gru_2_unique

December 12, 2018

In [1]:
```
# GRU WITH UNIQUE DATASET IMPLEMENTATION 2
# 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::

gru_size = 128
gru_layers = 5
batch_size = 10
learning_rate = 0.0001
epochs = 5
```

In [4]:
```
fileName = "data_all_unique_dnd_stratified_2"
checkpointName = "checkpoints/"+fileName+".ckpt"
print(checkpointName)
print(type(checkpointName))
```

```
checkpoints/data_all_unique_dnd_stratified_2.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 [0]: def gru_cell():
            # Basic GRU cell
            gru = tf.contrib.rnn.GRUCell(gru_size, reuse=tf.get_variable_scope().reuse)
            # Add dropout to the cell
            return tf.contrib.rnn.DropoutWrapper(gru, output_keep_prob=keep_prob)

        with tf.name_scope("RNN_layers"):
            # Stack up multiple GRU layers, for deep learning
            cell = tf.contrib.rnn.MultiRNNCell([gru_cell() for _ in range(gru_layers)])

            # Getting an initial state of all zeros
            initial_state = cell.zero_state(batch_size, tf.float32)

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=ti
            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=fee

            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_d
                    summary, batch_acc, val_state = sess.run([merged, accuracy, final
                    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 elasped =',duration,'sec(s)')
```

```
Epoch: 1/5 Iteration: 5 Train loss: 0.282
Epoch: 1/5 Iteration: 10 Train loss: 0.258
Epoch: 1/5 Iteration: 15 Train loss: 0.247
Epoch: 1/5 Iteration: 20 Train loss: 0.245
Epoch: 1/5 Iteration: 25 Train loss: 0.213
Val acc: 0.436
Epoch: 1/5 Iteration: 30 Train loss: 0.229
Epoch: 1/5 Iteration: 35 Train loss: 0.233
Epoch: 1/5 Iteration: 40 Train loss: 0.228
Epoch: 1/5 Iteration: 45 Train loss: 0.228
Epoch: 1/5 Iteration: 50 Train loss: 0.258
Val acc: 0.391
Epoch: 1/5 Iteration: 55 Train loss: 0.253
Epoch: 1/5 Iteration: 60 Train loss: 0.189
Epoch: 1/5 Iteration: 65 Train loss: 0.270
Epoch: 1/5 Iteration: 70 Train loss: 0.260
Epoch: 1/5 Iteration: 75 Train loss: 0.252
Val acc: 0.309
Epoch: 1/5 Iteration: 80 Train loss: 0.321
Epoch: 1/5 Iteration: 85 Train loss: 0.276
Epoch: 1/5 Iteration: 90 Train loss: 0.231
Epoch: 2/5 Iteration: 95 Train loss: 0.246
Epoch: 2/5 Iteration: 100 Train loss: 0.222
Val acc: 0.709
Epoch: 2/5 Iteration: 105 Train loss: 0.293
Epoch: 2/5 Iteration: 110 Train loss: 0.261
Epoch: 2/5 Iteration: 115 Train loss: 0.235
Epoch: 2/5 Iteration: 120 Train loss: 0.238
Epoch: 2/5 Iteration: 125 Train loss: 0.223
Val acc: 0.445
Epoch: 2/5 Iteration: 130 Train loss: 0.248
Epoch: 2/5 Iteration: 135 Train loss: 0.245
Epoch: 2/5 Iteration: 140 Train loss: 0.277
Epoch: 2/5 Iteration: 145 Train loss: 0.226
Epoch: 2/5 Iteration: 150 Train loss: 0.263
Val acc: 0.427
Epoch: 2/5 Iteration: 155 Train loss: 0.246
Epoch: 2/5 Iteration: 160 Train loss: 0.222
Epoch: 2/5 Iteration: 165 Train loss: 0.274
Epoch: 2/5 Iteration: 170 Train loss: 0.248
Epoch: 2/5 Iteration: 175 Train loss: 0.255
Val acc: 0.500
Epoch: 2/5 Iteration: 180 Train loss: 0.248
```

```
Epoch: 2/5 Iteration: 185 Train loss: 0.227
Epoch: 3/5 Iteration: 190 Train loss: 0.298
Epoch: 3/5 Iteration: 195 Train loss: 0.233
Epoch: 3/5 Iteration: 200 Train loss: 0.235
Val acc: 0.709
Epoch: 3/5 Iteration: 205 Train loss: 0.255
Epoch: 3/5 Iteration: 210 Train loss: 0.246
Epoch: 3/5 Iteration: 215 Train loss: 0.269
Epoch: 3/5 Iteration: 220 Train loss: 0.223
Epoch: 3/5 Iteration: 225 Train loss: 0.239
Val acc: 0.391
Epoch: 3/5 Iteration: 230 Train loss: 0.200
Epoch: 3/5 Iteration: 235 Train loss: 0.198
Epoch: 3/5 Iteration: 240 Train loss: 0.221
Epoch: 3/5 Iteration: 245 Train loss: 0.269
Epoch: 3/5 Iteration: 250 Train loss: 0.231
Val acc: 0.409
Epoch: 3/5 Iteration: 255 Train loss: 0.212
Epoch: 3/5 Iteration: 260 Train loss: 0.223
Epoch: 3/5 Iteration: 265 Train loss: 0.260
Epoch: 3/5 Iteration: 270 Train loss: 0.231
Epoch: 3/5 Iteration: 275 Train loss: 0.213
Val acc: 0.627
Epoch: 3/5 Iteration: 280 Train loss: 0.184
Epoch: 4/5 Iteration: 285 Train loss: 0.273
Epoch: 4/5 Iteration: 290 Train loss: 0.203
Epoch: 4/5 Iteration: 295 Train loss: 0.335
Epoch: 4/5 Iteration: 300 Train loss: 0.249
Val acc: 0.736
Epoch: 4/5 Iteration: 305 Train loss: 0.186
Epoch: 4/5 Iteration: 310 Train loss: 0.223
Epoch: 4/5 Iteration: 315 Train loss: 0.232
Epoch: 4/5 Iteration: 320 Train loss: 0.218
Epoch: 4/5 Iteration: 325 Train loss: 0.188
Val acc: 0.418
Epoch: 4/5 Iteration: 330 Train loss: 0.216
Epoch: 4/5 Iteration: 335 Train loss: 0.183
Epoch: 4/5 Iteration: 340 Train loss: 0.188
Epoch: 4/5 Iteration: 345 Train loss: 0.186
Epoch: 4/5 Iteration: 350 Train loss: 0.207
Val acc: 0.418
Epoch: 4/5 Iteration: 355 Train loss: 0.240
Epoch: 4/5 Iteration: 360 Train loss: 0.203
Epoch: 4/5 Iteration: 365 Train loss: 0.220
Epoch: 4/5 Iteration: 370 Train loss: 0.160
Epoch: 4/5 Iteration: 375 Train loss: 0.182
Val acc: 0.718
Epoch: 5/5 Iteration: 380 Train loss: 0.153
```

```
Epoch: 5/5 Iteration: 385 Train loss: 0.230
Epoch: 5/5 Iteration: 390 Train loss: 0.240
Epoch: 5/5 Iteration: 395 Train loss: 0.162
Epoch: 5/5 Iteration: 400 Train loss: 0.244
Val acc: 0.473
Epoch: 5/5 Iteration: 405 Train loss: 0.220
Epoch: 5/5 Iteration: 410 Train loss: 0.182
Epoch: 5/5 Iteration: 415 Train loss: 0.229
Epoch: 5/5 Iteration: 420 Train loss: 0.151
Epoch: 5/5 Iteration: 425 Train loss: 0.153
Val acc: 0.455
Epoch: 5/5 Iteration: 430 Train loss: 0.153
Epoch: 5/5 Iteration: 435 Train loss: 0.144
Epoch: 5/5 Iteration: 440 Train loss: 0.256
Epoch: 5/5 Iteration: 445 Train loss: 0.117
Epoch: 5/5 Iteration: 450 Train loss: 0.168
Val acc: 0.527
Epoch: 5/5 Iteration: 455 Train loss: 0.174
Epoch: 5/5 Iteration: 460 Train loss: 0.132
Epoch: 5/5 Iteration: 465 Train loss: 0.040
Epoch: 5/5 Iteration: 470 Train loss: 0.158
Time elasped = 3731.952135464 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/data_all_unique_dnd_stratified_2.ckpt
Test accuracy: 0.700
```

In [0]: