Navya Battu
Class Id : 8
ICP Id : 14

Objective : Implement the Text classification with CNN model, RNN model, LSTM model and Compare results of CNN, RNN and LSTM for text classification and describe, which model is best for text classification based on your results

Features :

1. CNN
2. RNN
3. LSTM
4. MNIST data

Configurations :

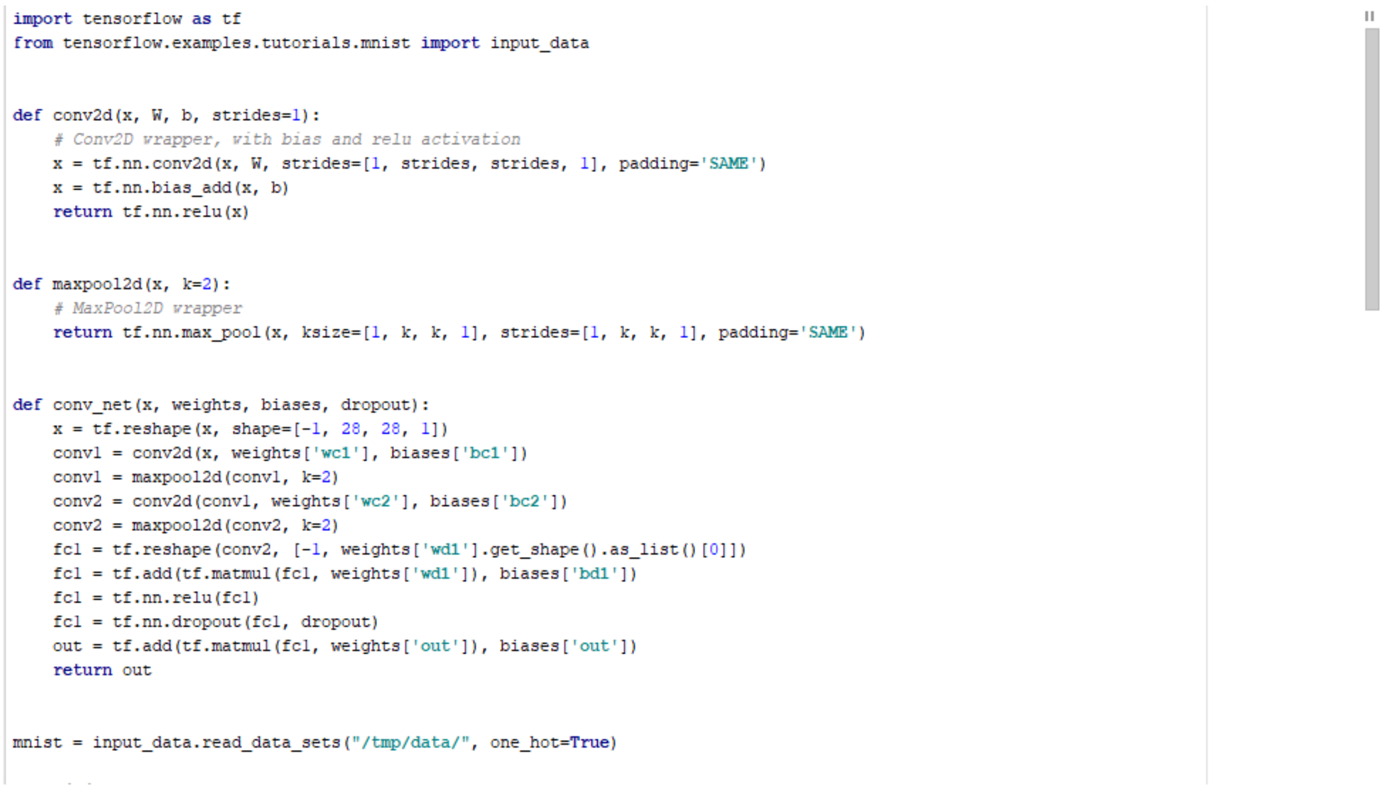
Spyder(ANACONDA)

Implementation & Deployment :

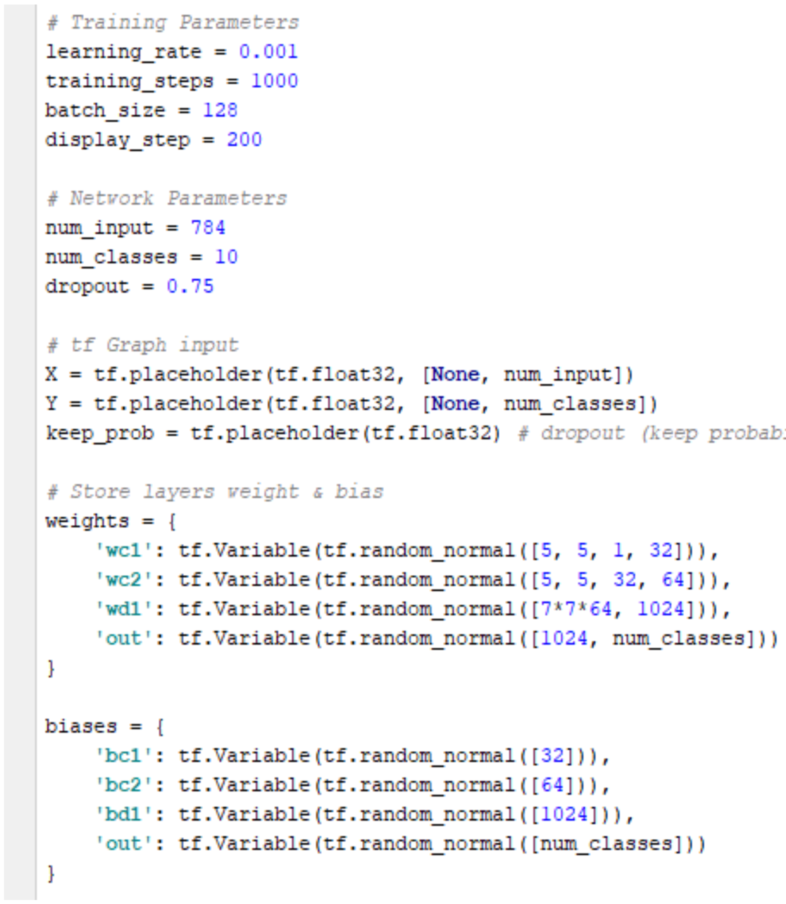
Question:

Input :

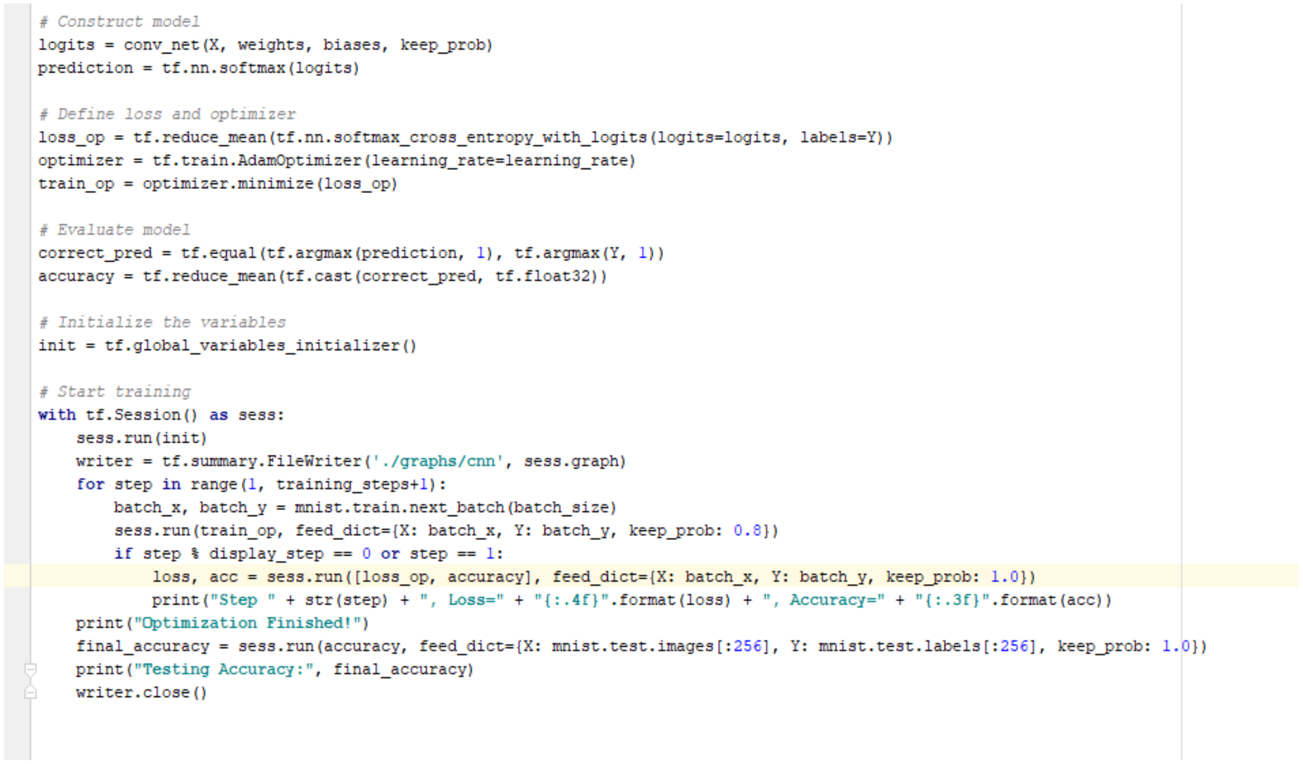
CNN :



```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data


def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)


def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME')


def conv_net(x, weights, biases, dropout):
    x = tf.reshape(x, shape=[-1, 28, 28, 1])
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    conv1 = maxpool2d(conv1, k=2)
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    conv2 = maxpool2d(conv2, k=2)
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    fc1 = tf.nn.dropout(fc1, dropout)
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out


mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

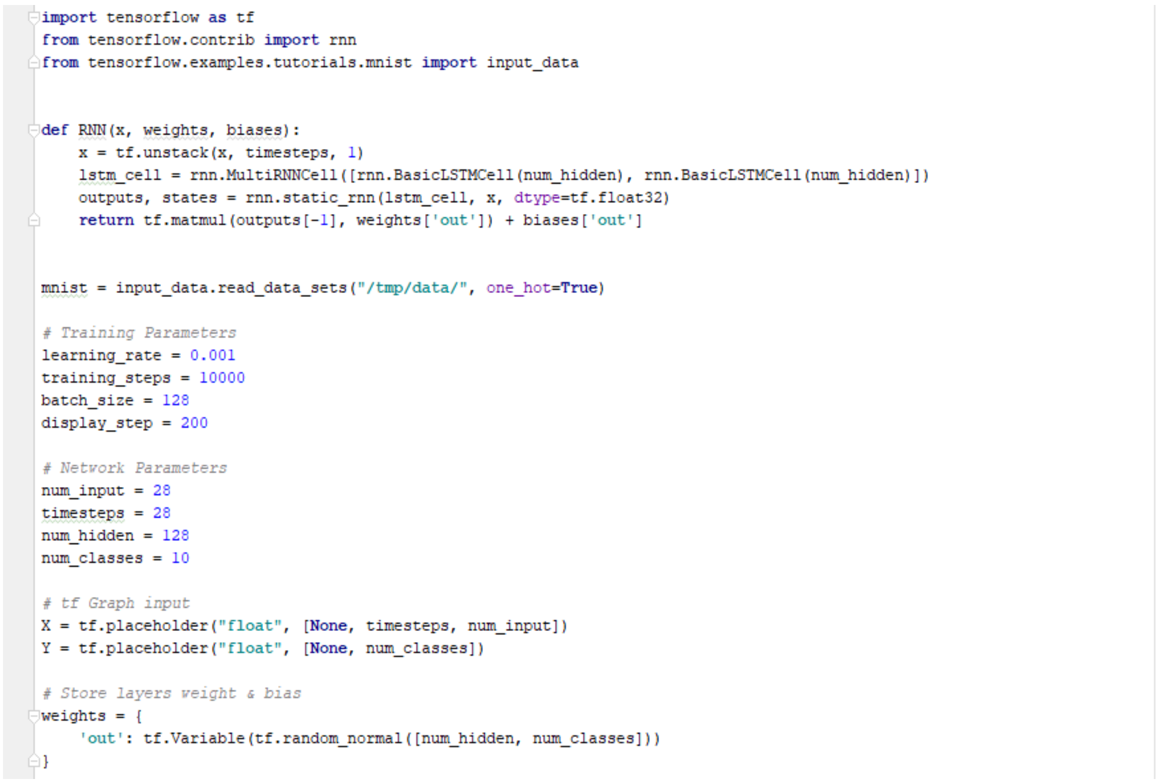


```python
# Training Parameters
learning_rate = 0.001
training_steps = 1000
batch_size = 128
display_step = 200

# Network Parameters
num_input = 784
num_classes = 10
dropout = 0.75

# tf Graph input
X = tf.placeholder(tf.float32, [None, num_input])
Y = tf.placeholder(tf.float32, [None, num_classes])
keep_prob = tf.placeholder(tf.float32) # dropout (keep probability)

# Store layers weight & bias
weights = {
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    'out': tf.Variable(tf.random_normal([1024, num_classes]))
}

biases = {
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([num_classes]))
}
```

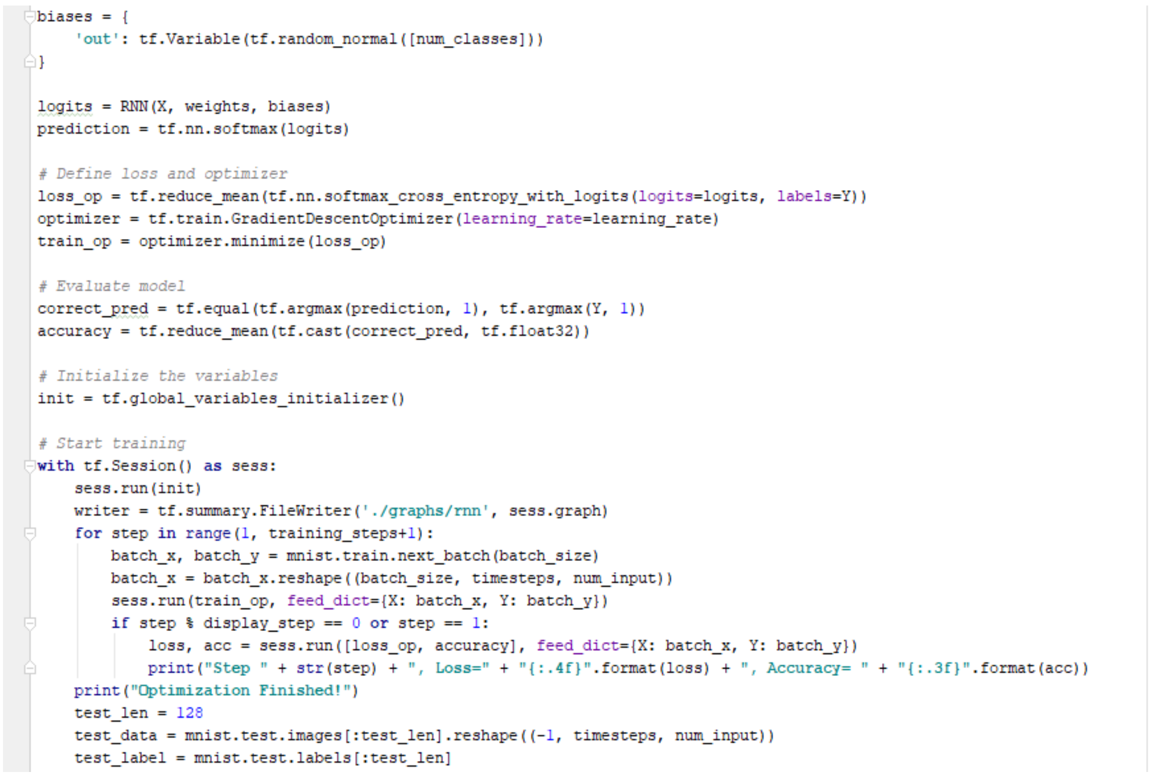


```python
# Construct model
logits = conv_net(X, weights, biases, keep_prob)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:
    sess.run(init)
    writer = tf.summary.FileWriter('./graphs/cnn', sess.graph)
    for step in range(1, training_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.8})
        if step % display_step == 0 or step == 1:
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x, Y: batch_y, keep_prob: 1.0})
            print("Step " + str(step) + ", Loss=" + "{:.4f}".format(loss) + ", Accuracy=" + "{:.3f}".format(acc))
    print("Optimization Finished!")
    final_accuracy = sess.run(accuracy, feed_dict={X: mnist.test.images[:256], Y: mnist.test.labels[:256], keep_prob: 1.0})
    print("Testing Accuracy:", final_accuracy)
    writer.close()
```

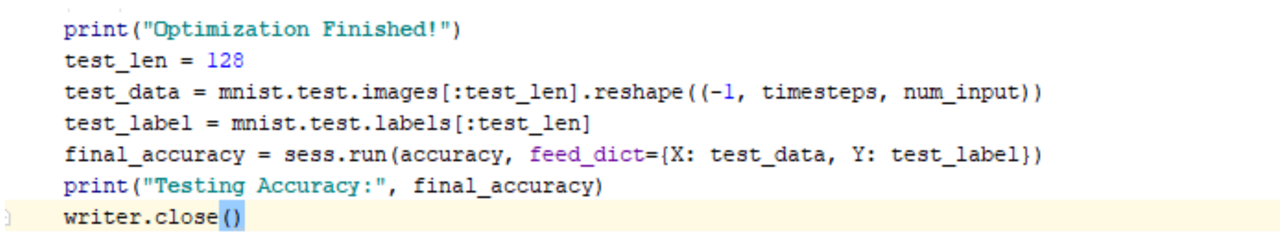
RNN :



```python
import tensorflow as tf
from tensorflow.contrib import rnn
from tensorflow.examples.tutorials.mnist import input_data


def RNN(x, weights, biases):
    x = tf.unstack(x, timesteps, 1)
    lstm_cell = rnn.MultiRNNCell([rnn.BasicLSTMCell(num_hidden), rnn.BasicLSTMCell(num_hidden)])
    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)
    return tf.matmul(outputs[-1], weights['out']) + biases['out']


mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

# Training Parameters
learning_rate = 0.001
training_steps = 10000
batch_size = 128
display_step = 200

# Network Parameters
num_input = 28
timesteps = 28
num_hidden = 128
num_classes = 10

# tf Graph input
X = tf.placeholder("float", [None, timesteps, num_input])
Y = tf.placeholder("float", [None, num_classes])

# Store layers weight & bias
weights = {
    'out': tf.Variable(tf.random_normal([num_hidden, num_classes]))
}
```

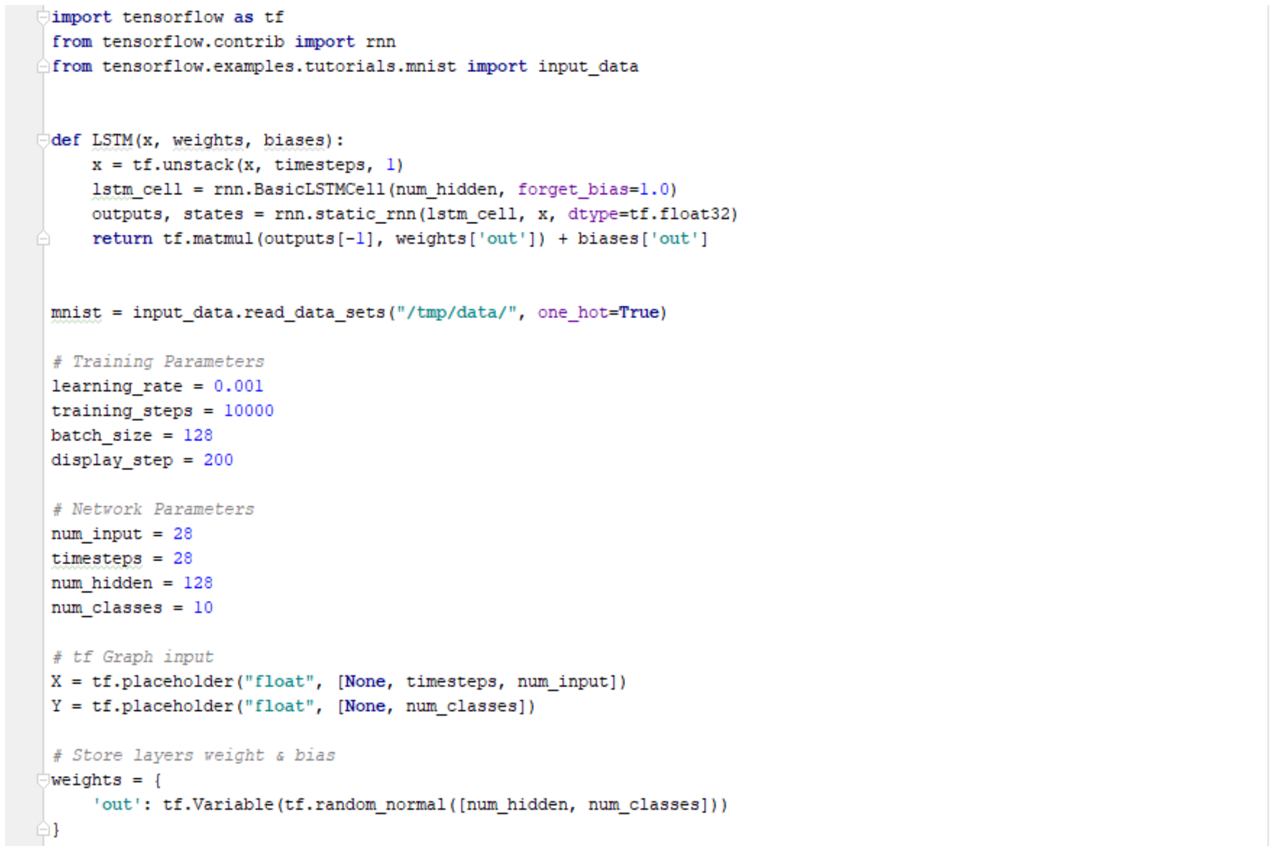


```python
biases = {
    'out': tf.Variable(tf.random_normal([num_classes]))
}

logits = RNN(X, weights, biases)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:
    sess.run(init)
    writer = tf.summary.FileWriter('./graphs/rnn', sess.graph)
    for step in range(1, training_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        batch_x = batch_x.reshape((batch_size, timesteps, num_input))
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
        if step % display_step == 0 or step == 1:
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x, Y: batch_y})
            print("Step " + str(step) + ", Loss=" + "{:.4f}".format(loss) + ", Accuracy= " + "{:.3f}".format(acc))
    print("Optimization Finished!")
    test_len = 128
    test_data = mnist.test.images[:test_len].reshape((-1, timesteps, num_input))
    test_label = mnist.test.labels[:test_len]
```

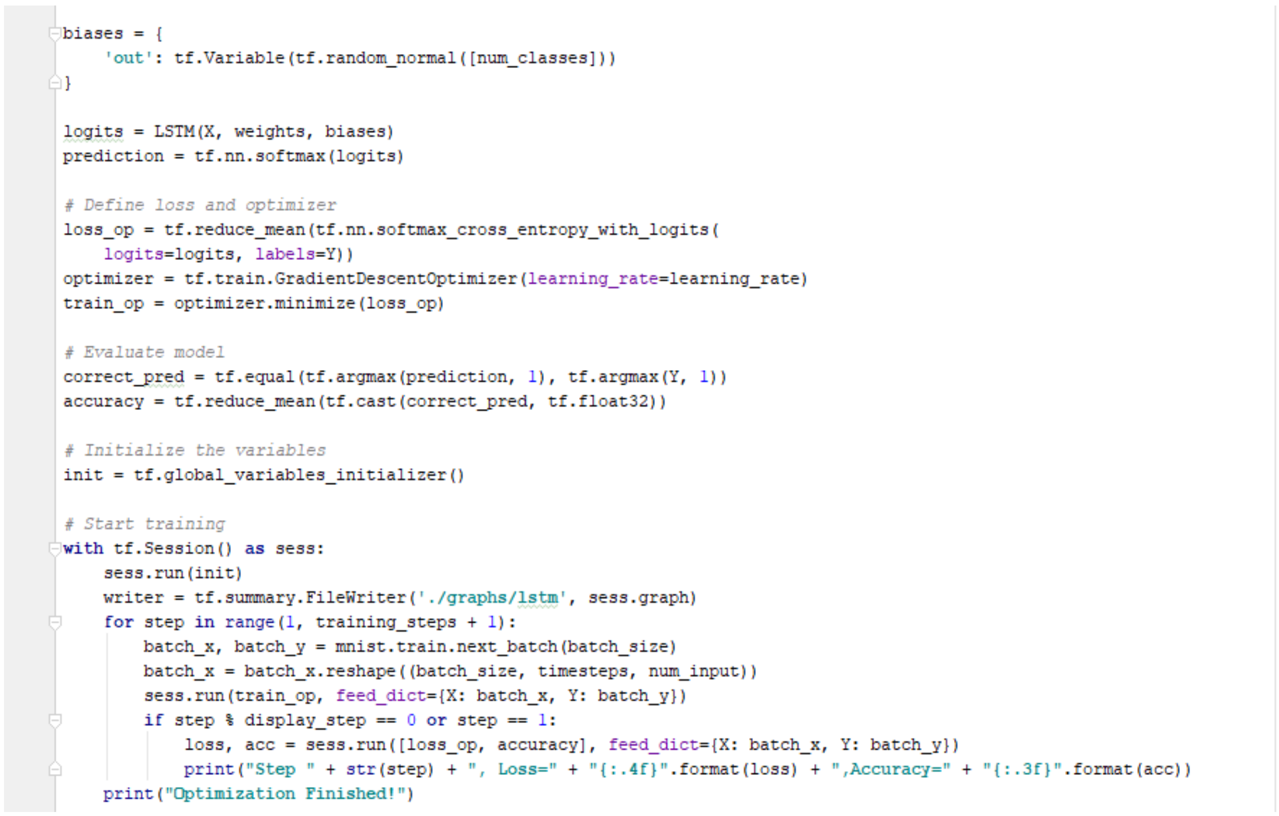


```
    print("Optimization Finished!")
    test_len = 128
    test_data = mnist.test.images[:test_len].reshape((-1, timesteps, num_input))
    test_label = mnist.test.labels[:test_len]
    final_accuracy = sess.run(accuracy, feed_dict={X: test_data, Y: test_label})
    print("Testing Accuracy:", final_accuracy)
    writer.close()
```

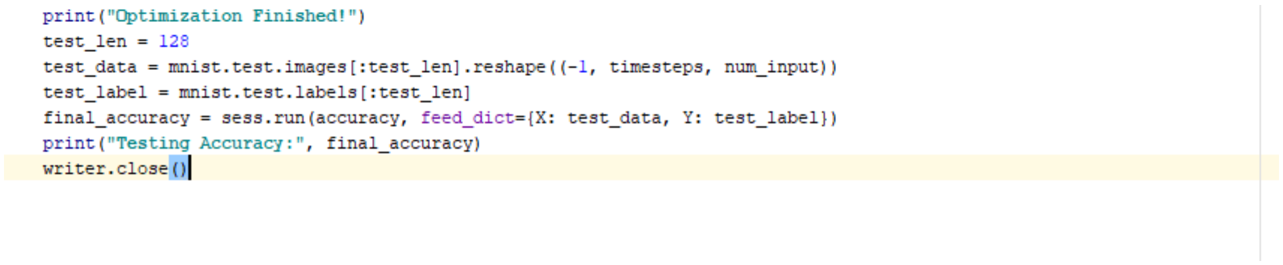
LSTM :



```
import tensorflow as tf
from tensorflow.contrib import rnn
from tensorflow.examples.tutorials.mnist import input_data


def LSTM(x, weights, biases):
    x = tf.unstack(x, timesteps, 1)
    lstm_cell = rnn.BasicLSTMCell(num_hidden, forget_bias=1.0)
    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)
    return tf.matmul(outputs[-1], weights['out']) + biases['out']


mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

# Training Parameters
learning_rate = 0.001
training_steps = 10000
batch_size = 128
display_step = 200

# Network Parameters
num_input = 28
timesteps = 28
num_hidden = 128
num_classes = 10

# tf Graph input
X = tf.placeholder("float", [None, timesteps, num_input])
Y = tf.placeholder("float", [None, num_classes])

# Store layers weight & bias
weights = {
    'out': tf.Variable(tf.random_normal([num_hidden, num_classes]))
}
```



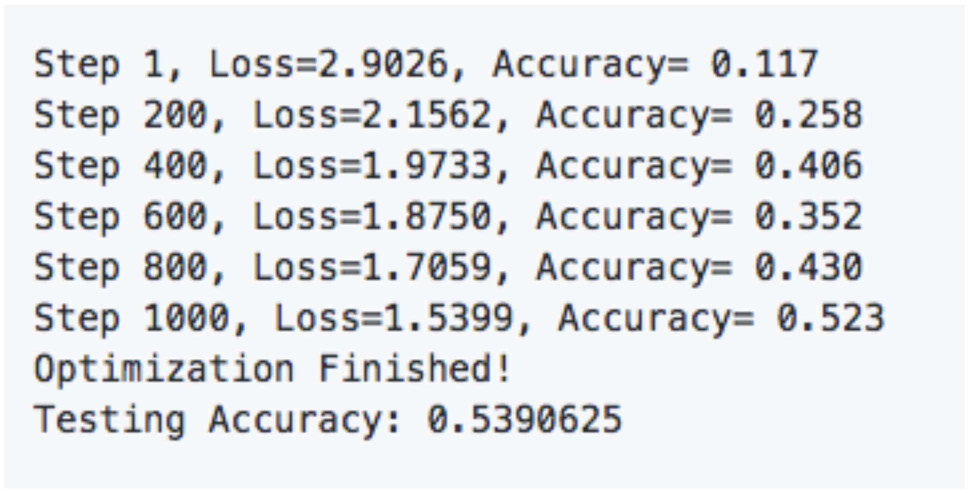
```python
biases = {
    'out': tf.Variable(tf.random_normal([num_classes]))
}

logits = LSTM(X, weights, biases)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:
    sess.run(init)
    writer = tf.summary.FileWriter('./graphs/lstm', sess.graph)
    for step in range(1, training_steps + 1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        batch_x = batch_x.reshape((batch_size, timesteps, num_input))
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
        if step % display_step == 0 or step == 1:
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x, Y: batch_y})
            print("Step " + str(step) + ", Loss=" + "{:.4f}".format(loss) + ",Accuracy=" + "{:.3f}".format(acc))
    print("Optimization Finished!")
```



```python
print("Optimization Finished!")
test_len = 128
test_data = mnist.test.images[:test_len].reshape((-1, timesteps, num_input))
test_label = mnist.test.labels[:test_len]
final_accuracy = sess.run(accuracy, feed_dict={X: test_data, Y: test_label})
print("Testing Accuracy:", final_accuracy)
writer.close()
```

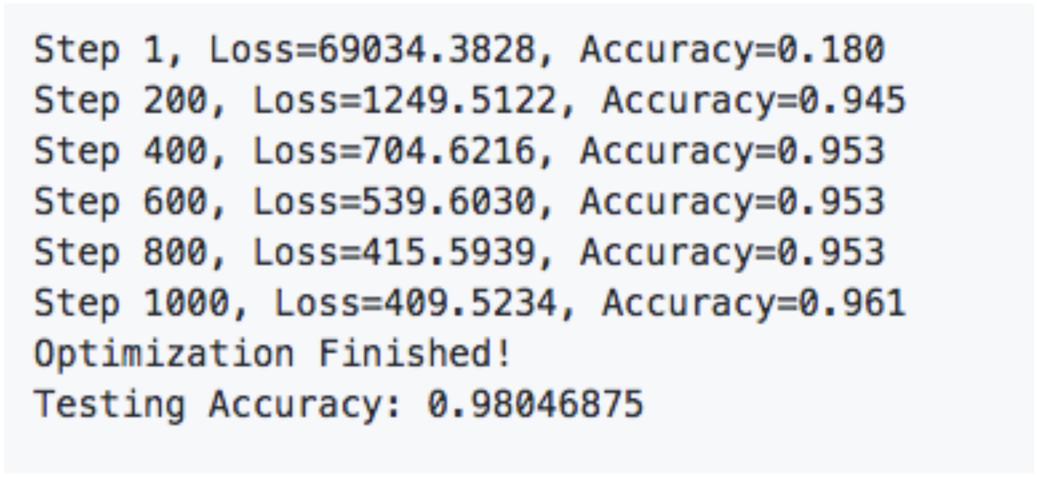
Output :

CNN :



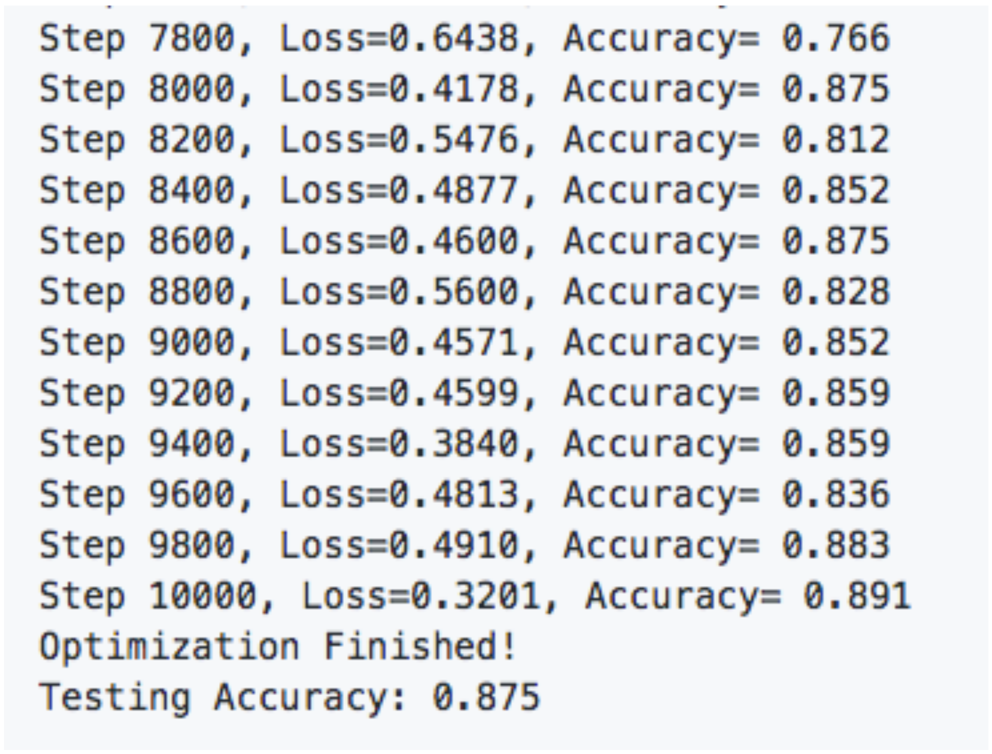
```
Step 1, Loss=2.9026, Accuracy= 0.117
Step 200, Loss=2.1562, Accuracy= 0.258
Step 400, Loss=1.9733, Accuracy= 0.406
Step 600, Loss=1.8750, Accuracy= 0.352
Step 800, Loss=1.7059, Accuracy= 0.430
Step 1000, Loss=1.5399, Accuracy= 0.523
Optimization Finished!
Testing Accuracy: 0.5390625
```

RNN



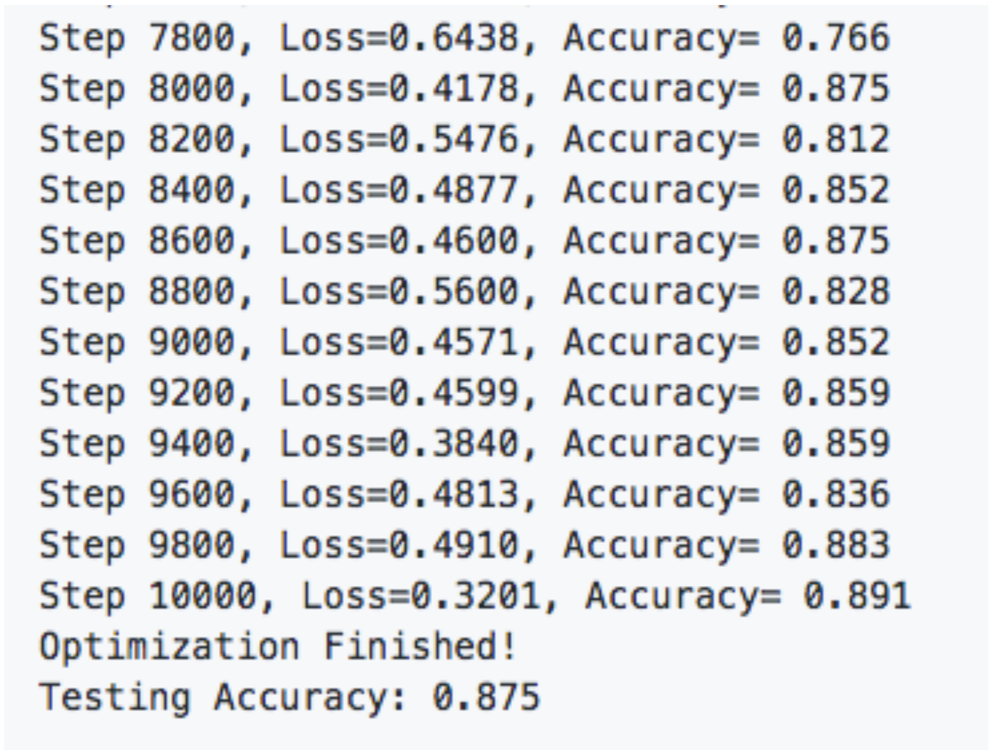
```
Step 1, Loss=69034.3828, Accuracy=0.180
Step 200, Loss=1249.5122, Accuracy=0.945
Step 400, Loss=704.6216, Accuracy=0.953
Step 600, Loss=539.6030, Accuracy=0.953
Step 800, Loss=415.5939, Accuracy=0.953
Step 1000, Loss=409.5234, Accuracy=0.961
Optimization Finished!
Testing Accuracy: 0.98046875
```

For 10000 steps :



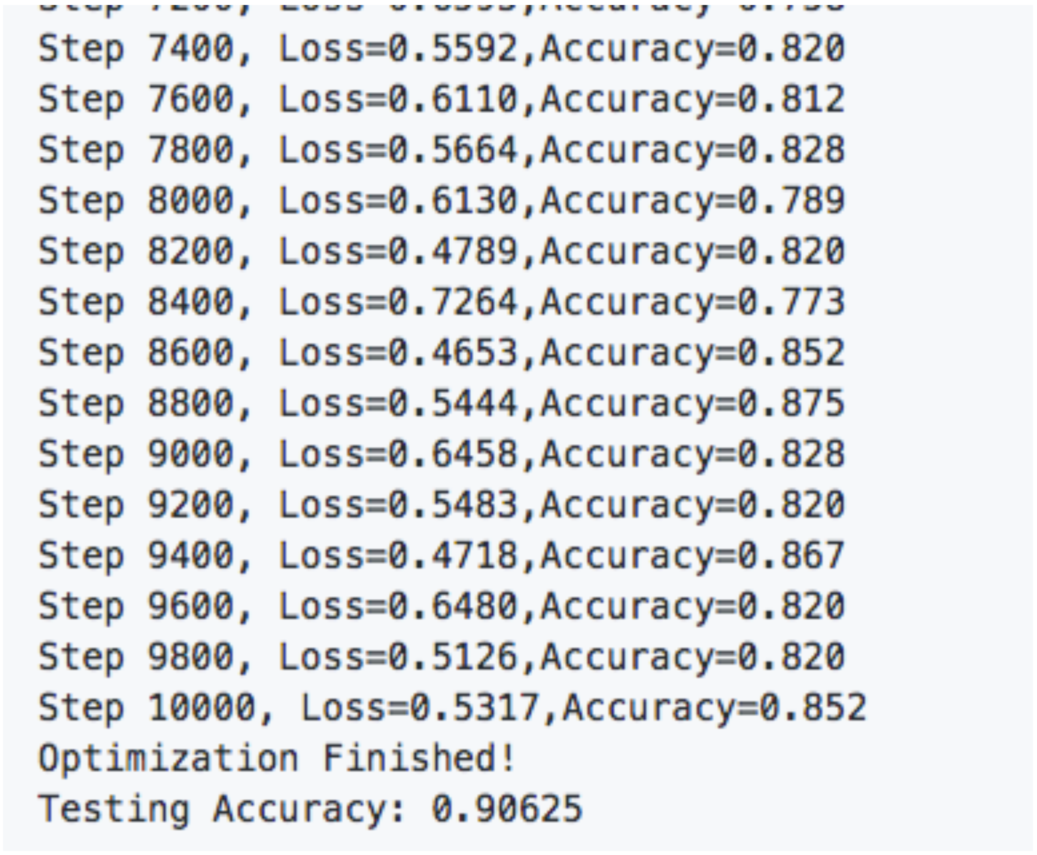
```
Step 7800, Loss=0.6438, Accuracy= 0.766
Step 8000, Loss=0.4178, Accuracy= 0.875
Step 8200, Loss=0.5476, Accuracy= 0.812
Step 8400, Loss=0.4877, Accuracy= 0.852
Step 8600, Loss=0.4600, Accuracy= 0.875
Step 8800, Loss=0.5600, Accuracy= 0.828
Step 9000, Loss=0.4571, Accuracy= 0.852
Step 9200, Loss=0.4599, Accuracy= 0.859
Step 9400, Loss=0.3840, Accuracy= 0.859
Step 9600, Loss=0.4813, Accuracy= 0.836
Step 9800, Loss=0.4910, Accuracy= 0.883
Step 10000, Loss=0.3201, Accuracy= 0.891
Optimization Finished!
Testing Accuracy: 0.875
```

LSTM



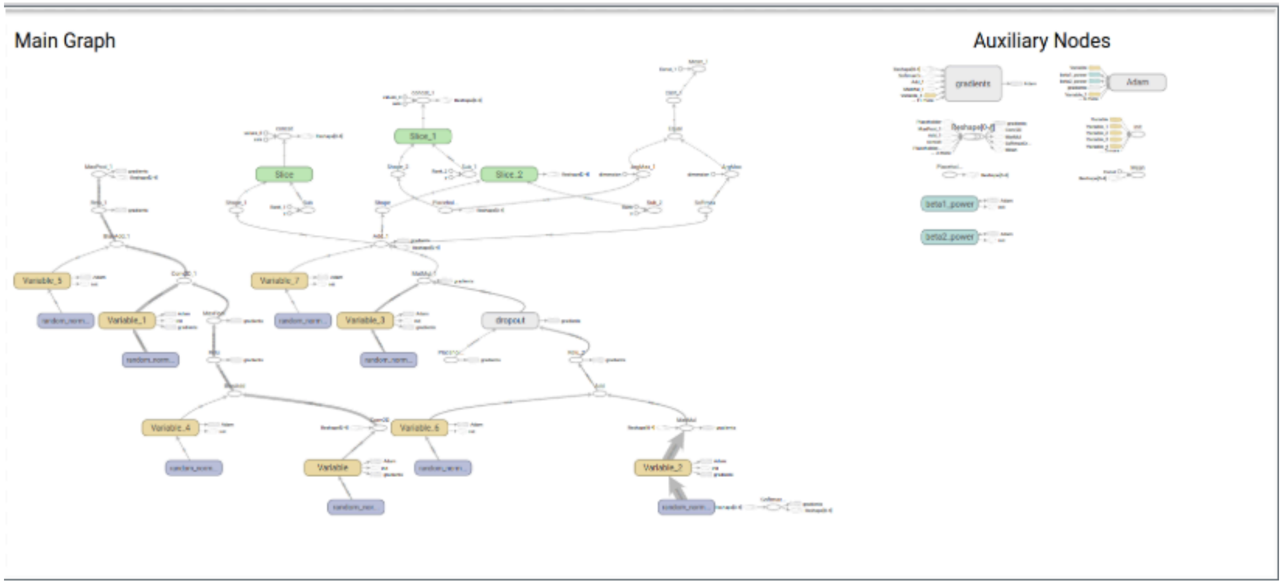
```
Step 7800, Loss=0.6438, Accuracy= 0.766
Step 8000, Loss=0.4178, Accuracy= 0.875
Step 8200, Loss=0.5476, Accuracy= 0.812
Step 8400, Loss=0.4877, Accuracy= 0.852
Step 8600, Loss=0.4600, Accuracy= 0.875
Step 8800, Loss=0.5600, Accuracy= 0.828
Step 9000, Loss=0.4571, Accuracy= 0.852
Step 9200, Loss=0.4599, Accuracy= 0.859
Step 9400, Loss=0.3840, Accuracy= 0.859
Step 9600, Loss=0.4813, Accuracy= 0.836
Step 9800, Loss=0.4910, Accuracy= 0.883
Step 10000, Loss=0.3201, Accuracy= 0.891
Optimization Finished!
Testing Accuracy: 0.875
```

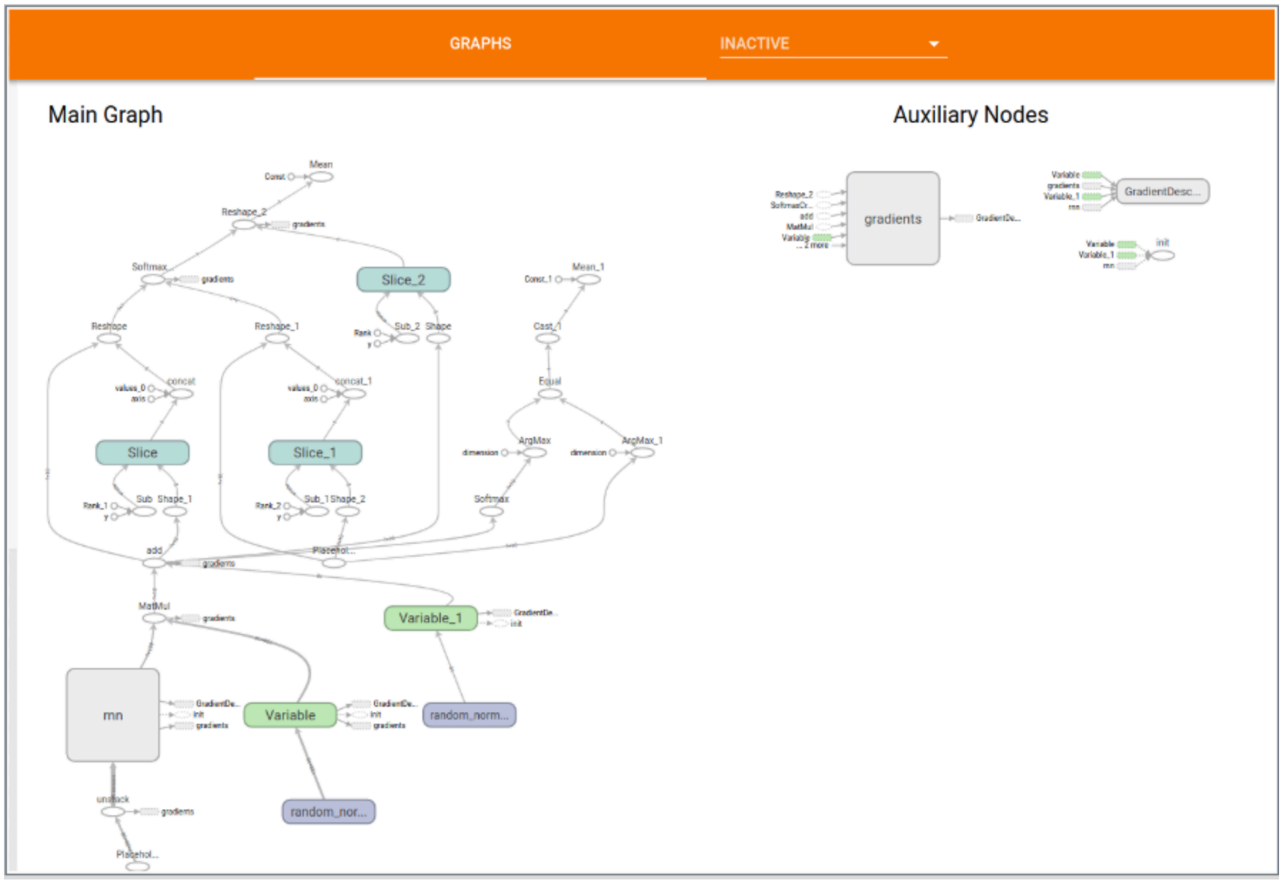
For 10000 steps :

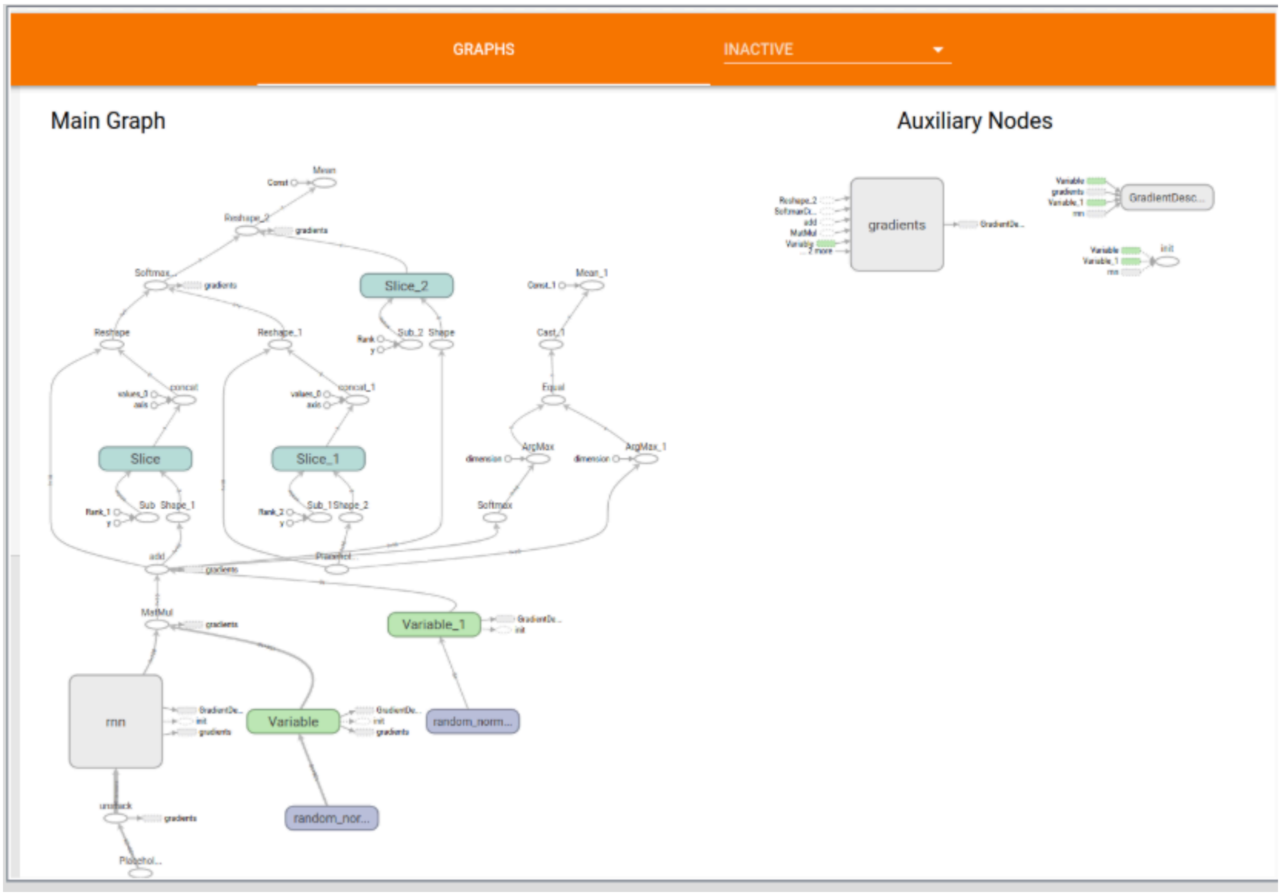


```
Step 7400, Loss=0.5592,Accuracy=0.820
Step 7600, Loss=0.6110,Accuracy=0.812
Step 7800, Loss=0.5664,Accuracy=0.828
Step 8000, Loss=0.6130,Accuracy=0.789
Step 8200, Loss=0.4789,Accuracy=0.820
Step 8400, Loss=0.7264,Accuracy=0.773
Step 8600, Loss=0.4653,Accuracy=0.852
Step 8800, Loss=0.5444,Accuracy=0.875
Step 9000, Loss=0.6458,Accuracy=0.828
Step 9200, Loss=0.5483,Accuracy=0.820
Step 9400, Loss=0.4718,Accuracy=0.867
Step 9600, Loss=0.6480,Accuracy=0.820
Step 9800, Loss=0.5126,Accuracy=0.820
Step 10000, Loss=0.5317,Accuracy=0.852
Optimization Finished!
Testing Accuracy: 0.90625
```

Graph :




GRAPHS
INACTIVE
Main Graph
Auxiliary Nodes
Mean
Const
Reshape_2
gradients
Softmax
Slice_2
Mean_1
Const_1
Reshape
Reshape_1
Rank
Sub_2
Shape
Cast_1
values_0
axis
concat
concat_1
Equal
Slice
Slice_1
ArgMax
ArgMax_1
dimension
Rank_1
Sub
Shape_1
Rank_2
Sub_1
Shape_2
Softmax
add
Placehol...
MatMul
Variable_1
GradientDe...
init
rnn
Variable
random_norm...
unstack
random_nor...
gradients
GradientDesc...
Reshape_2
SoftmaxCr...
Variable_1
... 2 more


GRAPHS
INACTIVE
Main Graph
Auxiliary Nodes
Mean
Const
Reshape_2
gradients
Softmax
Slice_2
Mean_1
Const_1
Reshape
Reshape_1
Rank
Sub_2
Shape
Cast_1
values_0
axis
concat
concat_1
Equal
Slice
Slice_1
ArgMax
ArgMax_1
dimension
Rank_1
Sub
Shape_1
Rank_2
Sub_1
Shape_2
Softmax
add
Placehol...
MatMul
Variable_1
GradientDe...
init
rnn
Variable
random_norm...
unstack
random_nor...
gradients
GradientDesc...
Reshape_2
SoftmaxCr...
Variable_1
... 2 more

Explanation :

1. Dataset loaded
2. Modification done as per the requirements
3. CNN, RNN, LSTM models need to run
4. Before that accuracy, loss, graph calculated and represented in tensorboard
5. Output of all the models will be evaluated and checking them for different parameters.

Conclusion :

Here LSTM performs better compare to other given models.
For MNIST dataset this model works better . Best model may differ based on the type of data we picked .

References :

http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/
https://towardsdatascience.com/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-d2ee64b9dd0b?gi=6f2045e19315
https://richliao.github.io/supervised/classification/2016/11/26/textclassifier-convolutional/
https://theneuralperspective.com/2016/10/06/recurrent-neural-networks-rnn-part-2-text-classification/
https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/