([4 * 4 * 16, 120])
    b_fc1 = bias_variable([120])
    h_pool2_flat = tf.reshape(h_pool2, [-1, 4 * 4 * 16])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

    # 第二全连接层
    W_fc2 = weight_variable([120, 84])
    b_fc2 = bias_variable([84])
    h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
    h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

    # 第三全连接层
    W_fc3 = weight_variable([84, 10])
    b_fc3 = bias_variable([10])
    y_conv = tf.matmul(h_fc2_drop, W_fc3) + b_fc3
'''

    cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels = y_, logits =
        y_conv)) # 交叉熵计算
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy) # 最小化交叉熵
    prediction = tf.argmax(y_conv, 1) # 预测结果
    correct_prediction = tf.equal(prediction, tf.argmax(y_, 1)) # 结果是否正确
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32)) # 准确率

    sess.run(tf.global_variables_initializer()) # 初始化所有变量
```

```
    saver = tf.train.Saver() # 参数保存/加载器

    with tf.device('/cpu:0'): # 指定在某设备上运行（如CPU、显卡）
        best_accuracy = 0.0 # 在验证集最高准确率
        best_num = 0 # 对应最高准确率的训练次数
        for i in range(1000): # 训练50000次
            batch = next_batch(50) # 每次提供50个训练样本
            # 每训练500次用验证集测试准确率，若高于最高准确率，记录并保存结果
            if i % 500 == 0:
                [validation_pixel, validation_label] = validation_set()
                train_accuracy = accuracy.eval(feed_dict = {x: validation_pixel, y_:
                    validation_label, keep_prob: 1.0})
                if (train_accuracy >= best_accuracy):
                    best_accuracy = train_accuracy
                    best_num = i
                    saver.save(sess, './model')
                print("step %d, training accuracy %g, best accuracy %g, best num %d" % (i,
                    train_accuracy, best_accuracy, best_num))
            train_step.run(feed_dict = {x: batch[0], y_: batch[1], keep_prob: 0.5}) # 训练

        load_path = saver.restore(sess, './model') # 加载训练最好结果
        result = [] # 结果向量
        # 每次记录500个结果，避免消耗过多资源（尤其是使用显卡时）
        i = 0
        while (i <= test_count):
            result[i: i + 500] = prediction.eval(feed_dict = {x: test_pixel[i: i + 500], keep_prob
                : 1.0})
            i += 500
        save_result(result) # 保存结果文件

# 主程序
def main():
    load_train_data() # 加载训练集
    load_test_data() # 加载测试集
    train_nn() # 训练神经网络

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

参考文献

[1] 人工智能导论, 马少平, 2017.

[2] Python 机器学习及实践：从零开始通往 Kaggle 竞赛之路, 范淼, 李超, 2017.