ASSIGNMENT 1D:

Real Text Classification Task.

NEURAL NETWORKS AND NATURAL LANGUAGE PROCESSING.

Carolina Raymond Rodrigues.
Student ID: 210607000
ec21209@qmul.ac.uk

Contents

List of Figures

List of Tables

1 Introduction

Sentiment analysis is a common NLP task that analyses affective states and subjective information from a given text. In this assignment, we will be applying different deep learning algorithms in order to classify a given aspect on a three-point scale: positive, neutral or negative. Therefore, reviews text datasets with aspect will be provided.

2 Pre-process the data, to adapt the models from Part C.

Before building our models, the first thing we have to do is to load the data. The dataset that we will be using is the Aspect-term sentiment analysis (ATSA), with 4,297 reviews for training and 500 for testing and validation, respectively.

Once we downloaded the data, we will build a vocabulary with the training dataset. The vocabulary has a size of 7898 words, where the first 4, represent "PAD", "START", "UNK" and "EOS" for padding the reviews, indicating start of the review, representing words that are not part of the vocabulary and defining the end of the review, respectively.

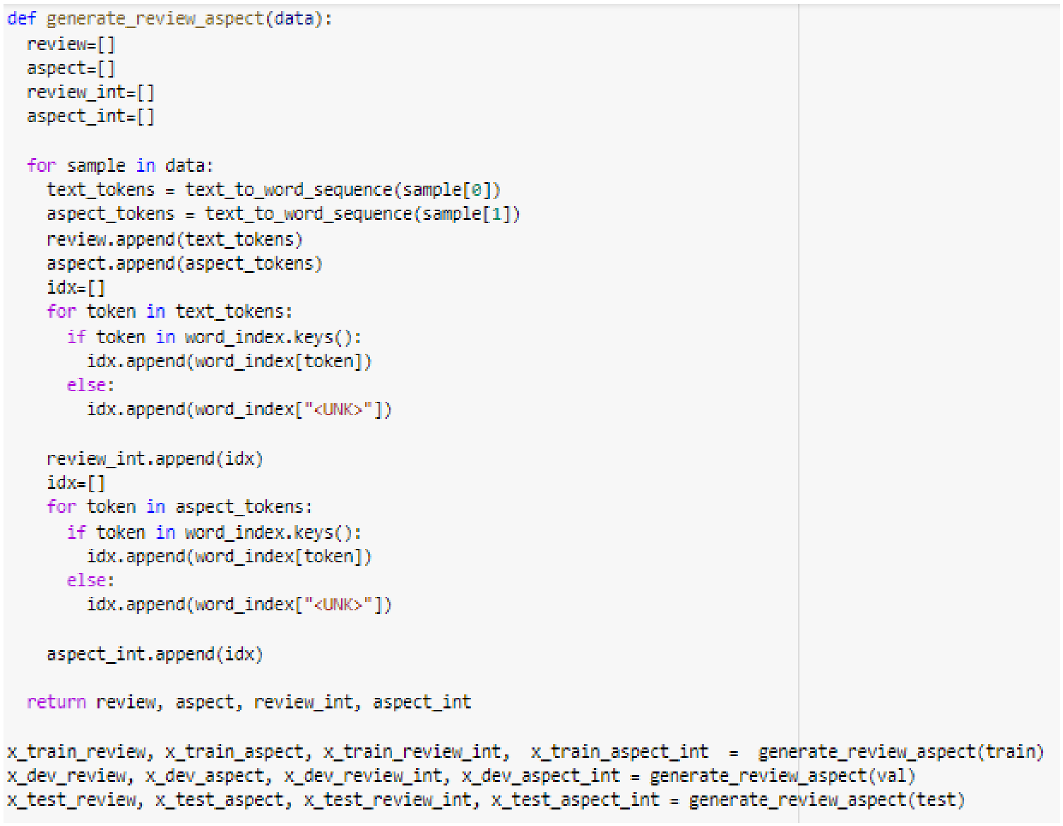
Now that we have the vocabulary and the data, we have to pre-process the given dataset. According to the word_index and the tokenizer function (text_to_word_sequence), we can convert the review text and aspect words to word tokens and integers separately. The implemented code for this can be seen in Figure 1, where a function generate_review_aspect(data) has been created to store the required information of the reviews in the training, validation and test datasets.

Moreover, when it comes to the labels, for each review an array of three elements will be used to represent the sentiment scale. To represent a positive sentiment the array [1,0,0] will be utilised. For negative it will be [0, 0, 1] and for neutral [0,1,0].

Finally, in order to introduce the inputs in our model, we have to combine the review and aspect into one sentence and then input it into the model. The first element must be the aspect and then it will be followed by the review. However, in order to distinguish the beginning of the later one, the string "START" will be included between both variables.

For each review, two new variables have been created following the implementation explained in the previous paragraph. One variable will contain the textual data, while the other one will contain the indices.

Additionally, since the reviews have a variable length, we must pad the inputs in order to improve the model's performance. Post padding will be done using the preprocessing.sequence.pad_sequences function from the Keras framework, where the length of the input will be 128.



```
def generate_review_aspect(data):
  review=[]
  aspect=[]
  review_int=[]
  aspect_int=[]

  for sample in data:
    text_tokens = text_to_word_sequence(sample[0])
    aspect_tokens = text_to_word_sequence(sample[1])
    review.append(text_tokens)
    aspect.append(aspect_tokens)
    idx=[]
    for token in text_tokens:
      if token in word_index.keys():
        idx.append(word_index[token])
      else:
        idx.append(word_index["<UNK>"])

    review_int.append(idx)
    idx=[]
    for token in aspect_tokens:
      if token in word_index.keys():
        idx.append(word_index[token])
      else:
        idx.append(word_index["<UNK>"])

    aspect_int.append(idx)

  return review, aspect, review_int, aspect_int

x_train_review, x_train_aspect, x_train_review_int,  x_train_aspect_int  =  generate_review_aspect(train)
x_dev_review, x_dev_aspect, x_dev_review_int, x_dev_aspect_int = generate_review_aspect(val)
x_test_review, x_test_aspect, x_test_review_int, x_test_aspect_int = generate_review_aspect(test)
```

Figure 1: Code to convert review text and aspect words to word tokens and integers separately .

3 Adapt your models without pre-trained word embeddings in Part C; train and evaluate it

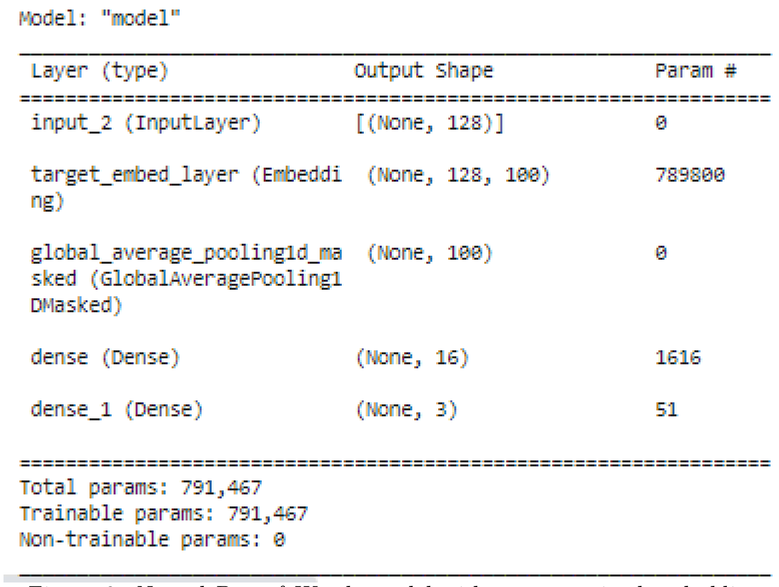
In this section we will build a model without pre-trained word embeddings. This dense vectorial representation will be jointly learned with the model.
Two models are used:

3.1 Neural Bag of Words without pre-trained embeddings.

In the Neural Bag of Word algorithm, words are trained separately, having no actual meaning as a sentence. Thus, the predictions made by the model are based on pure statistics and no semantical meaning is taken into account.
Following the tutorial given in the notebook, the model will follow the successive architecture: the first layer is the embedding layer. The next layer will compute average on all word vectors in a sentence. This will be done with help of the function GlobalAveragePooling1DMasked() function. The resulting output will be introduced in a fully connected layer whose outcome will be again introduced in a three output node and the softmax activation

function. Each output node will represent the three-point scale: positive, neutral and negative. The nodes will output values in a range between 0 to 1, representing probabilities. The sum of the output values should be 1.

In Figure 2 we can see a summary of this model.



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

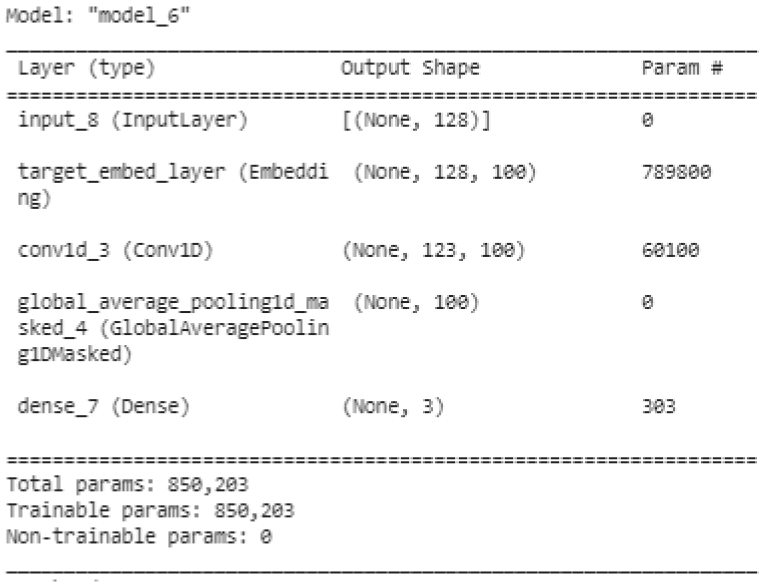
Figure 2: Neural Bag of Words model without pre-trained embeddings.

After training the model, we obtain poor results since the accuracy in the validation set is 55.63%. On the other hand, when it comes to the testing set, the accuracy is 55.61%. There is no overfitting. However, the results might improve if we use Convolutional Neural Netowrks (CNN).

3.2 CNN without pre-trained embeddings..

Unlike Bag of Words model, CNN identify patterns on the sentences regardless of words position. Therefore, important information may be extracted leading us to create models with better performance.

The CNN model, we will use the Conv1 function from the Keras framework. This is a one dimensional convolutional layer whose kernel size is equal to 6. The next layer is the pooling layer where max pooling is performed using the previously implemented GlobalAveragePooling1DMasked() function, where the maximum value of the resulting convolutional channels will be saved. Finally, the last layer is the output layer with three nodes and softmax activation function. Figure 3 summarises the model explained in this paragraph.



```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 128)]             0

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

 conv1d_3 (Conv1D)           (None, 123, 100)          60100

 global_average_pooling1d_ma  (None, 100)              0
 sked_4 (GlobalAveragePoolin
 g1DMasked)

 dense_7 (Dense)             (None, 3)                 303

=================================================================
Total params: 850,203
Trainable params: 850,203
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: CNN model without pre-trained embeddings.

As expected, the CNN model yields better results than the Neural Bag of Words. We obtain accuracy values of 75.28%, 59,76% and 62.05% for training, validation and test datasets, respectively.

4 Adapt your models with pre-trained word embeddings in Part C; train and evaluate it

In the previous section, our models were trained using word embeddings that were jointly learned with the model. Nevertheless, the performance of the models can be affected due to two reasons:

- Sparsity of training data due to the large amount of rare words, what may lead to a poor word embedding representation.
- Computationally expensive since the model has to learn the embeddings too.

Therefore, in this section the option of using pre-trained word embeddings is explored. Concretely, for this assignment we will be using GloVe embeddings, that derive the relationship between the words from Global Statistics.

Additionally, we must decide if the embeddings should be updated or not during the model's training. In the case of not, we will be talking about freezing pre-trained embeddings. Otherwise, we will fine-tune the embedding weights. In the given script it is required to freeze the embeddings, therefore they will not be updated.

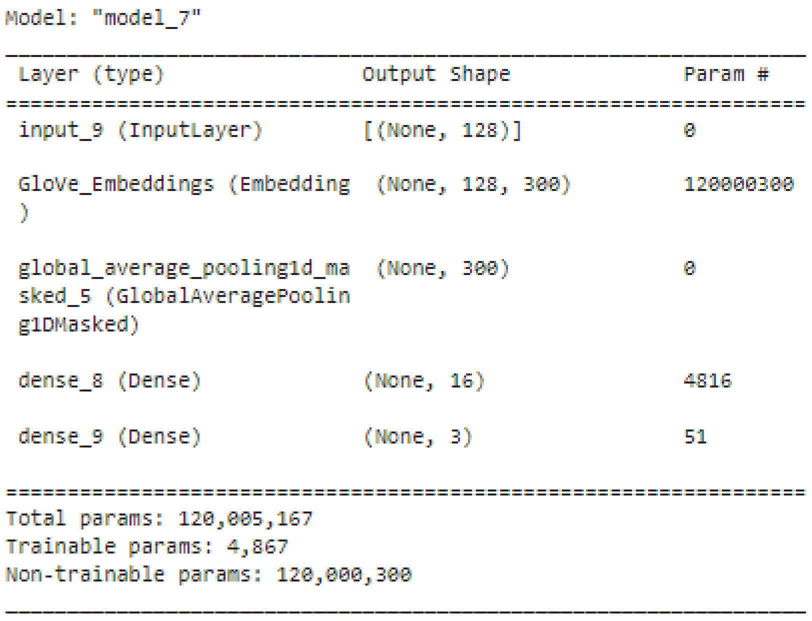
As in the previous section two models will be adapted to use pre-trained embeddings: Neural Bag of Words and CNN. The only change with respect to the models in the previous section is the embedding layer, which must be loaded. Two functions are used for this:

- readGloveFile(gloveFile): loads the GloVe file of the Standford NLP toolkit.
- createPretrainedEmbeddingLayer(wordToGlove, wordToIndex, isTrainable): build the embeddign layer. The argument isTrainable is a boolean. In the case of being "False" the embedding layer will be freezed during training. Otherwise, the weights will be fine-tuned. In our case the value will be set to "False".

Once we have used these functions, we will have an embedding layer whose embedding size is equal to 300. Now that we have the embedding later we will start adapting it to our models.

4.1 Neural Bag of Words using pre-trained embeddings.

After adapting the model in section 3.1, we obtain the model summary that can be visualised in Figure 4



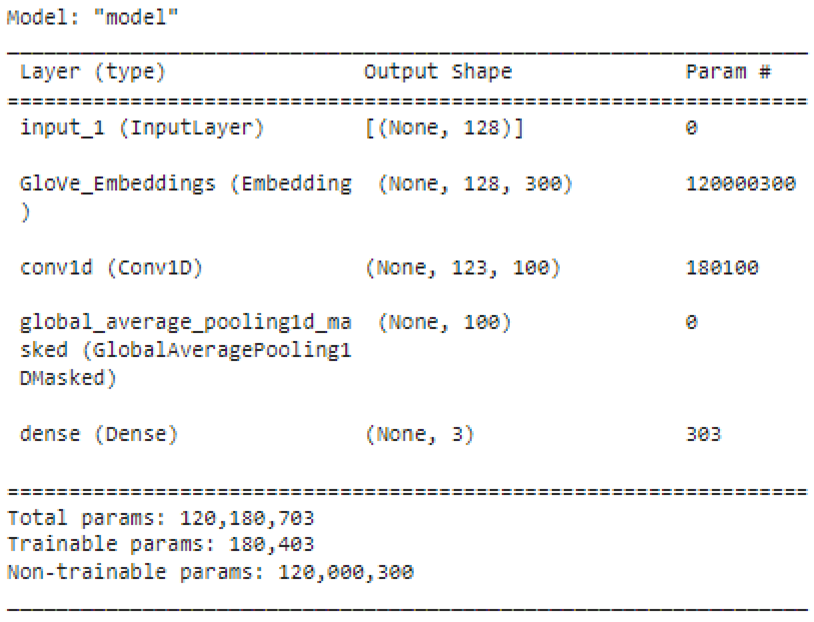
```
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 128)]             0

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

 global_average_pooling1d_ma  (None, 300)              0
 sked_5 (GlobalAveragePoolin
 g1DMasked)

 dense_8 (Dense)             (None, 16)                4816

 dense_9 (Dense)             (None, 3)                 51

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

Figure 4: Neural Bag of Words model with pre-trained embeddings.

Once the previous model is trained we obtain 58.78%, 55.93% and 58.53% of accuracy for the training, validation and test set respectively. Our model does not overfit, and in comparison with the one that does not use pre-trained embeddings, the improvement is quite insignificant. The main reason behind this is that even though the embedding layer was obtained differently, the final matrix had similar weights and therefore, the evaluation results were similar too.

4.2 CNN using pre-trained embeddings.

In this subsection, model 3.2 was adapted leading to the model summary that can be visualised in Figure 5



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

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

 conv1d (Conv1D)             (None, 123, 100)          180100

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

 dense (Dense)               (None, 3)                 303

=================================================================
Total params: 120,180,703
Trainable params: 180,403
Non-trainable params: 120,000,300
_________________________________________________________________
```

Figure 5: CNN with pre-trained embeddings.

After training the previous model, we obtain 66.25%, 63.59% and 63.32% of accuracy for the training, validation and test set respectively. Our model does not overfit, and in comparison with the one that does not use pre-trained embeddings, it just one point above. The main reason behind this is the same as the one given in subsection 4.1: even if the embedding layer was obtained differently, the final matrix had similar weights and therefore, the evaluation results were similar too.

5 Build and evaluate two more classifiers with multiple input.

In the previous models, we combined the reviews and aspects to input into the models. In this section a different approach is taken: the reviews and aspects will be two separate inputs for the model and they will use different layers to analyse them.

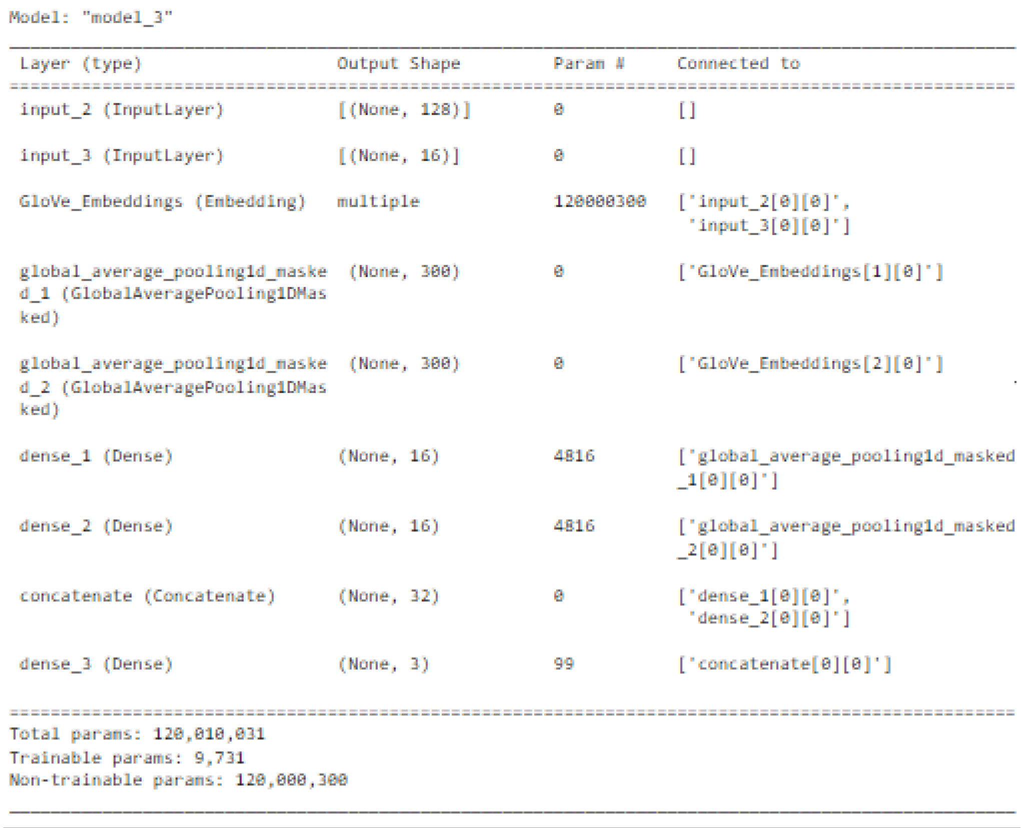
Before building the models, the first step is to pad the reviews and aspects separately with a maximum length of 128. This will be done using the preprocessing.sequence.pad_sequences() function of the Keras framework.

Once we have the padded inputs we can proceed to build the models.

5.1 Neural Bag of Words with multiple inputs.

For this model, we will first have a review embedding layer. The next layer will compute average on all word vectors in a input review. This will be done with help of the function GlobalAveragePooling1DMasked(). The resulting output will be introduced in a fully connected layer.

The same procedure explained in the previous paragraph will be applied for the aspect input. Once we have both of the outputs of the fully connected dense layer with 16 nodes, we will apply the dot product to both outcomes. The result of this will be a vector of 32 features. This vector will be introduced to a three output node layer with the softmax as the activation function. A summary of this explanation can be seen in Figure 6



```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 128)]        0           []

 input_3 (InputLayer)           [(None, 16)]         0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_2[0][0]',
                                                                  'input_3[0][0]']

 global_average_pooling1d_maske  (None, 300)         0           ['GloVe_Embeddings[1][0]']
 d_1 (GlobalAveragePooling1DMas
 ked)

 global_average_pooling1d_maske  (None, 300)         0           ['GloVe_Embeddings[2][0]']
 d_2 (GlobalAveragePooling1DMas
 ked)

 dense_1 (Dense)                (None, 16)           4816        ['global_average_pooling1d_masked
                                                                 _1[0][0]']

 dense_2 (Dense)                (None, 16)           4816        ['global_average_pooling1d_masked
                                                                 _2[0][0]']

 concatenate (Concatenate)      (None, 32)           0           ['dense_1[0][0]',
                                                                  'dense_2[0][0]']

 dense_3 (Dense)                (None, 3)            99          ['concatenate[0][0]']

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

Figure 6: Neural Bag of Words with multiple inputs.

For clarity sake, in Figure 7 provides a visualisation of the model in form of a graph.

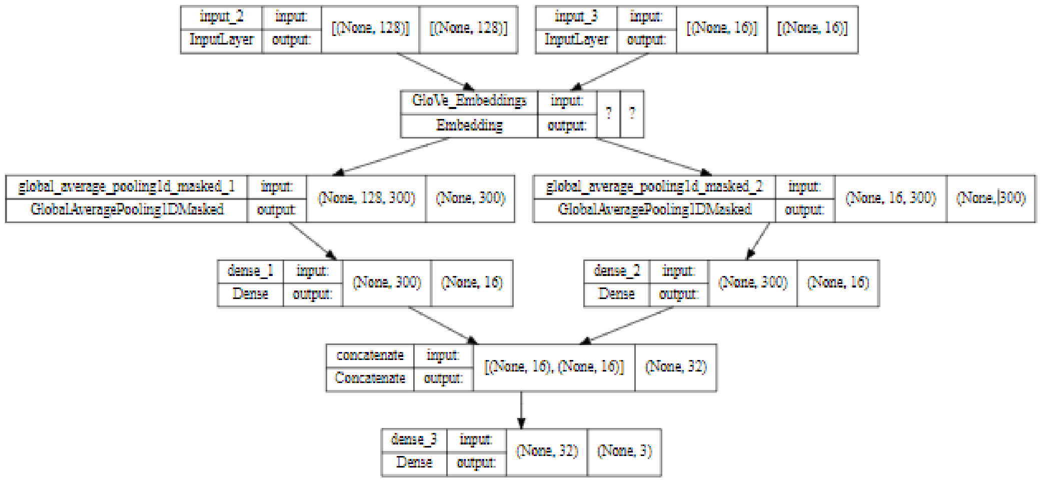


Figure 7: Model visualisation.

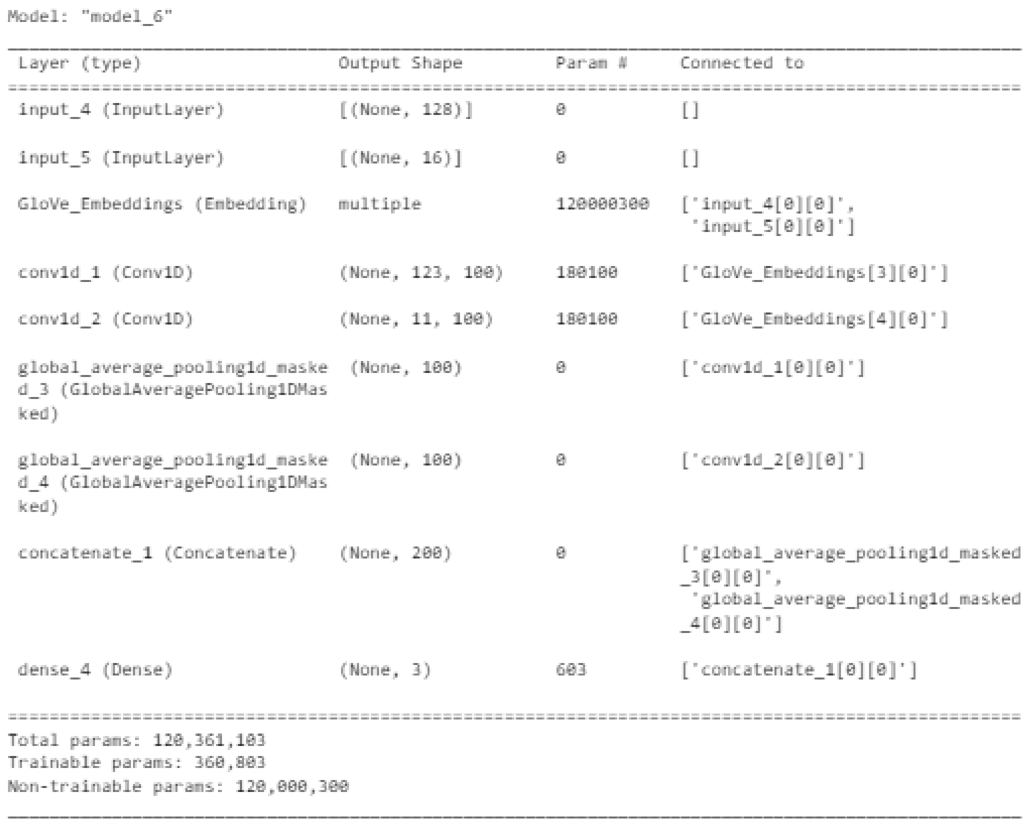
This model is compiled using a categorical_crossentropy classifier (three-point scale problem: the output is either negative, positive or neutral). Moreover, Adam optimiser is used (Stochastic Gradient Descent is used for training) and accuracy is used as performance metric.

After training the model we obtain 67.95%, 64.79% and 64.75% of accuracy for the training, validation and test set respectively. The model performance significantly improves if we separate both inputs, since not doing it provides an accuracy value for the validation set of 55.93%. Separating inputs help the model to find the relation between the review and the aspect in a more meaningful way. At first we will be learning patterns from both inputs separately and then after concatenating them and computing similarity, the model learns weights in accordance to the end-task. Thus, it leads to a better performance.

5.2 CNN with multiple inputs.

As it was done in the previous case, both inputs will go separately through a convolutional layer whose kernel size is equal to 6 and the dimensionality of the output space will be 100. Then, it will go through the pooling layer where max pooling is performed using the previously implemented GlobalAveragePooling1DMasked() function.

Once we have both of the outputs of the pooling layer with, we will apply the dot product to both outcomes.The result of this will be a vector of 200 features. This vector will be introduced to a three output node layer with the softmax as the activation function. A summary of this explanation can be seen in Figure 8



```
Model: "model_6"
____________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
====================================================================================================
 input_4 (InputLayer)           [(None, 128)]        0           []

 input_5 (InputLayer)           [(None, 16)]         0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_4[0][0]',
                                                                  'input_5[0][0]']

 conv1d_1 (Conv1D)              (None, 123, 100)     180100      ['GloVe_Embeddings[3][0]']

 conv1d_2 (Conv1D)              (None, 11, 100)      180100      ['GloVe_Embeddings[4][0]']

 global_average_pooling1d_maske  (None, 100)         0           ['conv1d_1[0][0]']
 d_3 (GlobalAveragePooling1DMas
 ked)

 global_average_pooling1d_maske  (None, 100)         0           ['conv1d_2[0][0]']
 d_4 (GlobalAveragePooling1DMas
 ked)

 concatenate_1 (Concatenate)    (None, 200)          0           ['global_average_pooling1d_masked
                                                                 _3[0][0]',
                                                                  'global_average_pooling1d_masked
                                                                 _4[0][0]']

 dense_4 (Dense)                (None, 3)            603         ['concatenate_1[0][0]']

====================================================================================================
Total params: 120,361,103
Trainable params: 360,803
Non-trainable params: 120,000,300
____________________________________________________________________________________________________
```

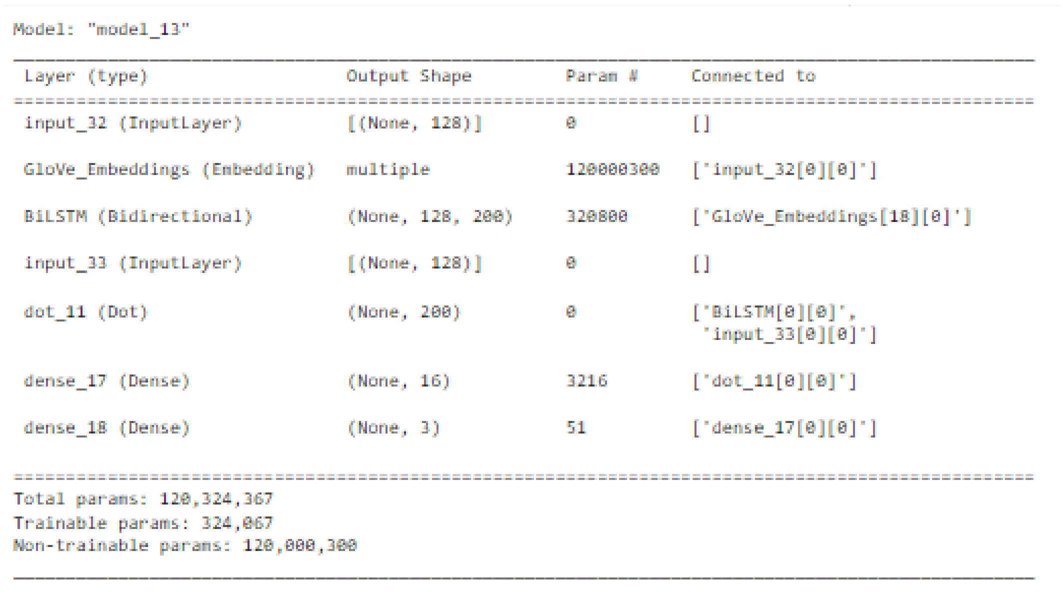
Figure 8: CNN with multiple output model summary.

This model is compiled using a categorical_crossentropy classifier, Adam optimiser and accuracy is used as performance metric. After training the model we obtain 72.08%, 64.79% and 62.35% of accuracy for the training, validation and test set respectively. In comparison to the model presented in section 4.2, adding two inputs does not improve much the model. With CNN, the model is able to find patterns on the reviews with or without separate inputs.

6 Build and evaluate the classifier extracting information from LSTM.

Taking into account that the aspects is part of the review, we can treat this problem as an "unknown target" sequence tagging models, and extract the aspect information from the reviews. The polarity of an aspect is usually determined by the content surrounding it. To transfer information from adjacent context to the aspect, we can employ a Long-Short-Term Memory (LSTM) model. Without evaluating the entire statement, we simply need to extract the aspect vector in order to calculate its polarity. This will be accomplished with the help of Bidirectional LSTM (BiLSTM)

BiLSTMs are made up of two LSTMs: one that takes input in one way and the other that takes it in the opposite direction. BiLSTMs effectively improve the quantity of data available to the network, providing the algorithm with more context.
The built BiLSTM model can be seen in the following figure, where we can see that the reviews are introduced in the BiLSTM layer and the resulting vector is concatenated to the aspect vector using the dot product. Later it will go through two fully connected layers, whose final outcome will be a 3 nodes layer, each one for the posible output (positive, neutral and negative).



```
Model: "model_13"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_32 (InputLayer)          [(None, 128)]        0           []

 GloVe_Embeddings (Embedding)   multiple             120000300   ['input_32[0][0]']

 BiLSTM (Bidirectional)         (None, 128, 200)     320800      ['GloVe_Embeddings[18][0]']

 input_33 (InputLayer)          [(None, 128)]        0           []

 dot_11 (Dot)                   (None, 200)          0           ['BiLSTM[0][0]',
                                                                  'input_33[0][0]']

 dense_17 (Dense)               (None, 16)           3216        ['dot_11[0][0]']

 dense_18 (Dense)               (None, 3)            51          ['dense_17[0][0]']

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

Figure 9: BiLSTM model summary.

A visual graph of the model can be seen in Figure 10.

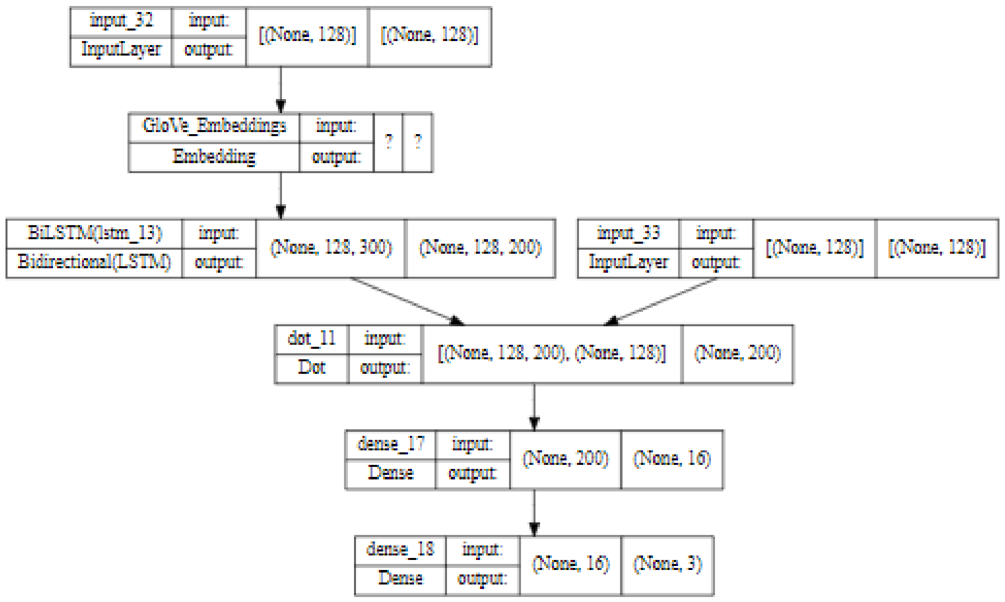


Figure 10: Visualisation BiLSTM model.

The BiLSTM model yields better results than the ones built in the previous sections. We obtain accuracy values of 93.81%, 72.67% and 71.33% for training, validation and test datasets, respectively. The main reason behind this, is that th BiLSTM enable additional training by traversing the input data twice and therefore, provides better predictions for this particular case.

Since BiLSTM is the best model we could come up with for this part of the assignment, it will be the one choosen to classify the sentiment of the aspects for the given dataset.