目录

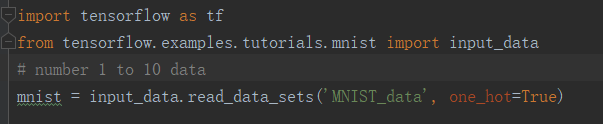
[一.数据集 1](#_Toc522270659)

[二.队伍构成 1](#_Toc522270660)

[三.性能测量指标 1](#_Toc522270661)

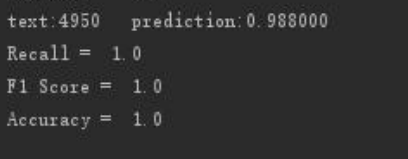
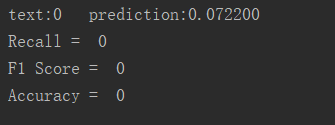
[四.卷积神经网络结构图（利用tensorboard绘制） 2](#_Toc522270662)  
五.其他参数变化图  
 1.交叉熵 3

[2.第一层神经网 weight、baises和output 3](#_Toc522270663)  
3.第二层神经网 weight、baises和output 5

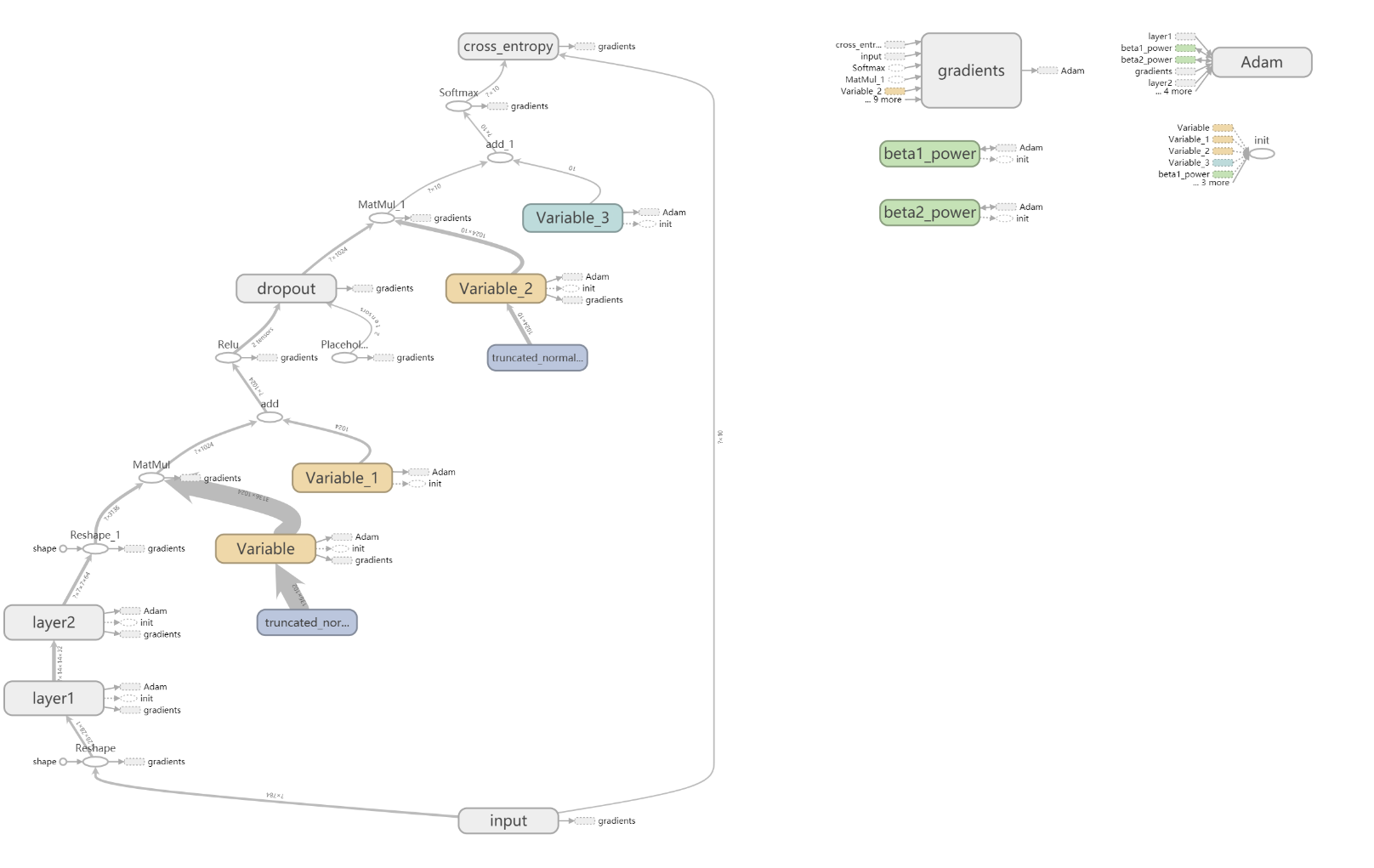
源码在第六页  
一.数据集  
tensorflow自带的手写体数字数据集，通过一下语句下载并导入调用

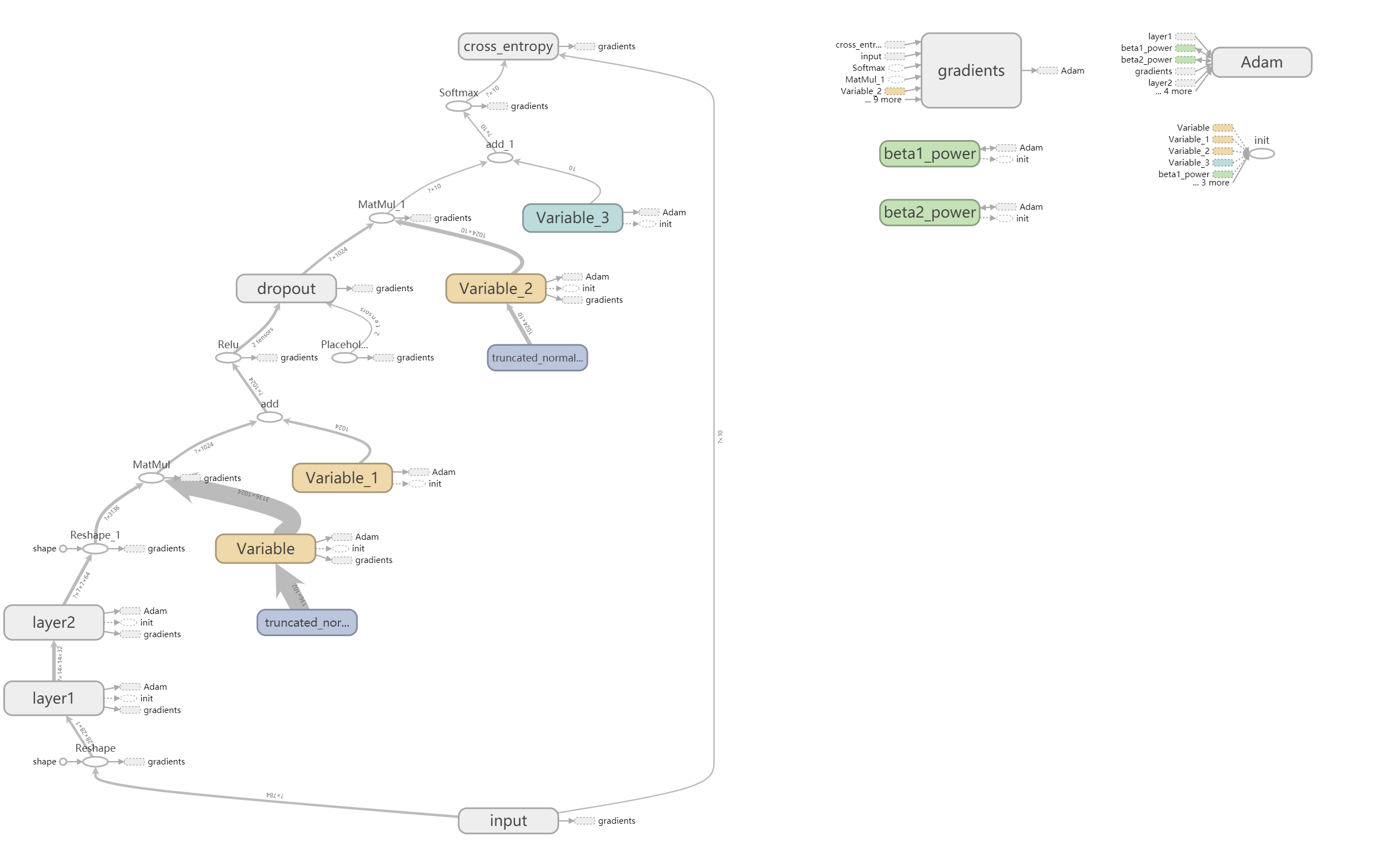
二.队伍构成  
管胡昊（队长） 班级：计1708 学号：20174725

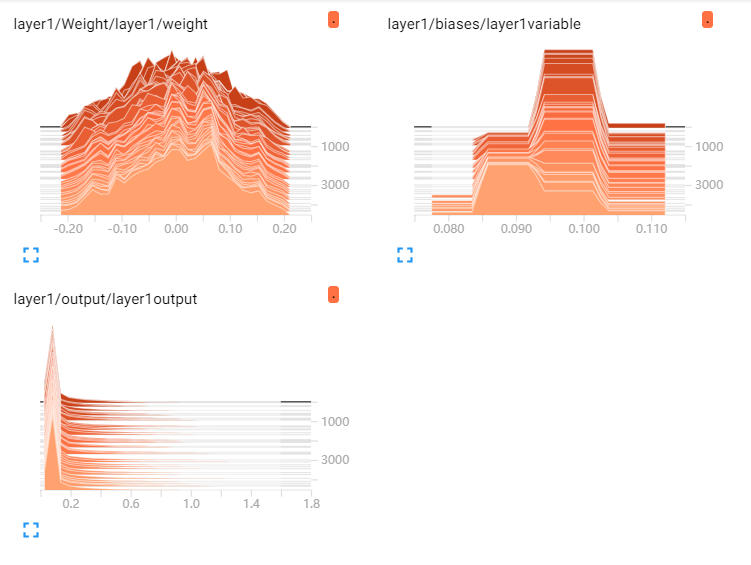
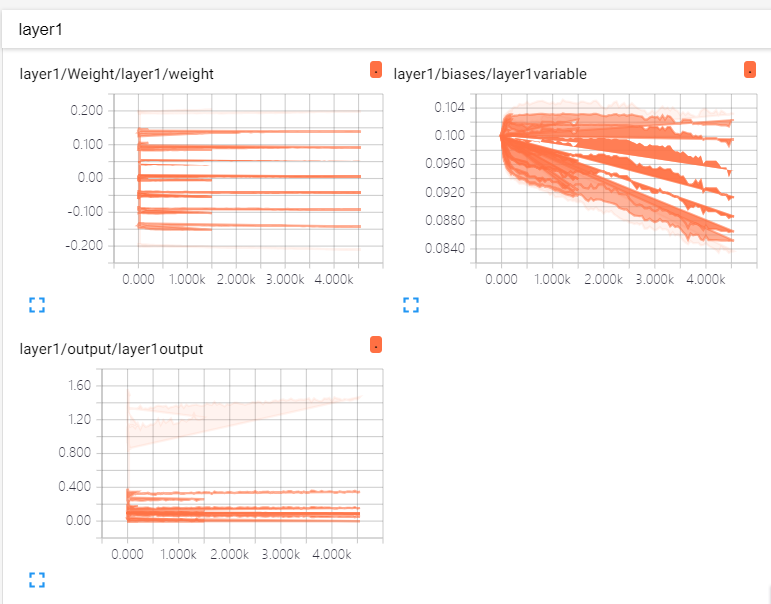
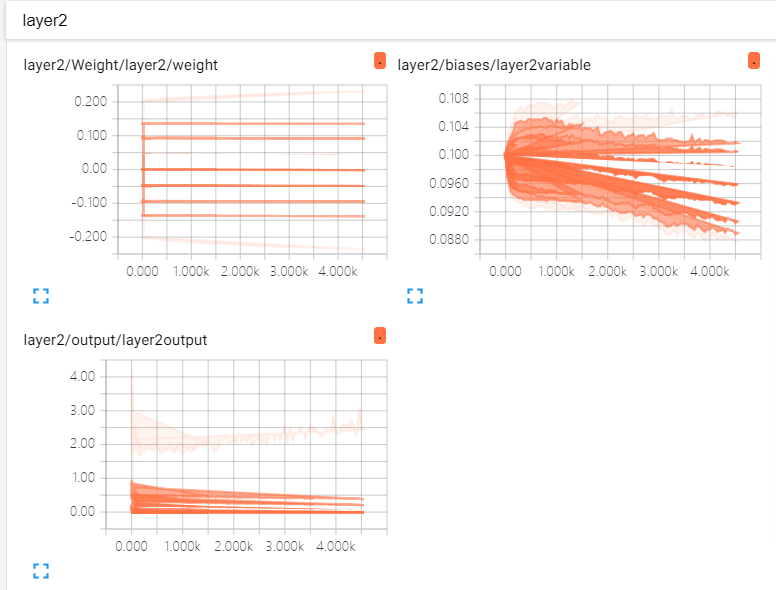
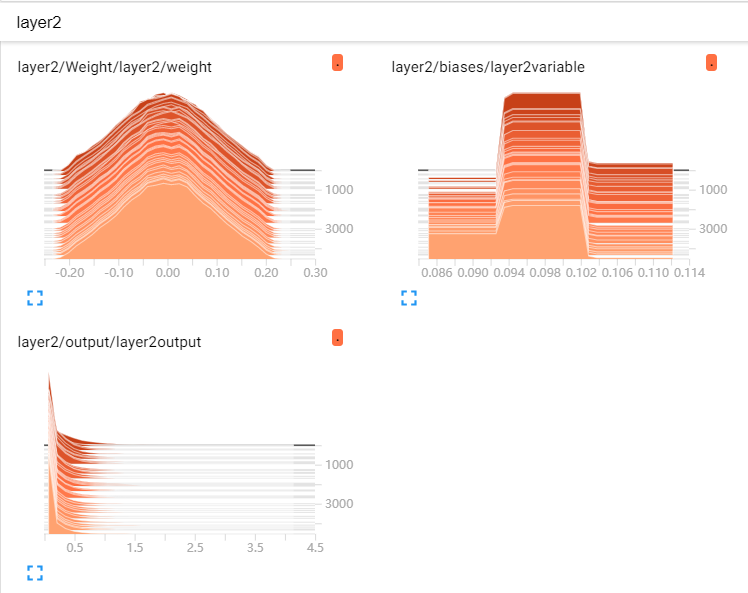
## 三.性能测量指标 数据集名称：MNIST



未训练时的各项参数 100个数据为一组5000组后的参数  
PS：recall和F1会在前100次内迅速提到1.0，并很少波动，不知是否哪里设置出现了问题

四.卷积神经网络结构图（利用tensorboard绘制）

  
  
  
  
  
  
  
  
  
  
  
  
五、其他参数变化  
  
1.交叉熵

2.第一层神经网 weight、baises和output  
  
  
  
  
  
  
  
  
  
  
  
  
3.第二层神经网 weight、baises和output  


**六.源码**https://paste.ubuntu.com/p/BgXVQWPzjT/  
  
import tensorflow as tf  
from tensorflow.examples.tutorials.mnist import input\_data  
# number 1 to 10 data  
mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)  
  
  
# tem  
def tf\_confusion\_metrics(model, actual\_classes, session, feed\_dict):  
 predictions = tf.argmax(model, 1)  
 actuals = tf.argmax(actual\_classes, 1)  
  
 ones\_like\_actuals = tf.ones\_like(actuals)  
 zeros\_like\_actuals = tf.zeros\_like(actuals)  
 ones\_like\_predictions = tf.ones\_like(predictions)  
 zeros\_like\_predictions = tf.zeros\_like(predictions)  
  
 tp\_op = tf.reduce\_sum(  
 tf.cast(  
 tf.logical\_and(  
 tf.equal(actuals, ones\_like\_actuals),  
 tf.equal(predictions, ones\_like\_predictions)  
 ),  
 "float"  
 )  
 )  
  
 tn\_op = tf.reduce\_sum(  
 tf.cast(  
 tf.logical\_and(  
 tf.equal(actuals, zeros\_like\_actuals),  
 tf.equal(predictions, zeros\_like\_predictions)  
 ),  
 "float"  
 )  
 )  
  
 fp\_op = tf.reduce\_sum(  
 tf.cast(  
 tf.logical\_and(  
 tf.equal(actuals, zeros\_like\_actuals),  
 tf.equal(predictions, ones\_like\_predictions)  
 ),  
 "float"  
 )  
 )  
  
 fn\_op = tf.reduce\_sum(  
 tf.cast(  
 tf.logical\_and(  
 tf.equal(actuals, ones\_like\_actuals),  
 tf.equal(predictions, zeros\_like\_predictions)  
 ),  
 "float"  
 )  
 )  
 tp, tn, fp, fn = \  
 session.run(  
 [tp\_op, tn\_op, fp\_op, fn\_op],  
 feed\_dict  
 )  
 if float(tp) + float(fn) == 0:  
 tpr = 0  
 fpr = 0  
 accuracy1 = 0  
 f1\_score1 = 0  
 else:  
 tpr = float(tp) / (float(tp) + float(fn))  
 fpr = float(fp) / (float(tp) + float(fn))  
 accuracy1 = (float(tp) + float(tn)) / (float(tp) + float(fp) + float(fn) + float(tn))  
 f1\_score1 = (2 \* float(tp)) / (2 \* float(tp) + float(fp) + float(fn))  
  
 recall1 = tpr  
 # precision1 = float(tp) / (float(tp) + float(fp))  
  
 # print('Precision = ', precision1)  
 print('Recall = ', recall1)  
 print('F1 Score = ', f1\_score1)  
 print('Accuracy = ', accuracy1)  
  
  
# tem  
  
  
def compute\_accuracy(v\_xs, v\_ys):  
 global prediction  
 y\_pre = sess.run(prediction, feed\_dict={xs: v\_xs, keep\_prob: 1})  
 correct\_prediction = tf.equal(tf.argmax(y\_pre, 1), tf.argmax(v\_ys, 1))  
 accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  
 result = sess.run(accuracy, feed\_dict={xs: v\_xs, ys: v\_ys, keep\_prob: 1})  
 print('text:%d prediction:%f' % (i, result))  
 tf\_confusion\_metrics(y\_pre, v\_ys, sess, feed\_dict={xs: v\_xs, ys: v\_ys, keep\_prob: 1})  
 return result  
  
  
def weight\_variable(shape):  
 initial = tf.truncated\_normal(shape, stddev=0.1)  
 return tf.Variable(initial)  
  
  
def bias\_variable(shape):  
 initial = tf.constant(0.1, shape=shape)  
 return tf.Variable(initial)  
  
  
def conv2d(x, W):  
 return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')  
  
  
def max\_pool\_2x2(x):  
 return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')  
  
  
# define placeholder for inputs to network  
with tf.name\_scope('input'):  
 xs = tf.placeholder(tf.float32, [None, 784])  
 ys = tf.placeholder(tf.float32, [None, 10])  
keep\_prob = tf.placeholder(tf.float32)  
x\_image = tf.reshape(xs, [-1, 28, 28, 1])  
  
  
# conv1 layer  
n\_layer = 1  
layer\_name = 'layer%s' % n\_layer  
with tf.name\_scope(layer\_name):  
 with tf.name\_scope('Weight'):  
 W\_conv1 = weight\_variable([5, 5, 1, 32])  
 tf.summary.histogram(layer\_name + '/weight', W\_conv1)  
 with tf.name\_scope('biases'):  
 b\_conv1 = bias\_variable([32])  
 tf.summary.histogram(layer\_name + 'variable', b\_conv1)  
 with tf.name\_scope('output'):  
 h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)  
 tf.summary.histogram(layer\_name + 'output', h\_conv1)  
 h\_pool1 = max\_pool\_2x2(h\_conv1)  
  
# conv2 layer  
n\_layer = 2  
layer\_name = 'layer%s' % n\_layer  
with tf.name\_scope(layer\_name):  
 with tf.name\_scope('Weight'):  
 W\_conv2 = weight\_variable([5, 5, 32, 64])  
 tf.summary.histogram(layer\_name + '/weight', W\_conv2)  
 with tf.name\_scope('biases'):  
 b\_conv2 = bias\_variable([64])  
 tf.summary.histogram(layer\_name + 'variable', b\_conv2)  
 with tf.name\_scope('output'):  
 h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)  
 tf.summary.histogram(layer\_name + 'output', h\_conv2)  
 h\_pool2 = max\_pool\_2x2(h\_conv2)  
  
# func1 layer  
W\_fc1 = weight\_variable([7\*7\*64, 1024])  
b\_fc1 = bias\_variable([1024])  
  
h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 7\*7\*64])  
h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1)+b\_fc1)  
h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)  
# func2 layer  
W\_fc2 = weight\_variable([1024, 10])  
b\_fc2 = bias\_variable([10])  
prediction = tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2)+b\_fc2)  
  
  
  
# the loss between prediction and real data  
with tf.name\_scope('cross\_entropy'):  
 cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(ys\*tf.log(prediction), reduction\_indices=[1]))  
 tf.summary.scalar('cross\_entropy', cross\_entropy)  
  
train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)  
  
sess = tf.Session()  
  
# important step  
sess.run(tf.global\_variables\_initializer())  
  
# show graph  
merged = tf.summary.merge\_all()  
writer = tf.summary.FileWriter('logs', sess.graph)  
  
  
for i in range(5000):  
 batch\_xs, batch\_ys = mnist.train.next\_batch(100)  
 sess.run(train\_step, feed\_dict={xs: batch\_xs, ys: batch\_ys, keep\_prob: 0.5})  
 if i % 50 == 0:  
 result = sess.run(merged, feed\_dict={xs: batch\_xs, ys: batch\_ys, keep\_prob: 1})  
 writer.add\_summary(result, i)  
 compute\_accuracy(mnist.test.images, mnist.test.labels)