**Kishan Sarpangala**

**M04307966**

**Project 12**

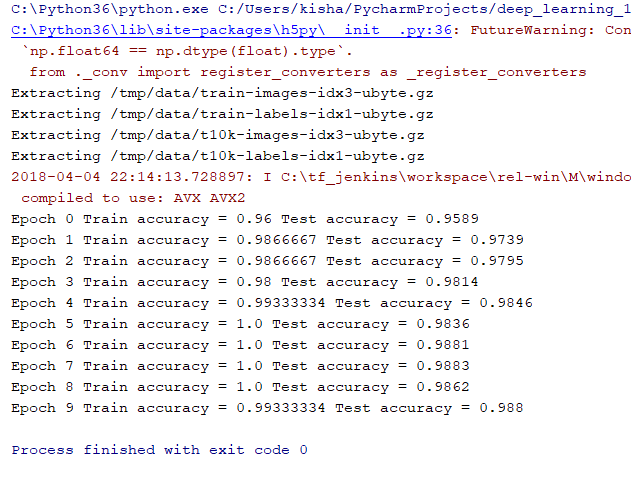
lstm1.py is RNN with the long short-term memory (BasicLSTMCell) for the MNIST data.  Run this program and submit a screenshot of the execution and result.  This may take 20 minutes (on my cheap laptop).

Replace BasicLSTMCell with GRUCell (the program should be named gru1.py then) and do the same.  You might need to replace also states[-1][1] with states[-1] to get the right shape (?, 150), according to Juntao Dong.  Jeremiah Greer showed another solution in class.

Computational graphs and node names for the two are attached for those of you who like to have a look.

**Answer:**

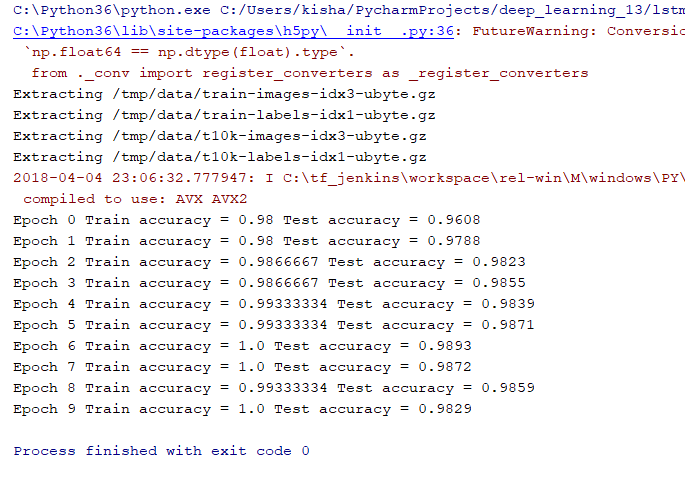
**Original Code: Output :**



After making changes in the code. After replacing the following

1. tf.contrib.rnn.BasicLSTMCell with tf.contrib.rnn.GRUCell
2. top\_layer\_h\_state = states[-1][1] with top\_layer\_h\_state = states[-1]

**Resultant Output:**



Code (with Changes as suggested in the question):

**import** tensorflow **as** tf  
  
n\_steps = 28  
n\_inputs = 28  
n\_neurons = 150  
n\_outputs = 10  
n\_layers = 3  
  
learning\_rate = 0.001  
  
X = tf.placeholder(tf.float32, [**None**, n\_steps, n\_inputs])  
y = tf.placeholder(tf.int32, [**None**])  
  
lstm\_cells = [tf.contrib.rnn.GRUCell(num\_units=n\_neurons)  
 **for** layer **in** range(n\_layers)]  
multi\_cell = tf.contrib.rnn.MultiRNNCell(lstm\_cells)  
outputs, states = tf.nn.dynamic\_rnn(multi\_cell, X, dtype=tf.float32)  
top\_layer\_h\_state = states[-1]  
logits = tf.layers.dense(top\_layer\_h\_state, n\_outputs, name=**"softmax"**)  
xentropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(labels=y, logits=logits)  
loss = tf.reduce\_mean(xentropy, name=**"loss"**)  
optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate)  
training\_op = optimizer.minimize(loss)  
correct = tf.nn.in\_top\_k(logits, y, 1)  
accuracy = tf.reduce\_mean(tf.cast(correct, tf.float32))  
   
init = tf.global\_variables\_initializer()  
  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
mnist = input\_data.read\_data\_sets(**"/tmp/data/"**)  
X\_test = mnist.test.images.reshape((-1, n\_steps, n\_inputs))  
y\_test = mnist.test.labels  
  
n\_epochs = 10  
batch\_size = 150  
  
**with** tf.Session() **as** sess:  
 init.run()  
 **for** epoch **in** range(n\_epochs):  
 **for** iteration **in** range(mnist.train.num\_examples // batch\_size):  
 X\_batch, y\_batch = mnist.train.next\_batch(batch\_size)  
 X\_batch = X\_batch.reshape((batch\_size, n\_steps, n\_inputs))  
 sess.run(training\_op, feed\_dict={X: X\_batch, y: y\_batch})  
 acc\_train = accuracy.eval(feed\_dict={X: X\_batch, y: y\_batch})  
 acc\_test = accuracy.eval(feed\_dict={X: X\_test, y: y\_test})  
 print(**"Epoch"**, epoch, **"Train accuracy ="**, acc\_train, **"Test accuracy ="**, acc\_test)

Code (Original without Changes):

**import** tensorflow **as** tf  
  
n\_steps = 28  
n\_inputs = 28  
n\_neurons = 150  
n\_outputs = 10  
n\_layers = 3  
  
learning\_rate = 0.001  
  
X = tf.placeholder(tf.float32, [**None**, n\_steps, n\_inputs])  
y = tf.placeholder(tf.int32, [**None**])  
  
lstm\_cells = [tf.contrib.rnn.BasicLSTMCell(num\_units=n\_neurons)  
 **for** layer **in** range(n\_layers)]  
multi\_cell = tf.contrib.rnn.MultiRNNCell(lstm\_cells)  
outputs, states = tf.nn.dynamic\_rnn(multi\_cell, X, dtype=tf.float32)  
top\_layer\_h\_state = states[-1][1]  
logits = tf.layers.dense(top\_layer\_h\_state, n\_outputs, name=**"softmax"**)  
xentropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(labels=y, logits=logits)  
loss = tf.reduce\_mean(xentropy, name=**"loss"**)  
optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate)  
training\_op = optimizer.minimize(loss)  
correct = tf.nn.in\_top\_k(logits, y, 1)  
accuracy = tf.reduce\_mean(tf.cast(correct, tf.float32))  
  
init = tf.global\_variables\_initializer()  
  
**from** tensorflow.examples.tutorials.mnist **import** input\_data  
  
mnist = input\_data.read\_data\_sets(**"/tmp/data/"**)  
X\_test = mnist.test.images.reshape((-1, n\_steps, n\_inputs))  
y\_test = mnist.test.labels  
  
n\_epochs = 10  
batch\_size = 150  
  
**with** tf.Session() **as** sess:  
 init.run()  
 **for** epoch **in** range(n\_epochs):  
 **for** iteration **in** range(mnist.train.num\_examples // batch\_size):  
 X\_batch, y\_batch = mnist.train.next\_batch(batch\_size)  
 X\_batch = X\_batch.reshape((batch\_size, n\_steps, n\_inputs))  
 sess.run(training\_op, feed\_dict={X: X\_batch, y: y\_batch})  
 acc\_train = accuracy.eval(feed\_dict={X: X\_batch, y: y\_batch})  
 acc\_test = accuracy.eval(feed\_dict={X: X\_test, y: y\_test})  
 print(**"Epoch"**, epoch, **"Train accuracy ="**, acc\_train, **"Test accuracy ="**, acc\_test)