Name: Juliann Weir-Jackson

Project 3: Sentiment Analysis on US Airline Tweets

Date: December 14, 2021

Abstract:

This project is a study of different neural network models used to conduct sentiment analysis on tweets of US airlines. The standard model of a Simple RNN, LSTM and GRU is used and built on to create different models. A preexisting dataset of US airline tweets was used, sentiments and negative reason for most tweets were included in the dataset. The performance of each model used was recorded and used for comparison.

Introduction:

The goal of this project will be to perform a sentiment analysis on tweets about US airlines. Each tweet can be categorized as negative positive or neutral. In order to preprocess the data tokenization, normalization and noise removal will be done. First all the targets (airline sentiments) were converted to a scale, 0 representing a neutral sentiment, 1 representing a negative sentiment and 2 representing a positive sentiment. To preprocess each tweet, all html breaks, URLs, usernames, emojis were removed. Stop words, numbers and punctuation marks were also removed, each tweet was changed so all words were in lower case. This dataset also included a negative reason column which was added to the tweet prior to all preprocessing. The method that will be used to evaluate the sentiment of each tweet is, to first start with the standard model of SimpleRNN, LSTM and GRU and then add additional layers, after each change is made to the model the network will be executed and results will be recorded.

Literature Review:

"Sentiment Analysis also known as opinion mining is the process of determining the emotions behind words, sentences or documents which is very useful in analyzing public opinions" (Baktha, Tripathy). There are many ways in which sentiment analysis has been done some methodologies include: RNNs, LSTM, GRU, hybrids of the aforementioned methodologies and CNNS. In 2017 Baktha and Tripathy, investigated the use of RNNs specifically Vanilla RNN, LSTM and GRU, RMS Prop optimizer and Bidirectional RNNs. Three datasets were used: the amazon product reviews, Stanford Sentiment Treebank (SST-1), and SST-2. The investigation concluded that GRU model obtained the best accuracy on all three datasets, while the Vanilla RNN obtained the worst accuracy. The bidirectional structures also outperformed their unidirectional counter parts when tested on all datasets.

Yoon Kim, used several variations of CNNs trained on top of pretrained word vectors for sentence classification. CNN-Rand, CNN-Static, CNN non static and CNN-Multichannel were the variations used and were tested on five different datasets. They found that CNN-rand did not perform well on its own, using pretrained vectors there were remarkable increase in performance. The CNN-static model also performed remarkably well. For multi-channel and single channel models they found that results were mixed and stated that finetuning needed to be done.

Model:

The standard model of each methodology was used, after which the architecture of each model was changed and hyperparameters were added. The table below will identify the layers in each model.

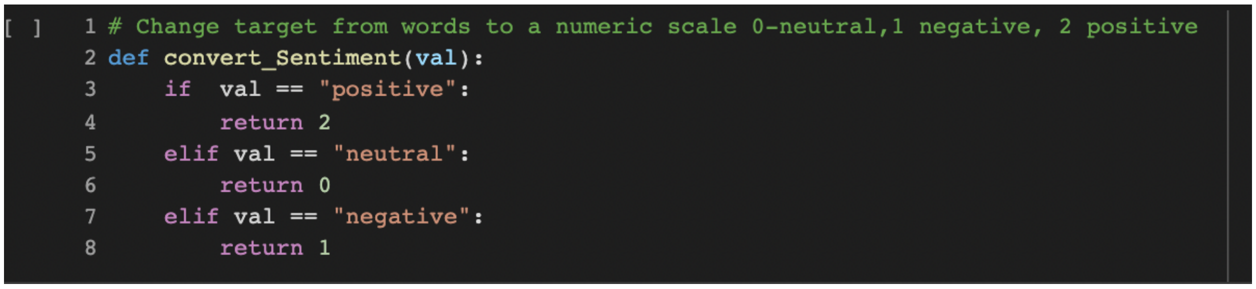
Layers and Hyperparameters	Text Vectorization	Embedding	Dense	SimpleRNN	Bidirectional SimpleRNN	LSTM	GRU	Bidirectional LSTM	Dropout	Regularization
Model 1	▪	▪	▪	▪						
Model 2	▪	▪	▪	▪	▪	▪			▪	
Model 3	▪	▪	▪	▪						▪
Model 4	▪	▪		▪	▪				▪	▪
Model 5	▪	▪	▪			▪				
Model 6	▪	▪					▪		▪	▪
Model 7	▪	▪	▪			▪		▪	▪	▪
Model 8	▪	▪	▪				▪	▪	▪	▪
Model 9	▪	▪	▪		▪		▪		▪	▪

Table showing the architecture of each model if the model has a ▭ present then the layer indicated is a part of the model.

Training:

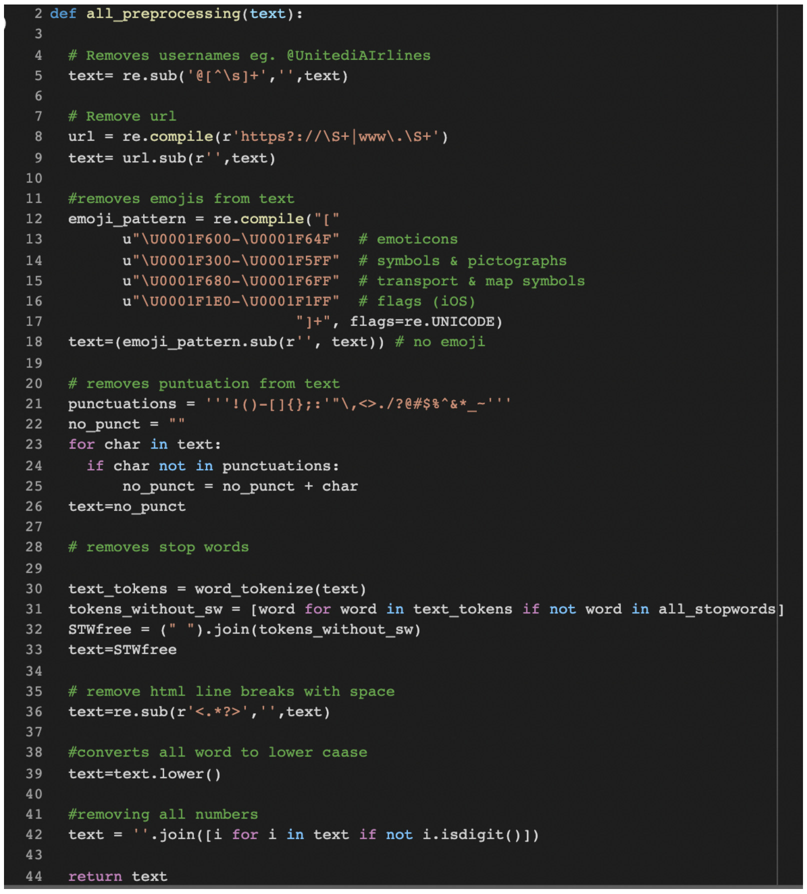
<u>Data Preprocessing:</u>

Data was read from a csv file into a data frame, from the data frame only 3 columns were needed, columns labeled target and data were created and the content of the airline sentiment and text of each tweet were read into them. Since the airline sentiments were given as words, positive negative or neutral a function was created to convert each sentiment to a number, 0 representing neutral ,1 representing negative and 2 representing positive.



```python
# Change target from words to a numeric scale 0-neutral,1 negative, 2 positive
def convert_Sentiment(val):
    if  val == "positive":
        return 2
    elif val == "neutral":
        return 0
    elif val == "negative":
        return 1
```

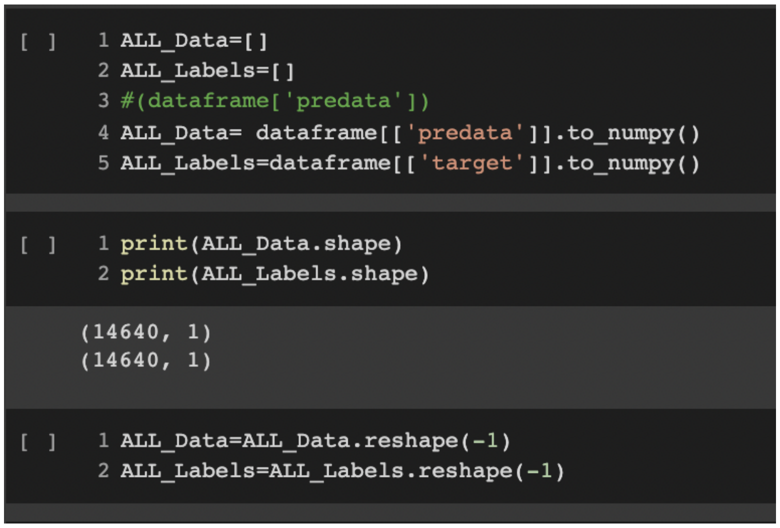
After sentiments were converted text data was preprocessed, a function called all_preprocessing was created. This function contained all the necessary function needed to remove noise from the data, it removed the username from all tweets, emojis, numbers, punctuation marks, URLS, html breaks and stop words. It also converted all text to lower case.



```python
def all_preprocessing(text):

  # Removes usernames eg. @UnitediAIrlines
  text= re.sub('@[^\s]+','',text)

  # Remove url
  url = re.compile(r'https?://\S+|www\.\S+')
  text= url.sub(r'',text)

  #removes emojis from text
  emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags=re.UNICODE)
  text=(emoji_pattern.sub(r'', text)) # no emoji

  # removes puntuation from text
  punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
  no_punct = ""
  for char in text:
    if char not in punctuations:
        no_punct = no_punct + char
  text=no_punct

  # removes stop words

  text_tokens = word_tokenize(text)
  tokens_without_sw = [word for word in text_tokens if not word in all_stopwords]
  STWfree = (" ").join(tokens_without_sw)
  text=STWfree

  # remove html line breaks with space
  text=re.sub(r'<.*?>','',text)

  #converts all word to lower caase
  text=text.lower()

  #removing all numbers
  text = ''.join([i for i in text if not i.isdigit()])

  return text
```

Once all preprocessed text was saved to the data frame all unnecessary columns were dropped.

Each record in the two columns left in the data frame were the converted to NumPy arrays and



```
ALL_Data=[]
ALL_Labels=[]
#(dataframe['predata'])
ALL_Data= dataframe[['predata']].to_numpy()
ALL_Labels=dataframe[['target']].to_numpy()
```

```
print(ALL_Data.shape)
print(ALL_Labels.shape)
```

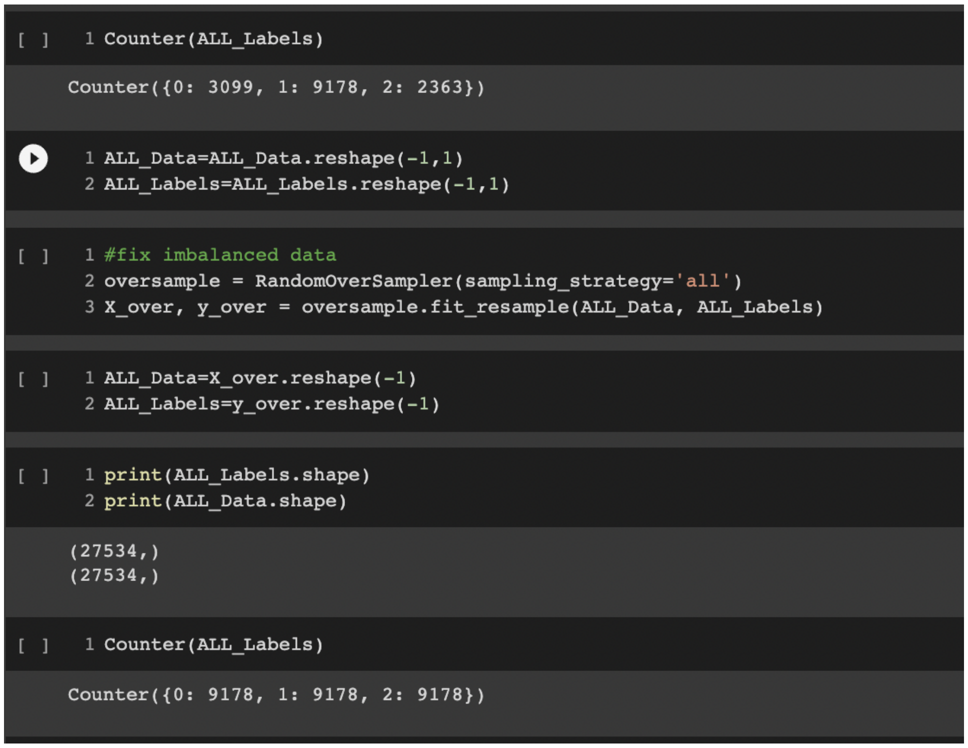
```
(14640, 1)
(14640, 1)
```

```
ALL_Data=ALL_Data.reshape(-1)
ALL_Labels=ALL_Labels.reshape(-1)
```

reshaped

The Counter() function was used to check the number of sentiments that belonged to each class.

The dataset was found to be imbalanced, and oversampling was used to balance the dataset.



```
Counter(ALL_Labels)
```

```
Counter({0: 3099, 1: 9178, 2: 2363})
```

```
ALL_Data=ALL_Data.reshape(-1,1)
ALL_Labels=ALL_Labels.reshape(-1,1)
```

```
#fix imbalanced data
oversample = RandomOverSampler(sampling_strategy='all')
X_over, y_over = oversample.fit_resample(ALL_Data, ALL_Labels)
```

```
ALL_Data=X_over.reshape(-1)
ALL_Labels=y_over.reshape(-1)
```

```
print(ALL_Labels.shape)
print(ALL_Data.shape)
```

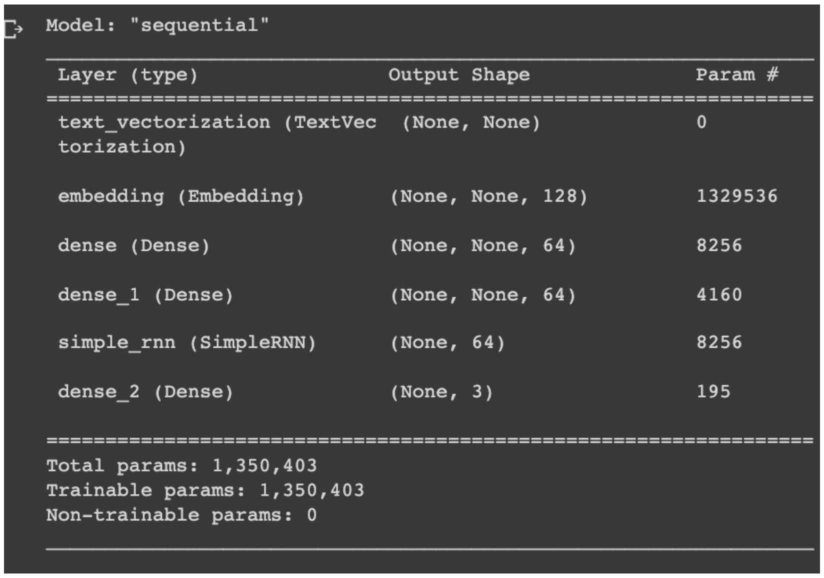
```
(27534,)
(27534,)
```

```
Counter(ALL_Labels)
```

```
Counter({0: 9178, 1: 9178, 2: 9178})
```

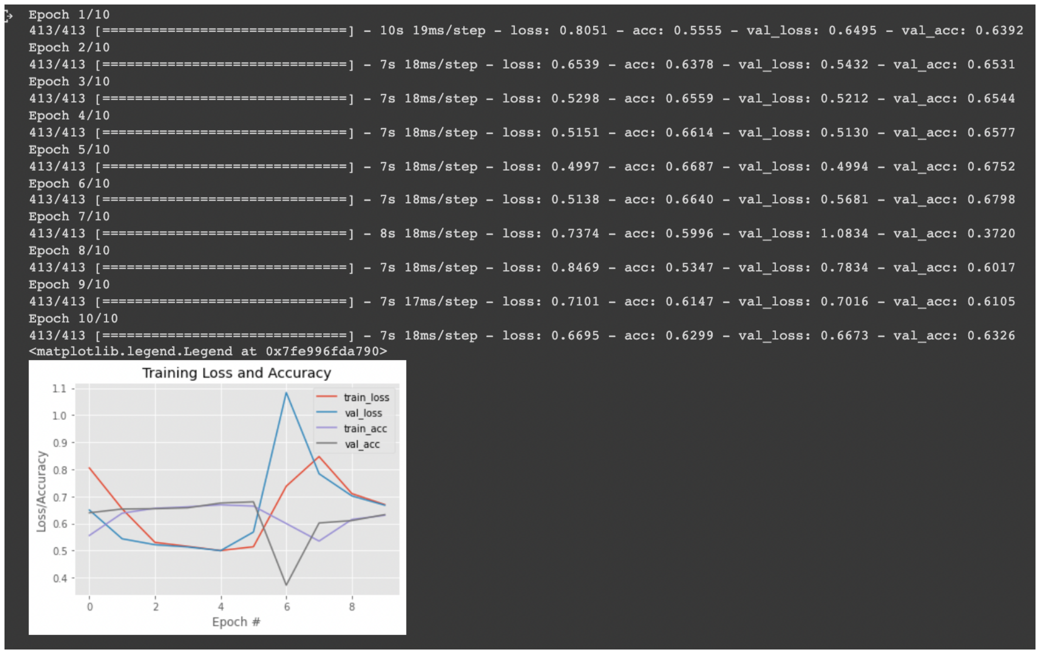
Benchmark:

For the benchmark, the model used was the SimpleRNN , consisting of a vectorization layer , embedding layer , two dense layers one fully connected recurrent net and a soft max classifier. Model Summary shown below:



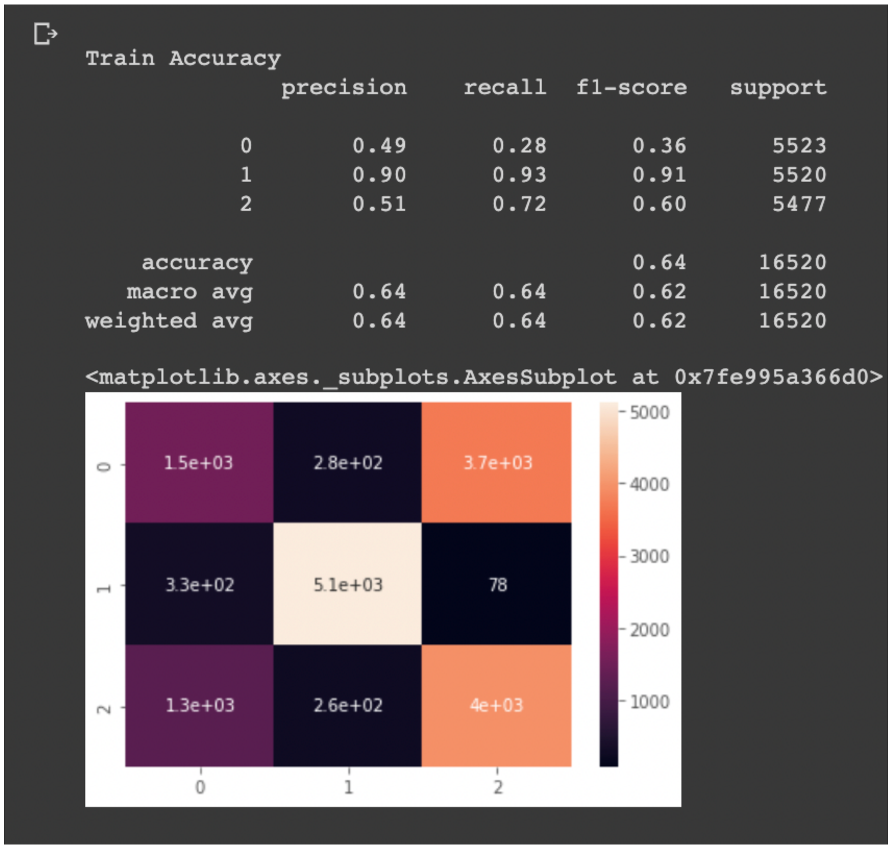
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 128)         1329536

 dense (Dense)               (None, None, 64)          8256

 dense_1 (Dense)             (None, None, 64)          4160

 simple_rnn (SimpleRNN)      (None, 64)                8256

 dense_2 (Dense)             (None, 3)                 195

=================================================================
Total params: 1,350,403
Trainable params: 1,350,403
Non-trainable params: 0
_________________________________________________________________
```

A validation loss and validation accuracy of 0.6673 and 0.6626 were recorded respectively. Not only was the accuracy bad but overfitting was occurring as the accuracy stagnated throughout all epochs.



```
Epoch 1/10
413/413 [==============================] - 10s 19ms/step - loss: 0.8051 - acc: 0.5555 - val_loss: 0.6495 - val_acc: 0.6392
Epoch 2/10
413/413 [==============================] - 7s 18ms/step - loss: 0.6539 - acc: 0.6378 - val_loss: 0.5432 - val_acc: 0.6531
Epoch 3/10
413/413 [==============================] - 7s 18ms/step - loss: 0.5298 - acc: 0.6559 - val_loss: 0.5212 - val_acc: 0.6544
Epoch 4/10
413/413 [==============================] - 7s 18ms/step - loss: 0.5151 - acc: 0.6614 - val_loss: 0.5130 - val_acc: 0.6577
Epoch 5/10
413/413 [==============================] - 7s 18ms/step - loss: 0.4997 - acc: 0.6687 - val_loss: 0.4994 - val_acc: 0.6752
Epoch 6/10
413/413 [==============================] - 7s 18ms/step - loss: 0.5138 - acc: 0.6640 - val_loss: 0.5681 - val_acc: 0.6798
Epoch 7/10
413/413 [==============================] - 8s 18ms/step - loss: 0.7374 - acc: 0.5996 - val_loss: 1.0834 - val_acc: 0.3720
Epoch 8/10
413/413 [==============================] - 7s 18ms/step - loss: 0.8469 - acc: 0.5347 - val_loss: 0.7834 - val_acc: 0.6017
Epoch 9/10
413/413 [==============================] - 7s 17ms/step - loss: 0.7101 - acc: 0.6147 - val_loss: 0.7016 - val_acc: 0.6105
Epoch 10/10
413/413 [==============================] - 7s 18ms/step - loss: 0.6695 - acc: 0.6299 - val_loss: 0.6673 - val_acc: 0.6326
<matplotlib.legend.Legend at 0x7fe996fda790>
```

Confusion matrix and classification report from benchmark:



```
Train Accuracy
              precision    recall  f1-score   support

           0       0.49      0.28      0.36      5523
           1       0.90      0.93      0.91      5520
           2       0.51      0.72      0.60      5477

    accuracy                           0.64     16520
   macro avg       0.64      0.64      0.62     16520
weighted avg       0.64      0.64      0.62     16520

<matplotlib.axes._subplots.AxesSubplot at 0x7fe995a366d0>
```

Class	True Positive	True Negative	False Positive	False Negative

0-Neutral	15000	9198	1630	3,980
1-Negative	5100	8,500	540	408
2-positive	4000	7210	3778	1560

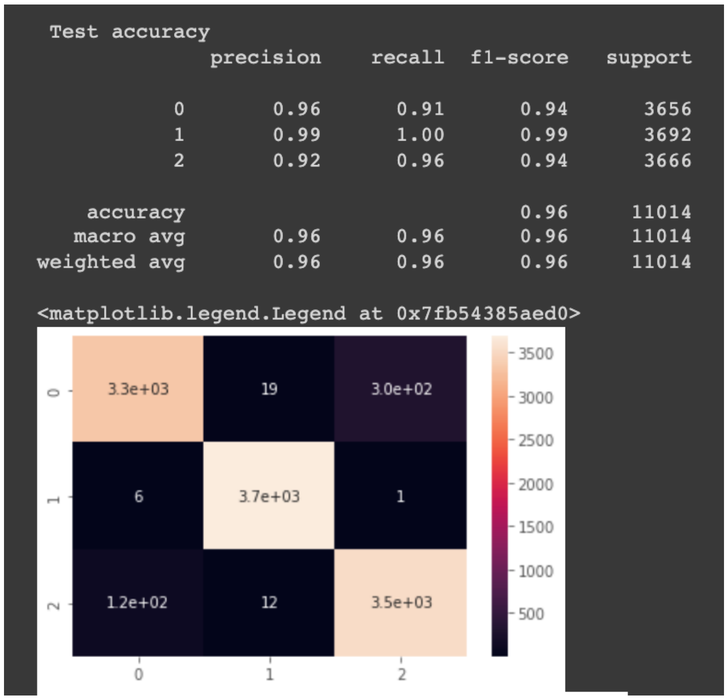
From the confusion matrix we can see for the neutral class the network confused mainly positive sentiments. For the negative class network confused mainly the neutral class and for the positive class mainly the neutral class was confused.

After noticing the overfitting in the benchmark model, the next three experiments focused on trying to mitigate overfitting. Experiment one implemented dropout layers, Experiment 2 implemented regularization on dense layers and Experiment 3 implemented early stopping by using callbacks. Even after implementing three methods to mitigate overfitting, it still occurred in all three models in each experiment.

In the following experiments the LSTM and GRU were used as well as bi-directional LSTM and bidirectional SimpleRNN. Each of the models were trained with dropout early stopping and regularization included as a way to mitigate overfitting.

Model	Training Loss after 10 epochs	Training Accuracy after 10 epochs	Validation Loss after 10 epochs	Validation Accuracy after 10 epochs
Benchmark: SimpleRNN	0.6695	0.6299	0.66673	0.6326
Bidirectional RNN and SimpleRNN	0.0482	0.9898	0.1836	0.9573
LSTM	0.0444	0.9893	0.1799	0.9537
Bidirectional LSTM and SimpleRNN	0.0508	0.9892	0.1919	0.9564
Bidirectional LSTM and LSTM	0.0616	0.9865	0.1835	0.9597
GRU	0.4690	0.6662	0.4741	0.6752
Bidirectional LSTM and GRU	0.0478	0.9880	0.1879	0.9561

All models shown also had dropout regularization and early stopping implemented when executed, however overfitting still occurred in many of the models regardless. The models in which the validation loss and training loss were almost equal namely the GRU model it was determined that the model was underfitting to more layers were added to increase the complexity of the model in order to handle the underfitting this can be seen in following experiment which added bidirectional LSTM to the GRU model. Experiment 10 focused on trying to tune the hyperparameters of the best performing model from the previous models. The test dataset was run on the best model using the dropout hyperparameters and regularization parameters that resulted in the best accuracy. Test classification report and confusion matrix



```
 Test accuracy
              precision    recall  f1-score   support

           0       0.96      0.91      0.94      3656
           1       0.99      1.00      0.99      3692
           2       0.92      0.96      0.94      3666

    accuracy                           0.96     11014
   macro avg       0.96      0.96      0.96     11014
weighted avg       0.96      0.96      0.96     11014

<matplotlib.legend.Legend at 0x7fb54385aed0>
```

Class	True Positive	True Negative	False Positive	False Negative
0-Neutral	3300	7213	126	319
1-Negative	3700	7220	31	7
2-positive	3500	7025	301	132

Conclusion:

To conclude sentiment analysis was successfully conducted on the dataset, based on the confusion matrix of the benchmark and the confusion matrix of the test data which was on run on the best performing model there was a significant increase in the networks capabilities of predicting sentiments based on text. Improvements that could have been made was to thoroughly tune hyperparameters in order tom mitigate overfitting more as many models were overfitting even with the use of dropout regularization an early stopping. Using a model with Simple RNN alone performs poorly, models with multiple recurrent network layers (bidirectional LTSM, LTSM GRU ,bidirectional SimpleRNN) tended to perform better that models with only one recurrent network layer.

References

Baktha and B. K. Tripathy, "Investigation of recurrent neural networks in the field of sentiment analysis," 2017 International Conference on Communication and Signal Processing (ICCSP), 2017, pp. 2047-2050, doi: 10.1109/ICCSP.2017.8286763

Kim,Yoon. *Convolutional Neural Networks for Sentence Classification*. New York University, 2014.