ECS7001P - NEURAL NETWORKS AND NLP

School of Electronic Engineering and Computer Science, Queen Mary University of London, UK

Assignment 1: Embeddings, Text Classification, And Machine Translation

SHIVANI GURUNG RAKESH: 210268155

A. Word Embeddings with Word2Vec

1. Preprocessing the training corpus

```
4 [['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane',
'Austen', '1811', ']'], ['CHAPTER', '1'], ['But', ',',
'then', ',', 'if', 'Mrs', '.', 'Dashwood', 'should', 'live',
'fifteen', 'years', 'we', 'shall', 'be', 'completely',
'taken', 'in', '."'], ['"', 'Fifteen', 'years', '!']]

2 ['sense sensibility jane austen', 'mrs dashwood live
fifteen years shall completely taken']
```

2. Creating the corpus vocabulary and preparing the dataset

```
Number of unique words: 10098

Sample word2idx:  [('sense', 0), ('sensibility', 1),
('jane', 2), ('austen', 3), ('family', 4), ('dashwood', 5),
('long', 6), ('settled', 7), ('sussex', 8), ('estate', 9)]

Sample idx2word: [(0, 'sense'), (1, 'sensibility'), (2,
'jane'), (3, 'austen'), (4, 'family'), (5, 'dashwood'), (6,
'long'), (7, 'settled'), (8, 'sussex'), (9, 'estate')]

Sample sents_as_id: [[0, 1, 2, 3], [68, 5, 194, 592, 33,
593, 285, 594]]
```

3. Building the skip-gram neural network architecture

```
Model: "model"
__________________________________________________________________________________________________
 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)      1009800     ['input_1[0][0]']

 context_embed_layer (Embedding  (None, 1, 100)      1009800     ['input_2[0][0]']
 )

 reshape (Reshape)              (None, 100)          0           ['target_embed_layer[0][0]']

 reshape_1 (Reshape)            (None, 100)          0           ['context_embed_layer[0][0]']

 dot (Dot)                      (None, 1)            0           ['reshape[0][0]',
                                                                  'reshape_1[0][0]']

 dense (Dense)                  (None, 1)            2           ['dot[0][0]']

==================================================================================================
Total params: 2,019,602
Trainable params: 2,019,602
Non-trainable params: 0
__________________________________________________________________________________________________
```

4. Training the models (and reading McCormick's tutorial)

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

The input would be a numeric/vector representation of textual data/word/string. The outcome is a single vector representation for each word that explains us how probable it is to be picked as the next word.

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

This layer takes as inputs a target word and a context word. The embedding in the preceding layer, as well as the embedding's change in the reshaping layer Finally, the dot product is taken into account.

c. What are the reasons this training approach is considered inefficient?

This training approach is considered inefficient because Internally the terms provided here do not sufficiently represent semantic connections since they do not capture the contextual meaning. A huge dataset, such as the one created artificially for word2vec, is required.

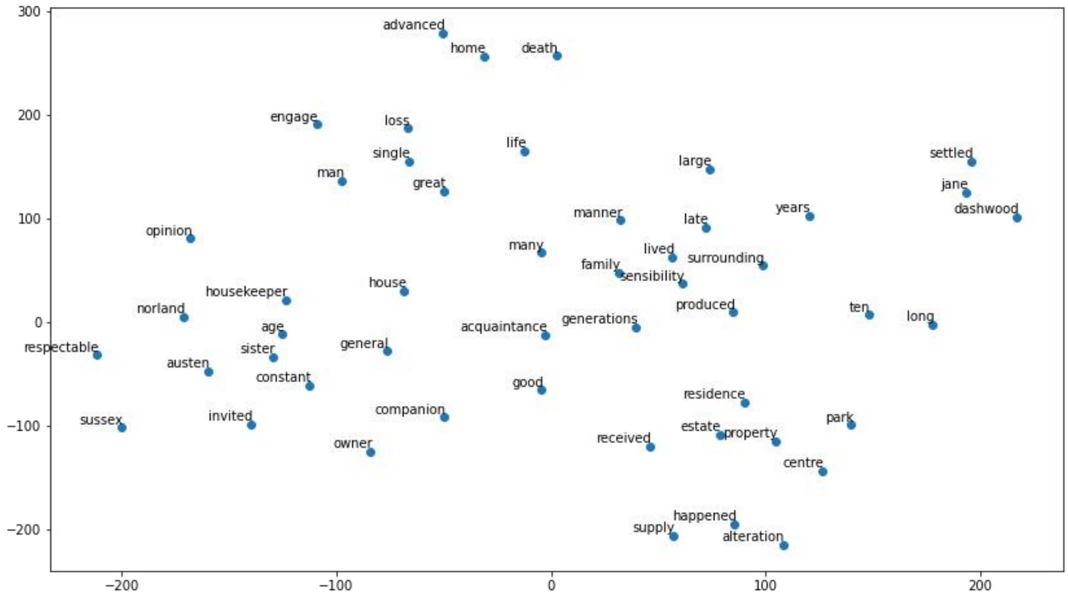
5. Getting the word embeddings

```python
from pandas import DataFrame

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

```
                    0         1         2         3         4         5  \
sense       -0.004377  0.011746 -0.012830 -0.009580 -0.020550 -0.014081
sensibility  0.003382  0.000622  0.022603  0.030473  0.017281  0.023543
jane         0.089320  0.010615 -0.082091  0.033990  0.015080  0.165932
austen      -0.026869 -0.021149  0.016017  0.043684 -0.000899 -0.000202
family       0.004196 -0.017450 -0.034747  0.027821  0.072123  0.012793
dashwood    -0.077659  0.020736 -0.064296 -0.015214 -0.024381  0.064664
long         0.080348  0.024557  0.025339 -0.112085  0.104181  0.079189
settled     -0.044766  0.028073  0.087832 -0.096838  0.034346 -0.003181
sussex      -0.028548  0.002718  0.023980 -0.012256  0.014442  0.012626
estate      -0.037755 -0.041832 -0.022449  0.004864 -0.003568  0.023726

                    6         7         8         9  ...        90        91  \
sense       -0.019290 -0.005767 -0.006455  0.002863  ...  0.009058  0.004660
sensibility -0.014133  0.044463 -0.008119 -0.000441  ... -0.000632  0.015275
jane        -0.010550 -0.045091  0.055103  0.051547  ...  0.073622 -0.043846
austen       0.005886 -0.034634 -0.003676  0.015691  ... -0.012423  0.008044
family      -0.048012  0.059493 -0.014547 -0.007315  ...  0.097147  0.085674
dashwood    -0.100502 -0.082606  0.022895  0.081411  ... -0.076113 -0.160559
long        -0.009862  0.161314  0.021617  0.029898  ... -0.028522  0.032630
settled     -0.006206 -0.041104 -0.005688 -0.060299  ... -0.051511  0.059097
sussex      -0.015412 -0.013735 -0.003959 -0.029497  ... -0.001287  0.012682
estate       0.025047  0.035452 -0.023745  0.011232  ... -0.028287 -0.001712
```

```
                   92        93        94        95        96        97  \
sense        0.022175  0.019816 -0.017966 -0.011483  0.018054 -0.017310
sensibility -0.042384  0.012944 -0.001387  0.004599  0.022433 -0.023866
jane         0.013386 -0.004306 -0.072449 -0.089178  0.035184 -0.076658
austen       0.021220 -0.009510  0.001668  0.006835  0.036758 -0.010606
family      -0.056270  0.012475  0.028407 -0.058864 -0.008543  0.054817
dashwood    -0.084238  0.078736  0.068596 -0.108069  0.096946  0.066056
long         0.017362  0.036617 -0.048536  0.082366 -0.083572 -0.040926
settled     -0.009625 -0.044698  0.009037 -0.088269  0.006469 -0.040118
sussex      -0.013260  0.038690 -0.005735  0.000106  0.002469  0.022721
estate      -0.035640  0.028941  0.042068  0.095949 -0.038173  0.049339

                   98        99
sense        0.000966  0.012991
sensibility -0.032437 -0.023564
jane        -0.036430  0.028108
austen      -0.005499  0.028972
family      -0.053529 -0.036122
dashwood    -0.042492 -0.012248
long         0.014095  0.047280
settled      0.038472  0.049939
sussex       0.021741  0.009505
estate      -0.059784 -0.018636

[10 rows x 100 columns]
```

6. Exploring and visualizing your word embeddings using t-SNE



B. Using LSTMs for Text Classification

1. Section 2, reading the inputs for the LSTM

```
print('Length of sample train_data before preprocessing:', len(train_data[0]))
print('Length of sample train_data after preprocessing:', len(padded_train_data[0]))
print('Sample train data:', padded_train_data[0])
```

```
Length of sample train_data before preprocessing: 218
Length of sample train_data after preprocessing: 500
Sample train data: [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    1   13   21   15   42  529  972 1621 1384   64  457 4467
   65 3940    3  172   35  255    4   24   99   42  837  111   49  669
    2    8   34  479  283    4  149    3  171  111  166    2  335  384
   38    3  171 4535 1110   16  545   37   12  446    3  191   49   15
```

```
   38    3  171 4535 1110   16  545   37   12  446    3  191   49   15
    5  146 2024   18   13   21    3 1919 4612  468    3   21   70   86
   11   15   42  529   37   75   14   12 1246    3   21   16  514   16
   11   15  625   17    2    4   61  385   11    7  315    7  105    4
    3 2222 5243   15  479   65 3784   32    3  129   11   15   37  618
    4   24  123   50   35  134   47   24 1414   32    5   21   11  214
   27   76   51    4   13  406   15   81    2    7    3  106  116 5951
   14  255    3    2    6 3765    4  722   35   70   42  529  475   25
  399  316   45    6    3    2 1028   12  103   87    3  380   14  296
   97   31 2070   55   25  140    5  193 7485   17    3  225   21   20
  133  475   25  479    4  143   29 5534   17   50   35   27  223   91
   24  103    3  225   64   15   37 1333   87   11   15  282    4   15
 4471  112  102   31   14   15 5344   18  177   31]
```

2. Building the model

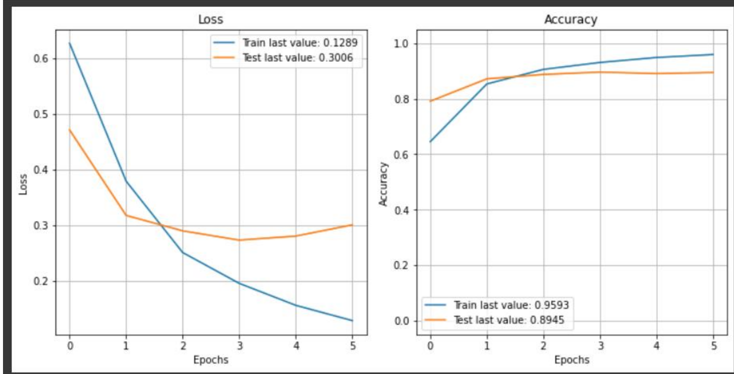
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500)]             0

 embed_layer (Embedding)     (None, 500, 100)          1000000

 lstm_1 (LSTM)               (None, 100)               80400

 output_layer (Dense)        (None, 1)                 101

=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

3. Section 4, training the model

```
history = model.fit(train_x, train_y, epochs=6, batch_size=1000, validation_data=(validation_x, validation_y))

Epoch 1/6
23/23 [==============================] - 37s 1s/step - loss: 0.6265 - accuracy: 0.6451 - val_loss: 0.4710 - val_accuracy: 0.7910
Epoch 2/6
23/23 [==============================] - 22s 949ms/step - loss: 0.3790 - accuracy: 0.8531 - val_loss: 0.3176 - val_accuracy: 0.8715
Epoch 3/6
23/23 [==============================] - 22s 955ms/step - loss: 0.2510 - accuracy: 0.9057 - val_loss: 0.2899 - val_accuracy: 0.8875
Epoch 4/6
23/23 [==============================] - 24s 1s/step - loss: 0.1957 - accuracy: 0.9307 - val_loss: 0.2733 - val_accuracy: 0.8960
Epoch 5/6
23/23 [==============================] - 22s 954ms/step - loss: 0.1563 - accuracy: 0.9486 - val_loss: 0.2805 - val_accuracy: 0.8905
Epoch 6/6
23/23 [==============================] - 22s 957ms/step - loss: 0.1289 - accuracy: 0.9593 - val_loss: 0.3006 - val_accuracy: 0.8945
```

```
plot_history(history.history, path="standard.png")
plt.show()
```



Based on the accuracy plot, what do you think the optimal stopping point for your model should have been?

```
[ ] len(test_data)

25000
```

4. Evaluating the model on the test data

Evaluate the model on the padded test data using the code in the following cell block.

```
[ ] # YOUR CODE TO EVALUATE THE MODEL ON TEST DATA GOES HERE
    results = model.evaluate(padded_test_data, test_labels)
    print('test_loss:', results[0], 'test_accuracy:', results[1])

782/782 [==============================] - 110s 139ms/step - loss: 0.3601 - accuracy: 0.8685
test_loss: 0.3600790500640869 test_accuracy: 0.8684800267219543
```

5. Section 6, extracting the word embedding

Sanity Check

Print the shape of the word embeddings using the line of code below. It should return (VOCAB_SIZE, EMBED_SIZE)

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

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

6. Visualizing the reviews

```
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

```
from pandas import DataFrame

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

```
                 0         1         2         3         4         5  \
woods      0.021322 -0.016948  0.016021 -0.001831  0.003546  0.018424
hanging   -0.028856  0.000362 -0.005027  0.004653  0.011076 -0.019683
woody     -0.002695  0.018942 -0.016365 -0.008425  0.020334 -0.023362
arranged  -0.014585 -0.000727 -0.016299 -0.011049  0.006553  0.013894
bringing   0.008422  0.010303 -0.001404  0.012576  0.019186 -0.013220
wooden     0.011720 -0.005625  0.011642 -0.028148  0.012258 -0.014192
errors     0.025149 -0.017638  0.017255 -0.006253 -0.015176  0.020370
dialogs   -0.026621  0.008397 -0.018563 -0.003514 -0.014060  0.020323
kids       0.011181  0.007972  0.001516  0.006886 -0.004066  0.011590
uplifting -0.012604 -0.005350 -0.017992 -0.005206 -0.020141  0.019626

                 6         7         8         9  ...        90        91  \
woods      0.016331 -0.010687  0.009252  0.009148  ... -0.010355  0.011949
hanging   -0.023211 -0.002756  0.008896  0.009968  ...  0.019486 -0.017137
woody      0.015943 -0.009274  0.007285  0.009084  ...  0.020612 -0.005199
arranged  -0.023375 -0.009402 -0.018564 -0.021615  ...  0.011507  0.013424
bringing  -0.015699  0.002351  0.001682 -0.003326  ...  0.019078 -0.006840
wooden     0.021057 -0.024747 -0.002334 -0.011708  ... -0.017786  0.016806
errors    -0.013985 -0.004677 -0.005856  0.010407  ... -0.000786 -0.013370
dialogs    0.009236  0.013456  0.016642  0.008468  ...  0.002678 -0.020391
kids      -0.019377 -0.005905  0.027780  0.002733  ...  0.016922  0.006767
uplifting -0.002850  0.009913  0.015541 -0.012341  ...  0.013410 -0.015321
```

```
                 92        93        94        95        96        97  \
woods     -0.001569 -0.015296  0.016905 -0.001719 -0.018573 -0.016062
hanging    0.017108 -0.015979  0.012520  0.012075  0.020443 -0.008092
woody     -0.025347 -0.002880  0.006627 -0.005779 -0.005327  0.021822
arranged   0.003250  0.020580 -0.019248 -0.013976  0.009721 -0.017364
bringing  -0.000961  0.005473  0.029043 -0.019207  0.028355  0.000540
wooden     0.016476 -0.018995  0.023646  0.005147  0.009694  0.003889
errors    -0.015242  0.011041 -0.020507  0.014845  0.016741 -0.005231
dialogs   -0.006387 -0.011488 -0.001304  0.000383 -0.012729 -0.006082
kids      -0.002744 -0.019494  0.006983 -0.016619  0.022346  0.013522
uplifting -0.003392  0.004667  0.018703 -0.008488  0.023051 -0.018911

                 98        99
woods      0.003212  0.010031
hanging    0.003593 -0.006954
woody      0.018661  0.006990
arranged  -0.022717  0.012371
bringing  -0.001960  0.021795
wooden    -0.005483  0.019206
errors    -0.010556 -0.011302
dialogs    0.014426 -0.007965
kids       0.008722  0.010755
uplifting -0.018329  0.010686

[10 rows x 100 columns]
```

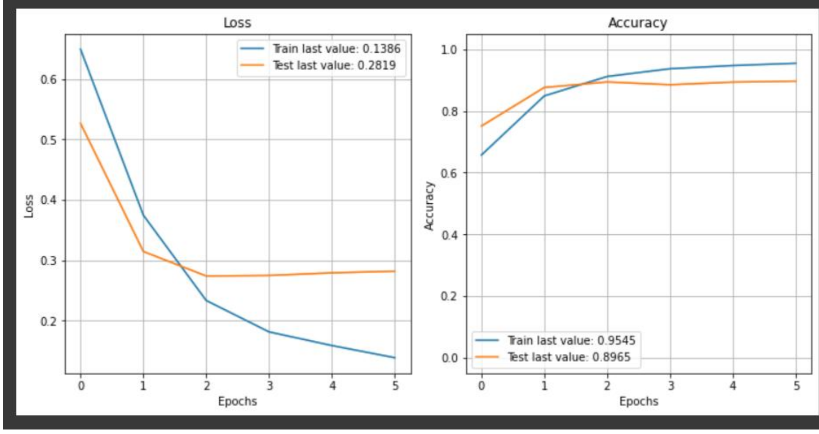
8.

Section 9

a. Create a new model that is a copy of 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?

```
[ ]  new_model.summary()

     Model: "model_1"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      input_2 (InputLayer)        [(None, 500)]             0

      embed_layer (Embedding)     (None, 500, 100)          1000000

      dropout_layer_1 (Dropout)   (None, 500, 100)          0

      lstm_1 (LSTM)               (None, 100)               80400

      dropout_layer_2 (Dropout)   (None, 100)               0

      output_layer (Dense)        (None, 1)                 101

     =================================================================
     Total params: 1,080,501
     Trainable params: 1,080,501
     Non-trainable params: 0
     _________________________________________________________________
```

```
plot_history(new_history.history, path="standard.png")
plt.show()
```

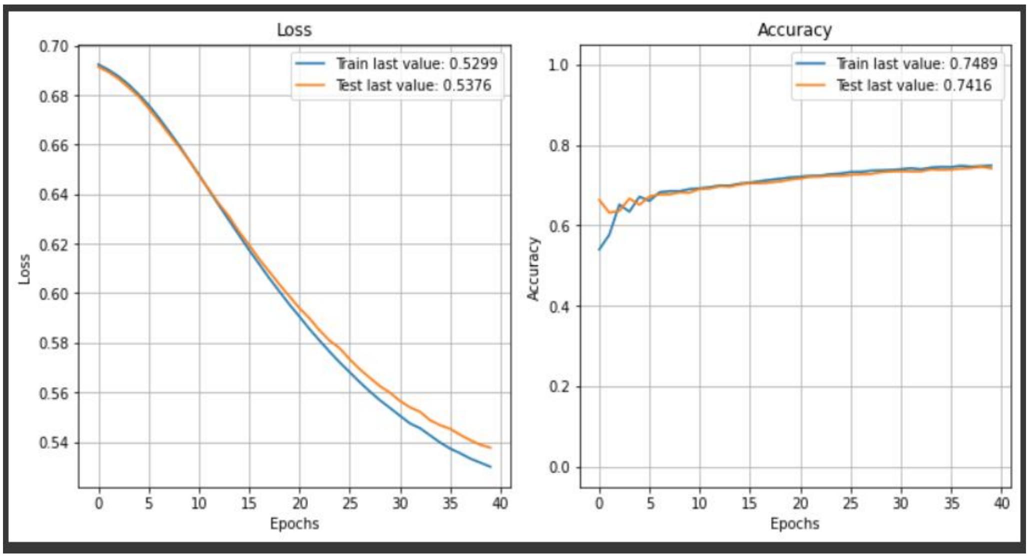


The graphs before and after we applied dropout to the model may be seen. Even after running the whole epochs, we can plainly observe that after adding the dropout, our validation loss does not rise. This demonstrates that our model is no longer over fitting. We can also see that after we introduced the dropouts, the validation accuracy has improved somewhat. We might be able to improve accuracy and reduce over fitting by increasing dropout rates even further.

C. Comparing The Classification Models

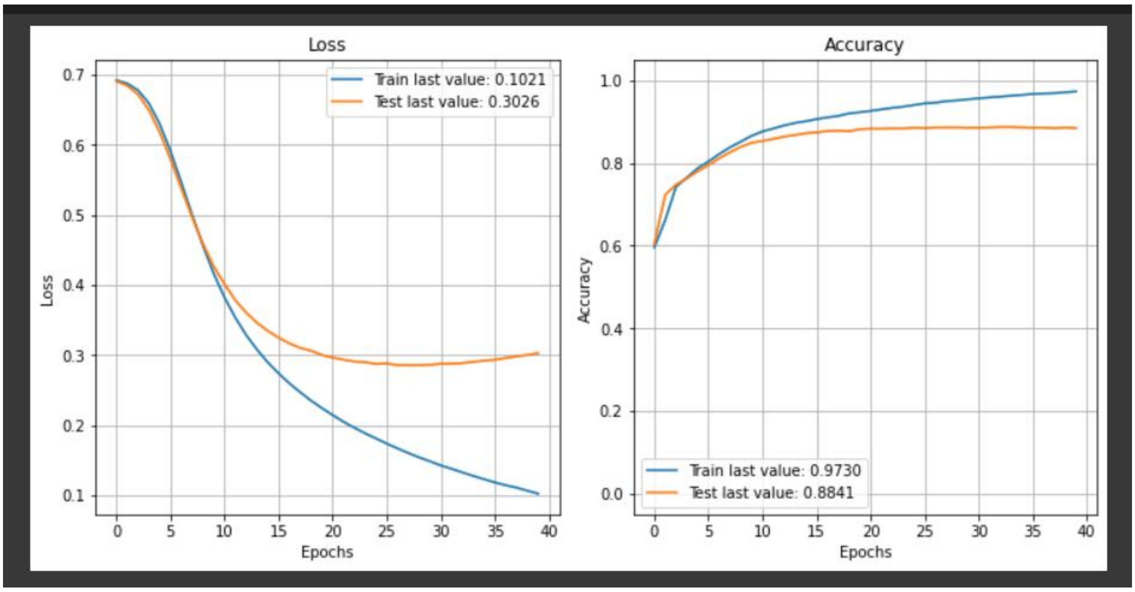
1. Build a neural network classifier using one-hot word vectors (Model 1), and train and evaluate it

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256)]             0

 lambda (Lambda)             (None, 256, 10000)        0

 global_average_pooling1d_ma  (None, 10000)            0
 sked (GlobalAveragePooling1
 DMasked)

 dense (Dense)               (None, 16)                160016

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 160,033
Trainable params: 160,033
Non-trainable params: 0
_________________________________________________________________
```



2. Modify your model to use a word embedding layer instead of one-hot vectors (Model 2), and to learn the values of these word embedding vectors along with the model

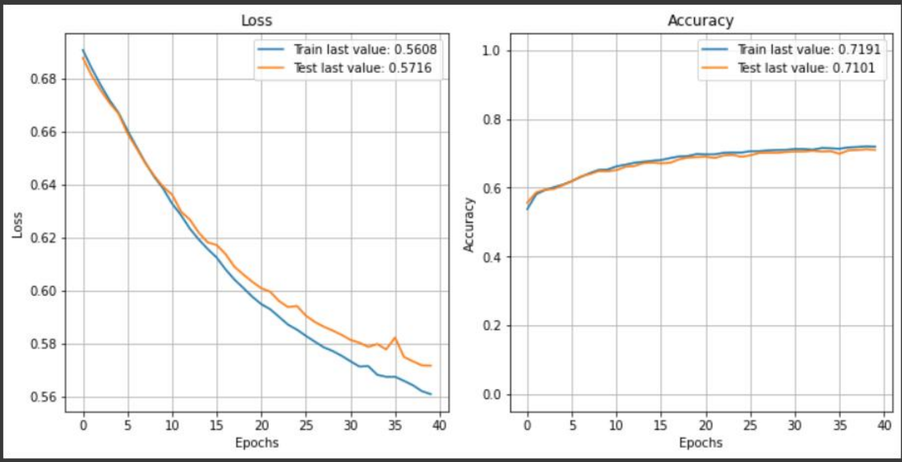
```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256)]             0

 embedding (Embedding)       (None, 256, 16)           160000

 global_average_pooling1d_ma  (None, 16)               0
 sked_1 (GlobalAveragePoolin
 g1DMasked)

 dense_2 (Dense)             (None, 16)                272

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 160,289
Trainable params: 160,289
Non-trainable params: 0
_________________________________________________________________
```



3. Adapt your model to load and use pre-trained word embeddings in- stead (Model 3); train and evaluate it and compare the effect of freez- ing and fine-tuning the embeddings

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256)]             0

 GloVe_Embeddings (Embedding  (None, 256, 300)         120000300
 )

 global_average_pooling1d_ma  (None, 300)              0
 sked_2 (GlobalAveragePoolin
 g1DMasked)

 dense_4 (Dense)             (None, 16)                4816

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 120,005,133
Trainable params: 4,833
Non-trainable params: 120,000,300
_________________________________________________________________
```

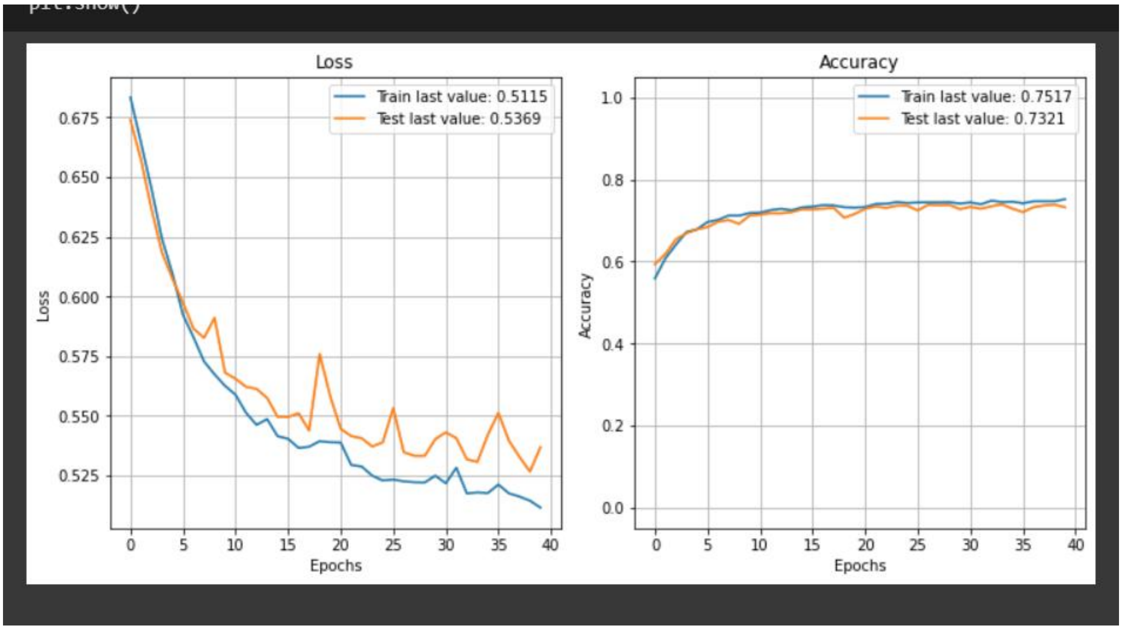
```
plot_history(history_3.history, path="standard.png")
plt.show()
```



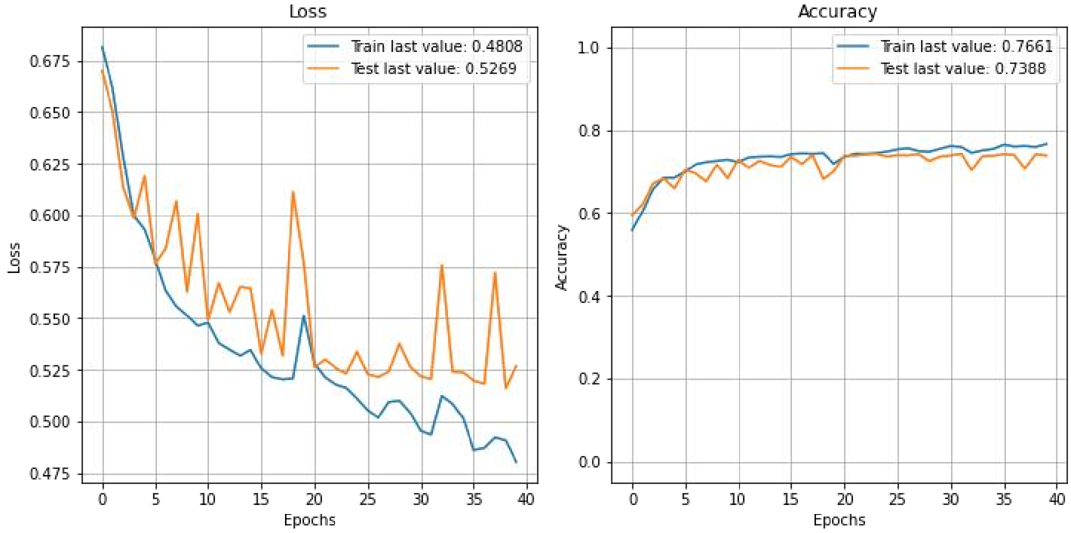
4. One way to improve the performance is to add another fully-connected layer to your network. Try this (Model 4) and see if it improves the performance. If not, what can you do to improve it?

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 256)]             0

 GloVe_Embeddings (Embedding  (None, 256, 300)         120000300
 )

 global_average_pooling1d_ma  (None, 300)              0
 sked_4 (GlobalAveragePoolin
 g1DMasked)

 dense_9 (Dense)             (None, 100)               30100

 dense_10 (Dense)            (None, 16)                1616

 dense_11 (Dense)            (None, 1)                 17

=================================================================
Total params: 120,032,033
Trainable params: 31,733
Non-trainable params: 120,000,300
_________________________________________________________________
```

- Training and validation loss plot after adding one dense layer



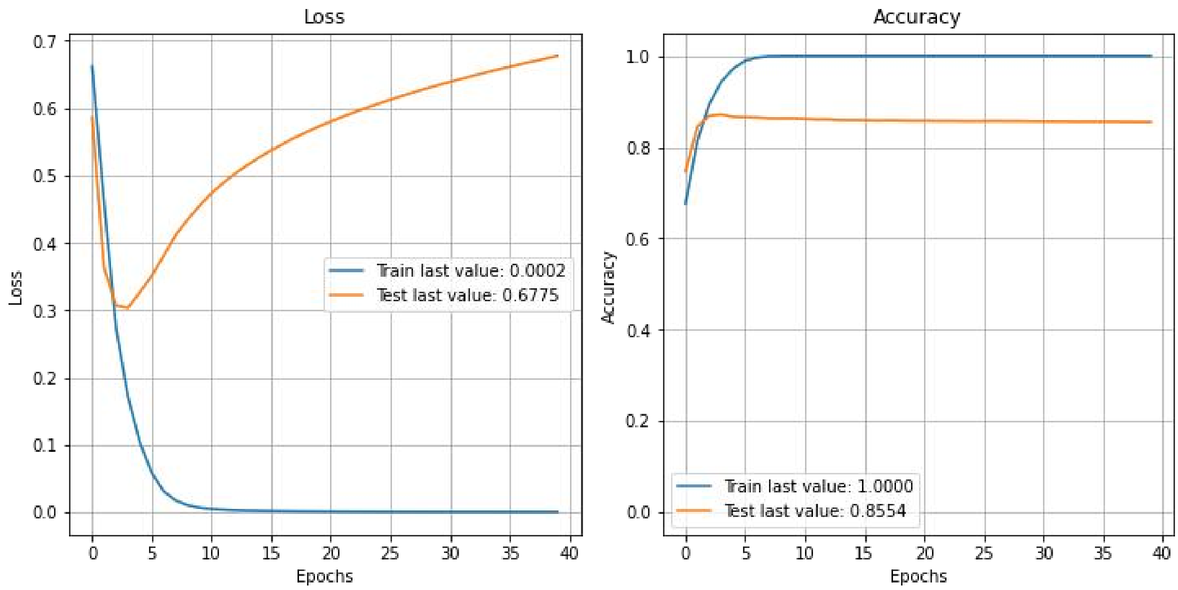
- Training and validation loss plot after adding two dense layer



As layers are added, the number of weights in the network increases, lowering the model's complexity. Without a large training set, an increasingly huge network is prone to overfit, lowering accuracy on test data. The two plots show that test accuracy is decreasing while train accuracy is increasing, signalling that our model is overfitting. We can improve performance by adding dropouts, adjusting hyperparameters, and even simplifying the model.

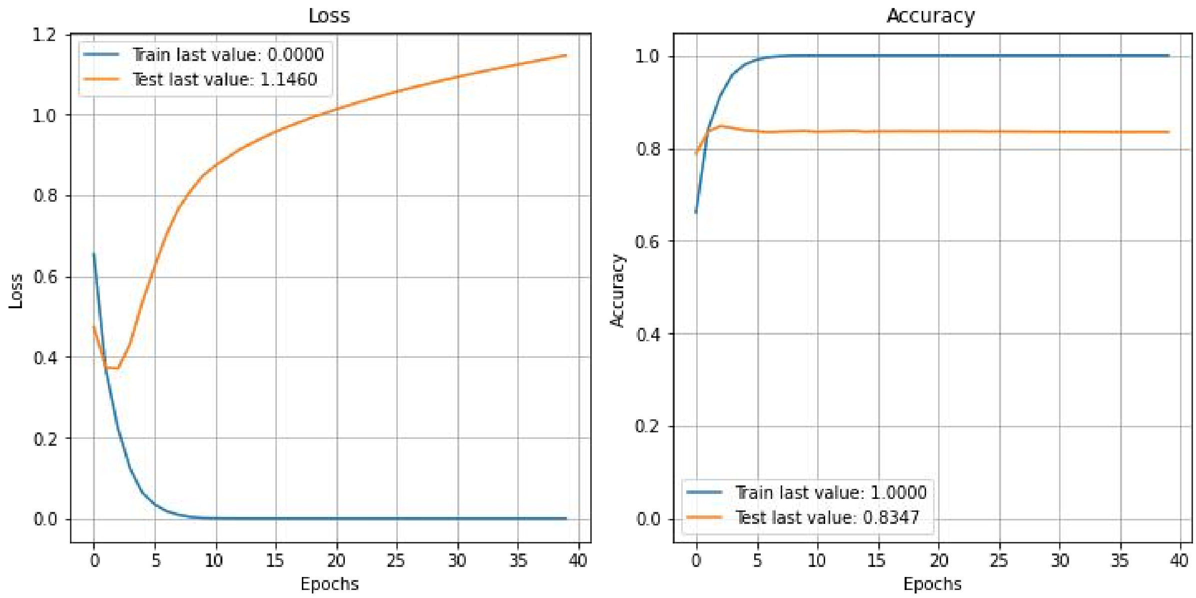
5. Build a CNN classifier (Model 5), and train and evaluate it. Then try adding extra convolutional layers, and conduct training and evaluation.
   - one convolutional layer

```
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 256)]             0

 embed_layer (Embedding)     (None, 256, 300)          3000000

 conv1d (Conv1D)             (None, 251, 100)          180100

 global_max_pooling1d (Globa  (None, 100)              0
 lMaxPooling1D)

 dense_16 (Dense)            (None, 1)                 101

=================================================================
Total params: 3,180,201
Trainable params: 3,180,201
Non-trainable params: 0
_________________________________________________________________
```



- Extra convolutional layer

```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 256)]             0

 embed_layer (Embedding)     (None, 256, 300)          3000000

 conv1d_1 (Conv1D)           (None, 251, 100)          180100

 conv1d_2 (Conv1D)           (None, 246, 100)          60100

 global_max_pooling1d_1 (Glo  (None, 100)              0
 balMaxPooling1D)

 dense_17 (Dense)            (None, 1)                 101

=================================================================
Total params: 3,240,301
Trainable params: 3,240,301
Non-trainable params: 0
_________________________________________________________________
```



The validation accuracy has decreased and the validation loss has increased as our model has began to overfit. The model's complexity rises as the layered complexity rises, which is presumably why it's overfit.

D. A Real Text Classification Task

1. Preprocess the data, to adapt the models from Parts C - Done in Jupiter notebook

2. Adapt your models without pre-trained word embeddings in Part C to this task

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128)]             0

 embedding (Embedding)       (None, 128, 100)          789800

 global_average_pooling1d (G  (None, 100)              0
 lobalAveragePooling1D)

 dense (Dense)               (None, 16)                1616

 dense_1 (Dense)             (None, 3)                 51

=================================================================
Total params: 791,467
Trainable params: 791,467
Non-trainable params: 0
```

The accuracy of the validation data was 55.9%. The accuracy of the test data was 58.6%.

3. Adapt your models with pre-trained word embeddings in Part C to this task (Model 2); train and evaluate it

Neural bag of words using pre-trained word embeddings

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 128)]             0

 GloVe_Embeddings (Embedding  (None, 128, 300)         120000300
 )

 global_average_pooling1d_2   (None, 300)              0
 (GlobalAveragePooling1D)

 dense_6 (Dense)             (None, 16)                4816

 dense_7 (Dense)             (None, 3)                 51

=================================================================
Total params: 120,005,167
Trainable params: 4,867
Non-trainable params: 120,000,300
```

4. Build and evaluate two more classifiers with multiple inputs (Model 3: separate inputs for text and aspect)

```
Model: "model_6"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_layer_6 (InputLayer)     [(None, 16)]         0           []

 input_layer_7 (InputLayer)     [(None, 128)]        0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_layer_6[0][0]',
                                                                  'input_layer_7[0][0]']

 global_average_pooling1d_4 (Gl  (None, 300)         0           ['GloVe_Embeddings[3][0]']
 obalAveragePooling1D)

 global_average_pooling1d_5 (Gl  (None, 300)         0           ['GloVe_Embeddings[4][0]']
 obalAveragePooling1D)

 dense_12 (Dense)               (None, 16)           4816        ['global_average_pooling1d_4[0][0
                                                                 ]']

 dense_13 (Dense)               (None, 16)           4816        ['global_average_pooling1d_5[0][0
                                                                 ]']

 concatenate (Concatenate)      (None, 32)           0           ['dense_12[0][0]',
                                                                  'dense_13[0][0]']

 dense_14 (Dense)               (None, 3)            99          ['concatenate[0][0]']

==================================================================================================
Total params: 120,010,031
Trainable params: 9,731
Non-trainable params: 120,000,300
__________________________________________________________________________________________________
```

5. Build and evaluate the classifier extracting information from LSTM (Model 4)

```
Model: "model_8"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_7 (InputLayer)           [(None, 128)]        0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_7[0][0]']

 bidirectional (Bidirectional)  (None, 128, 200)     320800      ['GloVe_Embeddings[7][0]']

 input_8 (InputLayer)           [(None, 128)]        0           []

 dot (Dot)                      (None, 200)          0           ['bidirectional[0][0]',
                                                                  'input_8[0][0]']

 dense_18 (Dense)               (None, 16)           3216        ['dot[0][0]']

 dense_19 (Dense)               (None, 3)            51          ['dense_18[0][0]']

==================================================================================================
Total params: 120,324,367
Trainable params: 324,067
Non-trainable params: 120,000,300
__________________________________________________________________________________________________
```

E. Neural Machine Translation

1. Implementing the encoder

```
"""
Task 1 encoder

Start
"""
# The train encoder
# (a.) Create two randomly initialized embedding lookups, one for the source, another for the target.
print('Task 1(a): Creating the embedding lookups...')
embeddings_source = Embedding(input_dim=self.vocab_source_size,output_dim=self.embedding_size,mask_zero=True,trainable=True)
embeddings_target = Embedding(input_dim=self.vocab_target_size,output_dim=self.embedding_size,mask_zero=True,trainable=True)

# (b.) Look up the embeddings for source words and for target words. Apply dropout to each encoded input
print('\nTask 1(b): Looking up source and target words...')
source_word_embeddings =  Dropout(self.embedding_dropout_rate)(embeddings_source(source_words))
target_words_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_target(target_words))

# (c.) An encoder LSTM() with return sequences set to True
print('\nTask 1(c): Creating an encoder')

encoder_outputs, encoder_state_h, encoder_state_c = LSTM(self.hidden_size,return_sequences=True,return_state=True)(source_word_embeddings)

"""
End Task 1
"""
```

We will begin by creating the embedding layers for the source and target. A random initialization will be used to start the embeddings, and they will be taught during training. The input dimensions will be the source and target dimensions. They will both be the same size for embedding and output. We will set "mask zero" to true because we wish to remove the padding. Next, dropouts will be added to the embeddings for both the source and destination. An estimation of the dropout rate has already been made.

The inputs will be sent to the embedding layers, who will then pass them on to the dropout layers. Finally, we'll create an LSTM layer to handle the output of the dropout layers. To get the outputs, set "return sequence" to true, and "return state" to true to access the LSTM's hidden states.

2. In this step we will be implementing the decoder

```
"""
Task 2 decoder for inference

Start
"""
# Task 1 (a.) Get the decoded outputs
print('\n Putting together the decoder states')
# get the intitial states for the decoder, decoder_states
# decoder states are the hidden and cell states from the training stage

decoder_states = [decoder_state_input_h, decoder_state_input_c]

# use decoder states as input to the decoder lstm to get the decoder outputs, h, and c for test time inference
decoder_outputs_test,decoder_state_output_h, decoder_state_output_c = decoder_lstm(target_words_embeddings,initial_state=decoder_states)


# Task 1 (b.) Add attention if attention
if self.use_attention:
    decoder_attention = AttentionLayer()
    decoder_outputs_test = decoder_attention([encoder_outputs_input,decoder_outputs_test])

# Task 1 (c.) pass the decoder_outputs_test (with or without attention) to the decoder dense layer
decoder_outputs_test = decoder_dense(decoder_outputs_test)

"""
End Task 2
"""
```

- Decoder interface model summary

Putting together the decoder states

```
                                     Decoder Inference Model summary
Model: "model_2"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_2 (InputLayer)           [(None, None)]       0           []

 embedding_1 (Embedding)        (None, None, 100)    250600      ['input_2[0][0]']

 dropout_1 (Dropout)            (None, None, 100)    0           ['embedding_1[0][0]']

 input_3 (InputLayer)           [(None, 200)]        0           []

 input_4 (InputLayer)           [(None, 200)]        0           []

 lstm_1 (LSTM)                  [(None, None, 200),  240800      ['dropout_1[0][0]',
                                 (None, 200),                     'input_3[0][0]',
                                 (None, 200)]                     'input_4[0][0]']

 input_5 (InputLayer)           [(None, None, 200)]  0           []

 dense (Dense)                  (None, None, 2506)   503706      ['lstm_1[1][0]']

=================================================================================================
Total params: 995,106
Trainable params: 995,106
Non-trainable params: 0
```

```
                                        Decoder Inference Model summary
Model: "model_5"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape          Param #     Connected to
=================================================================================================
 input_7 (InputLayer)           [(None, None)]        0           []

 embedding_3 (Embedding)        (None, None, 100)     250600      ['input_7[0][0]']

 dropout_3 (Dropout)            (None, None, 100)     0           ['embedding_3[0][0]']

 input_8 (InputLayer)           [(None, 200)]         0           []

 input_9 (InputLayer)           [(None, 200)]         0           []

 input_10 (InputLayer)          [(None, None, 200)]   0           []

 lstm_3 (LSTM)                  [(None, None, 200),   240800      ['dropout_3[0][0]',
                                 (None, 200),                      'input_8[0][0]',
                                 (None, 200)]                      'input_9[0][0]']

 attention_layer_1 (AttentionLa  (None, None, 400)    0           ['input_10[0][0]',
 yer)                                                              'lstm_3[1][0]']

 dense_1 (Dense)                (None, None, 2506)    1004906     ['attention_layer_1[0][0]']

=================================================================================================
Total params: 1,496,306
Trainable params: 1,496,306
Non-trainable params: 0
```

We use the decoder model to create the decoder states rather than the encoder states. We then feed the LSTM the list of encoder states. In the LSTM, the target embeddings are taken as input, and the starting state is used to build a decoder. Our LSTM produces three outputs, which are assigned to variable.
The attention layer condition will be added next, which is the same as the one in encoder. The LSTM or attention layer output will be transmitted to the last dense layer, which will assign probability for the following token.This test set had a BLEU score of 5.06

3. Adding attention

```
"""
Task 3 attention

Start
"""

luong_score = tf.matmul(decoder_outputs, encoder_outputs, transpose_b=True)
alignment = tf.nn.softmax(luong_score, axis=2)
context = tf.matmul(K.expand_dims(alignment,axis=2), K.expand_dims(encoder_outputs,axis=1))
encoder_vector = K.squeeze(context,axis=2)

"""
End Task 3
"""
```

```
 Putting together the decoder states
                                                  Decoder Inference Model summary
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                  Output Shape         Param #     Connected to
==================================================================================================
 input_7 (InputLayer)          [(None, None)]       0           []

 embedding_3 (Embedding)       (None, None, 100)    250600      ['input_7[0][0]']

 dropout_3 (Dropout)           (None, None, 100)    0           ['embedding_3[0][0]']

 input_8 (InputLayer)          [(None, 200)]        0           []

 input_9 (InputLayer)          [(None, 200)]        0           []

 input_10 (InputLayer)         [(None, None, 200)]  0           []

 lstm_3 (LSTM)                 [(None, None, 200),  240800      ['dropout_3[0][0]',
                                (None, 200),                     'input_8[0][0]',
                                (None, 200)]                     'input_9[0][0]']

 attention_layer_1 (AttentionLa  (None, None, 400)  0           ['input_10[0][0]',
 yer)                                                            'lstm_3[1][0]']

 dense_1 (Dense)               (None, None, 2506)   1004906     ['attention_layer_1[0][0]']

==================================================================================================
Total params: 1,496,306
Trainable params: 1,496,306
Non-trainable params: 0
```

We'll now use the NMT's attention layer to boost our BLEU score. To begin, we must determine the "luong score," which is obtained by multiplying the decoder and encoder outputs in a matrix. We must transpose the "encoder outputs" since the shapes do not match the decoder output. Instead of transposing and multiplying, we used tensor flow's "matmul" function to combine the two steps.
Once the score has been computed, we should softmax the dimension with the size "max source sent len". We are doing this for axis 2. This modification is performed using softmax in TensorFlow.

The next step is to increase the dimension of "encoder output" and our softmax output so that we can multiply them element by element. Using the encoder vector, we multiplied the dimensions once they were enlarged. To get the same dimension as before, we must conduct a sum because multiplication updates the dimension. Our encoder vector is derived by summing the "max source sent len" values.This test set had a BLEU score of 15.33