姓名：向建宇

学号：1801210687

**采用多线程对图像数据（大于50M）进行基于神经网络的训练分析，得出准确率，并与单线程做效率（训练时间）的比较。**

TensorFlow提供两个类帮助实现多线程，一个是tf.train.Coordinator，另一个是tf.train.QueueRunner。Coordinator主要用来实现多个线程同时停止，QueueRunner用来创建一系列线程。

**Coordinator**

根据官方文档，Coordinator主要有三个方法：

* tf.train.Coordinator.should\_stop: returns True if the threads should stop.
* tf.train.Coordinator.request\_stop: requests that threads should stop.
* tf.train.Coordinator.join: waits until the specified threads have stopped.

接下来使用Coordinator和不使用Coordinator对书上第一节的内容做训练分析：

**1. 创建文件列表，通过文件列表创建输入文件队列，读取文件为本章第一节创建的文件。**

|  |
| --- |
| files = tf.train.match\_filenames\_once("output.tfrecords")  filename\_queue = tf.train.string\_input\_producer(files, shuffle=False) |

#### 2. 解析TFRecord文件里的数据。

|  |
| --- |
| # 读取文件。  reader = tf.TFRecordReader()  \_,serialized\_example = reader.read(filename\_queue)  # 解析读取的样例。  features = tf.parse\_single\_example(  serialized\_example,  features={  'image\_raw':tf.FixedLenFeature([],tf.string),  'pixels':tf.FixedLenFeature([],tf.int64),  'label':tf.FixedLenFeature([],tf.int64)  })  decoded\_images = tf.decode\_raw(features['image\_raw'],tf.uint8)  retyped\_images = tf.cast(decoded\_images, tf.float32)  labels = tf.cast(features['label'],tf.int32)  #pixels = tf.cast(features['pixels'],tf.int32)  images = tf.reshape(retyped\_images, [784]) |

#### 3. 将文件以100个为一组打包。[¶](http://localhost:8888/notebooks/6.%20%E8%BE%93%E5%85%A5%E6%95%B0%E6%8D%AE%E5%A4%84%E7%90%86%E6%A1%86%E6%9E%B6.ipynb#4.-将文件以100个为一组打包。)

|  |
| --- |
| min\_after\_dequeue = 10000  batch\_size = 100  capacity = min\_after\_dequeue + 3 \* batch\_size  image\_batch, label\_batch = tf.train.shuffle\_batch([images, labels],  batch\_size=batch\_size,  capacity=capacity,  min\_after\_dequeue=min\_after\_dequeue) |

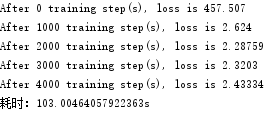
#### 4. 训练模型。

|  |
| --- |
| def inference(input\_tensor, weights1, biases1, weights2, biases2):  layer1 = tf.nn.relu(tf.matmul(input\_tensor, weights1) + biases1)  return tf.matmul(layer1, weights2) + biases2  # 模型相关的参数  INPUT\_NODE = 784  OUTPUT\_NODE = 10  LAYER1\_NODE = 500  REGULARAZTION\_RATE = 0.0001  TRAINING\_STEPS = 5000  weights1 = tf.Variable(tf.truncated\_normal([INPUT\_NODE, LAYER1\_NODE], stddev=0.1))  biases1 = tf.Variable(tf.constant(0.1, shape=[LAYER1\_NODE]))  weights2 = tf.Variable(tf.truncated\_normal([LAYER1\_NODE, OUTPUT\_NODE], stddev=0.1))  biases2 = tf.Variable(tf.constant(0.1, shape=[OUTPUT\_NODE]))  y = inference(image\_batch, weights1, biases1, weights2, biases2)    # 计算交叉熵及其平均值  cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=y, labels=label\_batch)  cross\_entropy\_mean = tf.reduce\_mean(cross\_entropy)    # 损失函数的计算  regularizer = tf.contrib.layers.l2\_regularizer(REGULARAZTION\_RATE)  regularaztion = regularizer(weights1) + regularizer(weights2)  loss = cross\_entropy\_mean + regularaztion  # 优化损失函数  train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)    # 初始化会话，并开始训练过程。  with tf.Session() as sess:  # tf.global\_variables\_initializer().run()  sess.run((tf.global\_variables\_initializer(),  tf.local\_variables\_initializer()))  coord = tf.train.Coordinator()  threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)  # 循环的训练神经网络。  for i in range(TRAINING\_STEPS):  if i % 1000 == 0:  print("After %d training step(s), loss is %g " % (i, sess.run(loss)))    sess.run(train\_step)  coord.request\_stop()  coord.join(threads) |

上面的代码使用多线程和输入队列，使得运算加快

结果：

设定两个代码都使用4000次迭代，耗时为：



多线程耗时103秒，

修改代码后，使用单线程大概耗时303秒

对比起来多线程加快了学习速度。