# COMP9444 Report

By Chengbin Feng z5109259

The task of our assignment sentiment analysis involves taking in an input sequence of words and determining whether the sentiment is positive, negative, or neutral.

**In preprocess() method:** I clean the remove the words in stop words and punctuations. I also lower the case of each reviews and output as a list format.

**In define_graph() method:**

Firstly, I specify two placeholders one is one for the inputs into the network, and one for the labels.

```
input_data= tf.placeholder(dtype=tf.float32,shape=[BATCH_SIZE,MAX_WORDS_IN_REVIEW, EMBEDDING_SIZE], name='input_data')
labels = tf.placeholder(dtype=tf.float32,shape=[BATCH_SIZE,2], name='labels')
```

The *input_data* is a placeholder to hold each training example that we include in our batch. The *labels* placeholder represents a set of values, each either [1, 0] or [0, 1] which represent whether each training example is positive or negative.

```
#Reference: https://github.com/tensorflow/tensorflow/issues/16186
def lstm_cell():
    lstm = tf.contrib.rnn.BasicLSTMCell(BATCH_SIZE, forget_bias=1.0, state_is_tuple=True, dtype=tf.float32)
#    lstm = tf.contrib.rnn.DropoutWrapper(cell=lstm, output_keep_prob=tf.placeholder_with_default(0.75, shape=[], name='dropout_keep_prob'))
    return lstm

l_cell = tf.contrib.rnn.MultiRNNCell([lstm_cell() for _ in range(2)]) #Double lstm

init = lstm.zero_state(BATCH_SIZE, dtype=tf.float32)
outputs, states= tf.nn.dynamic_rnn(lstm, input_data, initial_state=init)
```

Then I'll feed both the two LSTM cells and the 3-D tensor full of input data into a function called tf.nn.dynamic_rnn. This function is in charge of unrolling the whole network and creating a pathway for the data to flow through the RNN graph.

```
dense = tf.layers.dense(outputs[:,-1], 50, activation=tf.nn.relu)
drop_second = tf.layers.dropout(dense, rate=(1-dropout_keep_prob))
```

And try to reduce the overfiting in dense layer by dropout technical.

```
logits= tf.layers.dense(inputs=states.h, units=2, activation=None)
```

Logits will get the of except output of size 2 densely-connected layer function interface.

```
dense = tf.layers.dense(outputs[:,-1], 50, activation=tf.nn.relu)
drop_second = tf.layers.dropout(dense, rate=(1-dropout_keep_prob))
```

```
loss=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,labels=labels), name='loss')
optimizer = tf.train.AdamOptimizer(learning_rate=0.0001).minimize(loss)
```

I define a standard cross entropy loss with a softmax layer put on top of the final prediction values. For the optimizer, we'll use Adam and the assigned learning rate of .0001in order to get the more accuracy weight to improve the total accuracy.

```
correct_prediction = tf.equal(tf.argmax(logits,1),tf.argmax(labels,1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32), name='accuracy')
```

I define the correct prediction and accuracy metrics to track how the network is doing. The correct prediction formulation works by looking at the index of the maximum value of the 2 output values, and then seeing whether it matches with the training labels.