



SCHOOL OF COMPUTER SCIENCE
UNIVERSITI SAINS MALAYSIA

CPC353/CPT346: NATURAL LANGUAGE PROCESSING

Semester 1, Academic Session: 2021/2022

Lecturer :

Dr Tan Tien Ping

Assignment 2:

Sentiment Analysis on News Headlines

Student:

Chan Siang Sheng (142413)

Introduction

As technology has started to bloom, digital news has become a new method for the news media industry to deliver their news to the public in an instant and easier way. News headlines are one of the important components to attract the public's attention, it should be meaningful, short and concise to easily give the gist of the article to the reader.

With so many news headlines available, we can make use of them by experimenting with sentiment analysis. Sentiment analysis is a Natural Language Processing (NLP) technique used to determine if the words or sentences are positive, neutral or negative. For example, a news headline such as “Malaysia celebrates its 66th Independence Day” is analyzed as positive and it can be deduced that the keyword of “Independence” has the feature of positivity, hence if the other news includes “Independence” in its news headline, it should be analyzed as positive as well unless there are the other keywords that brought neutrality and negativity which would affect the outcome.

This sentiment analysis is not only helpful in deducing the positivity of news headlines, it can be used in other scenarios as well such as in a feedback survey, to determine the general opinion from the customer, so that surveyor can determine the factor to the customers' need.

Methodology

Sentiment analysis is a technique from Natural Language Processing and we will be experimenting with news headlines. This experiment will be carried out in Google Colaboratory with Python as the programming language. There are multiple imported libraries such as nltk, tensorflow, numpy, pandas, etc which are required to assist in the sentimental analysis process. The whole experiment will be carried out in several steps and explained chronologically in the sections below.

Data Collection

For this experiment we will be using an amount of 61692 news headlines as trained data, another 1000 news headlines as test data and another 1000 more for validation. Each dataset should be saved in an excel (.csv) format. Each dataset should contain a list of news headlines as the first column and sentiment outcome as the second column. A depiction of the columns from test.csv can be seen in Figure 1. All news headlines from the train.csv are annotated by students of CPC353/ CPT346.

	A	B	C
1	fgv explains accounting treatment of replanting cost	1	
2	rhb founder to take up stake in priceworth via special share issuance	1	
3	time to buy 37% stake in thai telecoms operator for rm186.7m	1	
4	malaysia ranks low in graft survey	-1	
5	'no cover-up'	-1	
6	malaysia ready to decide on nuclear power	0	
7	wz satu may trend higher, says rhb retail research	1	
8	samchem may rise higher, says rhb retail research	1	
9	higher asps, firmer volume growth to sustain earnings for rubber products se	1	
10	close to half of malaysian women quit work due to halt in career progression	-1	
11	record spending for fab equipment expected in 2017 and 2018, says semi	0	
12	global passenger traffic up 9.6% y-o-y in january, says iata	0	
13	saudi pledges big projects to re-energise economy	1	
14	china's zte to pay over us\$800 mil to settle with us over iran sales - source	0	
15	us trade deficit jumps to five-year high on imports	-1	
16	malaysia is one of two asia pacific nations with severe graft issues - ti survey	-1	

Figure 1: First 16 Rows of News Headlines with Its Respective Sentiment in test.csv.

The files are saved in my own Github Repository, so in this experiment, I will download the files through Github's raw files link and read the csv with pandas library. An example of codes is shown in Figure 2.

```
2 import pandas as pd
3 !wget -O 'train.csv' 'https://raw.githubusercontent.com/aaron6347/CPT346-Asn2/main/sentiment/train.csv'
4 !wget -O 'test.csv' 'https://raw.githubusercontent.com/aaron6347/CPT346-Asn2/main/sentiment/test.csv'
5 !wget -O 'validation.csv' 'https://raw.githubusercontent.com/aaron6347/CPT346-Asn2/main/sentiment/validation.csv'
6 train_data = pd.read_csv('train.csv', header=None, names=['sentence', 'sentiment'])
7 test_data = pd.read_csv('test.csv', header=None, names=['sentence', 'sentiment'])
8 validation_data = pd.read_csv('validation.csv', header=None, names=['sentence', 'sentiment'])
9 train_sentence = train_data['sentence']
10 train_sentiment = train_data['sentiment']
```

Figure 2: Code of Downloading and Reading Files.

Data Preprocessing

Once we have loaded our dataset, we need to preprocess our data. This is due to the news headlines containing irregularities of upper and lower case keywords, stop words as well as keywords with different word forms such as plural and singular, verbs, etc. We need to filter out these to produce tokens that are normalized in small capitalized and lemmatized. And if we didn't do any data preprocessing, we will have multiple tokens that have the same semantics but different wording or different forms or tokens that are not meaningful, hence affecting the true outcome and wasting memory space. For instance, "Malaysia" needs to be preprocessed to "malaysia" as there are other news headlines that use "malaysia" instead of "Malaysia" and both should have the same semantics. The example is shown in Figure 3 below.

6204	eg industries gets offer to buy 17.5% stake in singapore-listed firm for rm21.22m	0
6205	covid-19: <u>malaysia</u> records 2,690 new cases, with 10 more deaths	0
6206	prasarana finds breach of fiduciary duty by former management over makkah metro southe	0
6299	Chiau family MGO on Chin Hin Group Property not fair and not reasonable, says independen	-1
6300	Too much bureaucracy in doing business in <u>Malaysia</u> , says PM	-1
6301	HLIB Research expects a record-smashing FY20 for Bursa Malaysia	1

Figure 3: Uncapitalized and Capitalized "Malaysia" in News Headlines from train.csv.

We will be using the nltk library and its packages such as "wordnet", "stopwords", "average_perceptron_tagger" and "punkt" to do data preprocessing. An example of the code is shown in Figure 4.

```
2 import nltk
3 from nltk.stem import WordNetLemmatizer
4 from nltk.corpus import stopwords
5 from nltk.tokenize import word_tokenize
6 import string
7 nltk.download('wordnet')
8 nltk.download('stopwords')
9 nltk.download('averaged_perceptron_tagger')
10 nltk.download('punkt')
11
12 def preprocessing(sentence_data):
13     set_stopwords = set(stopwords.words('english'))
14     lemmatizer = WordNetLemmatizer()
15     for sentence in sentence_data:
16         #remove punctuations from sentences
17         no_punctuation_sentence = sentence.translate(str.maketrans('', '', string.punctuation))
18         #split sentences into tokens
19         tokens = word_tokenize(no_punctuation_sentence)
20         #lowercase tokens, remove stopwords and lemmatize tokens
21         preprocessed_tokens = [lemmatizer.lemmatize(token.lower()) for token in tokens if token.lower() not in set_stopwords]
22         #join back all preprocessed tokens into sentence format
23         preprocessed_sentence = " ".join(preprocessed_tokens)
24         #replace sentence
25         sentence_data = sentence_data.replace(to_replace = sentence, value = preprocessed_sentence)
26     return sentence_data
```

Figure 4: Code of Data Preprocessing.

Glove Word Embedding Vector

Next, we will be applying Glove word embedding to our data. Word embedding is a technique to represent the words in a semantically-meaningful dense valued vector. [1] There are several reasons to apply the word embedding. Firstly, word embedding can preserve the syntactical and semantic information by extracting features into a vector. Secondly, insufficient amount of training data, word embedding can provide generalization and boost the performance by summarizing the similarities of tokens.

Besides that, we are using word embedding, specifically the Glove method. Glove method is published by Stanford, Glove method is preferable compared to the other methods such as the Continuously Bag of Words (CBOW) method or Skip-Gram method. CBOW method and Skip-Gram method are used to predict the context word from a targeted word while Glove method performs matrix factorization on word-context matrix. [2] The training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

However, the Glove word embedding model doesn't cover all the words or tokens. When we apply the Glove word embedding to our news headlines, if the words is present in the Glove embedding model, the word will be placed with an amount of relevant features of vector, else if the word is not found in the Glove model, the vector will be replaced with multiples of 0 to represent zero feature. The amount of relevant features can be selected by the user and in this experiment we will be using 100 which can be obtained from glove.6B.100d.txt. Figure 5 displays the matrix of 100 features of "malaysia" from glove.6b.100d.txt.

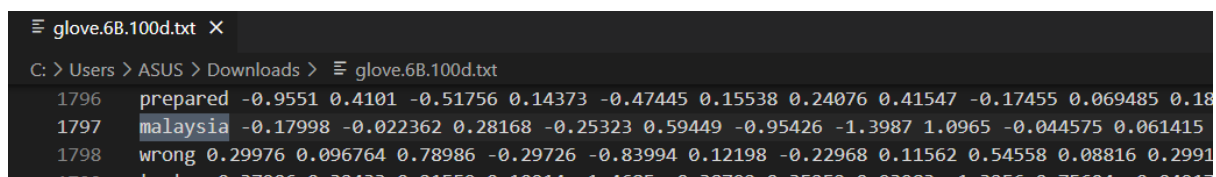
The image is a screenshot of a text editor window titled 'glove.6B.100d.txt'. The window shows a list of word vectors. The second line, corresponding to index 1797, is highlighted and shows the vector for the word 'malaysia'. The vector consists of 100 numerical values, with the first few being -0.17998, -0.022362, 0.28168, -0.25323, 0.59449, -0.95426, -1.3987, 1.0965, -0.044575, and 0.061415. The window also shows the file path 'C: > Users > ASUS > Downloads > glove.6B.100d.txt' and other lines of the file, such as '1796 prepared' and '1798 wrong'.

Figure 5: Matrix of 100 Features of “malaysia” from glove.6b.100d.txt.

Besides the Glove, we will need to use the gensim library to load the Glove word embedding into word2vec and eventually create a Glove model with KeyedVectors function. Next, we will define a user defined function for vectorization. Inside the function, we will try to match each token with the words that exist in the Glove model. [5] If the token exists, we will get the vector of the word from the Glove model and append it into a new list. Otherwise, the tokens that are not found in the Glove model will have zeros in its vector. We will run this function with each dataset including the Glove model, thus producing tokens with features that are in vector format and it is readily available for neural network model uses. An example of the codes is displayed in Figure 6.

```

1 import numpy as np
2 import tensorflow as tf
3 from tensorflow import keras
4 from tensorflow.keras.utils import to_categorical
5
6 def vector_representation(model, data, label):
7     not_found = []
8     #to determine the dimensionality of vector
9     D = model.get_vector('king').shape[0]
10
11     X = []
12     max = 0
13     emptycount = 0
14     for sentence in data:
15         tokens = word_tokenize(sentence)
16         vecs = []
17         m = 0
18
19         for word in tokens:
20             try:
21                 vec = model.get_vector(word)
22                 vecs.append(vec)
23                 m += 1
24             except KeyError:
25                 not_found.append(word)
26                 vecs.append(np.zeros(D))
27         if len(vecs) > 0:
28             vecs = np.array(vecs)
29             X.append(vecs)
30             if len(vecs) > max:
31                 max = len(vecs)
32         if np.sum(vecs)==0:
33             emptycount += 1
34     print("Number of samples with no words found: %s / %s" % (emptycount, len(data)))
35     print(not_found[:5])
36     padded_X = keras.preprocessing.sequence.pad_sequences(X, maxlen=64, dtype='float32')
37     encoded_sentiment = to_categorical(label, num_classes=3)
38
39     return padded_X, encoded_sentiment

```

Figure 6: Code of Vectorization.

Neural Network Model

After we have successfully word embedding our data, finally the last step is to build a neural network and run the data in it. Our neural network model is a Bidirectional Long-Short Term Memory Network Model (LSTM), it consists of 3 layers and it has 3 different outputs which are positive, neutral and negative. The design of this model is very simple to showcase the Bidirectional LSTM Model without any additional layers involved. Figure 7 below illustrates the overview of the neural network model.

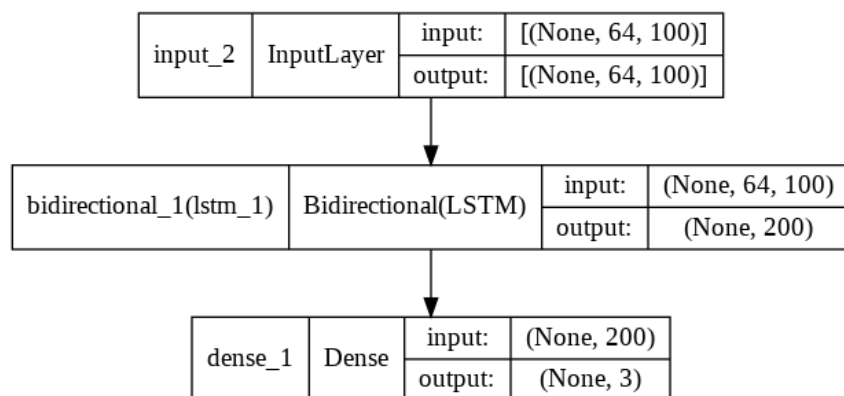


Figure 7: Overview of Bidirectional LSTM Model

We have the first layer to accept our input, the dimension of the input should be 64 rows and 100 columns. Next, the second layer is the Bidirectional Long-Short Term Memory (LSTM) Layer, the activation function is Rectified Linear Unit (ReLU) and the output from this layer can get information from the past and future states simultaneously due to the design of bidirectional. [3] Figure 8 below shows the structure of the Bidirectional LSTM Layer. ReLU is a linear piecewise function that gives the input either positive or zero result and there is infinite range of positive values, hence this will result in the model able to map the positive values but unable to map the negative values appropriately, however it is also able to overcome the vanishing gradient problem, allowing the models to learn faster and perform better. [4] A depiction of the ReLU graph can be seen in Figure 9.

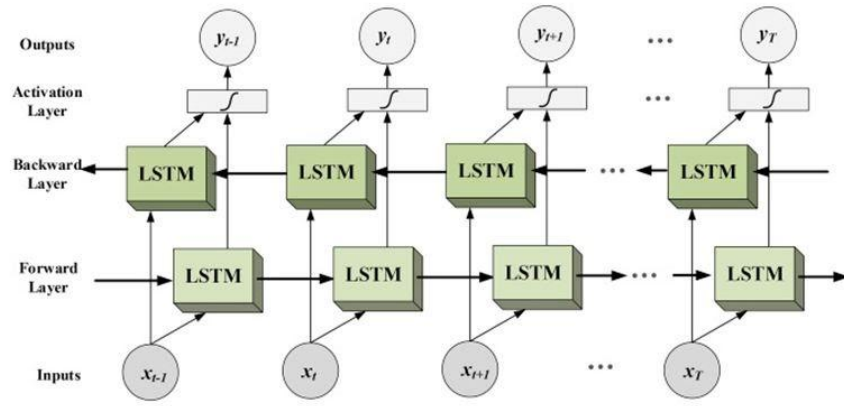


Figure 8: Bidirectional Long-Short Term Memory (LSTM) Layer.

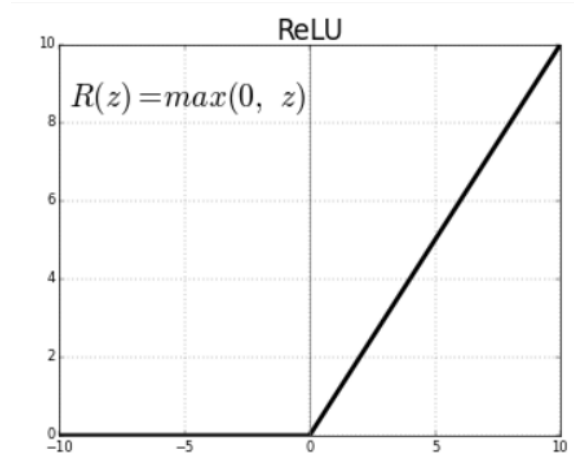
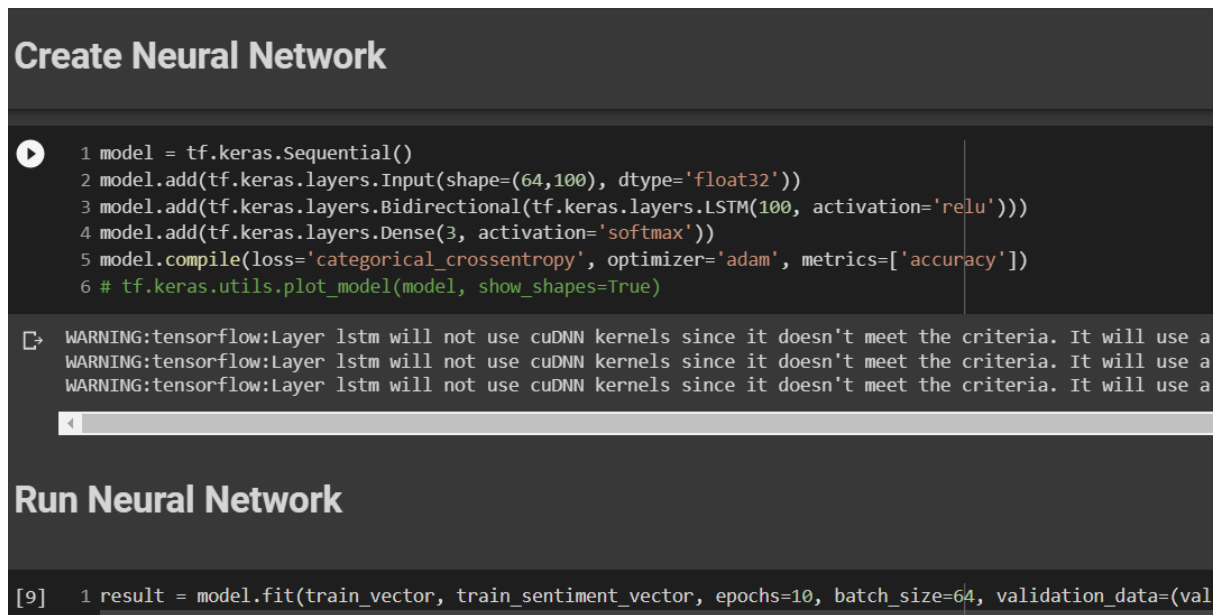


Figure 9: Graph of Rectified Linear Unit (ReLU).

Lastly, the third layer is the output layer and the activation function used is the softmax function. Softmax function is used for the multiclass classification purpose, the output has the probability in each “positive”, “negative” or “neutral” classification, the highest values will be selected as the outcome. The sum of the probabilities in all three classification is 1 and the values of each classification will be 0 to 1.

To create the neural network layers, we need to use the tensorflow library, we make a variable to the model and add the layers to it by programming it sequentially. After adding all layers, we will compile the model with a cross entropy loss function, adamax optimizer and use accuracy as the metric. Last but not least, we run the model with 10 epochs by using the fit function and use the validation data for the validation process. An example of the code for building a neural network model is shown in Figure 10.



```
Create Neural Network

1 model = tf.keras.Sequential()
2 model.add(tf.keras.layers.Input(shape=(64,100), dtype='float32'))
3 model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(100, activation='relu')))
4 model.add(tf.keras.layers.Dense(3, activation='softmax'))
5 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
6 # tf.keras.utils.plot_model(model, show_shapes=True)

WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a

Run Neural Network

[9] 1 result = model.fit(train_vector, train_sentiment_vector, epochs=10, batch_size=64, validation_data=(val
```

Figure 10: Code of Building Bidirectional LSTM Model.

Evaluation & Analysis

Once we are done with training the neural network, we will reuse the model to test on the test data to evaluate its accuracy as well as the loss. We will use the evaluation function of the model with the test data. An example of the evaluation code is shown in Figure 11.

```
1 loss, acc = model.evaluate(test_vector, test_sentiment_vector)
2 print("Loss: {}, Accuracy: {}".format(loss, acc))
```

Figure 11: Code of Bidirectional LSTM Model's Evaluation.

To analyze the performance in visual presentation, we will be using sklearn and seaborn library to make use of confusion matrix. We will use a 3x3 confusion matrix and each row and column will represent each three classification, namely “Neutral”, “Positive” and “Negative” in this orderly manner for the to_categorical representation of 0, 1, 2. The y-axis is the actual classification while the x-axis serves as prediction from the model. As for the cell of the matrix, it represents the probability of the actual classification with predicted

classification from the neural network model. The code for the confusion matrix is shown in Figure 12. For example, in Figure 13 a confusion matrix, the model successfully predicted the news headlines of neutral sentiment correctly with 0.6 probability but predicted positive sentiment for the news headline of actual neutral sentiment with 0.27 probability.

```
1 from sklearn.metrics import confusion_matrix
2 import seaborn as sns
3
4 labels=['Neutral','Positive','Negative']
5 pred = model.predict(test_vector)
6 pred_cat = tf.argmax(pred, axis=-1)
7 actual_cat = tf.argmax(test_sentiment_vector, axis=-1)
8 confusion_mat = tf.math.confusion_matrix(actual_cat, pred_cat)
9 confusion_mat = confusion_mat/confusion_mat.numpy().sum(axis=1)[:, tf.newaxis]
10 sns.heatmap(confusion_mat, annot=True, xticklabels=labels, yticklabels=labels)
11 plt.xlabel("Predicted")
12 plt.ylabel("Actual")
```

Figure 12: Code of Confusion Matrix.



Figure 13: Example of Confusion Matrix

Result & Analysis

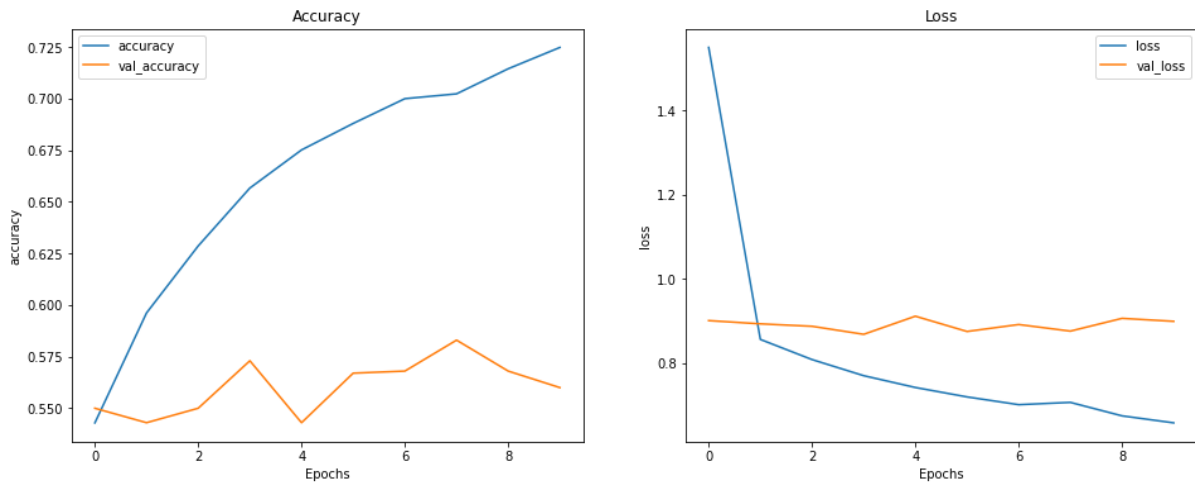


Figure 14: History of Accuracy and Loss Values from Bidirectional LSTM Model.

Based on Figure 14, we can observe that the training accuracy increases linearly from 0.543 to 0.725 while the training loss decreases from 1.55 until 0.657 for the whole 10 rounds of epoch. Conversely, validation accuracy fluctuated in the range of 0.9 and 0.8, while the validation loss also fluctuated in the range of 0.54 to 0.58 for the whole 10 rounds of epoch. This implies that the training data is fit for the model but not the validation data.

The training accuracy and loss are performing as expected, the model is able to increase the accuracy while decreasing the loss. On the other hand, the validation accuracy and loss didn't perform as expected, they fluctuated and didn't progress significantly as the training accuracy did. This situation can be due to several factors, one could be the size of validation data isn't big enough. If the size of validation data isn't big enough, the small changes from data can cause huge fluctuation, if the size of validation is more sufficient, small changes from the data would cause the changes in graph insignificantly and only substantial amount of changes can cause the graph to increase or decrease. Another possible factor is that the validation data isn't representative of the whole news headlines. In addition, our model has a very simple design and it fails to train the validation data accurately.

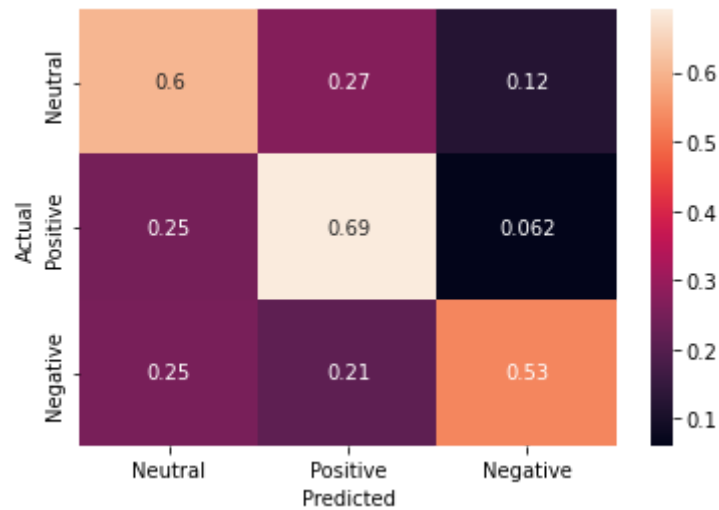


Figure 15: Confusion Matrix from Bidirectional LSTM Model.

From Figure 15, a 3x3 confusion matrix is used to illustrate the actual semantic against the semantic prediction from the neural network, we can use this confusion matrix to analyze the performance of the neural network model. This Bidirectional LSTM Model successfully predicted 60% of the neutral sentiment, 69% of the positive sentiment and 53% of the negative sentiment correctly.

Conclusively, this simple Bidirectional LSTM Model has managed to perform the sentiment analysis of news headlines on test data with 0.623 accuracy and 0.838 of loss. The model has fair and evenly good probabilities in predicting the sentiment correctly in each three classification. Moving forward, we will be making improvements and adjustments to this model to analyze its performance and compare it with the first neural network model.

Improvement on Model

After our first design of the Bidirectional LSTM Model, we can still make some changes to the layers to further improve the model. First, we add a masking layer before our bidirectional Long-Short Term Memory (LSTM) layer. The masking layer is used to skip processing some of the input due to the padding in Glove word embedding vectors. Additionally, we add a 0.5 of recurrent dropout to the bidirectional LSTM layer, it is used to prevent overfitting. Lastly, we will add a new layer after the bidirectional LSTM layer with ReLU activation function to increase the weights in the network. Figure 16 depicts the overview of an improved neural network model.

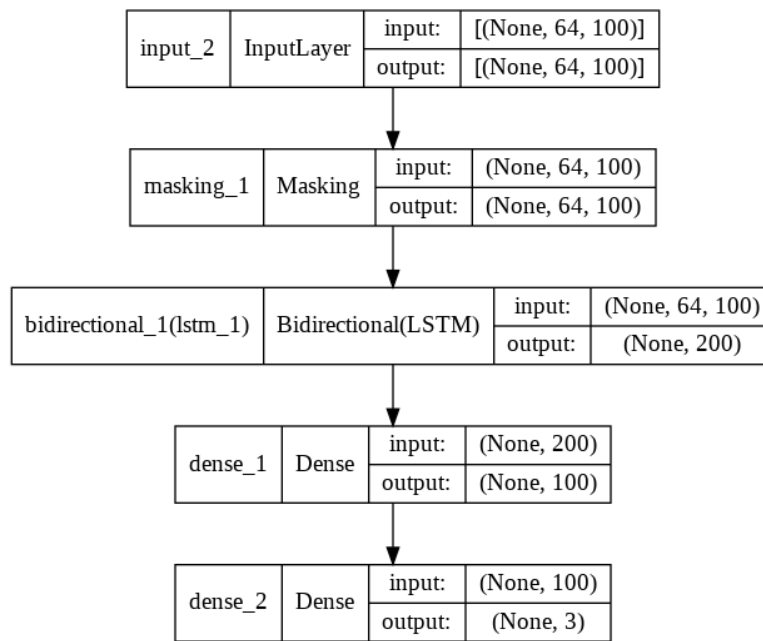


Figure 16: Overview of Improved Bidirectional LSTM Model.

Result from Improved Model

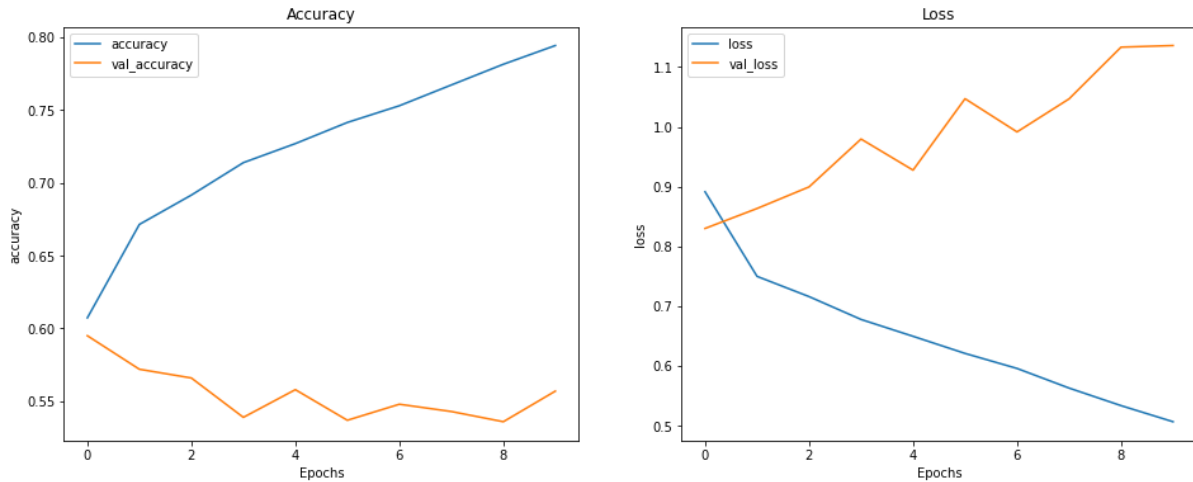


Figure 17: History of Accuracy and Loss Values from Improved Bidirectional LSTM Model.

Based on Figure 17, we can observe that the training accuracy increases linearly from 0.617 to 0.811 while the training loss decreases from 0.829 until 0.467 for the whole 10 rounds of epoch. On the other hand, validation accuracy fluctuated between 0.55 to 0.58, while the validation loss increased from 0.89 to 1.19 for the whole 10 rounds of epoch.

Refer back to Figure 14, we can compare this accuracy and loss values with our first model, the training accuracy and training loss were performing the same as in the first model, both models have their graphs increase and decrease linearly. However, the validation loss doesn't fluctuate as much as in the first model. This indicates that the improved model managed to approximate the validation data better than the first model even though the validation loss is increased.

Besides the pattern of the graph, both models have a very similar result in training and validation. The first model has the highest training accuracy as 0.72 while the improved model has its highest training accuracy at 0.81. The first model also has its training loss as lowest as 0.65 while the improved model has its lowest training loss at 0.46. The changes can be said as substantially small but the training accuracy has been improved and the training loss is also lowered.



Figure 18: Confusion Matrix from Improved Bidirectional LSTM Model.

From Figure 18, a 3x3 confusion matrix is used to illustrate the actual semantic against the semantic prediction from the neural network, we can use this confusion matrix to analyze the performance of the neural network model. The bidirectional LSTM model successfully predicted 52% of the neutral sentiment, 66% of the positive sentiment and 63% of the negative sentiment correctly.

Compared to the first model in Figure 15, the first model performed better than the improved model with 0.6230 accuracy against 0.6079 accuracy. The first model had better probability in predicting neutral and positive sentiment correctly with 0.6 and 0.69 probabilities compared to the improved model with 0.52 and 0.66 probabilities respectively. This indicates that even if the model has been improved, the accuracy does not guarantee a better result in each classification and can even be decreased as there are no guarantees in it. Beside that, it is possible to have a better probability in one classification after the improvement although the improved model has lower accuracy than the first model.

Conclusively, the improved Bidirectional LSTM Model has managed to produce a little better performance compared to the first model in testing data. However, the performance of the improved model with the test data is not better as in the first model. The improved model managed to produce an accuracy of 0.608 and loss of 1.015 while the first model has an accuracy of 0.623 and loss of 0.838. This indicates that an improved model doesn't reflect a better result but a more precise approximation result to the data. This could also potentially indicate that the size of 1000 test data isn't sufficient, because the adding of layers will increase the weight and without a sufficient amount of data, the large network might overfit, hence lowering the accuracy and increasing the loss at the same time.

A compilation of results between two Bidirectional LSTM Models for each dataset is shown in Table 1 while Table 2 displays the confusion matrices between two Bidirectional LSTM Models.

		Bidirectional LSTM Model	Improved Bidirectional LSTM Model
training dataset	Accuracy	0.7248	0.8107
	Loss	0.6570	0.4668
validation dataset	Accuracy	0.5600	0.5570
	Loss	0.8986	1.1851
test dataset	Accuracy	0.6230	0.6079
	Loss	0.8383	1.0152

Table 1: Compilation of Results between Two Bidirectional LSTM Models for Each Dataset.

	Predicted Neutral		Predicted Positive		Predicted Negative	
Actual Neutral	0.60	0.52	0.27	0.32	0.12	0.16
Actual Positive	0.25	0.26	0.69	0.66	0.062	0.082
Actual Negative	0.25	0.18	0.21	0.19	0.53	0.63

Bidirectional LSTM Model	-	Blue
Improved Bidirectional LSTM Model	-	Red

Table 2: Compilation of Confusion Matrices between Two Bidirectional LSTM Models.

Conclusion

In a nutshell, the Bidirectional LSTM Model was able to predict the sentiment of news headlines in three different classifications, namely “Positive”, “Negative” and “Neutral”.

However, when we did some improvements or adjustments to the model to make it improved, the resulting accuracy and loss produced from the improved model doesn't seem to be any improved from the first model. We can conclude that the changes made to the model seem to be only resulting in the model taking more time to train, and the accuracy didn't increase nor as the loss decreases. The improved model can be said that it only reflects a more precise approximation or more accurate result compared to the first model. Notably, the quantity of layers isn't directly proportional to a better result.

Besides the changes to the model, quality controls for the data could also contribute to the better result in accuracy and lower loss. For example, the data collection can include the other age of audience to reflect true sentiment analysis on the news headlines.

Reference

1. N. Latysheva, “Why do we use word embeddings in NLP?,” *Medium*, 10-Sep-2019. [Online]. Available: <https://towardsdatascience.com/why-do-we-use-embeddings-in-nlp-2f20e1b632d2#:~:text=Represent%20words%20as%20semantically%2Dmeaningful,a%20lot%20of%20training%20data>. [Accessed: 28-Jan-2022].
2. S. Bhattacharyya, M. Ramnani, and S. Mukherjee, “Word2Vec vs glove - a comparative guide to word embedding techniques,” *Analytics India Magazine*, 19-Oct-2021. [Online]. Available: <https://analyticsindiamag.com/word2vec-vs-glove-a-comparative-guide-to-word-embedding-techniques/>. [Accessed: 28-Jan-2022].
3. S. Bhattacharyya, M. Ramnani, and S. Mukherjee, “Complete Guide to Bidirectional LSTM (with python codes),” *Analytics India Magazine*, 20-Nov-2021. [Online]. Available: [https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-code/s/#:~:text=Bidirectional%20long%2Dshort%20term%20memory,forward\(past%20to%20future\)](https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-code/s/#:~:text=Bidirectional%20long%2Dshort%20term%20memory,forward(past%20to%20future)). [Accessed: 28-Jan-2022].
4. S. Sharma, “Activation functions in neural networks,” *Medium*, 04-Jul-2021. [Online]. Available: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. [Accessed: 28-Jan-2022].
5. N. Latysheva, “Why do we use word embeddings in NLP?,” *Medium*, 10-Sep-2019. [Online]. Available: <https://towardsdatascience.com/why-do-we-use-embeddings-in-nlp-2f20e1b632d2#:~:text=Represent%20words%20as%20semantically%2Dmeaningful,a%20lot%20of%20training%20data>. [Accessed: 28-Jan-2022].