姓名：向建宇

学号：1801210687

**参考minist最佳实践例子程序，使用tensorflow神经网络训练数据集，计算准确率。**

由于程序越来越复杂，所以我们要将代码模块化，以方便代码间的复用，所以分为三个部分：前向传播，反向传播和测试。

1.**前向传播：**

下面是全部代码，详解已经在注释里说的很清楚了，主要是定义了前向传播的过程以及神经网络中的参数：注意这里将INPUT\_NODE修改为256，因为老师给的数据是256维的。

|  |
| --- |
| **import** tensorflow **as** tf  *#定义神经网络结构相关的参数* INPUT\_NODE = 256 *#输入结点个数* OUTPUT\_NODE = 10 *#输出维数* LAYER1\_NODE = 500 *#隐藏层结点个数 #通过tf.get\_variable函数来获取变量。 get\_variable获取已存在的变量（要求不仅名字，而且初始化方法等各个参数都一样），如果不存在，就新建一个。 #需要注意的是tf.get\_variable() 要配合reuse和tf.variable\_scope() 使用。* **def** get\_weight\_variable(shape, regularizer):  weights = tf.get\_variable(**"weights"**, shape, initializer=tf.truncated\_normal\_initializer(stddev=0.1))  **if** regularizer != **None**: tf.add\_to\_collection(**'losses'**, regularizer(weights))  **return** weights   **def** inference(input\_tensor, regularizer):  **with** tf.variable\_scope(**'layer1'**): *#这里的weights是命名空间layer1中的* weights = get\_weight\_variable([INPUT\_NODE, LAYER1\_NODE], regularizer)  biases = tf.get\_variable(**"biases"**, [LAYER1\_NODE], initializer=tf.constant\_initializer(0.0))  layer1 = tf.nn.relu(tf.matmul(input\_tensor, weights) + biases)   **with** tf.variable\_scope(**'layer2'**):*#这里的weights是命名空间layer2中的* weights = get\_weight\_variable([LAYER1\_NODE, OUTPUT\_NODE], regularizer)  biases = tf.get\_variable(**"biases"**, [OUTPUT\_NODE], initializer=tf.constant\_initializer(0.0))  layer2 = tf.matmul(layer1, weights) + biases   **return** layer2 |

**2.反向传播：**

定义了神经网络的训练过程：这里其它代码没变，参考了MNIST的train，在原有基础上使用正则化，衰减率，滑动平均，学习率指数衰减，使得代码更健壮。这里修改了数据的读入（详见下面代码的load\_data函数），将老师给的数据进行了预处理，并在喂入神经网络时修改了喂入量的随机算法（详见tf.Session下的处理）

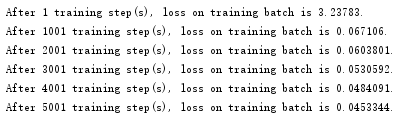
|  |
| --- |
| **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data **import** mnist\_inference **import** os **from** numpy **import** \*  *#定义神经网络结构相关的参数。* BATCH\_SIZE = 100 LEARNING\_RATE\_BASE = 0.8 LEARNING\_RATE\_DECAY = 0.99 REGULARIZATION\_RATE = 0.0001 TRAINING\_STEPS = 30000 MOVING\_AVERAGE\_DECAY = 0.99 MODEL\_SAVE\_PATH=**"MNIST\_model/"** MODEL\_NAME=**"mnist\_model"** *#定义训练过程。* **def** train(xs,ys):  x = tf.placeholder(tf.float32, [**None**, mnist\_inference.INPUT\_NODE], name=**'x-input'**)  y\_ = tf.placeholder(tf.float32, [**None**, mnist\_inference.OUTPUT\_NODE], name=**'y-input'**)   regularizer = tf.contrib.layers.l2\_regularizer(REGULARIZATION\_RATE)  y = mnist\_inference.inference(x, regularizer)  global\_step = tf.Variable(0, trainable=**False**)   variable\_averages = tf.train.ExponentialMovingAverage(MOVING\_AVERAGE\_DECAY, global\_step)  variables\_averages\_op = variable\_averages.apply(tf.trainable\_variables())  cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=y, labels=tf.argmax(y\_, 1))  cross\_entropy\_mean = tf.reduce\_mean(cross\_entropy)  loss = cross\_entropy\_mean + tf.add\_n(tf.get\_collection(**'losses'**))  learning\_rate = tf.train.exponential\_decay( *#tf.train.exponential\_decay(learning\_rate, global\_, decay\_steps, decay\_rate, staircase=True/False)* LEARNING\_RATE\_BASE, *#每decay\_steps轮衰减decay\_rate* global\_step,  BATCH\_SIZE, LEARNING\_RATE\_DECAY,  staircase=**True**)  train\_step = tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss, global\_step=global\_step)  **with** tf.control\_dependencies([train\_step, variables\_averages\_op]):  train\_op = tf.no\_op(name=**'train'**)   saver = tf.train.Saver()  **with** tf.Session() **as** sess:  tf.global\_variables\_initializer().run()   **for** i **in** range(TRAINING\_STEPS):  start = (i \* BATCH\_SIZE) % 1500  end = (i \* BATCH\_SIZE) % 1500 + BATCH\_SIZE  *# xs, ys = mnist.train.next\_batch(BATCH\_SIZE)* \_,loss\_value, step = sess.run([train\_op, loss, global\_step], feed\_dict={x: xs[start:end], y\_: ys[start:end]})  **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)  *#主程序入口。* **def** main(argv=**None**):  *#input\_data.read\_data\_sets函数生成的类会自动将MNIST数据集划分为train=55000、validation=5000和test=10000三个数据集。  # mnist = input\_data.read\_data\_sets("../../../datasets/MNIST\_data", one\_hot=True)* xs,ys,txs,tys= load\_date()  train(xs,ys)  **""" 处理数据 """ def** load\_date():  input\_xs = []  input\_ys = []  percent = 0.1  lines = open(**"semeion.data"**).readlines()  **for** line **in** lines:  list = line.strip().split(**' '**)  input\_x = list[:-10]  input\_y = list[-10:]  input\_xs.append(input\_x)  input\_ys.append(input\_y)   num = int(percent \* len(input\_xs))  train\_x = mat(input\_xs[:-num])  train\_y = mat(input\_ys[:-num])  test\_x = mat(input\_xs[-num:])  test\_y = mat(input\_ys[-num:])  **return** train\_x,train\_y,test\_x,test\_y  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  tf.app.run() |

3.测试：

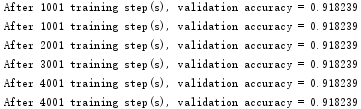
定义了测试过程，其中直接用saver类读取了前面训练好的模型，并使用了写在训练程序里的load\_data方法得到测试集，其中选用了数据集的10%作为测试集

|  |
| --- |
| **import** time **import** tensorflow **as** tf **from** tensorflow.examples.tutorials.mnist **import** input\_data **import** mnist\_inference **import** mnist\_train  *# 加载的时间间隔。* EVAL\_INTERVAL\_SECS = 10 *#每10秒加载一次最新的模型* **def** evaluate(txs,tys):  **with** tf.Graph().as\_default() **as** g:  x = tf.placeholder(tf.float32, [**None**, mnist\_inference.INPUT\_NODE], name=**'x-input'**)  y\_ = tf.placeholder(tf.float32, [**None**, mnist\_inference.OUTPUT\_NODE], name=**'y-input'**)  validate\_feed = {x: txs, y\_: tys}   y = mnist\_inference.inference(x, **None**)  correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))   variable\_averages = tf.train.ExponentialMovingAverage(mnist\_train.MOVING\_AVERAGE\_DECAY)  variables\_to\_restore = variable\_averages.variables\_to\_restore()  saver = tf.train.Saver(variables\_to\_restore)   **while True**:  **with** tf.Session() **as** sess:  ckpt = tf.train.get\_checkpoint\_state(mnist\_train.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=validate\_feed)  print(**"After %s training step(s), validation accuracy = %g"** % (global\_step, accuracy\_score))  **else**:  print(**'No checkpoint file found'**)  **return** time.sleep(EVAL\_INTERVAL\_SECS)  **def** main(argv=**None**):  xs, ys, txs, tys = mnist\_train.load\_date()  evaluate(txs,tys)  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  main() |

4.结果截图：



训练结果如图所示，这里我取的是前90%当训练集，而没有取随机数据，所以损失函数可能会偏大，导致后面正确率下降



正确率接近92%，比MNIST低了6.5个百分点，我觉得主要原因还是因为上面没有使用随机数，并且数据集太少只用1590条的原因。后面还会继续改进