Introduction

Implementing the Text Classification with CNN, RNN and LSTM model with new dataset and displaying the graphs in the tensor board. At the end compare results and decide which is the better model.

Objectives

1) To perform the text classification with Convolutional Neural Networks model
2) To perform the text classification with Recurrent Neural Networks model
3) To perform the text classification with Long Short-Term Memory
4) Then compare the results of three models and decide the best model

Approaches/Methods

In the beginning, I choose the MNIST as the input file. Here we will develop to perform the CNN, RNN and LSTM to perform the text evaluation and display the graphs in tensor board with filewriter and summaries. Then compare the model by their accuracies

Workflow

1-> Import the dataset

2-> Change some parameters in the code

3-> Find the accuracy and loss for each model

4-> Plot the graph in TensorFlow for each model

5-> Use same parameters in all the models and evaluate all the models

6-> The compare all the models and find the best model

Dataset

MNIST Dataset

Parameters

- Learning Rate
- Training Steps
- Batch Size
- Display Step

Evaluation

Evaluation of Convolutional Neural Networks Model:

```
def cnn(x, weights, biases, dropout):
    x = tf.reshape(x, shape=[-1, 28, 28, 1])
    y = conv2d(x, weights['wc1'], biases['bc1'])
    y = maxpool2d(y, k=2)
    z = conv2d(y, weights['wc2'], biases['bc2'])
    z = maxpool2d(z, k=2)
    out1 = tf.reshape(z, [-1, weights['wd1'].get_shape().as_list()[0]])
    out1 = tf.add(tf.matmul(out1, weights['wd1']), biases['bd1'])
    out1 = tf.nn.relu(out1)
    out1 = tf.nn.dropout(out1, dropout)
    out = tf.add(tf.matmul(out1, weights['out']), biases['out'])
    return out
```

Evaluation of Recurrent Neural Network Model:

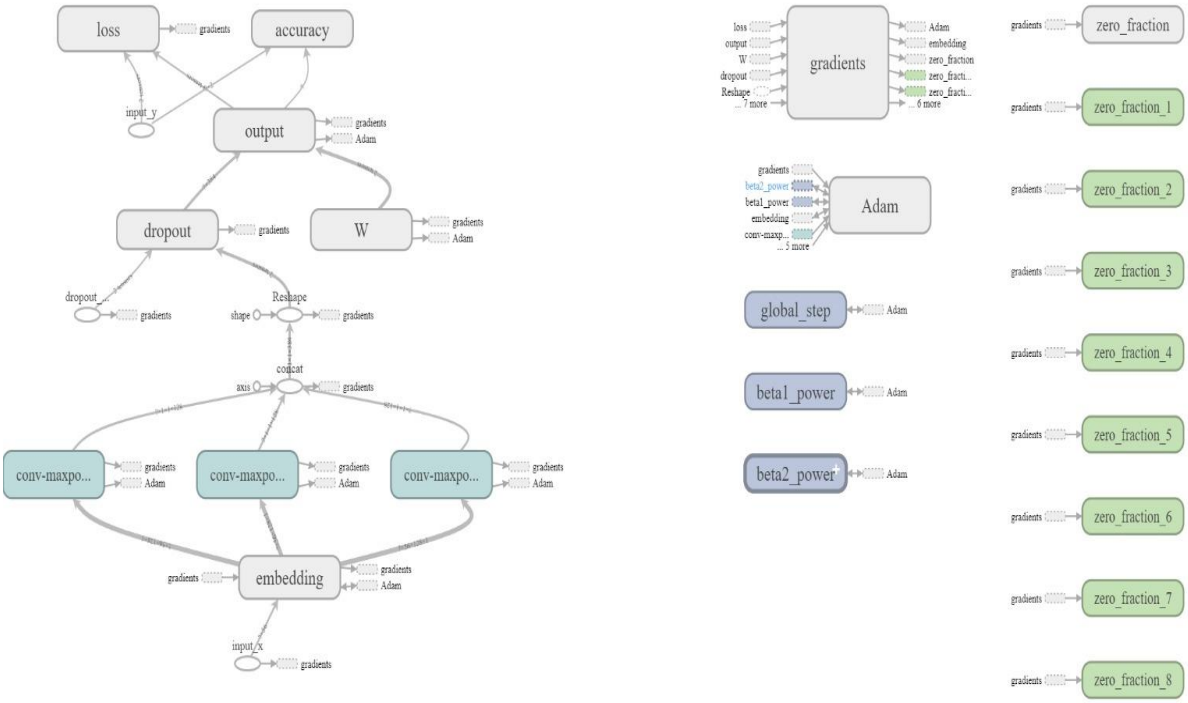
```
def RNN(x, weights, biases):
    x = tf.unstack(x, timesteps, 1)
    l = rnn.MultiRNNCell([rnn.BasicLSTMCell(num_hidden), rnn.BasicLSTMCell(num_hidden)])
    outputs, states = rnn.static_rnn(l, x, dtype=tf.float32)
    return tf.matmul(outputs[-1], weights['out']) + biases['out']
```

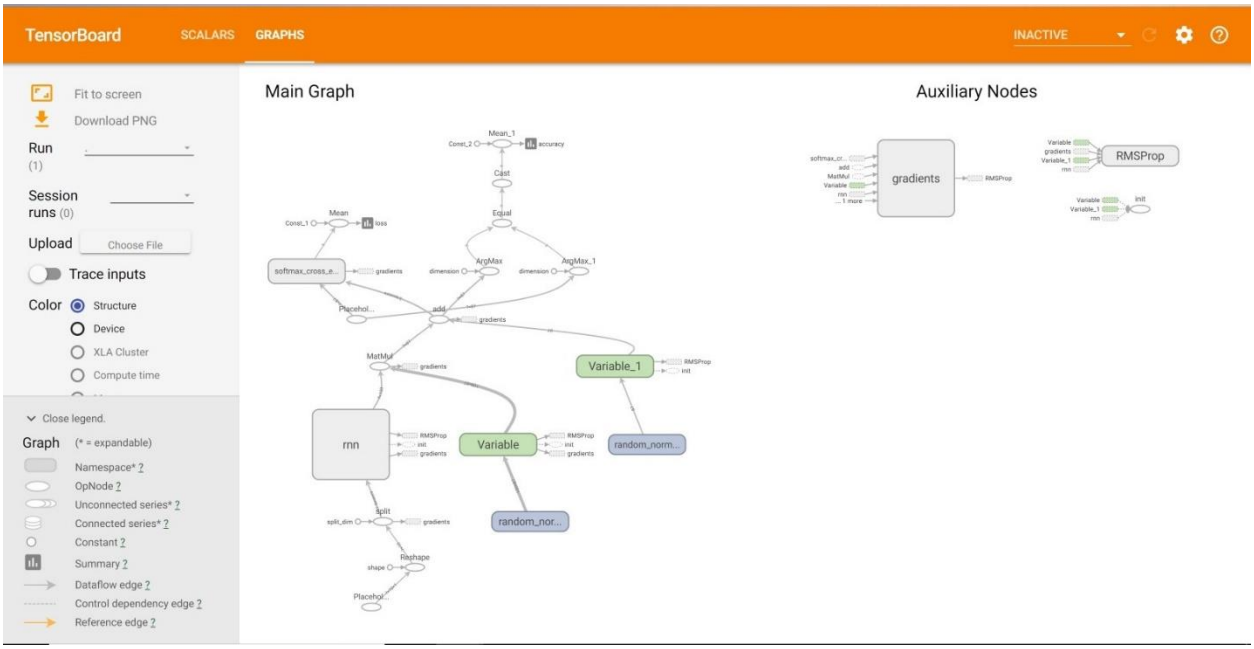
Evaluation of Long Short Term Memory: (LSTM units are building units of layers of a RNN)

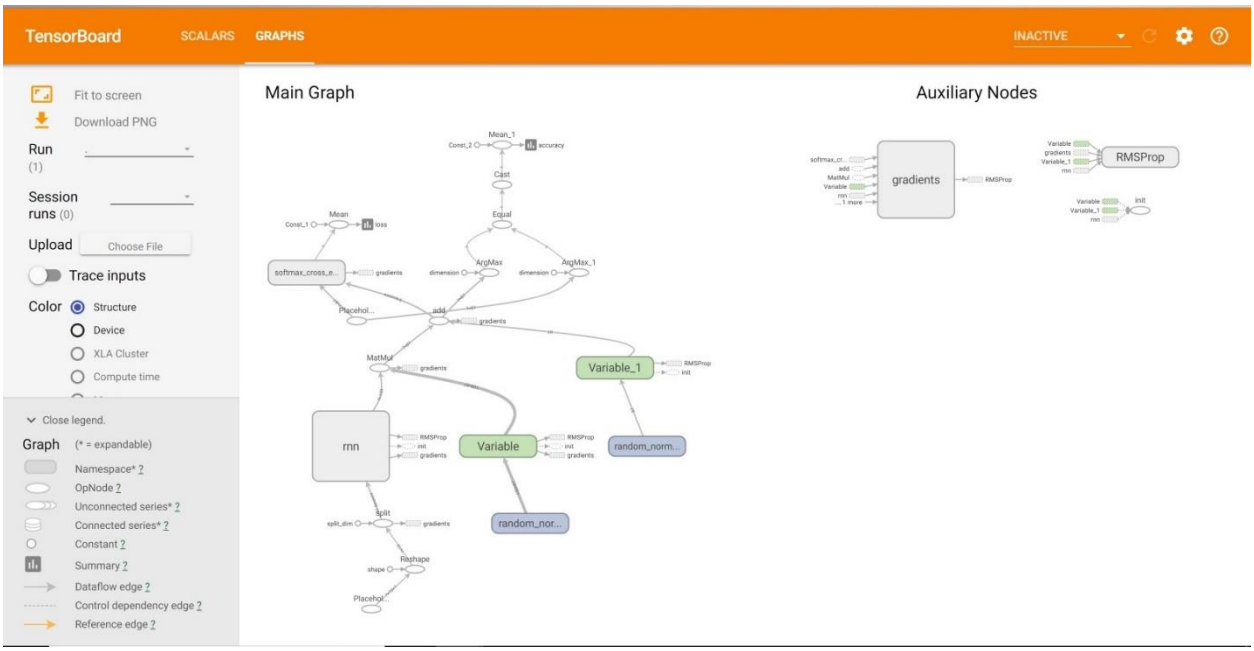
```
def LSTM(x, weights, biases):
    x = tf.unstack(x, timesteps, 1)
    l = rnn.BasicLSTMCell(num_hidden, forget_bias=1.0)
    out, states = rnn.static_rnn(l, x, dtype=tf.float32)
    return tf.matmul(out[-1], weights['out']) + biases['out']
```

The display step here is 200. So for every 200 steps it will display the output and learning rate is taken as 0.001 and training steps as 1000. The batch size is 200.

Graphs in Tensor Board for CNN, RNN and LSTM respectively:




TensorBoard
SCALARS
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Close legend.
Graph (* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
Main Graph
Auxiliary Nodes
gradients
RMSProp
rnn
Variable
Variable_1
random_norm...
random_nor...


TensorBoard
SCALARS
GRAPHS
INACTIVE
Fit to screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Close legend.
Graph (* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
Main Graph
Auxiliary Nodes
gradients
RMSProp
rnn
Variable
Variable_1
random_norm...
random_nor...

Output:

CNN

```
Step 1, Loss=71034.3828, Accuracy=0.170
Step 200, Loss=1249.5122, Accuracy=0.924
Step 400, Loss=704.6216, Accuracy=0.943
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: 0.97035926
```

RNN

```
Step 1, Loss=2.9026, Accuracy= 0.109
Step 200, Loss=2.1562, Accuracy= 0.198
Step 400, Loss=1.9733, Accuracy= 0.402
Step 600, Loss=1.8750, Accuracy= 0.352
Step 800, Loss=1.7059, Accuracy= 0.430
Step 1000, Loss=1.5399, Accuracy= 0.513
Optimization Finished!
Testing Accuracy: 0.5240765
```

for 10000 training steps

```
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 1, Loss=2.5708,Accuracy=0.086
Step 200, Loss=2.0772,Accuracy=0.328
Step 400, Loss=1.9447,Accuracy=0.297
Step 600, Loss=1.8180,Accuracy=0.398
Step 800, Loss=1.6701,Accuracy=0.484
Step 1000, Loss=1.4923,Accuracy=0.523
Optimization Finished!
Testing Accuracy: 0.4609375
```

for 10000 training steps

```
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
```

Conclusion

We conclude that Convolutional Neural Network Model is best among the three models depending on the accuracy and loss outputs of the models. If we increase more number of training steps, the accuracy increases, and loss increases but the time complexity also increases. LSTM stands in the second position and RNN in third position.