ECS7001P - NN & NLP
Assignment 1: Word Representation, Text Classification and Machine Translation - v2

By-Swati Thapa

PART-A

1. Pre-processing the training corpus [2 marks].

Your report must include in your report the output of the 'Sanity check' at the end of this Section

Answer:

Before pre-processing:

```
print("length before the preprocessed output:",len(austen))
```

```
length before the preprocessed output: 16498
```

After pre-processing:

```
def preprocess_corpus(corpus):
    corpus = [[string.lower() for string in sublist] for sublist in corpus]
    stop_words = stopwords.words('english') #usage of stopwords
    punctuation=list(string.punctuation) #usage of punctuation to delete unwanted punctuation
    stop=stop_words+punctuation #now all words excepted stopwords and unwanted punctuation will be ignored
    stop.extend(['."',"--"]) #adding more punctuations
    filtered_sentence= [[w for w in sublist if not w in stop] for sublist in corpus]
    filtered_sentence=[[i for i in sublist if not i.isdigit()] for sublist in filtered_sentence]


    filtered_sentence= [sublist for sublist in filtered_sentence if len(sublist) >= 4]
    return filtered_sentence


normalized_corpus = preprocess_corpus(austen)
print('The new length of the preprocessed output:',len(normalized_corpus))
```

```
The new length of the preprocessed output: 12716
```

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

Again, you must include the output of the sanity checks at the end of the Section

Answer:

```
[13] print('Number of unique words:', len(word2idx))

Number of unique words: 10285
```

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

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

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

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

```
[16] #sents_as_ids=normalized_corpus_test_1
     print('\nSample sents_as_id:', prepareSentsAsId(preprocessed_sample))

Sample sents_as_id: [[0, 1, 2, 3], [68, 5, 195, 593, 33, 594, 286, 595]]
```

3. Building the skip-gram neural network architecture [5 marks].

To get the credit, you must include the output of the model.summary() command in the sanity check part

Answer:

```
model.summary()

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)      1028500     ['input_1[0][0]']

 context_embed_layer (Embedding  (None, 1, 100)      1028500     ['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: 2,057,002
Trainable params: 2,057,002
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

Answer:

a. What would the inputs and outputs to the model be?
   Answer:
   The inputs and outputs of the model are vectors.

   The input vector is a one-hot vector representation with 1s at the position of its occurrence in the vocabulary and the remaining are 0s.

The output vector is the probability of a word given a word adjacent (neighbour) to the input word.

When training the model, we provide two inputs, one-hot representation of two words and only one output, if the second word is in the context (environment / skip-gram) of the first word.

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

```
[ ] # The input is an array of target indices e.g. [2, 45, 7, 23,...9]
    target_word = Input((1,), dtype='int32')


    # feed the words into the model using the Keras <Embedding> layer. This is the hidden layer
    # from whose weights we will get the word embeddings.
    target_embedding = Embedding(VOCAB_SIZE, EMBED_SIZE, name='target_embed_layer',
                                 embeddings_initializer='glorot_uniform',
                                 input_length=1)(target_word)


    # at this point, the input would of the shape (num_inputs x 1 x embed_size) and has to be flattened
    # or reshaped into a (num_inputs x embed_size) tensor.
    target_input = Reshape((EMBED_SIZE, ))(target_embedding)
```

B. Write similar code for the 'context_word' input.

```
[ ] # your code for the context_word goes here
    # The input is an array of target indices e.g. [2, 45, 7, 23,...9]
    context_word = Input((1,), dtype='int32')


    # feed the words into the model using the Keras <Embedding> layer. This is the hidden layer
    # from whose weights we will get the word embeddings.
    context_embedding = Embedding(VOCAB_SIZE, EMBED_SIZE, name='context_embed_layer',
                                 embeddings_initializer='glorot_uniform',
                                 input_length=1)(context_word)


    # at this point, the input would of the shape (num_inputs x 1 x embed_size) and has to be flattened
    # or reshaped into a (num_inputs x embed_size) tensor.
    context_input = Reshape((EMBED_SIZE, ))(context_embedding)
```

- **C. Merge the inputs.**

  Recall, each training instance is a (target_word, context_word) combination. Since we are trying to learn the degree of closeness between the two words, the model will compute the cosine distance between the two inputs using the layer. https://keras.io/layers/merge/, hence fusing the two inputs into one.

```
[ ] merged_inputs = Dot(axes=-1, normalize=False)([target_input, context_input])
```

- **D. The Output Layer**

  Pass the merged inputs (now a vector with a single number the cosine distance between the two input vectors for each word) into a sigmoid activated neuron. The output of this layer is the output of the model.

  Hint: Use the layer (https://keras.io/layers/core/), with a 'sigmoid' activation function.

```
[ ] # your code for the output layer goes here
    import tensorflow as tf
    out_layer = tf.keras.layers.Dense(1, activation='sigmoid')(merged_inputs)
```

- **E. Initialize the model:**

```
[ ] # label is the output of step D.
    model = Model(inputs=[target_word, context_word], outputs=[out_layer])
```

- **F. Compile the model using the <model.compile> command.** Use Loss = 'mean_squared_error', optimizer = 'rmsprop'.

```
[ ] model.compile(loss="mean_squared_error", optimizer="rmsprop")
```

c. What are the reasons this training approach is considered inefficient?
Answer:
The final step is computationally expensive and the training approach is inefficient, as the probabilities of all words in the vocabulary need to be calculated, and not all words need to be in context.

d. Training the model

```
Processed all 12715 sentences
Epoch: 1 Loss: 0.7499145418405533

Processed all 12715 sentences
Epoch: 2 Loss: 0.7473815083503723

Processed all 12715 sentences
Epoch: 3 Loss: 0.7450541406869888

Processed all 12715 sentences
Epoch: 4 Loss: 0.7424442023038864

Processed all 12715 sentences
Epoch: 5 Loss: 0.7394593507051468
```

5. Getting the word embeddings [1 marks].
To get the credit, you must include in your report the output of the instruction printing the DataFrame (summarized)

Answer:

```
word_df=DataFrame(word_embeddings, index=idx2word.values())
word_df
```

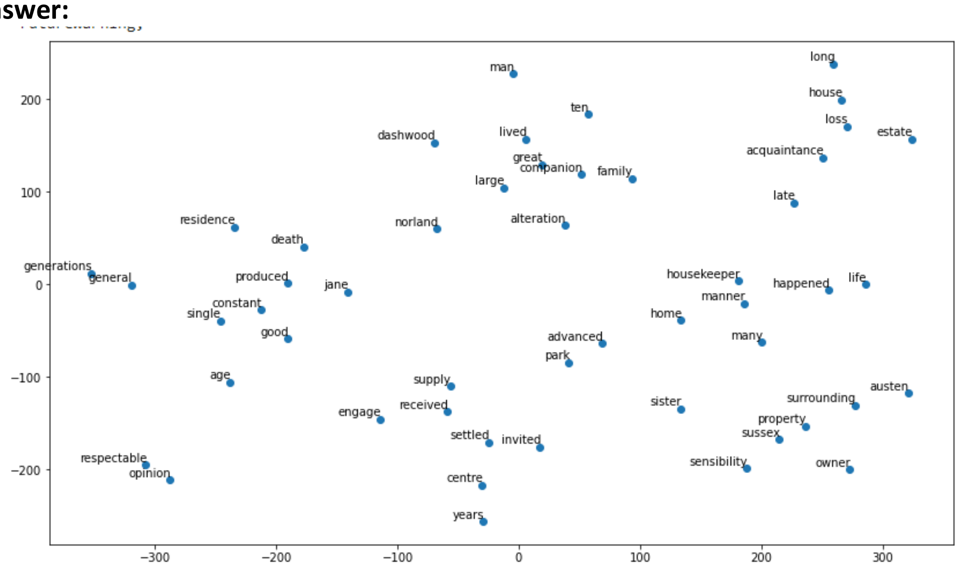
	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96
sense	0.011421	-0.014969	0.010833	-0.014109	-0.011458	-0.022278	-0.016098	-0.004706	0.022191	0.009888	...	0.015120	0.007817	-0.000025	-0.007100	0.020506	-0.002781	-0.020601
sensibility	0.026893	0.019133	-0.010789	0.005050	-0.037341	-0.020763	-0.021511	-0.010876	-0.021492	-0.021791	...	0.026239	-0.029278	0.019793	0.016487	-0.002518	-0.028056	0.03072
jane	-0.025423	0.016799	-0.006891	-0.036728	-0.027626	-0.021483	-0.011140	0.029163	0.000469	0.011607	...	-0.027268	-0.025377	-0.013829	0.019149	-0.034874	0.021220	-0.03112
austen	-0.005710	-0.023441	-0.006604	-0.010186	-0.019812	-0.000386	0.023650	0.016155	0.026603	-0.025344	...	0.004194	-0.003936	-0.031051	0.029627	-0.026625	0.009824	0.026641
family	-0.020713	-0.020265	0.009221	0.012659	0.001446	-0.009731	0.023887	0.004627	-0.007519	-0.021781	...	-0.020493	0.022704	0.011963	-0.002346	-0.020900	0.022921	0.00152
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
depriving	0.005735	-0.000271	0.016626	0.015267	-0.003449	-0.010014	0.010958	-0.015334	0.000005	0.021348	...	0.016769	0.017322	-0.022789	0.007582	-0.014869	-0.006077	0.01594
requited	0.000546	0.011585	0.009229	-0.009010	0.002357	0.006917	-0.009854	-0.000806	-0.008544	-0.006800	...	0.019124	0.009250	-0.019691	0.008051	0.021526	-0.004634	0.00164
supplies	0.023791	0.000309	-0.007642	0.006409	-0.006066	-0.003926	0.013844	0.009408	0.021634	-0.004767	...	-0.020727	-0.021149	-0.004513	-0.019734	-0.020098	0.003801	0.020931
accessions	-0.015418	0.002031	-0.019518	-0.021644	0.011509	-0.015024	-0.011472	0.015115	-0.022306	0.006589	...	-0.012950	-0.015341	-0.002836	-0.020590	0.012014	-0.010508	-0.01868
national	-0.003307	0.006839	0.023635	0.016531	-0.021248	0.015303	-0.018532	-0.007529	-0.013235	-0.012244	...	0.012245	0.009625	0.017901	-0.016998	0.015799	-0.017933	0.01386

10285 rows × 100 columns

6. Exploring and visualizing your word embeddings using t-SNE [2 marks].

Include in the report the output of the plt.annotate command

Answer:



PART-B

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

For this part, show the output you obtain from the sanity check.

Answer:

```
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])
```

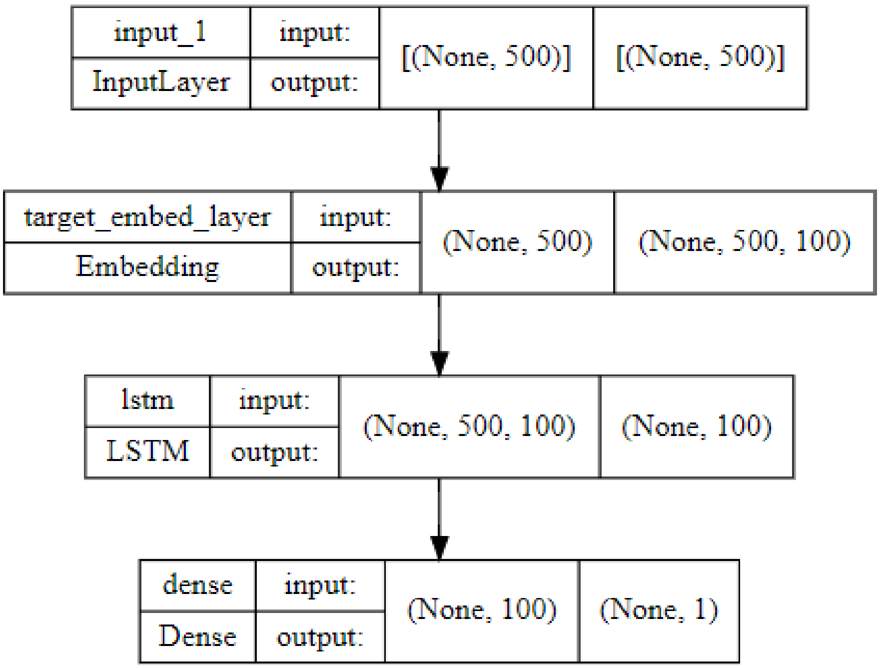
```
Length of sample train_data before preprocessing: 218
Length of sample train_data after preprocessing: 500
Sample train data: [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    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    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
    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 (section 3 of the script): [4 marks].

For this part, show the structure of the model you obtain.

Answer:

```
model.summary()

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

 target_embed_layer (Embeddi  (None, 500, 100)         1000000
 ng)

 lstm (LSTM)                 (None, 100)               80400

 dense (Dense)               (None, 1)                 101

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



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

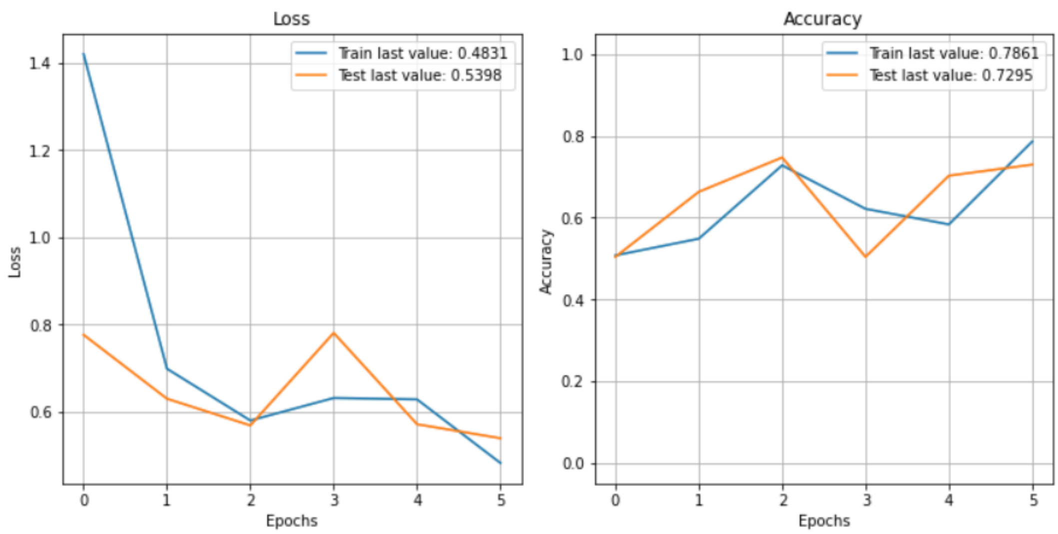
For this part, show the plot of training and validation accuracy through the epochs and comment on the optimal stopping point for the model.

Answer:

Optimal stopping point:

It can be observed from the below graphs loss had reached to its minimum around 0.6 and accuracy to 0.7 in epoch 2 itself. In further epoch loss has increased and accuracy have decreased for few more epochs indicating our model has over fit. It was better to stop training the model after epoch 2 for stable model.

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



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

For this part, show the output of the command print int test loss and test accuracy.

Answer:

```
[21] # 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 [==============================] - 118s 147ms/step - loss: 0.5506 - accuracy: 0.7330
test_loss: 0.5506224036216736 test_accuracy: 0.7329599857330322
```

5. Section 6, extracting the word embeddings [1 marks].

For this part, show the output of model.summary()

Answer:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 500)]             0         
                                                                 
 target_embed_layer (Embeddi  (None, 500, 100)         1000000   
 ng)                                                             
                                                                 
 lstm (LSTM)                 (None, 100)               80400     
                                                                 
 dense (Dense)               (None, 1)                 101       
                                                                 
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

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.

Answer:

```
idx2word
```

```
{1410: 'woods',
 2347: 'hanging',
 2291: 'woody',
 6748: 'arranged',
 2340: 'bringing',
 1638: 'wooden',
 4012: 'errors',
 3232: 'dialogs',
 361: 'kids',
 5036: 'uplifting',
 7095: 'controversy',
 9880: 'projection',
 7182: 'stern',
 5623: 'morally',
 5285: 'wang',
 180: 'want',
 2105: 'travel',
 6704: 'barbra',
 3932: 'dinosaurs',
 354: 'wrong',
 4762: 'subplots',
 9094: 'welcomed',
 6705: 'butcher',
 1182: 'fit',
 1929: 'screaming',
 4289: 'fix',
```

```
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 c
```

Full output:

```
<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 [1 marks].

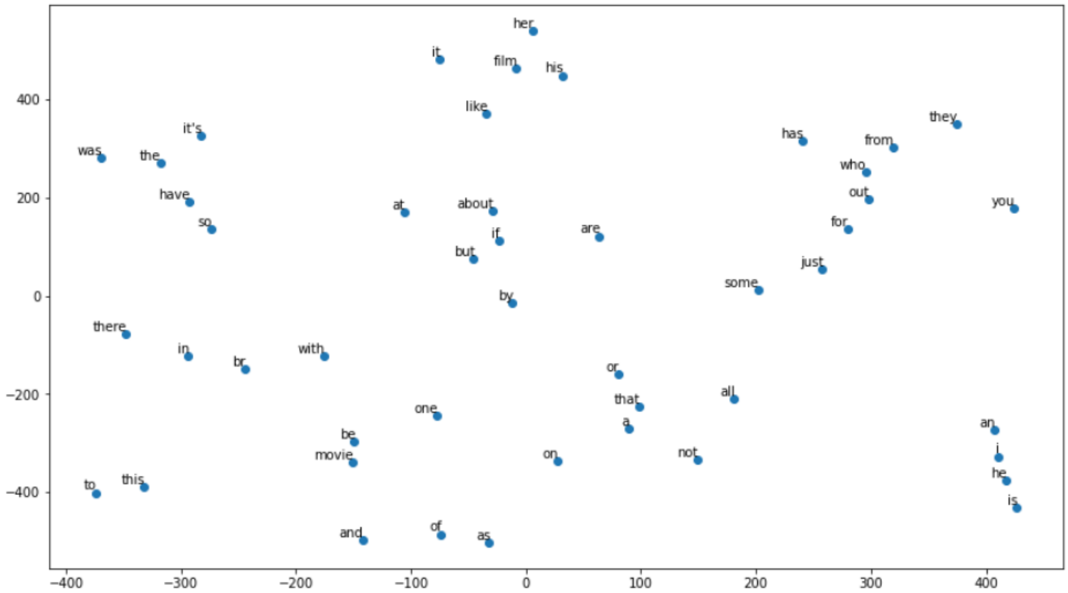
For this part, you should include the word embeddings for 10 of the words.

Answer:

```
full_df.head(10)
```

	word	0	1	2	3	4	5	6	7	8	...	90	91	92	93	94	95	96	97	98	99
0	woods	-0.019047	-0.002726	-0.023506	-0.012242	0.018512	-0.004842	-0.001334	0.023618	0.018694	...	0.020619	-0.021203	-0.003388	-0.006076	0.010034	0.020930	0.017838	0.012095	-0.010440	-0.001656
1	hanging	-0.023455	0.022903	0.015579	0.024484	-0.028078	0.004645	-0.011594	-0.006102	0.021463	...	0.007411	0.013870	0.004521	0.007414	0.005756	-0.004162	0.008085	-0.008218	0.002497	0.016609
2	woody	-0.018744	-0.013389	0.015424	-0.009389	-0.007300	-0.013153	-0.012117	-0.001019	-0.006265	...	-0.002932	0.003509	0.008756	-0.006475	0.009486	-0.001705	0.000385	0.016800	-0.004257	0.018372
3	arranged	-0.002259	-0.007126	0.016470	0.011843	0.010162	-0.006637	0.004898	-0.003348	-0.015230	...	0.014097	0.013031	0.014674	-0.004024	-0.024447	-0.004249	0.010542	0.008678	-0.008957	0.019900
4	bringing	-0.026006	0.005137	0.020172	0.032875	-0.012804	-0.013644	0.010825	-0.006213	0.003700	...	-0.002371	-0.005895	-0.024197	0.003256	-0.014065	-0.016816	-0.014207	-0.015323	0.000157	-0.012180
5	wooden	0.010720	-0.012827	0.015667	0.000669	-0.003770	-0.004345	-0.025865	-0.008908	0.027886	...	0.010232	-0.007770	0.007727	0.022760	0.003314	-0.024586	0.002035	-0.013008	0.018921	0.010646
6	errors	-0.005711	0.016372	-0.008042	0.017728	-0.017253	-0.007973	0.015858	0.012899	-0.005173	...	0.001101	-0.026779	-0.014976	0.002464	0.015370	-0.019347	-0.004184	-0.006851	0.006859	-0.008376
7	dialogs	-0.020183	0.007457	-0.019386	0.032144	0.023949	0.006002	0.016384	-0.008352	0.029656	...	-0.014323	0.009991	0.002567	0.031275	-0.003118	-0.010924	-0.006789	0.003200	0.011572	0.021074
8	kids	0.008666	-0.017093	-0.005807	0.004040	-0.011783	0.000798	-0.025670	-0.022568	-0.002256	...	0.003039	-0.024170	-0.000361	-0.015800	0.016455	0.003849	-0.005937	0.017247	0.013366	0.027088
9	uplifting	-0.018160	-0.007608	-0.011009	0.006236	0.016880	-0.011226	0.018973	-0.013647	-0.011022	...	0.018039	-0.001280	-0.001686	-0.000118	-0.024261	0.005970	-0.007267	-0.001252	-0.013261	0.007336

10 rows × 101 columns



8. Section 9 [2 marks].

For this paper, you have to write down your answers to the questions. 2 points each for questions 1 and 2.

Answer:

Dropout: Dropout is a regularization technique that improves model performance by avoiding the model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.

Observation:

It can be observed that accuracy has increased for the model where we have used dropout.

Batch size 1.
If batch size is 1 it is stochastic gradient descent. Meaning model will trained on batch each containing 1 data sample. It will converge faster but requires extremely high computational time which is not feasible especially on larger dataset.

Batch size as 32:
Also we can say as mini batch stochastic gradient descent. It will converge slower than batch size 1 but will be faster than stochastic gradient descent

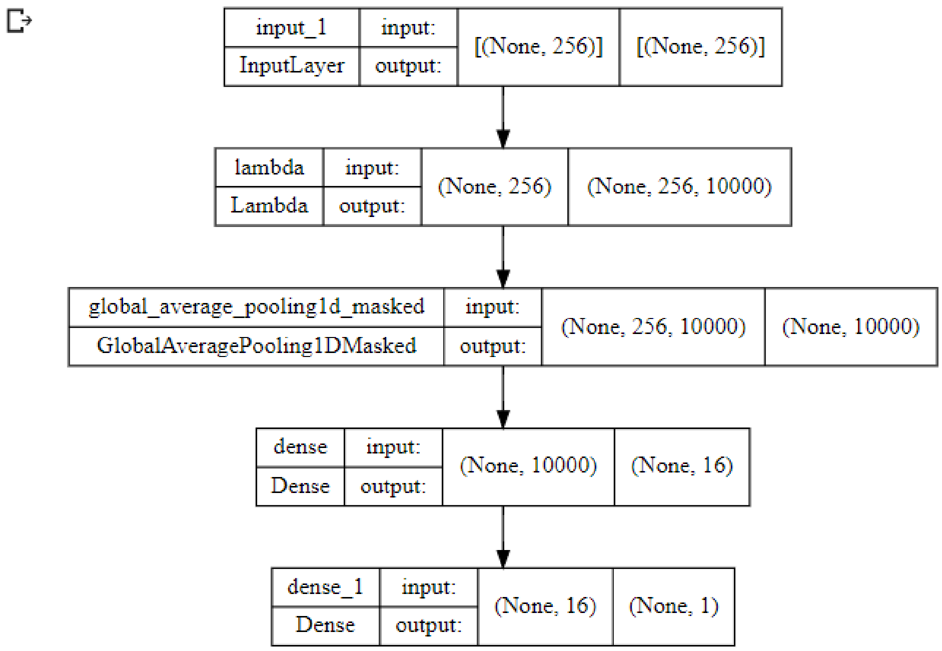
Batch size is len(data):
We can also name it as Batch Gradient Descent. We will be considering all dataset for every step therefore it is not at all sufficient.

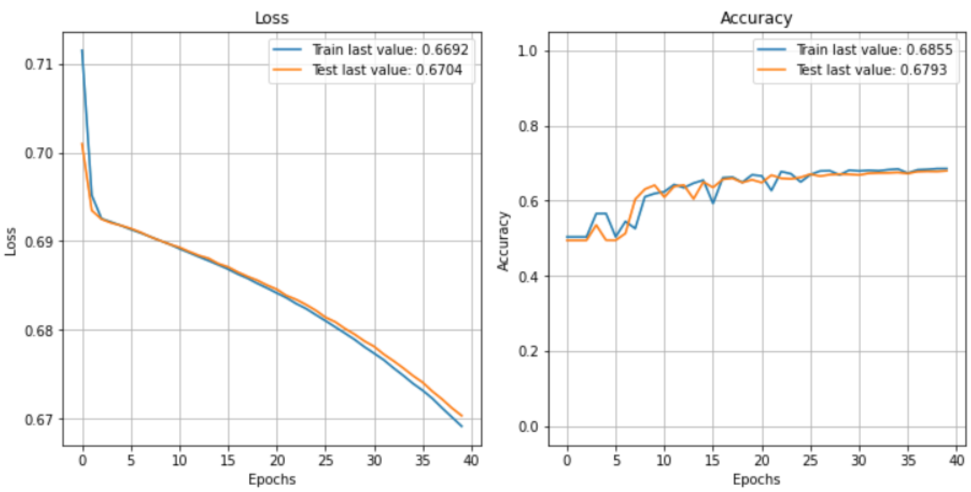
PART-C

1.Build a neural network classifier using one-hot word vectors (Model 1), and train and evaluate it [5 marks].
In your report, include the output of the model.summary() command to show your model structure, and plot training and validation loss in a graph.

Answer:

```
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
_________________________________________________________________
```





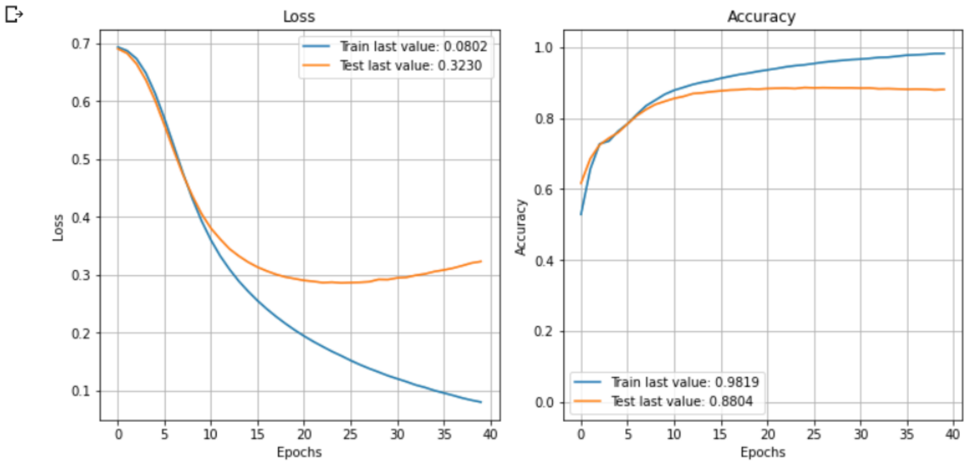
Training and validation loss are almost similar indicating this model is not over fit. But the overall accuracy for this model is not great it is just around 68%.

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 [5 marks].

Again, include the output of model.summary(), and plot training and validation loss.

Answer:

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 256)]             0         
                                                                 
 target_embed_layer (Embeddi  (None, 256, 100)         1000000   
 ng)                                                             
                                                                 
 global_average_pooling1d_ma  (None, 100)              0         
 sked_1 (GlobalAveragePoolin                                     
 g1DMasked)                                                      
                                                                 
 dense_2 (Dense)             (None, 16)                1616      
                                                                 
 dense_3 (Dense)             (None, 1)                 17        
                                                                 
=================================================================
Total params: 1,001,633
Trainable params: 1,001,633
Non-trainable params: 0
_________________________________________________________________
```



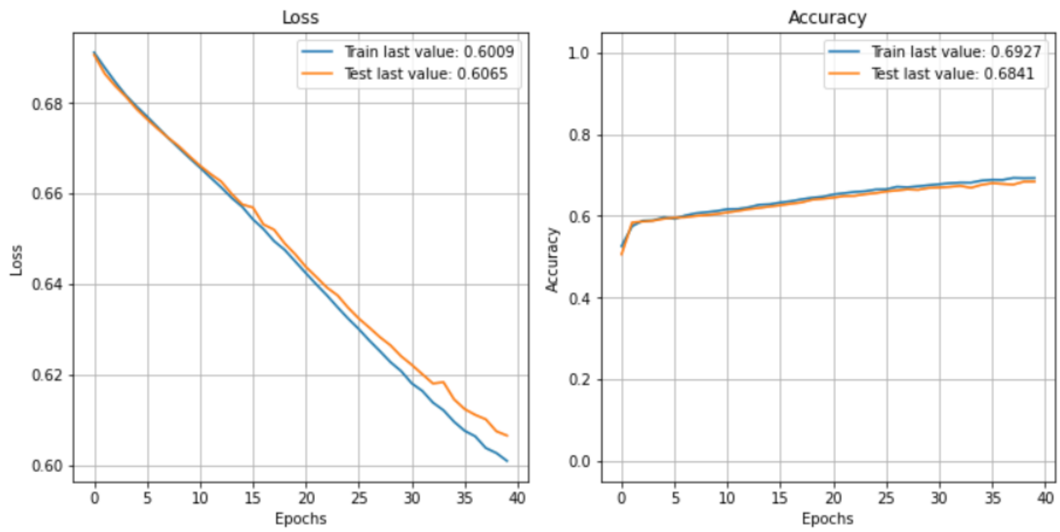
3. Adapt your model to load and use pre-trained word embeddings instead (Model 3); train and evaluate it and compare the effect of freezing and fine-tuning the embeddings [5 marks].

Again, include the output of model.summary(), and plot training and validation loss.

Answer:

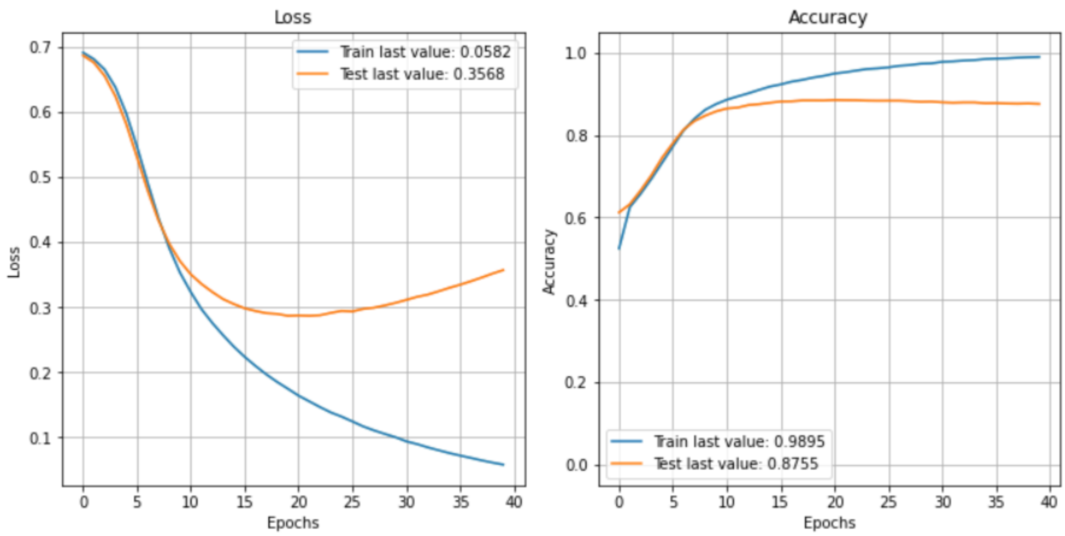
Model summary with freezing weights

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (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
_________________________________________________________________
```



Fine-tuning:

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256)]             0

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

 global_average_pooling1d_ma  (None, 300)              0
 sked_3 (GlobalAveragePoolin
 g1DMasked)

 dense_6 (Dense)             (None, 16)                4816

 dense_7 (Dense)             (None, 1)                 17

=================================================================
Total params: 120,005,133
Trainable params: 120,005,133
Non-trainable params: 0
_________________________________________________________________
```



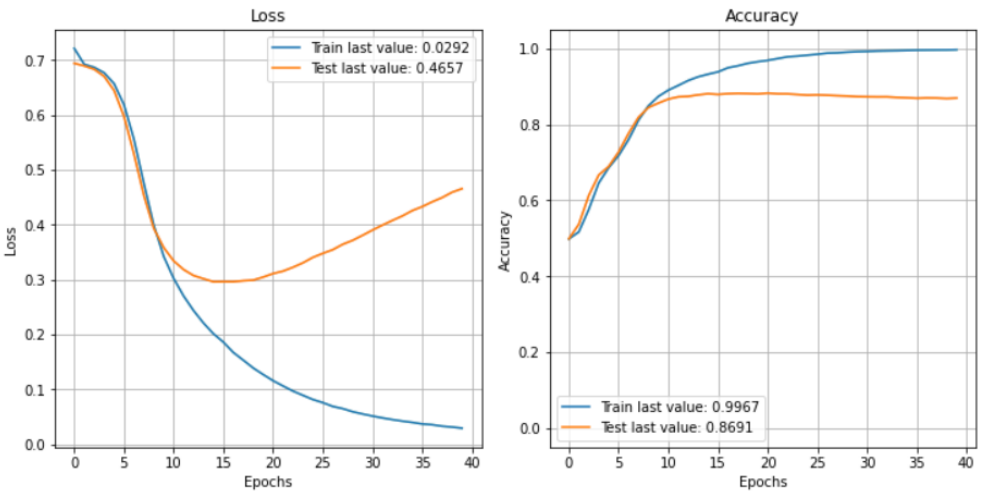
4. One way to improve the performance is to add another fully-connected layer to your network. Try this (Model 4) and see if it improves the performance. If not, what can you do to improve it? [5 marks]

Plot the training and validation loss of the new model, and other models you try. In your report, describe the differences you see and discuss why they occur.

Answer:

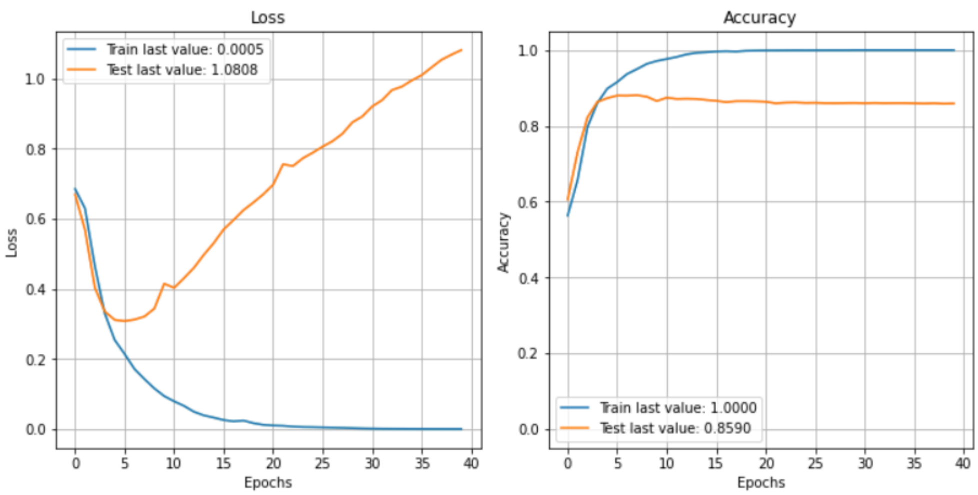
Adding one dense layer:

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (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: 120,032,033
Non-trainable params: 0
_________________________________________________________________
```



Adding two dense layer:

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 256)]             0         
                                                                 
 GloVe_Embeddings (Embedding  (None, 256, 300)         120000300 
 )                                                               
                                                                 
 global_average_pooling1d_ma  (None, 300)              0         
 sked_5 (GlobalAveragePoolin                                     
 g1DMasked)                                                      
                                                                 
 dense_12 (Dense)            (None, 300)               90300     
                                                                 
 dense_13 (Dense)            (None, 100)               30100     
                                                                 
 dense_14 (Dense)            (None, 16)                1616      
                                                                 
 dense_15 (Dense)            (None, 1)                 17        
                                                                 
=================================================================
Total params: 120,122,333
Trainable params: 120,122,333
Non-trainable params: 0
_________________________________________________________________
```



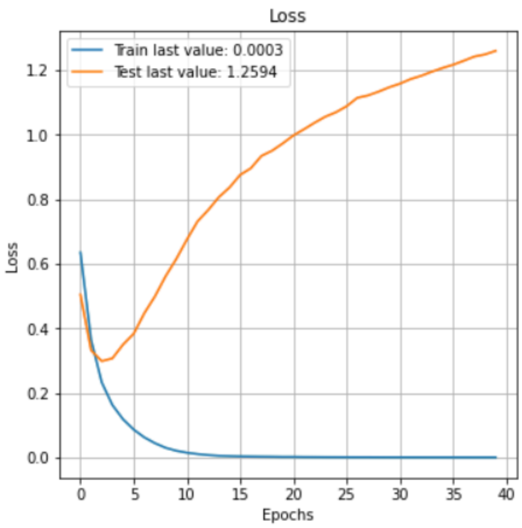
In model 3-1 (Neural network model) we get the accuracy of 0.67 with loss of 0.6073. Adding one extra dense layer give pretty good accuracy of 0.8678 with loss 0.4949 but adding one more additional dense layer gave me very slight decrease in accuracy 0f 0.8438 and loss of 1.1788. Sometimes running notebook from top gives little higher accuracy in extra two dense layer.

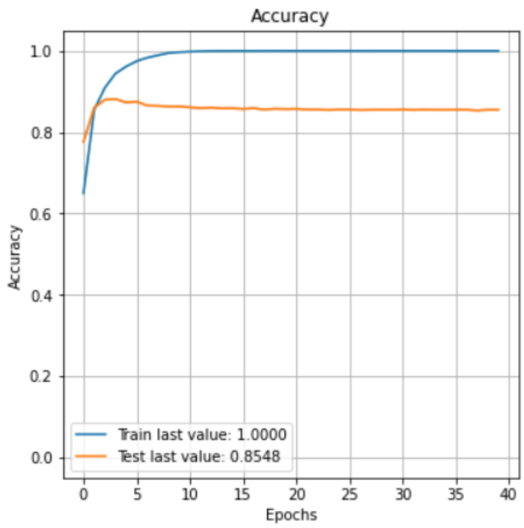
5. Build a CNN classifier (Model 5), and train and evaluate it. Then try adding extra convolutional layers, and conduct training and evaluation. [5 marks].

Again, include the output of model.summary(), plot training and validation loss, describe the differences you see and discuss why they occur

Answer:

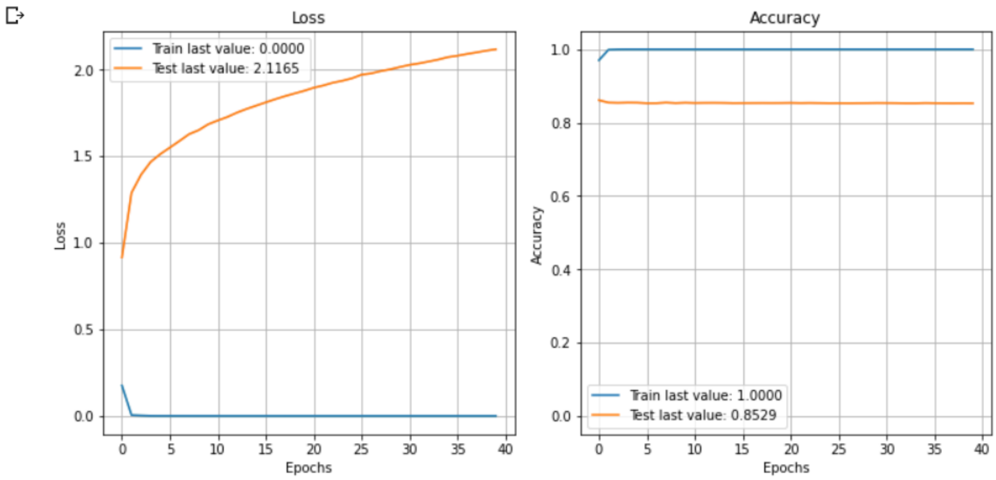
```
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256)]             0

 target_embed_layer (Embeddi  (None, 256, 300)         3000000
 ng)

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

 global_average_pooling1d_ma  (None, 100)              0
 sked_6 (GlobalAveragePoolin
 g1DMasked)

 dense_16 (Dense)            (None, 1)                 101

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





Adding extra CNN layer:

```
Model: "model_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256)]             0

 target_embed_layer (Embeddi  (None, 256, 300)         3000000
 ng)

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

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

 global_average_pooling1d_ma  (None, 100)              0
 sked_7 (GlobalAveragePoolin
 g1DMasked)

 dense_17 (Dense)            (None, 1)                 101

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



Although adding layer here reduces the training loss, the evaluation accuracy is worse than the model without the extra convolutional layer.

Adding more layers can help to extract more features. But we can do that up to a certain extent. After some point, instead of extracting features, we tend to over fit the data. Overfitting can lead to errors in on form or another, such as false positives. It is not easy to choose the number of units in a hidden layer or the number of hidden layers in a neural network.

PART-D

1. Preprocess the data, to adapt the models from Parts C [4 marks].

Answer:

a. Add <PAD>, <START>, <UNK>, <EOS> to start in our word_index. word_index here represents dictionary where each unique word in train has been assigned a unique number.

```
voc = []
from keras.preprocessing.text import text_to_word_sequence
for example in train:
  text_tokens = text_to_word_sequence(example[0])
  aspect_tokens = text_to_word_sequence(example[1])
  voc.extend(aspect_tokens)
  voc.extend(text_tokens)
voc = set(voc)
print(len(voc))

word_index = dict()
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2
word_index["<EOS>"] = 3
for w in voc:
  word_index[w] = len(word_index)
print(len(word_index))
```

```
7894
7898
```

b. Extract:
x_train_review: contain review data from train
x_train_aspect: contain aspect data from train
x_train_review_int: In this each word presented in x_train_review has been replaced with its unique integer value from word_index.
x_train_aspect_int: In this each word presented in x_train_aspect has been replaced with its unique integer value from word_index.

Do similar procedure for validation and testing data. If words are new in testing and validation data which is not present in word_index dictionary then that unknown word is replaced by <UNK> and <UNK> dictionary value is assigned.

```
[10] # If use the previous word_index, you can get a print result like:
     assert len(x_train_aspect) == len(train)
     assert len(x_train_aspect) == len(x_train_aspect_int)
     assert len(x_test_aspect) == len(test)
     assert len(x_test_aspect) == len(x_test_aspect_int)
     print("x_train_review[0]:")
     print(x_train_review[0])
     print("x_train_aspect[0]:")
     print(x_train_aspect[0])
     print("x_train_review_int[0]:")
     print(x_train_review_int[0])
     print("x_train_aspect_int[0]:")
     print(x_train_aspect_int[0])
```

```
x_train_review[0]:
['the', 'decor', 'is', 'not', 'special', 'at', 'all', 'but', 'their', 'food', 'and', 'amazing', 'prices', 'make', 'up', 'for', 'it']
x_train_aspect[0]:
['decor']
x_train_review_int[0]:
[4766, 1533, 7783, 1468, 1161, 7607, 226, 6464, 7883, 2839, 58, 4070, 418, 6191, 5215, 902, 7766]
x_train_aspect_int[0]:
[1533]
```

c. Now initialize <START> to all review list of train, test and validation.
Making it [‘<START>’, “the”, “décor”, “is”>]
Later concatenate it’s respective aspect with its review data like:
[“décor”, ‘<START>’, “the”, “décor”, “is”>] for all train, test and validation.
Now again replace each word with its unique integer in the list. Making it:
[7226, 1, 7769, 6837]
Similarly do it for test and validation data too.

d. Now to keep the length of all sentence equal pad it.

```
[16] # Don't forget the to use np.array function to wrap the ouput of pad_sequences function, such as: x_train_pad = np.array(x_train_pad)
     # Only pad the *_int varibles
     print("Before paded:")
     print(x_train[0])
     print(x_train_int[0])
     print("After paded:")
     print(x_train_pad[0])
```

```
Before paded:
['decor', '<START>', 'the', 'decor', 'is', 'not', 'special', 'at', 'all', 'but', 'their', 'food', 'and', 'amazing', 'prices', 'make', 'up', 'for', 'it']
[1533, 1, 4766, 1533, 7783, 1468, 1161, 7607, 226, 6464, 7883, 2839, 58, 4070, 418, 6191, 5215, 902, 7766]
After paded:
[1533    1 4766 1533 7783 1468 1161 7607  226 6464 7883 2839   58 4070
  418 6191 5215  902 7766    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]
```

e. Apply the above approach for GLOVE embedding:

```
[41] assert len(x_train_review_glove) == len(train)
     assert len(x_train_aspect_glove) == len(x_train_aspect_int)
     assert len(x_test_review_glove) == len(test)
     assert len(x_test_aspect_glove) == len(x_test_aspect_int)
     print("x_train_review_glove[0]:")
     print(x_train_review_glove[0])
     print("x_train_aspect_glove[0]:")
     print(x_train_aspect_glove[0])
```

```
x_train_review_glove[0]:
['the', 'decor', 'is', 'not', 'special', 'at', 'all', 'but', 'their', 'food', 'and', 'amazing', 'prices', 'make', 'up', 'for', 'it']
x_train_aspect_glove[0]:
['decor']
```

```
[43] assert len(x_train_review_glove) == len(train)
     assert len(x_train_aspect_glove) == len(x_train_aspect_int)
     assert len(x_test_review_glove) == len(test)
     assert len(x_test_aspect_glove) == len(x_test_aspect_int)
     print("x_train_review_glove[0]:")
     print(x_train_review_glove[0])
     print("x_train_aspect_glove[0]:")
     print(x_train_aspect_glove[0])
```

```
x_train_review_glove[0]:
[357266, 118926, 192973, 264550, 338995, 62065, 51582, 87775, 357354, 151204, 54718, 53201, 292136, 231458, 373317, 151349, 193716]
x_train_aspect_glove[0]:
[118926]
```

```
Before paded:
[118926, 1, 357266, 118926, 192973, 264550, 338995, 62065, 51582, 87775, 357354, 151204, 54718, 53201, 292136, 231458, 373317, 151349, 193716]
After paded:
[118926      1 357266 118926 192973 264550 338995  62065  51582  87775
 357354 151204  54718  53201 292136 231458 373317 151349 193716      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]
```

2. Adapt your models without pre-trained word embeddings in Part C to this task (Model 1); train and evaluate it [4 marks].

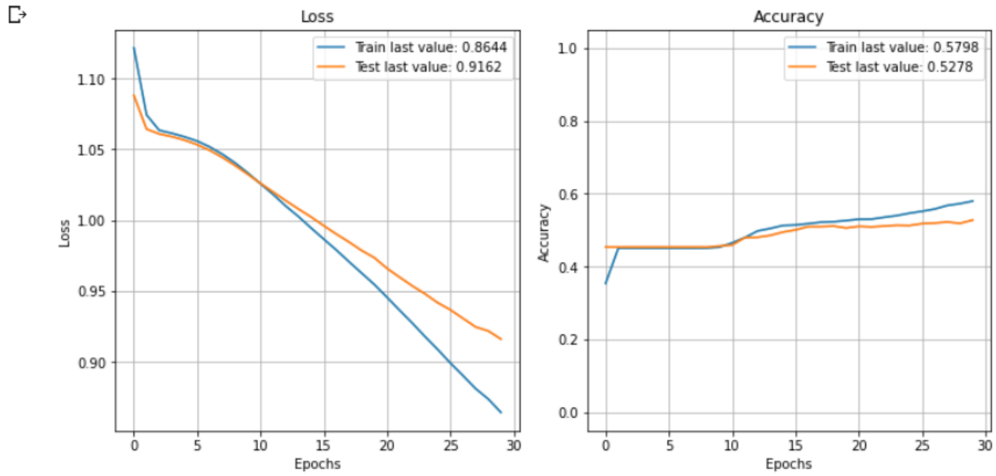
In your report (a jupyter notebook), include the output of the model.summary() command to show your model structure, and training epochs, and evaluation results.

Answer:

2.1 Neural bag of words without pre-trained word embeddings

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

 target_embed_layer (Embeddi  (None, 128, 100)         789800
 ng)

 global_average_pooling1d_ma  (None, 100)              0
 sked (GlobalAveragePooling1
 DMasked)

 dense (Dense)               (None, 16)                1616

 dense_1 (Dense)             (None, 3)                 51

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

```
results = model2.evaluate(x_test_pad,y_test)

42/42 [==============================] - 0s 2ms/step - loss: 0.9024 - accuracy: 0.5524
```



3. Adapt your models with pre-trained word embeddings in Part C to this task (Model 2); train and evaluate it [6 marks].

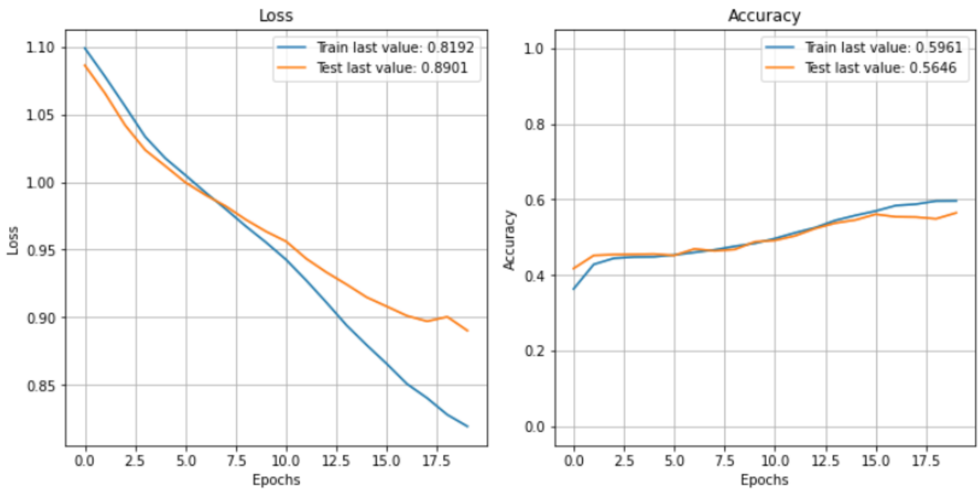
In your report, include the output of the model.summary() command to show your model structure, and training epochs, and evaluation results. Describe the differences pre-training makes, and explain why they happen.

Answer:

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128)]             0

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

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

 dense_6 (Dense)             (None, 16)                4816

 dense_7 (Dense)             (None, 3)                 51

=================================================================
Total params: 120,005,167
Trainable params: 120,005,167
Non-trainable params: 0
_________________________________________________________________
```

```
[58] results=model3.evaluate(x_test_pad_glove,y_test)

     42/42 [==============================] - 0s 2ms/step - loss: 0.8847 - accuracy: 0.5808
```

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



There is accuracy increase while we use GLOVE embedding as GLOVE embedding is usually trained on large dataset therefore it is able to capture semantic and syntactic meaning in an efficient way.

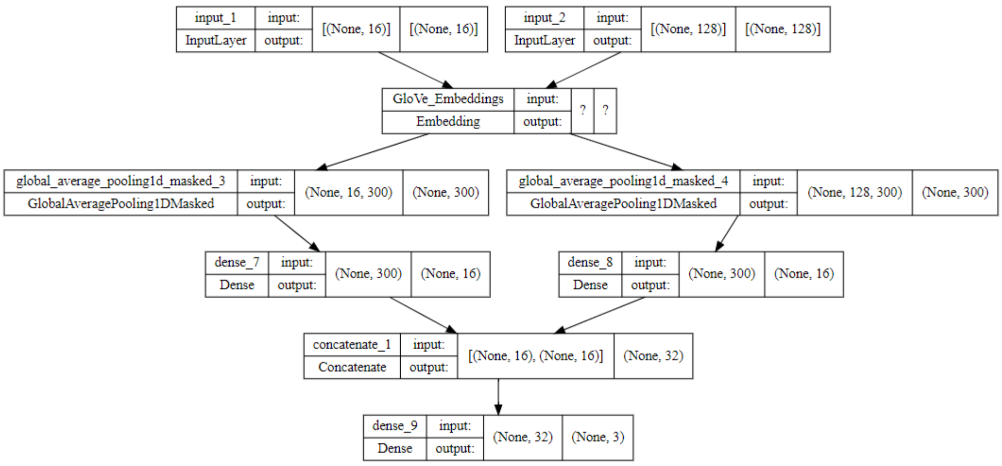
4. Build and evaluate two more classifiers with multiple input (Model 3) separate inputs for text and aspect) [7 marks].

In your report, include the output of the model.summary() command to show your model structure, and training epochs, and evaluation results. Describe the differences in performance and discuss why they occur.

Answer:

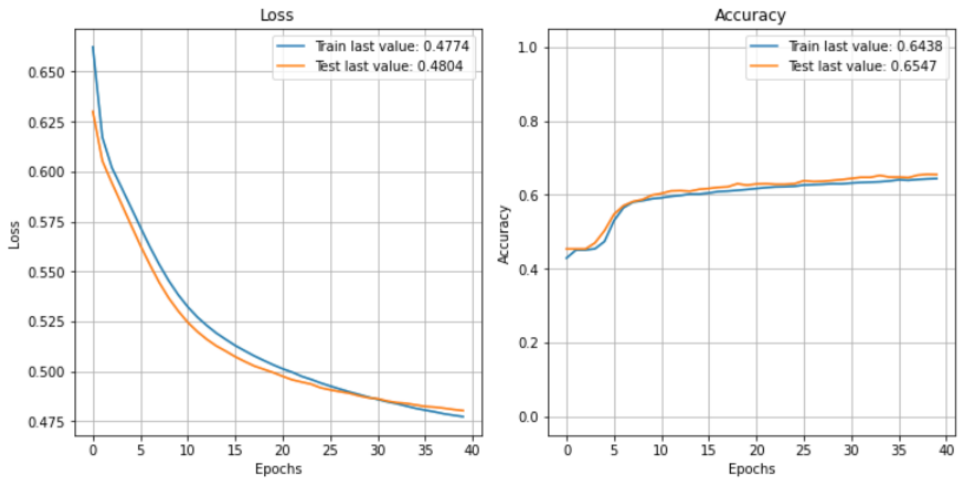
Model 1:

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 16)]         0           []

 input_2 (InputLayer)           [(None, 128)]        0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_1[0][0]',
                                                                  'input_2[0][0]']

 global_average_pooling1d_maske  (None, 300)         0           ['GloVe_Embeddings[0][0]']
 d_3 (GlobalAveragePooling1DMas
 ked)

 global_average_pooling1d_maske  (None, 300)         0           ['GloVe_Embeddings[1][0]']
 d_4 (GlobalAveragePooling1DMas
 ked)

 dense_7 (Dense)                (None, 16)           4816        ['global_average_pooling1d_masked
                                                                 _3[0][0]']

 dense_8 (Dense)                (None, 16)           4816        ['global_average_pooling1d_masked
                                                                 _4[0][0]']

 concatenate_1 (Concatenate)    (None, 32)           0           ['dense_7[0][0]',
                                                                  'dense_8[0][0]']

 dense_9 (Dense)                (None, 3)            99          ['concatenate_1[0][0]']

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



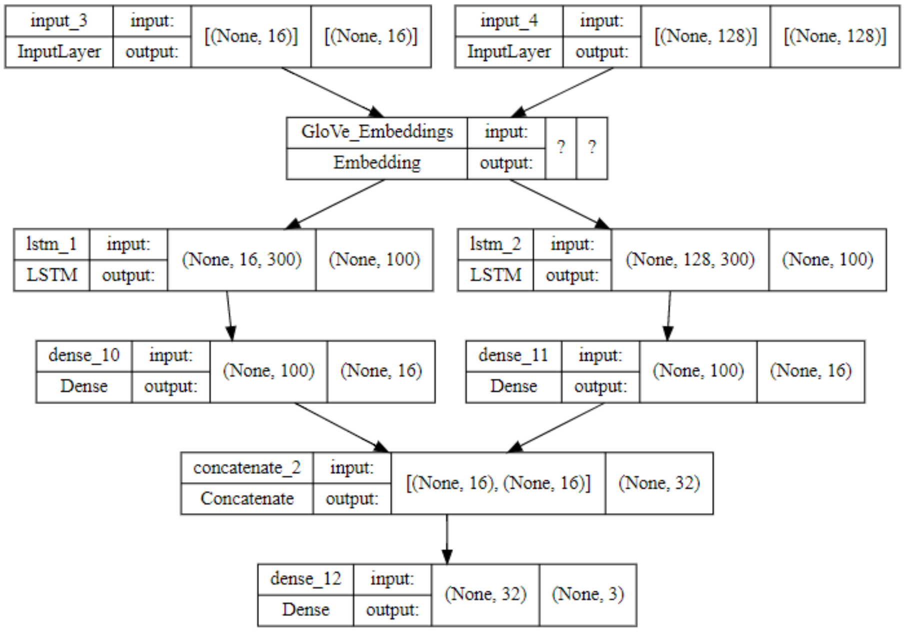
```
[64] results = model4.evaluate([x_test_aspect_pad_glove,x_test_review_pad_glove],y_test)
     print(results)

42/42 [==============================] - 0s 2ms/step - loss: 0.4736 - accuracy: 0.6609
[0.4736023545265198, 0.6609281301498413]
```

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



Model 2:

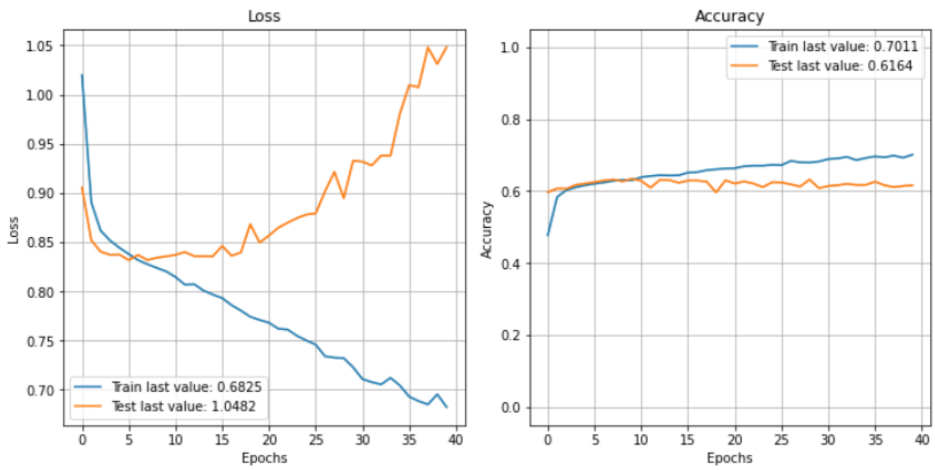
```
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 16)]         0           []

 input_4 (InputLayer)           [(None, 128)]        0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_3[0][0]',
                                                                  'input_4[0][0]']

 lstm_1 (LSTM)                  (None, 100)          160400      ['GloVe_Embeddings[0][0]']

 lstm_2 (LSTM)                  (None, 100)          160400      ['GloVe_Embeddings[1][0]']

 dense_10 (Dense)               (None, 16)           1616        ['lstm_1[0][0]']

 dense_11 (Dense)               (None, 16)           1616        ['lstm_2[0][0]']

 concatenate_2 (Concatenate)    (None, 32)           0           ['dense_10[0][0]',
                                                                  'dense_11[0][0]']

 dense_12 (Dense)               (None, 3)            99          ['concatenate_2[0][0]']

==================================================================================================
Total params: 120,324,431
Trainable params: 324,131
Non-trainable params: 120,000,300
__________________________________________________________________________________________________
```




```
results = model5.evaluate([x_test_aspect_pad_glove,x_test_review_pad_glove],y_test)
print(results)
```

```
42/42 [==============================] - 0s 7ms/step - loss: 1.0784 - accuracy: 0.6040
[1.0783694982528687, 0.6040419340133667]
```

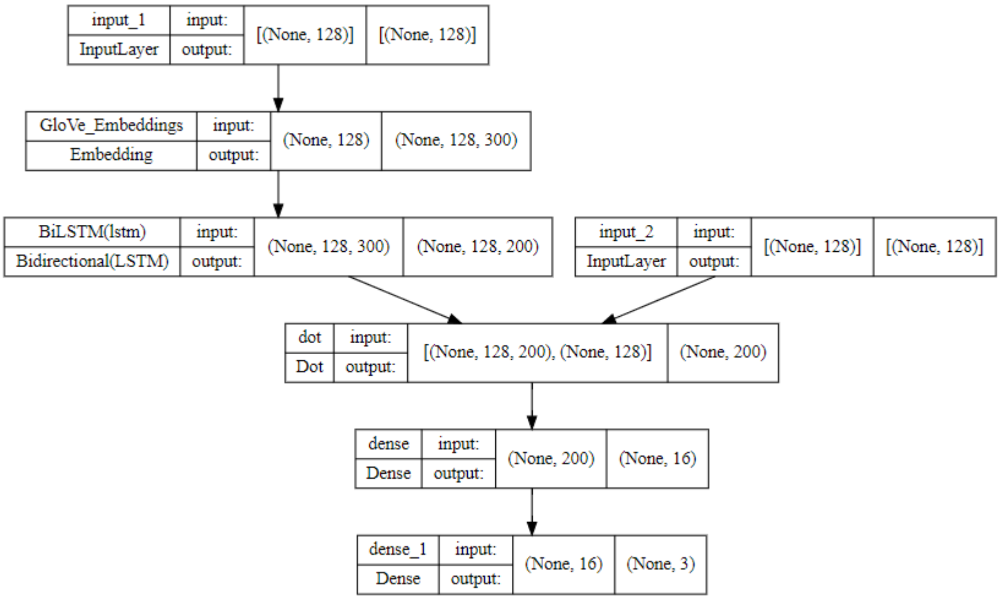
```
[77] plot_history(history.history, path="standard.png")
     plt.show()
```



5. Build and evaluate the classifier extracting information from LSTM (Model 4) [4 marks].
In your report, include the output of the model.summary() command to show your model structure, and training epochs, and evaluation results.

Answer:

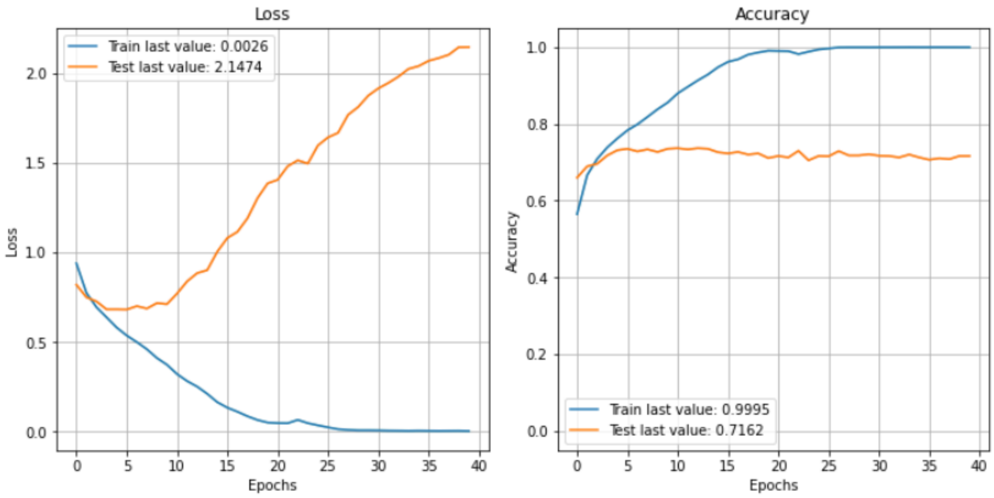
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1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

 GloVe_Embeddings (Embedding)   (None, 128, 300)     120000300   ['input_1[0][0]']

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

 input_2 (InputLayer)           [(None, 128)]        0           []

 dot (Dot)                      (None, 200)          0           ['BiLSTM[0][0]',
                                                                  'input_2[0][0]']

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

 dense_1 (Dense)                (None, 3)            51          ['dense[0][0]']

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



```
results = model6.evaluate([x_test_review_pad_glove,x_test_aspect_mask_pad],y_test)
print(results)
```

```
42/42 [==============================] - 0s 10ms/step - loss: 2.2029 - accuracy: 0.7096
[2.202911138534546, 0.7095808386802673]
```

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



Part-E

1. Task 1: Implementing the encoder [7 marks].

Your report must include the code and an explanation.

Answer:

The embedding source and embedding target layers are created to randomly initialise the embeddings for each words in the vocabulary (the embeddings will be trained during the training). The Embedding layers have a vocab size of input dim and an embedding size of output dim. mask_zero is set to True to remove paddings.

By sending the current inputs (source words and target words) through the Embedding layers, embeddings were created. Source and target word embeddings are referred to as source words embeddings and target words embeddings, respectively.
The Dropout layers are used to apply Dropout to the embeddings. embedding dropout rate is the dropout rate for the word embeddings.

Created an LSTM layer to process the source words embeddings, with return sequences set to True to get all tokens' outputs (call it encoder_outputs) and return state is set to True to retrieve the encoder LSTM's hidden state (encoder_state_h) and cell state (encoder_state_c).

```
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...')

'''The embedding source and embedding target layers are created to randomly initialise the embeddings for each words in the vocabulary (the embeddings will be
trained during the training).
The Embedding layers have a vocab size of input dim and an embedding size of output dim. mask_zero is set to True to remove paddings '''

''''''
embeddings_source = Embedding(input_dim=self.vocab_source_size, mask_zero=True,output_dim=self.embedding_size, name='source_embed_layer',
                              embeddings_initializer='glorot_uniform',
                              input_length=source_words.shape[1])(source_words)
embeddings_target = Embedding(input_dim=self.vocab_target_size, mask_zero=True,output_dim=self.embedding_size, name='target_embed_layer',
                              embeddings_initializer='glorot_uniform',
                              input_length=target_words.shape[1])(target_words)

# (b.) Look up the embeddings for source words and for target words. Apply dropout to each encoded input
'''By sending the current inputs (source words and target words) through the Embedding layers, embeddings were created.
  Source and target word embeddings are referred to as source words embeddings and target words embeddings, respectively.
  The Dropout layers are used to apply Dropout to the embeddings. embedding dropout rate is the dropout rate for the word embeddings.'''

print('\nTask 1(b): Looking up source and target words...')
source_word_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_source)

target_words_embeddings = Dropout(self.embedding_dropout_rate)(embeddings_target)

'''Created an LSTM layer to process the source words embeddings, with return sequences set to True to get all
tokens' outputs (call it encoder_outputs) and return state is set to True
to retrieve the encoder LSTM's hidden state (encoder_state_h) and cell state (encoder_state_c). '''

# (c.) An encoder LSTM() with return sequences set to True
print('\nTask 1(c): Creating an encoder')
encoder = LSTM(self.hidden_size, recurrent_dropout=self.hidden_dropout_rate,return_sequences=True,return_state=True, name = 'encoder')
encoder_outputs, encoder_state_h, encoder_state_c = encoder(source_word_embeddings)
"""
End Task 1
```

2. Task 2: Implementing the decoder [8 marks].

Again, you must include the code, an explanation, the BLEU score, and a sample of the output.

Answer:

Decoder_states list is created where decoder_model decoder_state_input_h and decoder_state_input_c are put together.

The target_word_embeddings and decoder_states are passed on to the decoder_lstm.

If statement for the attention model similar as the decoder for training.

Passed the output of the attention layer (for attention model) into the final layer of the decoder (decoder_dense) to assign probabilities of the next tokens.

```
Task 2 decoder for inference

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

'''The states passed to the decoder_model  decoder_state_input_h and decoder_state_input_c are put together to create a list decoder_states. '''

decoder_states = [decoder_state_input_h, decoder_state_input_c]

'''The target_word_embeddings and decoder_states are passed on to the decoder_lstm.'''
# 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)


'''If statement for the attention model similar as the decoder for training.'''
# 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])


'''Passed the output of the attention layer (for attention model) into the final layer of the decoder (decoder_dense) to assign probabilities of the next tokens.'''
# 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)
self.decoder_model = Model([target_words,decoder_state_input_h,decoder_state_input_c,encoder_outputs_input],
                           [decoder_outputs_test,decoder_state_output_h,decoder_state_output_c])

# you can now view the model summary
print('\t\t\t\t\t\t Decoder Inference Model summary')
print(self.decoder_model.summary())
"""
End Task 2

"""
```

Output:

```
Time used for epoch 10: 0 m 40 s
Evaluating on dev set after epoch 10/10:
The input sentence:  ['it', '&apos;s', 'a', '<unk>', '<unk>', ',', 'and', 'it', '&apos;s', 'a', '<unk>', '<unk>', ',', 'and', 'it', '&apos;s', '<unk>', '
The output sentence:  [['there', 'are', 'four', '<unk>', '<unk>', 'that', ',', 'each', 'time', 'this', 'ring', '<unk>', 'it', ',', 'as', 'it', '<unk>', '
source_words: [['có', '4', 'vi', 'điều', 'khiển', ',', 'để', 'mỗi', 'lần', 'vòng', 'này', 'quay', 'khi', 'nó', 'qua', 'phía', 'sau', 'hình', 'ảnh', ',',
Model BLEU score: 5.00
Time used for evaluate on dev set: 0 m 7 s
Training finished!
Time used for training: 7 m 12 s
Evaluating on test set:
The input sentence:  ['<unk>', '<unk>', ':', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>', '<unk>', '.']
The output sentence:  [['the', 'second', 'quote', 'is', 'from', 'the', 'head', 'of', 'the', 'u.k.', 'financial', 'services', '<unk>', '.']]
source_words: [['trích', 'dẫn', 'thứ', 'hai', 'đến', 'từ', 'người', 'đứng', 'đầu', 'cơ', 'quan', 'quản', 'lý', 'dịch', 'vụ', 'tài', 'chính', 'vương', 'qu
Model BLEU score: 5.50
Time used for evaluate on test set: 0 m 6 s
```

BLEU score → 5.50

3. Adding attention [10 marks].

Again, you must include the code, an explanation, the BLEU score, and a sample of the output.

Answer:

Shape of our inputs (encoder_outputs, decoder_outputs), the encoder_outputs has a shape of [batch_size, max_source_sent_len, hidden_size] and the decoder_outputs has a shape of [batch_size, max_target_sent_len, hidden_size].

Using permute_dimensions method to transpose the last two dimensions of the decoder_outputs to make it shape becomes [batch_size, hidden_size, max_target_sent_len].

Performed matrix multiplication of inputs encoder_outputs and decoder_outputs to generate the ou tput luong_score with shape of [batch_size, max_source_sent_len, max_target_sent_len].

A softmax is applied to the dimension that have a size of max_sourse_sent_len to create an attention score for the encoder_outputs.

Created the encoder_vector by doing element-wise multiplication between the encoder_outputs and their attention scores (luong_score). Since, the shape of the luong_score is actually not the same as the encoder_outputs, used expand_dims method to expand dimensions for both of them. Summed the max_source_sent_len dimension to create the encoder_vector.

```
Task 3 attention

Start
"""


'''Using permute_dimensions method to transpose the last two dimensions of the decoder_outputs to make it shape becomes [batch_size, hidden_size, max_target_sent_len]
.'''
decoder_outputs_transposed = K.permute_dimensions(decoder_outputs, (0, 2, 1))
print(f"decoder_outputs_transposed: {decoder_outputs_transposed.shape}")

'''Performed matrix multiplication of inputs encoder_outputs and decoder_outputs to generate the output luong_score with shape of [batch_size, max_source_sent_len,
max_target_sent_len].'''
# luong_score shape [batch_size, max_source_sent_len, max_target_sent_len]
luong_score = K.batch_dot(encoder_outputs, decoder_outputs_transposed, axes=[2,1])
print(f"luong_score(before softmax): {luong_score.shape}")

'''A softmax is applied to the dimension that have a size of max_sourse_sent_len to create an attention score for the encoder_outputs.'''
luong_score = K.softmax(luong_score,axis=1)

# luong_score shape [batch_size, max_source_sent_len, max_target_sent_len, 1]
# luong_scoreis one dimension after softmax, add one dimension so we can do matrix multiplication
luong_score = tf.expand_dims(luong_score, axis=3)
print(f"luong_score(add dimension): {luong_score.shape}")

#encoder_outputs shape [batch_size, max_source_sent_len, 1, hidden_size]
encoder_outputs = tf.expand_dims(encoder_outputs, axis=2)
print(f"encoder_outputs: {encoder_outputs.shape}")

encoder_vector =  luong_score * encoder_outputs
print(f"encoder_vector: {encoder_vector.shape}")
encoder_vector = K.sum(encoder_vector, axis=1)
print(f"encoder_vector: {encoder_vector.shape}")

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

Output:

```
Time used for evaluate on dev set: 0 m 7 s
Starting training epoch 10/10
240/240 [==============================] - 24s 101ms/step - loss: 0.9115 - accuracy: 0.5516
Time used for epoch 10: 0 m 40 s
Evaluating on dev set after epoch 10/10:
The input sentence:  ['there', 'are', 'four', '<unk>', ',', 'to', 'each', 'of', 'these', '<unk>', 'and', '<unk>', ',', 'it', 'turned', 'out', 'to', 'the', 'memory',
The output sentence:  [['there', 'are', 'four', '<unk>', '<unk>', 'that', ',', 'each', 'time', 'this', 'ring', '<unk>', 'it', ',', 'as', 'it', '<unk>', 'the', '<unk
source_words: [['có', '4', 'vi', 'điều', 'khiển', ',', 'để', 'mỗi', 'lần', 'vòng', 'này', 'quay', 'khi', 'nó', 'qua', 'phía', 'sau', 'hình', 'ảnh', ',', 'nó', 'ghi'
Model BLEU score: 15.07
Time used for evaluate on dev set: 0 m 7 s
Training finished!
Time used for training: 5 m 53 s
Evaluating on test set:
The input sentence:  ['the', 'second', 'signal', 'came', 'from', 'the', '<unk>', 'virus', '<unk>', 'the', '<unk>', 'virus', '<unk>', '.']
The output sentence:  [['the', 'second', 'quote', 'is', 'from', 'the', 'head', 'of', 'the', 'u.k.', 'financial', 'services', '<unk>', '.']]
source_words: [['trích', 'dẫn', 'thứ', 'hai', 'đến', 'từ', 'người', 'đứng', 'đầu', 'cơ', 'quan', 'quản', 'lý', 'dịch', 'vụ', 'tài', 'chính', 'vương', 'quốc', 'anh',
Model BLEU score: 15.18
Time used for evaluate on test set: 0 m 7 s
```

BLEU score -> 15.18