ECS 7001 - NN & NLP

Assignment 1: Word Representation and Text Classification with Neural Networks

---

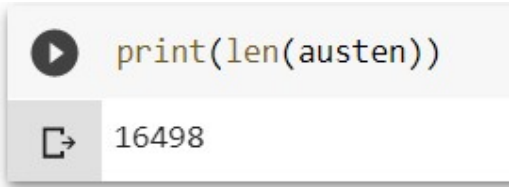
Name: BIKASH KUMAR TIWARY

Student Id: 190778031

Part A: Word Embeddings with Word2Vec [20 marks]

1. Pre-processing the training corpus [3 marks].

Sanity Check:



```
print(len(austen))
```

```
16498
```

```
[15] corpus(austen)[10]
```

```
'son steady respectable young man amply provided fortune mother large half devolved coming age'
```

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

```
print(list(word2idx.items())[:10])
```

```
[('could', 1), ('would', 2), ('mr', 3), ('mrs', 4), ('must', 5), ('said', 6), ('one', 7), ('much', 8), ('miss', 9), ('every', 10)]
```

```
print(list(idx2word.items())[:10])
```

```
[(1, 'could'), (2, 'would'), (3, 'mr'), (4, 'mrs'), (5, 'must'), (6, 'said'), (7, 'one'), (8, 'much'), (9, 'miss'), (10, 'every')]
```

```
print(sents_as_ids[:3])
```

```
[[313, 1398, 76, 4338], [236], [112, 102, 60, 347, 2429]]
```

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

```
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)       1017200     input_1[0][0]
__________________________________________________________________________________________________
context_embed_layer (Embedding) (None, 1, 100)       1017200     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: 2,034,402
Trainable params: 2,034,402
Non-trainable params: 0
__________________________________________________________________________________________________
```

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

One point for each answer to one of the three questions

What would the inputs and outputs to the model be?

- **Input would be target and context, and label is the output**

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

- **Keras is a simple tool for constructing neural network. A high level framework.**
- **Can be used by creating layers**

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

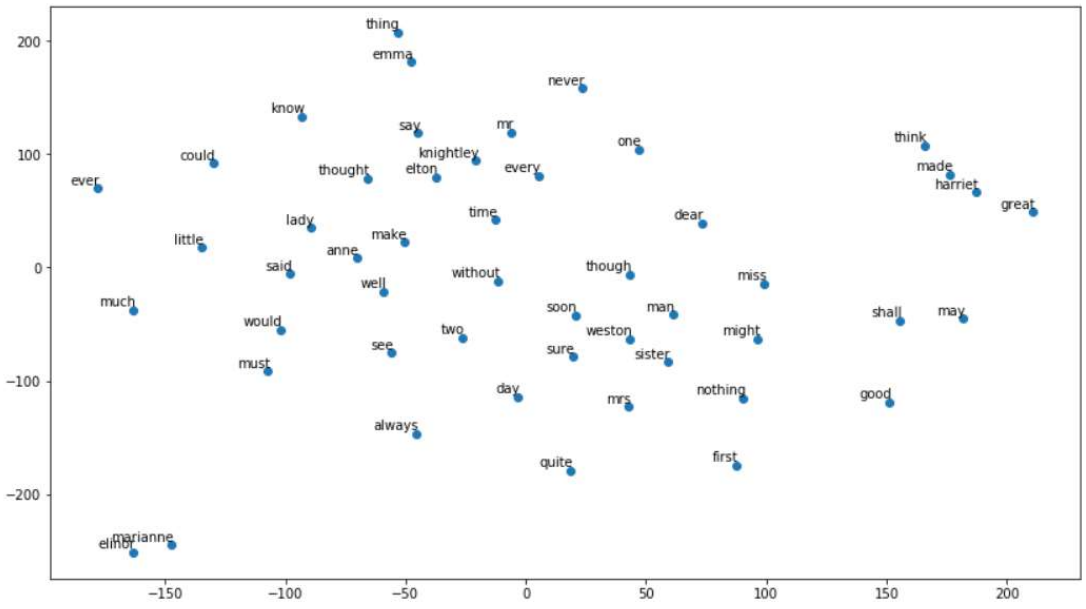
- **Word sense is not captured separately. All are represented in one vector**

5. Getting the word embeddings [2 marks].

	0	1	2	...	97	98	99
could	-0.057503	0.078258	-0.150479	...	-0.125350	-0.040457	0.058685
would	-0.072480	0.118796	0.061979	...	-0.228607	0.139279	0.057542
mr	-0.083789	0.132328	0.044928	...	-0.303188	-0.072274	0.103361
mrs	0.159288	-0.003670	-0.197643	...	-0.334622	0.098793	0.016832
must	-0.045403	0.211744	0.069021	...	-0.197327	-0.075032	-0.393683
said	0.068981	0.222880	0.094180	...	-0.345576	0.003286	0.088417
one	-0.163962	0.091581	-0.293868	...	0.017417	0.027737	0.012774
much	-0.143721	0.106195	0.008758	...	-0.174893	-0.113461	-0.212503
miss	0.055836	0.099174	0.018014	...	-0.159134	0.150408	0.021515
every	0.253846	0.311054	0.002639	...	-0.196641	-0.143397	-0.012405

[10 rows x 100 columns]

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



Part B: Basic Text Classification [25 marks]

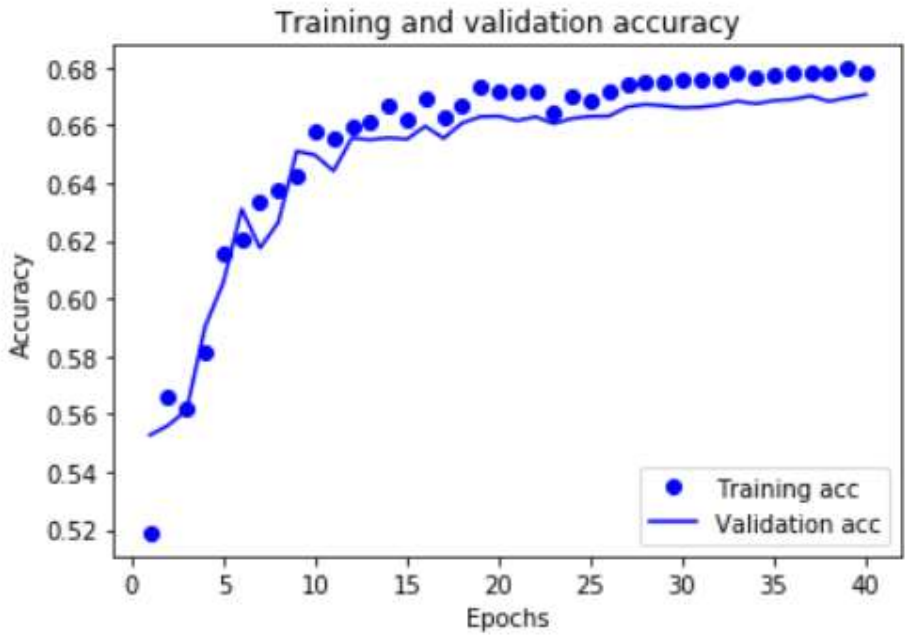
1. Build a neural network classifier using one-hot word vectors, and train and evaluate it [5 marks].

   The first layer is an one-hot layer. The second layer is to compute average on all word vectors in a sentence without considering padding. The output vector is piped through a fully-connected layer. The last layer is connected with a single output node with the sigmoid activation function.

   visualize the model summary.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_4 (Lambda)            (None, 256, 10000)        0
_________________________________________________________________
global_average_pooling1d_mas (None, 10000)             0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 10001
=================================================================
Total params: 10,001
Trainable params: 10,001
Non-trainable params: 0
_________________________________________________________________
```

Accuracy graph:

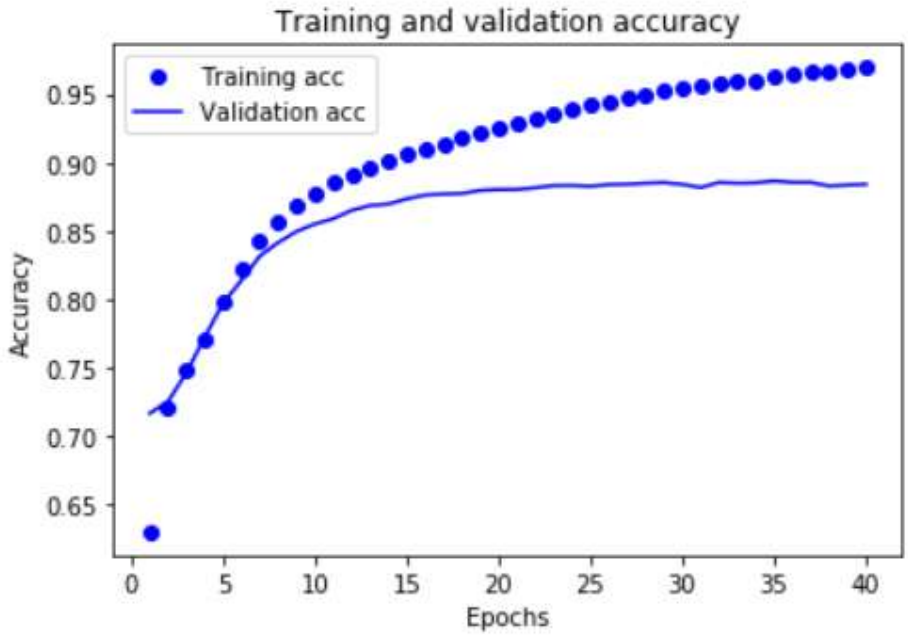


2. Modify your model to use a word embedding layer instead of one-hot vectors, and to learn the values of these word embedding vectors along with the model [5 marks].

Model summary:

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 256)         2560000
_________________________________________________________________
global_average_pooling1d_mas (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 2,560,257
Trainable params: 2,560,257
Non-trainable params: 0
_________________________________________________________________
```

Accuracy Graph:



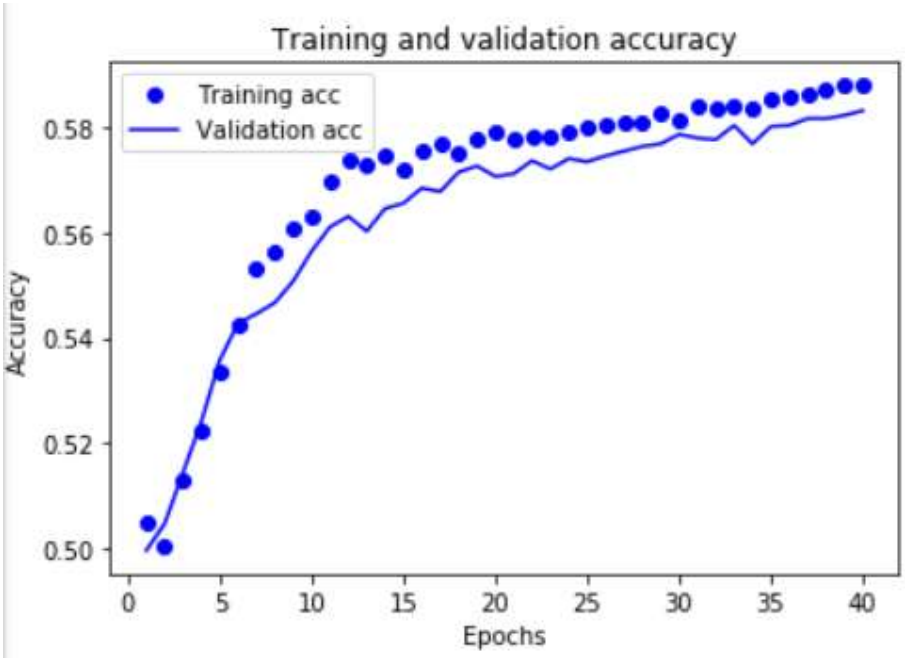
3. Adapt your model to load and use pre-trained word embeddings instead (either the embeddings you built in part A or from another pre-trained model), and train and evaluate it [7 marks].

Pre-trained word embedding

Model Summary:

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 50)          20000050
_________________________________________________________________
global_average_pooling1d_mas (None, 50)                0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 51
=================================================================
Total params: 20,000,101
Trainable params: 51
Non-trainable params: 20,000,050
_________________________________________________________________
```

Accuracy Graph:



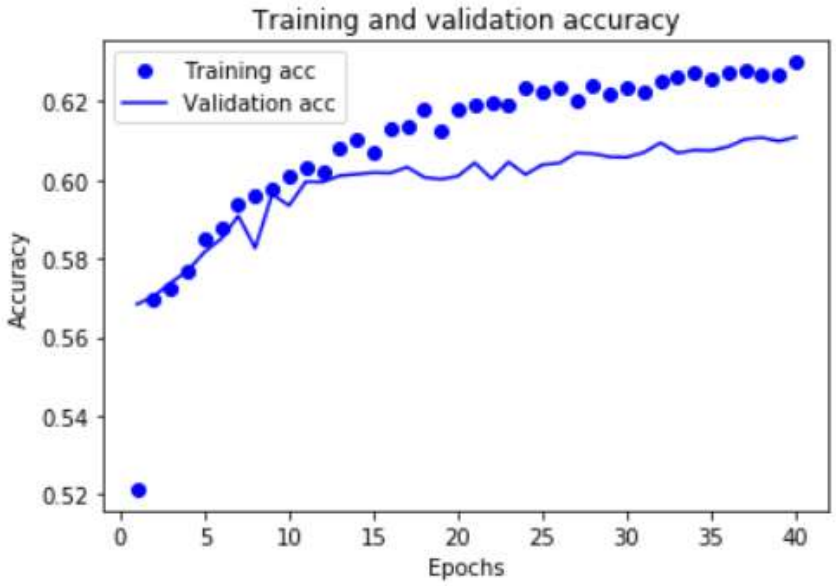
4. One way to improve the performance is to add another fully-connected layer to your network. Try this, and explain why the performance does not improve. What can you do to improve the situation? [8 marks]

Addition of another hidden layer

```
model3.add(Dense(50, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))
```

Below displays the model structure and accuracy plot:

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, None, 50)          20000050
_________________________________________________________________
global_average_pooling1d_mas (None, 50)                0
_________________________________________________________________
dense_12 (Dense)             (None, 50)                2550
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 51
=================================================================
Total params: 20,002,651
Trainable params: 2,601
Non-trainable params: 20,000,050
_________________________________________________________________
```



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

1. Section 2, Readying the inputs for the LSTM [2 marks].
   For this part, show the output you obtain from the sanity check.

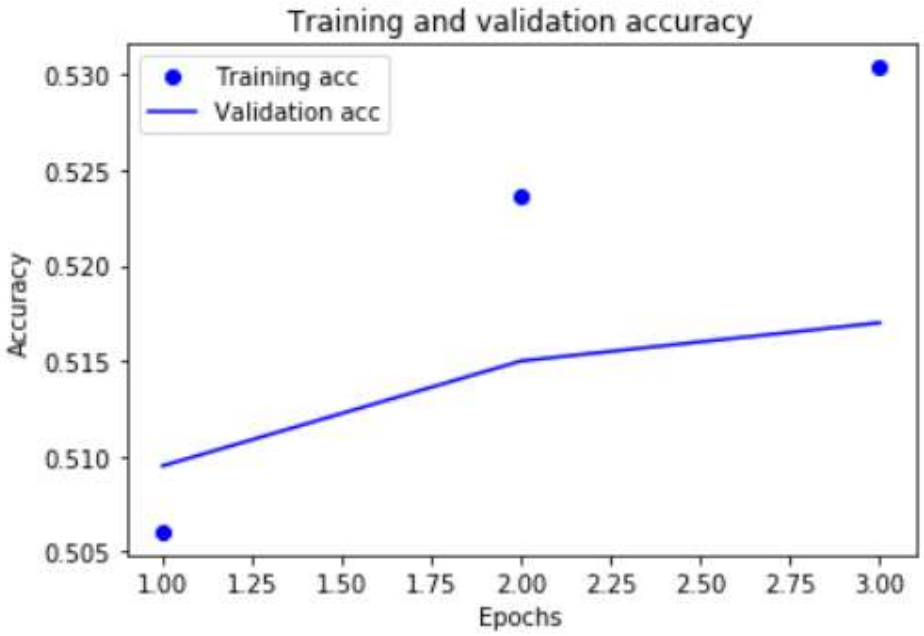
```
Length of sample train_data before preprocessing: 218
Length of sample train_data after preprocessing: 500
```

2. Building the model (section 3 of the script): [6 marks].
   For this part, show the structure of the model you obtain.

```
Model: "model_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_29 (InputLayer)        (None, 500)               0
_________________________________________________________________
embedding_26 (Embedding)     (None, 500, 100)          1000000
_________________________________________________________________
lstm_23 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_23 (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].
   For this part, show the plot of training and validation accuracy through the epochs and comment on the optimal stopping point for the model.

   Optimal stopping point will be at Epochs = 2. From Epochs = 3 onwards it will be overfitting.



4. Evaluating the model on the test data (section 5) [2 marks].
   For this part, show the output of the command printint test loss and test accuracy.

Code block to get and print test_loss and test_accuracy.

```
results = model.evaluate(test_data_after, test_labels)
print('test_loss:', results[0], 'test_accuracy:', results[1])
```

Output:

```
25000/25000 [==============================] - 95s 4ms/step
test_loss: 0.696229334564209 test_accuracy: 0.5118
```

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

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

6. Visualizing the reviews [1 mark].
   For this part, you should include in the report the output of the command printing out the idx2word map.

```
# View a sample review text using the lines of code below
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
```

```
[ ]  print(train_data[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
```

7. Visualizing the word embeddings [2 marks].
   For this part, you should include the word embeddings for 10 of the words.

   Below displays the visualization of word embedding

```
from pandas import DataFrame
print(DataFrame(word_embeddings, index=idx2word.values()).head(10))
```

```
                0         1         2   ...        97        98        99
woods        0.014584 -0.023229  0.034505  ...  0.002564  0.000413  0.003026
hanging      0.012257  0.003929 -0.023911  ... -0.016317  0.021681  0.031446
woody       -0.035216 -0.006086  0.008985  ...  0.037916 -0.020993 -0.029235
arranged    -0.010463 -0.001388 -0.017231  ...  0.009159  0.007215 -0.019347
bringing     0.002758  0.011330 -0.013720  ...  0.067569  0.038217 -0.034302
wooden       0.050368  0.021010  0.008729  ... -0.010500 -0.061428  0.087212
errors       0.007921  0.031989 -0.049887  ... -0.006778 -0.029282  0.007567
dialogs      0.041077 -0.051241 -0.038249  ... -0.007028  0.033451 -0.033306
kids        -0.064942  0.010439 -0.026839  ...  0.040594 -0.006789 -0.016402
uplifting    0.079436  0.018429 -0.024320  ... -0.046516 -0.056419  0.075209

[10 rows x 100 columns]
```

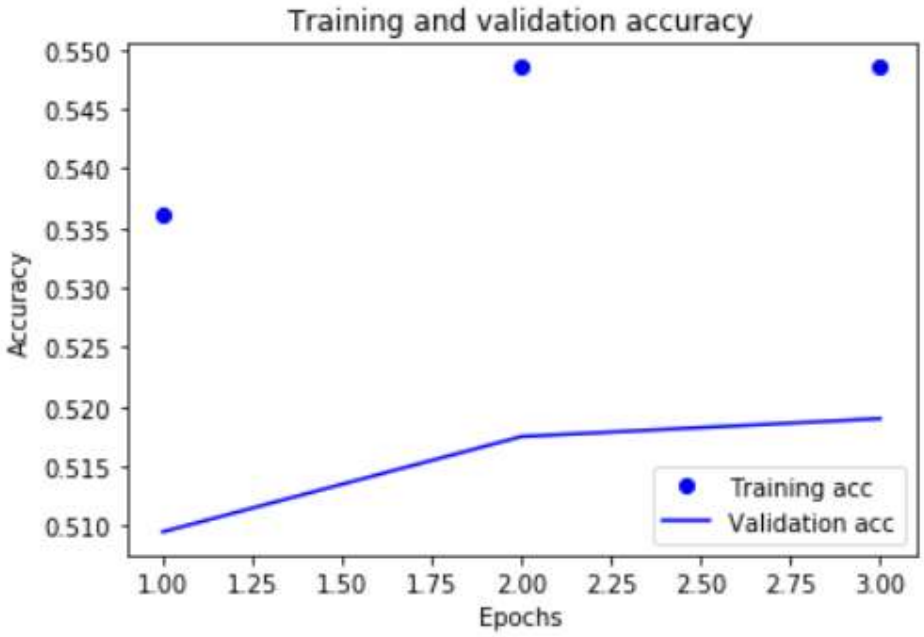
8. Section 9 [4 marks].
   For this paper, you have to write down your answers to the questions. 2 points each for questions 1 and 2.

   1. 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 the LSTM 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?

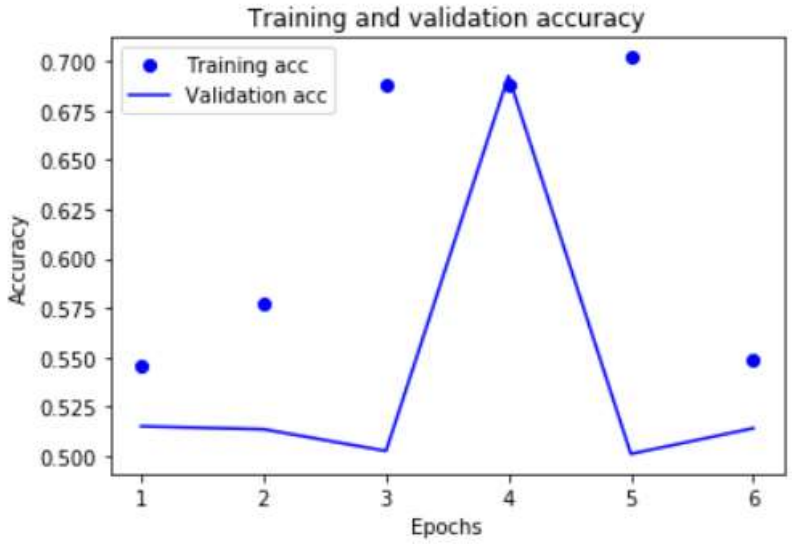
   Answer-

   Reduce Overfitting with dropout regularization. Furthermore, the network becomes less sensitive to the specific weights of neurons.

   Eposhs = 3



Eposhs = 6

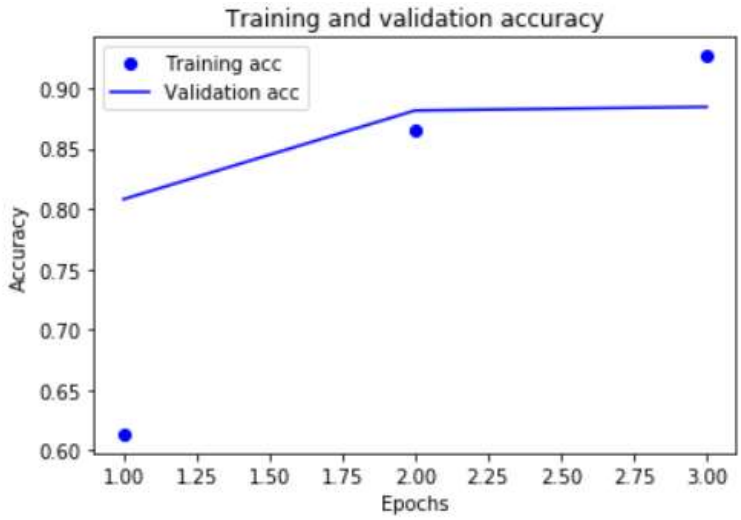


2. Experiment with compiling the model with batch sizes of 1, 32, len(training_data). What do you observe?

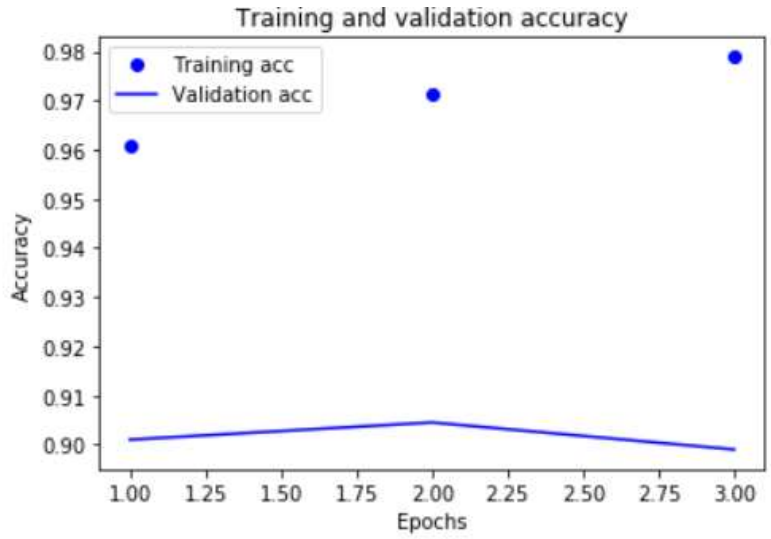
Answer- Batch size is inversely proportional to time taken. So as the batch size increased the execution was faster.

Observed overfitting with increase in batch size

Batch size = 1



Batch size = 32



Batch size = len(train_data) was not executable

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

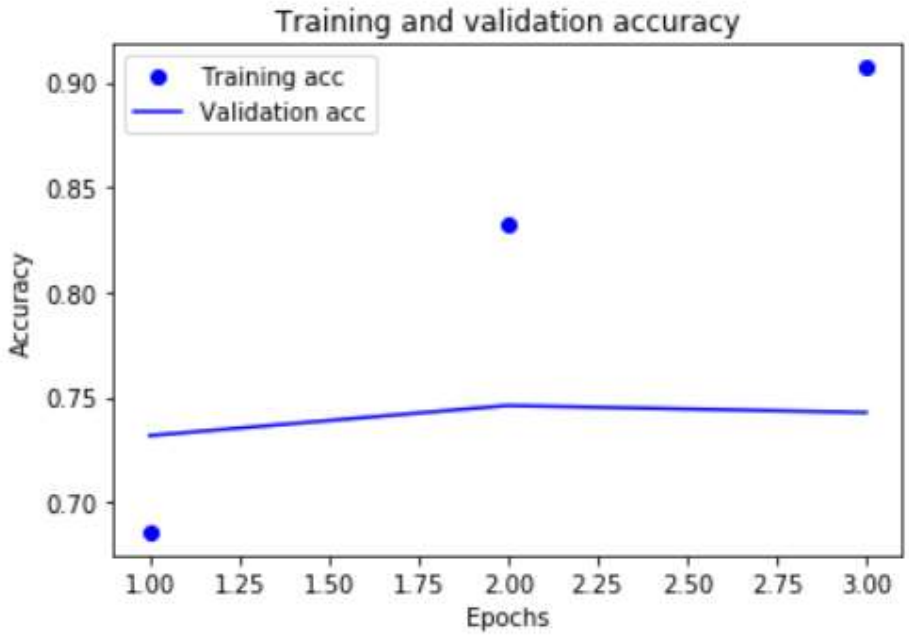
1. Build and evaluate a basic classifier for Subtask A, adapting one of your models from Parts B and C above [10 marks].

Adapted model in Part C

Below shows the model structure obtained:

```
Model: "model_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_31 (InputLayer)        (None, 40)                0
_________________________________________________________________
embedding_30 (Embedding)     (None, 40, 100)           2027900
_________________________________________________________________
lstm_30 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 101
=================================================================
Total params: 2,108,401
Trainable params: 2,108,401
Non-trainable params: 0
_________________________________________________________________
```

Furthermore, below shoes the show the Training and validation accuracy.



Evaluation of basic classifier for Subtask A, using Part C model.

```
860/860 [==============================] - 0s 361us/step
test_loss: 0.505951326808264 test_accuracy: 0.789534884552623
```