

深度学习

Tensorflow

DAY03

模型保存与加载

模型保存与加载

模型保存与加载

什么是模型保存与加载

模型保存与加载API

案例1：模型保存/加载

模型保存与加载

什么是模型保存与加载

- 模型训练可能是一个很长的过程，如果每次执行预测之前都重新训练，会非常耗时，所以几乎所有人工智能框架都提供了模型保存与加载功能，使得模型训练完成后，可以保存到文件中，供其它程序使用或继续训练

非会员专享



模型保存与加载API

- 模型保存与加载通过`tf.train.Saver`对象完成，实例化对象：

```
saver = tf.train.Saver(var_list=None, max_to_keep=5)
```

- `var_list`: 要保存和还原的变量，可以是一个dict或一个列表
- `max_to_keep`: 要保留的最近检查点文件的最大数量。创建新文件时，会删除较旧的文件（如`max_to_keep=5`表示保留5个检查点文件）

- 保存：`saver.save(sess, '/tmp/ckpt/model')`
- 加载：`saver.restore(sess, '/tmp/ckpt/model')`



案例1：模型保存/加载

```
30 saver = tf.train.Saver() #实例化Saver
31 with tf.Session() as sess: # 通过Session运行op
32     sess.run(init_op)
33     print("weight:", weight.eval(), " bias:", bias.eval()) # 打印初始权重、偏移值
34     fw = tf.summary.FileWriter("../summary/", graph=sess.graph) # 指定事件文件
35     # 训练之前，加载之前训练的模型，覆盖之前的参数
36     if os.path.exists("../model/linear_model/checkpoint"):
37         saver.restore(sess, "../model/linear_model/")
38
39     for i in range(500): # 循环执行训练
40         sess.run(train_op) # 执行训练
41         summary = sess.run(merged) # 运行合并后的tensor
42         fw.add_summary(summary, i)
43         print(i, ":", i, "weight:", weight.eval(), " bias:", bias.eval())
44
45     saver.save(sess, "../model/linear_model/")
```



案例1：模型保存/加载（续）

- 从执行结果可以看出，如果模型之前经过训练，直接从之前的参数值开始执行迭代，而不是从第一次给的初始值开始

```
0 : 0 weight: [[2.0043766]] bias: 4.9921255
1 : 1 weight: [[2.0041678]] bias: 4.992144
2 : 2 weight: [[2.0041957]] bias: 4.992287
3 : 3 weight: [[2.0040917]] bias: 4.9923406
4 : 4 weight: [[2.003974]] bias: 4.9924107
5 : 5 weight: [[2.0041006]] bias: 4.9926014
6 : 6 weight: [[2.003941]] bias: 4.9926243
7 : 7 weight: [[2.0038793]] bias: 4.9927034
8 : 8 weight: [[2.0038292]] bias: 4.992787
```



数据读取

数据读取

数据读取

文件读取机制

文件读取API

案例2: CSV文件读取

图片文件读取

案例3: 图片文件读取

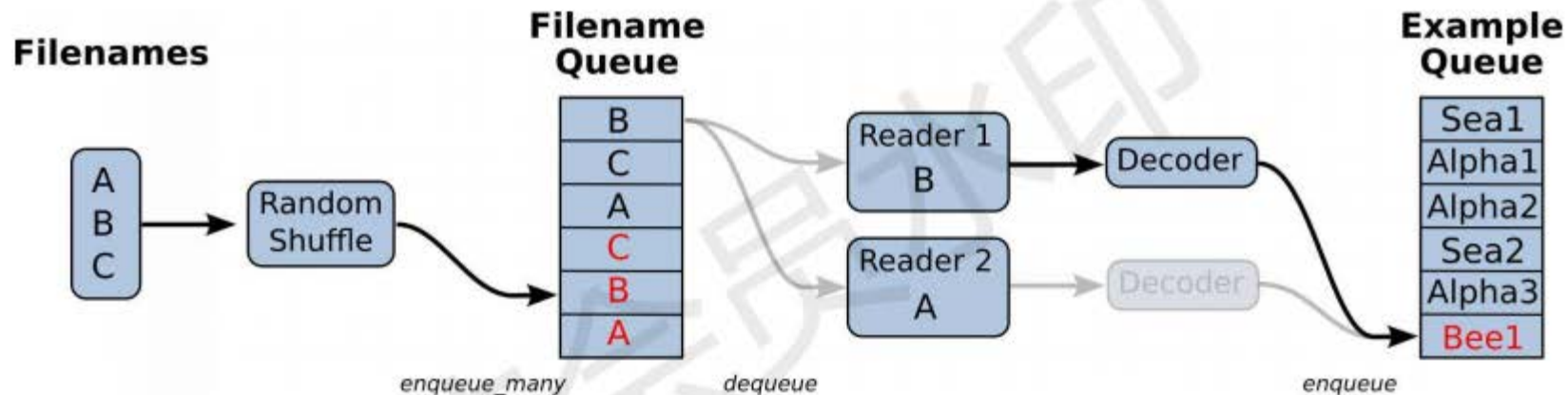
数据读取

文件读取机制

- TensorFlow文件读取分为三个步骤：
 - ✓ 第一步：将要读取的文件放入文件名队列
 - ✓ 第二步：读取文件内容，并实行解码
 - ✓ 第三步：批处理，按照指定笔数构建成一个批次取出



文件读取机制（续）



文件读取API

- 文件队列构造：生成一个先入先出的队列，文件阅读器会需要它来读取数据

`tf.train.string_input_producer(string_tensor, shuffle=True)`

- `string_tensor`: 含有文件名的一阶张量
- `shuffle`: 是否打乱文件顺序

返回：文件队列



文件读取API（续1）

➤ 文件读取：

- ✓ 文本文件读取：`tf.TextLineReader`
 - 读取CSV文件，默认按行读取
- ✓ 二进制文件读取：`tf.FixedLengthRecordReader(record_bytes)`
 - 读取每个记录是固定字节的二进制文件
 - `record_bytes`: 每次读取的字节数
- ✓ 通用读取方法：`read(file_queue)`
 - 从队列中读取指定数量（行，字节）的内容
 - 返回值：一个tensor元组，（文件名, value）



文件读取API（续2）

➤ 文件内容解码：

✓ 解码文本文件：`tf.decode_csv(records, record_defaults)`

- 将CSV文件内容转换为张量，与`tf.TextLineReader`搭配使用
- 参数：`records`: 字符串，对应文件中的一行
`record_defaults`: 类型
- 返回：`tensor`对象列表

✓ 解码二进制文件：`tf.decode_raw(input_bytes, out_type)`

- 将字节转换为由数字表示的张量，与`tf.FixedLengthRecordReader`搭配使用
- 参数：`input_bytes` - 待转换字节
`out_type` - 输出类型
- 返回：转换结果



案例2：CSV文件读取

```
import tensorflow as tf
import os
def csv_read(filelist):
    # 2. 构建文件队列
    file_queue = tf.train.string_input_producer(filelist)
    # 3. 构建csv reader, 读取队列内容(一行)
    reader = tf.TextLineReader()
    k, v = reader.read(file_queue)
    # 4. 对每行内容进行解码
    ## record_defaults: 指定每一个样本每一列的类型, 指定默认值
    records = [["None"], ["None"]]
    example, label = tf.decode_csv(v, record_defaults=records) # 每行两个值
    # 5. 批处理
    # batch_size: 跟队列大小无关, 只决定本批次取多少数据
    example_batch, label_batch = tf.train.batch([example, label],
                                                batch_size=9,
                                                num_threads=1,
                                                capacity=9)
    return example_batch, label_batch
```



案例2: CSV文件读取 (续)

```
if __name__ == "__main__":  
    # 1. 找到文件, 构造一个列表  
    dir_name = "./test_data/"  
    file_names = os.listdir(dir_name)  
    file_list = []  
    for f in file_names:  
        file_list.append(os.path.join(dir_name, f)) # 拼接目录和文件名  
    example, label = csv_read(file_list)  
    # 开启session运行结果  
    with tf.Session() as sess:  
        coord = tf.train.Coordinator() # 定义线程协调器  
        # 开启读取文件线程  
        # 调用 tf.train.start_queue_runners 之后, 才会真正把tensor推入内存序列中  
        # 供计算单元调用, 否则会由于内存序列为空, 数据流图会处于一直等待状态  
        threads = tf.train.start_queue_runners(sess, coord=coord)  
        print(sess.run([example, label])) # 打印读取的内容  
        # 回收线程  
        coord.request_stop()  
        coord.join(threads)
```



图片文件读取API

- 图像读取器：tf.WholeFileReader
 - ✓ 功能：将文件的全部内容作为值输出的reader
 - ✓ read方法：读取文件内容，返回文件名和文件内容
 - 图像解码器：
 - ✓ tf.image.decode_jpeg(constants)：解码jpeg格式
 - ✓ tf.image.decode_png(constants)：解码png格式
- 返回值：3-D张量，[height, width, channels]



图片文件读取API（续1）

- 修改图像大小：`tf.image.resize_images(images, size)`
 - ✓ `images`：图片数据，3-D或4-D张量
 - 3-D：[长，宽，通道]
 - 4-D：[数量，长，宽，通道]
 - ✓ `size`：1-D int32张量，[长、宽]（不需要传通道数）



案例3：图片文件读取

```
2 import tensorflow as tf
3 import os
4
5 def img_read(filelist):
6     # 1. 构建文件队列
7     file_queue = tf.train.string_input_producer(filelist)
8     # 2. 构建reader读取文件内容，默认读取一张图片
9     reader = tf.WholeFileReader()
10    k, v = reader.read(file_queue)
11
12    # 3. 对每行内容进行解码
13    img = tf.image.decode_jpeg(v) # 每行两个值
14
15    # 4. 批处理，图片需要处理成统一大小
16    img_resized = tf.image.resize(img, [200, 200]) # 200*200
17    img_resized.set_shape([200, 200, 3]) # 固定样本形状，批处理时对数据形状有要求
18    img_batch = tf.train.batch([img_resized],
19                                batch_size=10,
20                                num_threads=1)
21    return img_batch
```



案例3：图片文件读取（续）

```
24 ▶ if __name__ == "__main__":  
25     # 1. 找到文件，构造一个列表  
26     dir_name = "../data/test_img/"  
27     file_names = os.listdir(dir_name)  
28     file_list = []  
29     for f in file_names:  
30         file_list.append(os.path.join(dir_name, f)) # 拼接目录和文件名  
31     imgs = img_read(file_list)  
32     # 开启session运行结果  
33     with tf.Session() as sess:  
34         coord = tf.train.Coordinator() # 定义线程协调器  
35         # 开启读取文件线程  
36         # 调用 tf.train.start_queue_runners 之后，才会真正把tensor推入内存序列中  
37         # 供计算单元调用，否则会由于内存序列为空，数据流图会处于一直等待状态  
38         # 返回一组线程  
39         threads = tf.train.start_queue_runners(sess, coord=coord)  
40         print(sess.run([imgs])) # 打印读取的内容  
41         # 回收线程  
42         coord.request_stop()  
43         coord.join(threads)
```



图像识别

图像识别

手写体识别

MNIST数据集

任务目标

网络结构

相关API

关键代码

服装识别

数据集介绍

任务目标

网络结构

关键代码

手写体识别

MNIST数据集

- 手写数字的数据集，来自美国国家标准与技术研究所（National Institute of Standards and Technology, NIST），发布与1998年
- 样本来自250个不同人的手写数字，50%高中生，50%是人口普查局的工作人员
- 数字从0~9，图片大小是28×28像素，训练数据集包含60000个样本，测试数据集包含10000个样本。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。
- 下载地址：
<http://yann.lecun.com/exdb/mnist/>



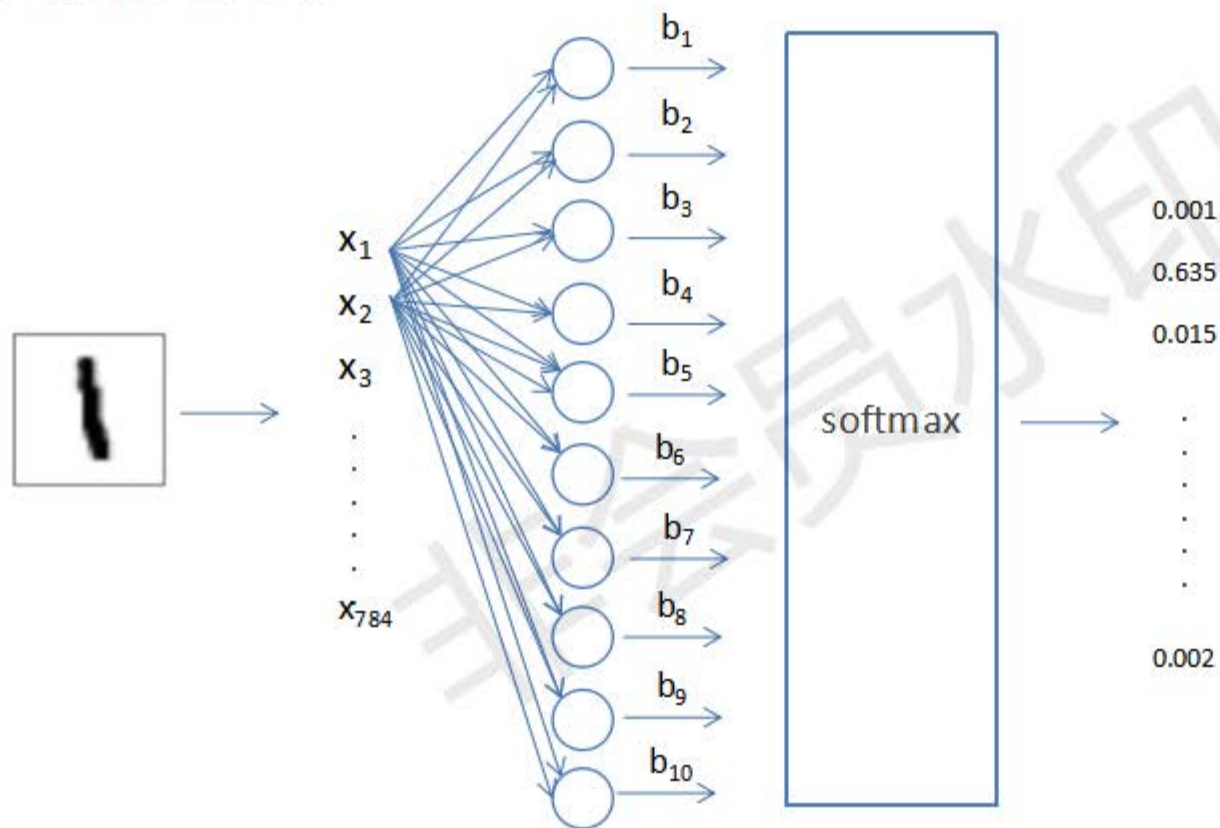
任务目标

- 根据训练集样本进行模型训练
- 保存模型
- 加载模型，用于新的手写体数字识别

非会员水印



网络结构



相关API

- `tf.matmul ()` : 执行矩阵乘法计算
- `tf.nn.softmax ()` : softmax激活函数
- `tf.reduce_sum ()` : 指定维度上求张量平均值
- `tf.train.GradientDescentOptimizer ()` : 优化器, 执行梯度下降
- `tf.argmax ()` : 返回张量最大元素的索引值



关键代码

- 定义数据

```
6 # 读入数据集(如果没有则在线下载), 并转换成独热编码
7 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
8
9 x = tf.placeholder(tf.float32, [None, 784]) # 占位符, 输入
10 y = tf.placeholder(tf.float32, [None, 10]) # 占位符, 输出
11
12 W = tf.Variable(tf.random_normal([784, 10])) # 权重
13 b = tf.Variable(tf.zeros([10])) # 偏移量
```



关键代码（续1）

- 模型搭建

```
15 # 构建模型
16 pred_y = tf.nn.softmax(tf.matmul(x, W) + b) # softmax分类
17 print("pred_y.shape:", pred_y.shape)
18 # 损失函数
19 cross_entropy = -tf.reduce_sum(y * tf.log(pred_y),
20                                reduction_indices=1) # 求交叉熵
21 cost = tf.reduce_mean(cross_entropy) # 求损失函数平均值
22
23 # 参数设置
24 lr = 0.01
25 # 梯度下降优化器
26 optimizer = tf.train.GradientDescentOptimizer(lr).minimize(cost)
```



关键代码（续2）

- 执行训练

```
34 with tf.Session() as sess:
35     sess.run(tf.global_variables_initializer())
36
37     # 循环开始训练
38     for epoch in range(training_epochs):
39         avg_cost = 0.0
40         total_batch = int(mnist.train.num_examples / batch_size) # 计算总批次
41
42         # 遍历全数据集
43         for i in range(total_batch):
44             batch_xs, batch_ys = mnist.train.next_batch(batch_size) # 读取一个批次样本
45             params = {x: batch_xs, y: batch_ys} # 训练参数
46
47             o, c = sess.run([optimizer, cost], feed_dict=params) # 执行训练
48
49             avg_cost += (c / total_batch) # 求平均损失值
50
51         print("epoch: %d, cost=%.9f" % (epoch + 1, avg_cost))
52
53     print("Finished!")
```



关键代码（续3）

- 模型评估

```
55     # 模型评估
56     correct_pred = tf.equal(tf.argmax(pred_y, 1), tf.argmax(y, 1))
57     # 计算准确率
58     accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
59     print("accuracy:", accuracy.eval({x: mnist.test.images,
60                                         y: mnist.test.labels}))
61     # 将模型保存到文件
62     save_path = saver.save(sess, model_path)
63     print("Model saved:", save_path)
64
```



关键代码（续4）

- 模型测试

```
65 # 测试模型
66 with tf.Session() as sess:
67     sess.run(tf.global_variables_initializer())
68     saver.restore(sess, model_path) # 加载模型
69
70     batch_xs, batch_ys = mnist.test.next_batch(2) # 读取2个测试样本
71     output = tf.argmax(pred_y, 1) # 预测结果值
72
73     output_val, predv = sess.run([output, pred_y], # 操作
74                                   feed_dict={x: batch_xs}) # 参数
75
76     print("预测结论:\n", output_val, "\n")
77     print("实际结果:\n", batch_ys, "\n")
78     print("预测概率:\n", predv, "\n")
```



执行结果

预测结论:

[4 0]

实际结果:

[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

预测概率:

[[1.4690894e-11 2.1458644e-24 1.9810063e-10 7.7833487e-17 9.9968827e-01

5.1569430e-07 2.1053264e-04 1.2249268e-09 8.1653903e-05 1.9109739e-05]

[9.9999952e-01 2.7850160e-22 1.5803887e-09 9.9736819e-10 2.4080727e-13

5.2853795e-07 5.2688409e-13 2.6116598e-10 1.3954279e-09 2.7404382e-10]]



案例4：实现手写体识别

- 见mnist_demo.py

非会员水印



服饰识别

数据集介绍

- 是来自 Zalando 文章的数据集，是时尚版的 MNIST。包括 60,000 个训练集数据，10,000 个测试集数据，每个数据为 28x28 灰度图像，一共有 10 类：

0	T-shirt/top	T恤
1	Trouser	裤子
2	Pullover	套衫
3	Dress	衣服
4	Coat	外套
5	Sandal	凉鞋
6	Shirt	衬衫
7	Sneaker	运动鞋
8	Bag	包
9	Ankle boot	短靴



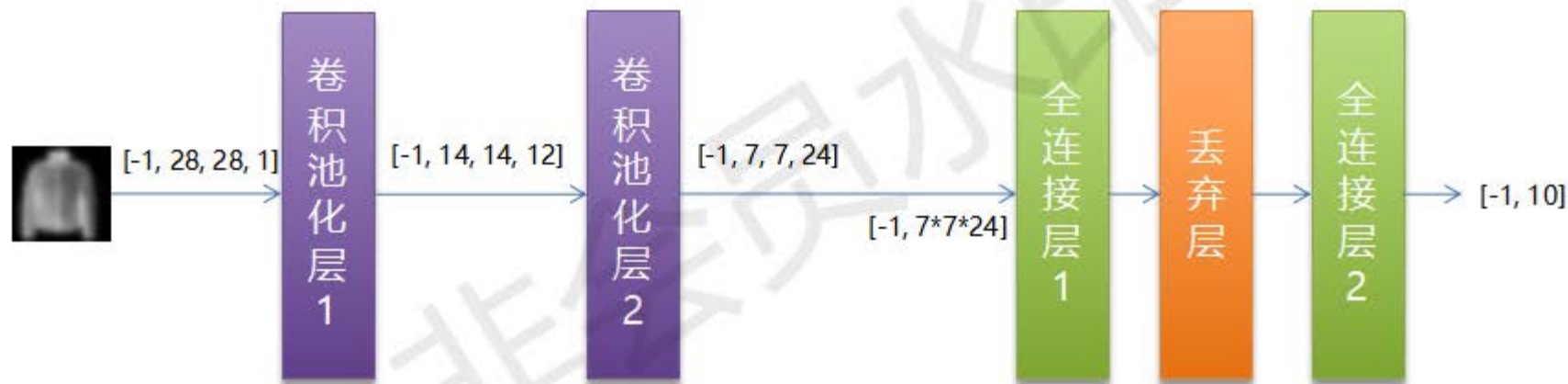
任务目标

- 根据训练集样本进行模型训练
- 保存模型
- 加载模型，用于新的服饰图片识别

非会员水印



网络结构



关键代码

```
10 class FashionMnist():
11     out_features1 = 12 # 第一层卷积输出特征数量
12     out_features2 = 24 # 第二层卷积输出特征数量
13     con_neurons = 512 # 全连接层神经元数量
14
15     def __init__(self, path):
16         self.sess = tf.Session()
17         self.data = read_data_sets(path, one_hot=True)
18         self.tt = None
19
20     # 权重初始化函数
21     def init_weight_variable(self, shape):
22         # 输出的随机数满足 截尾正态分布
23         # 截尾正态分布:产生的随机数与均值的差距不会超过两倍的标准差
24         initial = tf.truncated_normal(shape, stddev=0.1)
25         return tf.Variable(initial)
26
27     # 偏置初始化函数
28     def init_bias_variable(self, shape):
29         initial = tf.constant(1.0, shape=shape)
30         return tf.Variable(initial)
```



关键代码（续1）

```
32 # 二维卷积函数
33 def conv2d(self, x, W):
34     # input : 输入数据[batch, in_height, in_width, in_channels]
35     # filter : 卷积窗口[filter_height, filter_width, in_channels, out_channels]
36     # strides: 卷积核每次移动步数, 对应着输入的维度方向
37     # padding='SAME' : 输入和输出的张量形状相同
38     return tf.nn.conv2d(x, W,
39                           strides=[1, 1, 1, 1], # 每个维度上的步长值
40                           padding='SAME')
41
42 # 池化函数
43 def max_pool_2x2(self, x):
44     return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], # 池化窗口的大小
45                             strides=[1, 2, 2, 1], # 每个维度上的步长值
46                             padding='SAME')
```



关键代码（续2）

```
48 # 构建卷积层
49 def create_conv_pool_layer(self, input, input_features, out_features):
50     W_conv = self.init_weight_variable([5, 5, input_features, out_features]) # 权重
51     b_conv = self.init_bias_variable([out_features]) # 偏置
52     h_conv = tf.nn.relu(self.conv2d(input, W_conv) + b_conv) # relu激活
53     h_pool = self.max_pool_2x2(h_conv) # 2*2区域做最大池化
54     return h_pool
55
56 # 构建全连接层
57 def create_fc_layer(self, h_pool_flat, input_features, con_neurons):
58     W_fc = self.init_weight_variable([input_features, con_neurons]) # 初始化权重
59     b_fc = self.init_bias_variable([con_neurons]) # 初始化偏置
60     h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc) + b_fc) # 计算wx + b并做relu激活
61     return h_fc1
```



关键代码（续3）

```
63 # 神经网络构建
64 def build(self):
65     # 输入
66     self.x = tf.placeholder(tf.float32, shape=[None, 784]) # 28*28图像
67     x_image = tf.reshape(self.x, [-1, 28, 28, 1]) # 28*28单通道
68     self.y_ = tf.placeholder(tf.float32, shape=[None, 10]) # 输出, 对应10个类别
69     h_pool1 = self.create_conv_pool_layer(x_image, 1, self.out_features1) # 第一层
70     # 第二层: 第一层输出作为输入
71     h_pool2 = self.create_conv_pool_layer(h_pool1, self.out_features1, self.out_features2)
72     # 全连接层
73     h_pool2_flat_freatures = 7 * 7 * self.out_features2 # 为了做全连接计算, 所以需要变维
74     h_pool2_flat = tf.reshape(h_pool2, [-1, h_pool2_flat_freatures])
75     h_fc = self.create_fc_layer(h_pool2_flat, h_pool2_flat_freatures, self.con_neurons)
76     self.keep_prob = tf.placeholder("float")
77     h_fc1_drop = tf.nn.dropout(h_fc, self.keep_prob) # Dropout
78     # 输出层
79     W_fc = self.init_weight_variable([self.con_neurons, 10])
80     b_fc = self.init_bias_variable([10])
81     y_conv = tf.matmul(h_fc1_drop, W_fc) + b_fc
82     # 评价
83     correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(self.y_, 1))
84     self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32)) # 计算准确率
85     loss_func = tf.nn.softmax_cross_entropy_with_logits(labels=self.y_, logits=y_conv) # 损失函数
86     cross_entropy = tf.reduce_mean(loss_func)
87     optimizer = tf.train.AdamOptimizer(0.001) # 优化器
88     self.train_step = optimizer.minimize(cross_entropy)
```



关键代码（续4）

```
90 # 训练
91 def train(self):
92     self.sess.run(tf.global_variables_initializer()) # 初始化变量
93
94     merged = tf.summary.merge_all() # 将所有的摘要信息保存到磁盘
95     for i in range(500):
96         batch = self.data.train.next_batch(50)
97         params = {self.x: batch[0], self.y_: batch[1], self.keep_prob: 0.5}
98         t, acc = self.sess.run([self.train_step, self.accuracy], params)
99         # # 每20笔打印一次准确率
100         if i % 20 == 0:
101             print("passid: %d, acc: %f" % (i, acc))
102
103 # 评价
104 def eval(self, x, y, keep_prob):
105     params = {self.x: x, self.y_: y, self.keep_prob: 1.0}
106     test_acc = self.sess.run(self.accuracy, params)
107     print('Test accuracy %f' % test_acc)
108     return test_acc
```




关键代码（续5）

```
115 ▶ if __name__ == "__main__":  
116     mnist = FashionMnist('MNIST_data/')  
117     mnist.build()  
118     mnist.train()  
119  
120     print('\n----- Test -----')  
121     xs, ys = mnist.data.test.next_batch(100)  
122     mnist.eval(xs, ys, 1.0)  
123     mnist.close()
```



执行结果



passid: 260,	acc: 0.920000
passid: 280,	acc: 0.920000
passid: 300,	acc: 0.780000
passid: 320,	acc: 0.820000
passid: 340,	acc: 0.980000
passid: 360,	acc: 0.940000
passid: 380,	acc: 0.840000
passid: 400,	acc: 0.880000
passid: 420,	acc: 0.940000
passid: 440,	acc: 0.840000
passid: 460,	acc: 0.940000
passid: 480,	acc: 0.920000
----- Test -----	
Test accuracy 0.950000	



案例5：实现服饰识别

- 见fashion_mnist_demo.py

非会员水印



今日总结

- 模型保存与加载
- 文件读取
- 图像分类：手写体识别、服饰识别