ECS 7001 - NN & NLP

Assignment 1: Word Representation and Text Classification with Neural Networks

Neeraj Vashistha - 190573735

Part A: Word Embeddings with Word2Vec [20 mark]

1. Preprocessing the training corpus [3 marks]

```
[5] print('Length of processed corpus:', len(normalized_corpus))

    Length of processed corpus: 12498

[6] print('Processed line:', normalized_corpus[10])

    Processed line: therefore succession norland estate really important
```

2. Creating the corpus vocabulary and preparing the dataset [3 marks].

```
[16] print('\nSample word2idx: ', list(word2idx.items())[:10])

     Sample word2idx:  [('description', 1), ('invitation', 2), ('chili', 3), ('waives', 4), ('b

[17] print('\nSample idx2word:', list(idx2word.items())[:10])

     Sample idx2word: [(1, 'description'), (2, 'invitation'), (3, 'chili'), (4, 'waives'), (5,

[18] print('\nSample normalized corpus:', normalized_corpus[:3])

     Sample normalized corpus: ['sense sensibility jane austen', 'family dashwood long settled

[19] print('\nAbove sentence as a list of ids:' , sents_as_ids[:30])

     Above sentence as a list of ids: [[8036, 115, 9775, 1630], [5271, 6265, 8942, 8543, 1050],
```

3. Building the skip-gram neural network architecture [6 marks].

```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
target_embed_layer (Embedding)  (None, 1, 100)       1003900     input_1[0][0]
__________________________________________________________________________________________________
context_embed_layer (Embedding) (None, 1, 100)       1003900     input_2[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 100)          0           target_embed_layer[0][0]
__________________________________________________________________________________________________
reshape_2 (Reshape)             (None, 100)          0           context_embed_layer[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                     (None, 1)            0           reshape_1[0][0]
                                                                 reshape_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            2           dot_1[0][0]
==================================================================================================
Total params: 2,007,802
Trainable params: 2,007,802
Non-trainable params: 0
__________________________________________________________________________________________________
```

4. Training the models (and reading McCormick's tutorial) [3 marks].

a. What would the inputs and outputs to the model be?

The input is the numeric/vector representation of textual data/word/string. The output is a single vector representation for each word which gives us the probability of being chosen as the next word.

b. How would you use the Keras framework to create this architecture?

An Input layer with two inputs, target word and context word. And embedding of the above layer and the transforming the embedding in reshaping layer. Finally taking the dot product.

c. Can you think of reasons why this model is considered to be inefficient?

The words which are presented here do not capture the contextual meaning internally thus the result we get does not capture semantic relationship very well. We need rich dataset such as which is artificially created for word2vec.

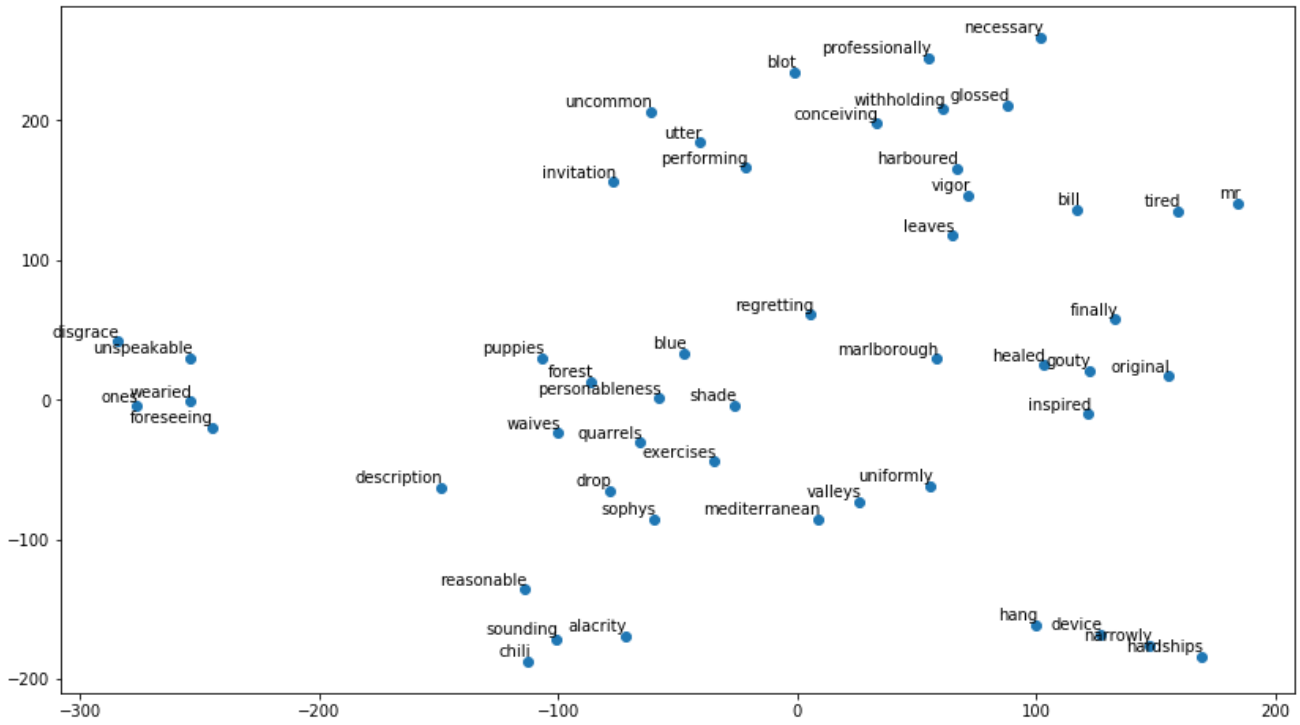
5. Getting the word embeddings [2 marks].

```
import pandas as pd
pd.DataFrame(word_embeddings, index=list(idx2word.values())).head(10)
```

	0	1	2	3	4	5	6	7	8	9	10
description	0.038180	-0.007630	-0.009093	0.004218	0.039224	0.008442	0.018509	0.040620	-0.024994	-0.005944	0.001864
invitation	0.000421	-0.007219	-0.028518	0.012713	0.046222	-0.008220	0.003519	0.028150	0.008995	-0.030596	0.005321
chili	0.014828	0.022766	-0.013250	0.000273	-0.009606	0.015507	0.027722	-0.013990	0.002698	-0.008393	-0.012705
waives	0.002003	-0.006559	-0.000457	-0.024515	0.020607	0.004022	0.012542	-0.007948	-0.014813	-0.001263	-0.032664
blue	-0.002439	0.031489	-0.010717	-0.031881	-0.003537	-0.004970	0.005637	0.042412	0.006596	0.034581	0.018694
alacrity	-0.001178	-0.007398	-0.008243	-0.004119	0.024026	0.019021	-0.006032	0.020286	-0.024673	0.000541	0.007428
tired	0.005797	0.000861	0.014825	0.016738	0.016778	-0.009721	0.022126	0.023859	-0.015631	0.023875	0.011542
regretting	0.006677	0.023356	0.000030	-0.005693	0.031373	-0.009469	-0.006472	-0.004011	-0.003569	0.014311	-0.005218
conceiving	0.030556	0.025950	-0.032347	-0.017791	0.000483	-0.003746	0.028112	0.013282	-0.027896	0.023116	0.006793
forest	0.015756	0.030446	-0.011458	-0.015530	0.008336	0.002182	0.018296	-0.002791	0.014764	-0.003694	-0.003388

10 rows × 100 columns

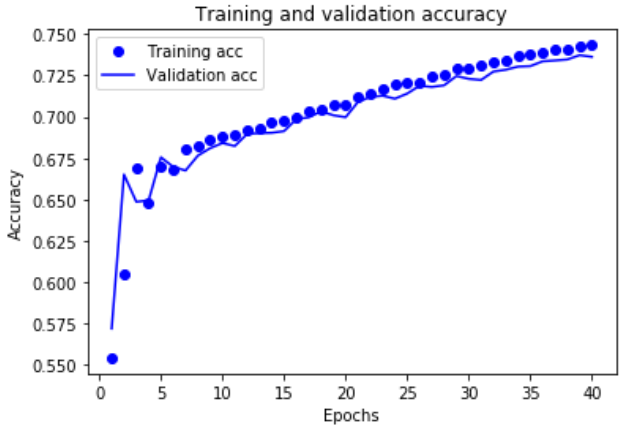
6. Exploring and visualizing your word embeddings using t-SNE [3 marks].



Part B: Basic Text Classification [25 marks]

1. Build a neural network classifier using one-hot word vectors, and train and evaluate it [5 marks].

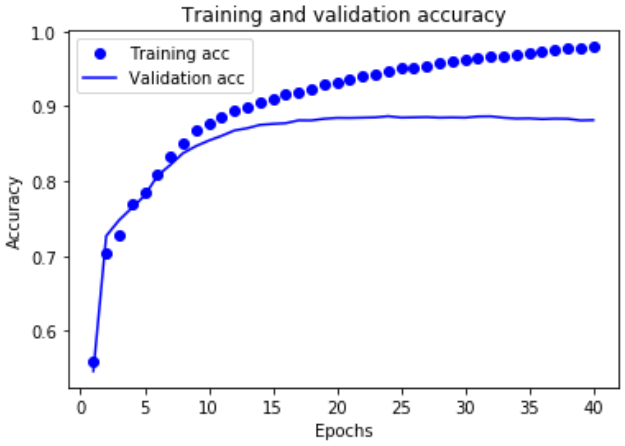
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_4 (Lambda)            (None, 256, 10000)        0
_________________________________________________________________
global_average_pooling1d_mas (None, 10000)             0
_________________________________________________________________
dense_8 (Dense)              (None, 16)                160016
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,033
Trainable params: 160,033
Non-trainable params: 0
_________________________________________________________________
```




```
[ ] results = model.evaluate(X_test_enc, y_test)

25000/25000 [==============================] - 4s 148us/step

[ ] print(results)
    # loss, accuracay

[0.545550052242279, 0.73616]
```

2. Modify your model to use a word embedding layer instead of one-hot vectors, and to learn the values of these word embedding vectors along with the model [5 marks].

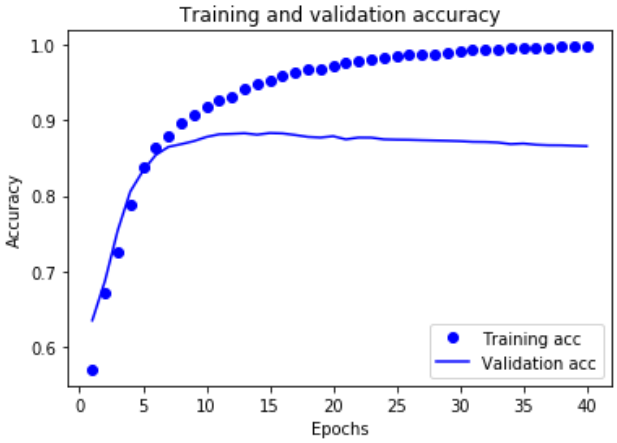
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 256, 10000)        0
_________________________________________________________________
global_average_pooling1d_mas (None, 10000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                160016
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,033
Trainable params: 160,033
Non-trainable params: 0
_________________________________________________________________
```




```
[ ] results = model2.evaluate(X_test_enc, y_test)

25000/25000 [==============================] - 1s 35us/step

[ ] print (results)

[0.32691408640861513, 0.87344]
```

3. Adapt your model to load and use pre-trained word embeddings instead (either the mbeddings you built in part A or from another pre-trained model), and train and evaluate it [7 marks].

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 300)         120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_5 (Dense)              (None, 16)                4816
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 17
=================================================================
Total params: 120,005,133
Trainable params: 120,005,133
Non-trainable params: 0
_________________________________________________________________
```

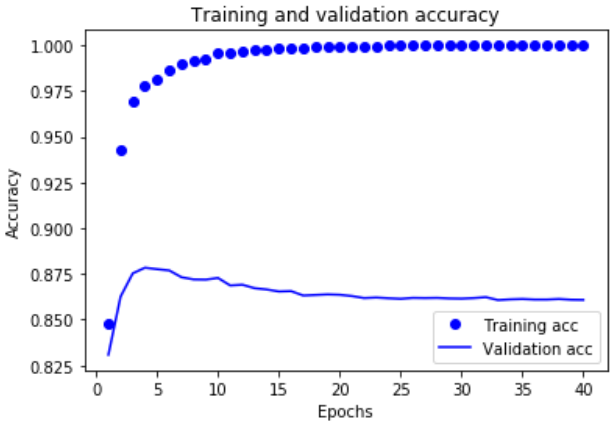



```
results = model3.evaluate(X_test_enc, y_test)
print(results)
```

```
25000/25000 [==============================] - 1s 40us/step
[0.5434409149956703, 0.8536]
```

4. One way to improve the performance is to add another fully-connected layer to your network. Try this, and explain why the performance does not improve. What can you do to improve the situation? [8 marks]

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 300)         120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_7 (Dense)              (None, 16)                4816
_________________________________________________________________
dense_8 (Dense)              (None, 16)                272
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 17
=================================================================
Total params: 120,005,405
Trainable params: 120,005,405
Non-trainable params: 0
_________________________________________________________________
```




```
results = model3.evaluate(X_test_enc, y_test)
print(results)
```

```
25000/25000 [==============================] - 1s 43us/step
[0.946351080160141, 0.84748]
```

Part C: Using LSTMs for Text Classification [25 marks]

1. Section 2, Readying the inputs for the LSTM [2 marks].

```
print('Length of sample train_data before preprocessing:', len(train_data[1]), type(train_data[1]))
print('Length of sample train_data after preprocessing:', len(preprocessed_train_data[0]), type(preprocessed_train_data[1]))
```

```
Length of sample train_data before preprocessing: 189 <class 'list'>
Length of sample train_data after preprocessing: 500 <class 'numpy.ndarray'>
```

2. Building the model (section 3 of the script): [6 marks].

```
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 100)          1000000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

3. Section 4, training the model [6 marks].

```
[9] history = model.fit(preprocessed_train_data,
                        train_labels,
                        epochs=3,
                        batch_size=512,
                        validation_split=0.08,
                        verbose=1)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is d

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is depre

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is depr

Train on 23000 samples, validate on 2000 samples
Epoch 1/3
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_sess

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is d

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_init

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initia

23000/23000 [==============================] - 44s 2ms/step - loss: 0.6694 - acc: 0.6613 - val_loss: 0.6239 - val_acc: 0.7195
Epoch 2/3
23000/23000 [==============================] - 43s 2ms/step - loss: 0.4548 - acc: 0.8093 - val_loss: 0.3876 - val_acc: 0.8395
Epoch 3/3
23000/23000 [==============================] - 43s 2ms/step - loss: 0.2847 - acc: 0.8890 - val_loss: 0.3232 - val_acc: 0.8750
```

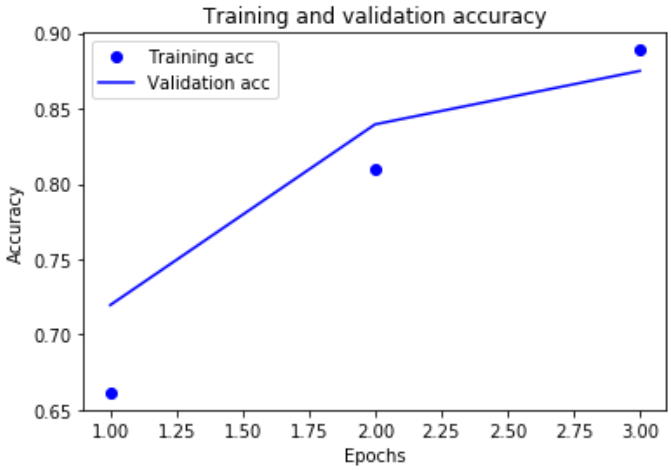
4. Evaluating the model on the test data (section 5) [2 marks].

```
[10] results = model.evaluate(processed_test_data, test_labels)
```

```
25000/25000 [==============================] - 282s 11ms/step
```

```
[11] print(results)
     # loss, accuracay
```

```
[0.3415896670007706, 0.86064]
```



5. Section 6, extracting the word embeddings [2 marks].

```
[14] embed_layer = model.get_layer('embedding_1').get_weights()[0]
```

```
print('Shape of word_embeddings:', embed_layer.shape)
```

```
Shape of word_embeddings: (10000, 100)
```

```
] print(model.summary())
```

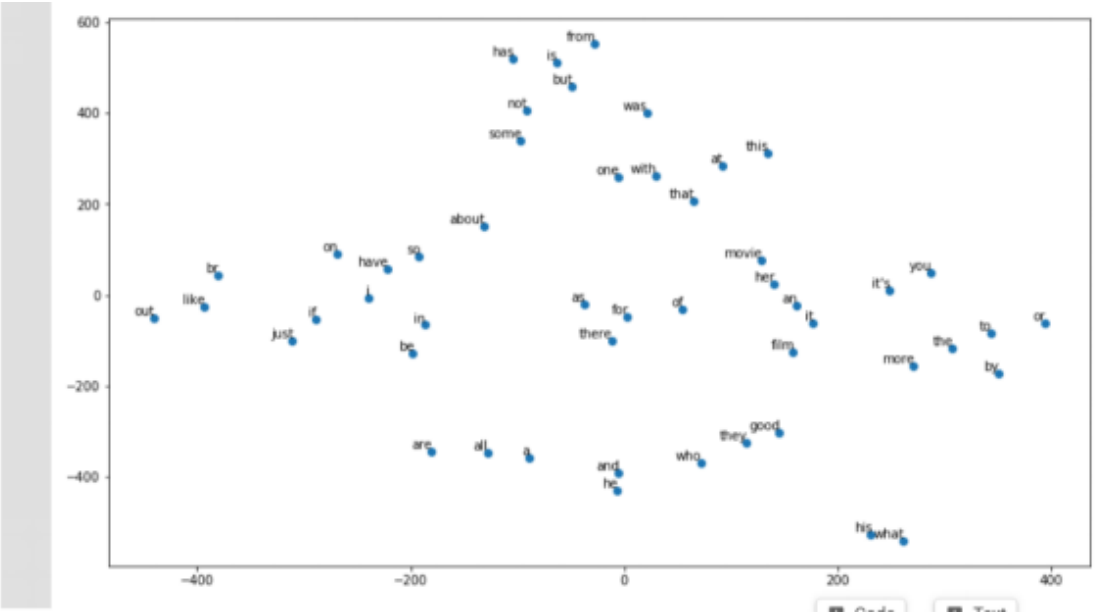
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 100)          1000000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

6. Visualizing the reviews [1 mark].

```
idx2word = {v: k for k,v in word2idx.items()}
print(' '.join(idx2word[idx] for idx in train_data[0]))
```

```
<START> this film was just brilliant casting location scenery story direction everyone's really suited the part
they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being
director <UNK> father came from the same scottish island as myself so i loved the fact there was a real
connection with this film the witty remarks throughout the film were great it was just brilliant so much that i
bought the film as soon as it was released for <UNK> and would recommend it to everyone to watch and the fly
fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it
must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of norman
and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that
play them all grown up are such a big profile for the whole film but these children are amazing and should be
praised for what they have done don't you think the whole story was so lovely because it was true and was
someone's life after all that was shared with us all
```

7. Visualizing the word embeddings [2 marks].



8. Section 9 [4 marks].
   a. Create a new model that is a copy of the model step 3. To this new model, add two dropout layers, one between the embedding layer and the LSTM layer and another between theLSTM layer and the output layer. Repeat steps 4 and 5 for this model. What do you observe? How about if you train this new model for 6 epochs instead?

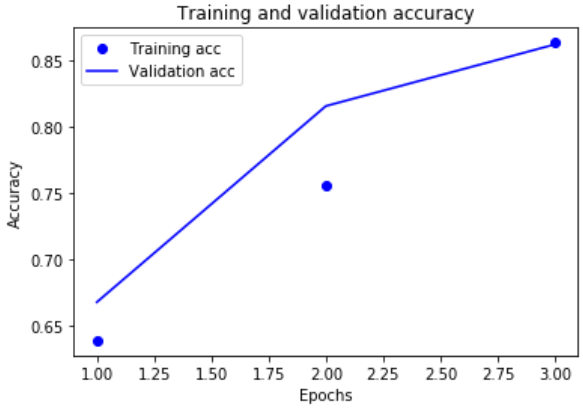
Model with dropout with 3 epoch

```
model2 = Sequential()
EMBED_SIZE = 100
# model.add()
model2.add(Embedding(VOCAB_SIZE,EMBED_SIZE,input_length=MAXIMUM_LENGTH))
model2.add(Dropout(0.2))
model2.add(LSTM(100, activation='tanh'))
model2.add(Dropout(0.2))
model2.add(Dense(1,activation='sigmoid',input_shape=(1,)))
print(model2.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 500, 100)          1000000
_________________________________________________________________
dropout_3 (Dropout)          (None, 500, 100)          0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
None
```

```
7] results = model2.evaluate(processed_test_data, test_labels)
   print(results)
```

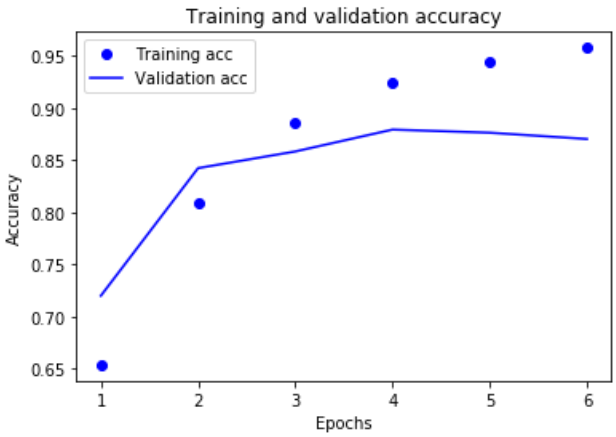
```
25000/25000 [==============================] - 295s 12ms/step
[0.33250678292274477, 0.85964]
```



Model with dropout with 6 epoch

```
results = model2.evaluate(processed_test_data, test_labels)
print(results)
```

```
25000/25000 [==============================] - 294s 12ms/step
[0.3607836242961884, 0.86712]
```



From above we can see that as we add dropouts and increase epoch, although there is a slight increase in accuracy, the model converges better here.

b. Experiment with compiling the model with batch sizes of 1, 32, len(training_data). What do you observe?

Smaller the batch size started converging at a local minima very early, as I increased the batch size there was a significant degradation in the quality of the model, as measured by its ability to generalize.

Part D: A Real Text Classification Task [30 marks]

1. Build and evaluate a basic classifier for Subtask A, adapting one of your models from Parts B and C above [10 marks]

In .ipynb notebook, used glove embedding and trained model using LSTM and CNN

2. Build and evaluate a basic classifier for Subtasks A-C combined, treating this as one single multi-class problem. There are 5 possible classes:NOT, OFF-UNT, OFF-TIN-IND, OFF-TIN-GRP, OFF-TIN-OTH (see the README with the dataset) [10 marks].

In .ipynb notebook, used glove embedding and trained model using LSTM

3. Try to improve your overall accuracy on Subtasks A-C. You could try one or more of: using a CNN instead; using a different classifier; treat the three tasks separately in a pipeline; modify embeddings, loss function etc - see ideas in Week 5 & 6 lectures. [10 marks].

In .ipynb notebook, used glove embedding and trained model using LSTM + CNN