

# lstm\_with\_unique\_7

December 12, 2018

```
In [1]: # LSTM WITH UNIQUE DATASET IMPLEMENTATION 7
        # Depression Analysis in Bangla
        # copyright (c) ABDUL HASIB UDDIN <abdulhasibuddin@gmail.com>
        # LICENSE: GNU General Public License v3.0
        # Courtesy: https://github.com/mchablani/deep-learning/blob/master/sentiment-rnn/Senti
```

```
In [0]: import numpy as np
        import tensorflow as tf
        from timeit import default_timer as timer
        from collections import Counter
        from string import punctuation
        from google.colab import files
```

```
In [0]: # Build the graph::
```

```
lstm_size = 128
lstm_layers = 5
batch_size = 50
learning_rate = 0.0001
epochs = 10
```

```
In [4]: fileName = "lstm_with_unique_7"
        checkpointName = "checkpoints/"+fileName+".ckpt"
        print(checkpointName)
        print(type(checkpointName))
```

```
checkpoints/lstm_with_unique_7.ckpt
<class 'str'>
```

```
In [5]: files.upload()
        files.upload()

        with open('data_all_unique_dnd_stratified_text.txt', 'r', encoding="utf8") as f:
            tweets = f.read()
        with open('data_all_unique_dnd_stratified_labels.txt', 'r', encoding="utf8") as f:
            labels_org = f.read()
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```
In [0]: # Data preprocessing::
        #all_text = ''.join([c for c in tweets if c not in punctuation])
        all_text = ''.join([c for c in tweets])
        tweets = all_text.split('\n')

        all_text = ' '.join(tweets)
        words = all_text.split()

In [0]: counts = Counter(words)
        vocab = sorted(counts, key=counts.get, reverse=True)
        vocab_to_int = {word: ii for ii, word in enumerate(vocab, 1)}

        tweets_ints = []
        for each in tweets:
            tweets_ints.append([vocab_to_int[word] for word in each.split()])
```

```
In [8]: # Encoding the labels::
        list_labels = []

        for l in labels_org.split():
            if l == "depressive":
                list_labels.append(1)
            else:
                list_labels.append(0)

        labels = np.array(list_labels)
        print(len(labels))
```

1176

```
In [9]: tweets_lens = Counter([len(x) for x in tweets_ints])
        print("Zero-length tweets: {}".format(tweets_lens[0]))
        print("Maximum tweets length: {}".format(max(tweets_lens)))
```

Zero-length tweets: 1

Maximum tweets length: 63

```
In [0]: # Filter out that tweets with 0 length
        tweets_ints = [r[0:200] for r in tweets_ints if len(r) > 0]
```

```
In [11]: from collections import Counter
        tweets_lens = Counter([len(x) for x in tweets_ints])
        print("Zero-length tweets: {}".format(tweets_lens[0]))
        print("Maximum tweet length: {}".format(max(tweets_lens)))
```

Zero-length tweets: 0  
Maximum tweet length: 63

```
In [0]: seq_len = 200
        features = np.zeros((len(tweets_ints), seq_len), dtype=int)
        # print(features[:10,:100])
        for i, row in enumerate(tweets_ints):
            features[i, -len(row):] = np.array(row)[:seq_len]
        #features[:10,:100]
```

```
In [13]: split_frac = 0.8
```

```
split_index = int(split_frac * len(features))

train_x, val_x = features[:split_index], features[split_index:]
train_y, val_y = labels[:split_index], labels[split_index:]

split_frac = 0.5
split_index = int(split_frac * len(val_x))

val_x, test_x = val_x[:split_index], val_x[split_index:]
val_y, test_y = val_y[:split_index], val_y[split_index:]

print("\t\t\tFeature Shapes:")
print("Train set: \t\t{}".format(train_x.shape),
      "\nValidation set: \t{}".format(val_x.shape),
      "\nTest set: \t\t{}".format(test_x.shape))
print("label set: \t\t{}".format(train_y.shape),
      "\nValidation label set: \t{}".format(val_y.shape),
      "\nTest label set: \t\t{}".format(test_y.shape))
```

```

                        Feature Shapes:
Train set:              (940, 200)
Validation set:         (118, 200)
Test set:               (118, 200)
label set:              (940,)
Validation label set:   (118,)
Test label set:         (118,)
```

```
In [0]: n_words = len(vocab_to_int) + 1 # Add 1 for 0 added to vocab
```

```
# Create the graph object
tf.reset_default_graph()
with tf.name_scope('inputs'):
    inputs_ = tf.placeholder(tf.int32, [None, None], name="inputs")
    labels_ = tf.placeholder(tf.int32, [None, None], name="labels")
    keep_prob = tf.placeholder(tf.float32, name="keep_prob")
```

```
In [0]: # Size of the embedding vectors (number of units in the embedding layer)
        embed_size = 300
```

```
        with tf.name_scope("Embeddings"):
            embedding = tf.Variable(tf.random_uniform((n_words, embed_size), -1, 1))
            embed = tf.nn.embedding_lookup(embedding, inputs_)
```

```
In [16]: def lstm_cell():
```

```
        # Your basic LSTM cell
```

```
        lstm = tf.contrib.rnn.BasicLSTMCell(lstm_size, reuse=tf.get_variable_scope().reuse)
```

```
        # Add dropout to the cell
```

```
        return tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_prob)
```

```
        with tf.name_scope("RNN_layers"):
```

```
            # Stack up multiple LSTM layers, for deep learning
```

```
            cell = tf.contrib.rnn.MultiRNNCell([lstm_cell() for _ in range(lstm_layers)])
```

```
            # Getting an initial state of all zeros
```

```
            initial_state = cell.zero_state(batch_size, tf.float32)
```

WARNING:tensorflow:From <ipython-input-16-678741cf60df>:3: BasicLSTMCell.\_\_init\_\_ (from tensorflow.nn.rnn\_cell) is deprecated and will be removed in the future. Instructions for updating:

This class is deprecated, please use tf.nn.rnn\_cell.LSTMCell, which supports all the feature t

```
In [0]: with tf.name_scope("RNN_forward"):
```

```
        outputs, final_state = tf.nn.dynamic_rnn(cell, embed, initial_state=initial_state)
```

```
In [0]: # Output::
```

```
        with tf.name_scope('predictions'):
```

```
            predictions = tf.contrib.layers.fully_connected(outputs[:, -1], 1, activation_fn=t
```

```
            tf.summary.histogram('predictions', predictions)
```

```
        with tf.name_scope('cost'):
```

```
            cost = tf.losses.mean_squared_error(labels_, predictions)
```

```
            tf.summary.scalar('cost', cost)
```

```
        with tf.name_scope('train'):
```

```
            optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

```
        merged = tf.summary.merge_all()
```

```
In [0]: # Validation accuracy::
```

```
        with tf.name_scope('validation'):
```

```
            correct_pred = tf.equal(tf.cast(tf.round(predictions), tf.int32), labels_)
```

```
            accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

```
In [0]: # Batching::
```

```
def get_batches(x, y, batch_size=100):
```

```
    n_batches = len(x)//batch_size
    x, y = x[:n_batches*batch_size], y[:n_batches*batch_size]
    for ii in range(0, len(x), batch_size):
        yield x[ii:ii+batch_size], y[ii:ii+batch_size]
```

```
In [21]: # Training::
```

```
    #epochs = 5
```

```
    saver = tf.train.Saver()
```

```
    start = timer()
```

```
    with tf.Session() as sess:
```

```
        sess.run(tf.global_variables_initializer())
```

```
        train_writer = tf.summary.FileWriter('./logs/tb/train', sess.graph)
```

```
        test_writer = tf.summary.FileWriter('./logs/tb/test', sess.graph)
```

```
        iteration = 1
```

```
        for e in range(1, epochs+1):
```

```
            state = sess.run(initial_state)
```

```
            for ii, (x, y) in enumerate(get_batches(train_x, train_y, batch_size), 1):
```

```
                feed = {inputs_: x,
                        labels_: y[:, None],
                        keep_prob: 1,
                        initial_state: state}
```

```
                summary, loss, state, _ = sess.run([merged, cost, final_state, optimizer]
```

```
#                loss, state, _ = sess.run([cost, final_state, optimizer], feed_dict=feed)
```

```
                train_writer.add_summary(summary, iteration)
```

```
            if iteration%5==0:
```

```
                print("Epoch: {}/{}".format(e, epochs),
                      "Iteration: {}".format(iteration),
                      "Train loss: {:.3f}".format(loss))
```

```
            if iteration%25==0:
```

```
                val_acc = []
```

```
                val_state = sess.run(cell.zero_state(batch_size, tf.float32))
```

```
                for x, y in get_batches(val_x, val_y, batch_size):
```

```
                    feed = {inputs_: x,
                            labels_: y[:, None],
                            keep_prob: 1,
                            initial_state: val_state}
```

```
#                    batch_acc, val_state = sess.run([accuracy, final_state], feed_dict=feed)
```

```
                    summary, batch_acc, val_state = sess.run([merged, accuracy, final_state], feed_dict=feed)
```

```
                    val_acc.append(batch_acc)
```

```
                print("Val acc: {:.3f}".format(np.mean(val_acc)))
```

```

        iteration +=1
        test_writer.add_summary(summary, iteration)
        saver.save(sess, checkpointName)
    #         tensorboard = TensorBoard(log_dir="logs/tweet_5000_all_sentiments_six_cl
    saver.save(sess, checkpointName)

```

```

    duration = timer() - start
    print('Time elapsed =',duration,'sec(s)')

```

```

Epoch: 1/10 Iteration: 5 Train loss: 0.248
Epoch: 1/10 Iteration: 10 Train loss: 0.250
Epoch: 1/10 Iteration: 15 Train loss: 0.258
Epoch: 2/10 Iteration: 20 Train loss: 0.253
Epoch: 2/10 Iteration: 25 Train loss: 0.248
Val acc: 0.450
Epoch: 2/10 Iteration: 30 Train loss: 0.246
Epoch: 2/10 Iteration: 35 Train loss: 0.268
Epoch: 3/10 Iteration: 40 Train loss: 0.245
Epoch: 3/10 Iteration: 45 Train loss: 0.245
Epoch: 3/10 Iteration: 50 Train loss: 0.244
Val acc: 0.420
Epoch: 4/10 Iteration: 55 Train loss: 0.245
Epoch: 4/10 Iteration: 60 Train loss: 0.246
Epoch: 4/10 Iteration: 65 Train loss: 0.244
Epoch: 4/10 Iteration: 70 Train loss: 0.256
Epoch: 5/10 Iteration: 75 Train loss: 0.244
Val acc: 0.440
Epoch: 5/10 Iteration: 80 Train loss: 0.243
Epoch: 5/10 Iteration: 85 Train loss: 0.244
Epoch: 5/10 Iteration: 90 Train loss: 0.262
Epoch: 6/10 Iteration: 95 Train loss: 0.240
Epoch: 6/10 Iteration: 100 Train loss: 0.232
Val acc: 0.440
Epoch: 6/10 Iteration: 105 Train loss: 0.254
Epoch: 7/10 Iteration: 110 Train loss: 0.231
Epoch: 7/10 Iteration: 115 Train loss: 0.222
Epoch: 7/10 Iteration: 120 Train loss: 0.209
Epoch: 7/10 Iteration: 125 Train loss: 0.277
Val acc: 0.560
Epoch: 8/10 Iteration: 130 Train loss: 0.211
Epoch: 8/10 Iteration: 135 Train loss: 0.201
Epoch: 8/10 Iteration: 140 Train loss: 0.170
Epoch: 9/10 Iteration: 145 Train loss: 0.161
Epoch: 9/10 Iteration: 150 Train loss: 0.145
Val acc: 0.530
Epoch: 9/10 Iteration: 155 Train loss: 0.149
Epoch: 9/10 Iteration: 160 Train loss: 0.189
Epoch: 10/10 Iteration: 165 Train loss: 0.191

```

Epoch: 10/10 Iteration: 170 Train loss: 0.141  
Epoch: 10/10 Iteration: 175 Train loss: 0.157  
Val acc: 0.550  
Epoch: 10/10 Iteration: 180 Train loss: 0.149  
Time elapsed = 1305.568099148 sec(s)

In [22]: # *Testing*::

```
test_acc = []
with tf.Session() as sess:
    saver.restore(sess, checkpointName)
    test_state = sess.run(cell.zero_state(batch_size, tf.float32))
    for ii, (x, y) in enumerate(get_batches(test_x, test_y, batch_size), 1):
        feed = {inputs_: x,
                 labels_: y[:, None],
                 keep_prob: 1,
                 initial_state: test_state}
        batch_acc, test_state = sess.run([accuracy, final_state], feed_dict=feed)
        test_acc.append(batch_acc)
    print("Test accuracy: {:.3f}".format(np.mean(test_acc)))
```

INFO:tensorflow:Restoring parameters from checkpoints/lstm\_with\_unique\_7.ckpt  
Test accuracy: 0.590

In [0]: