COVID-19 Tweets Sentiment Classification

Text Classification Group 17 - Project Group 37
Steven Wen, Yuehan Qi, Songyu Yang, Yuxuan Cai

Motivation

Our motivation for choosing the topic is started from reading the paper 'Deep Learning Based Text Classification: A comprehensive review'. we learn that the typical Text Classification tasks include sentiment analysis, news categorization and topic classification. And out of the three topics we find sentiment analysis the most interesting. And since we are trying to find a problem that is most relevant COVID-19 becomes the target.

Problem Description

We are trying to determine the sentiment from the tweets by analyzing the tweets by using four or more different analysis models. The main objective is to compare the different models and the accuracy to determine the best outcome.

Contribution

Steven Wen: Data preprocessing, RNN(Bi-GRU) model implementation, Motivation, Problem Description, conclusion
Yuehan Qi: LSTM implementation, conclusion and future work
Songyu Yang: CNN implementation, conclusion
Yuxuan Cai: Bi-LSTM implementation, conclusion

Related Work

RNN

For RNN model, compare with the paper Deep Learning Based Text Classification: A comprehensive review. I have used a specific data set and did tuning towards one instead of just implementing a basic model inorder to compare the performance. The strength of the paper compared to ours is that it discusses the same model on different dataset and how each performed, but the weakness is the lack of tuning discussed in the paper. Compare one of the similar data sets that used in the paper – IMDB Dataset which only has two different sentiments, while our dataset has over 5 different sentiments. The dataset we used is more challenging, which results in the accuracy in the paper is around 85% while for 3 sentiment my accuracy is 84% and for 5 is 72%.

CNN

From paper *Fake News Classification Based on Content Level Features*, the authors evaluate four machine learning models and three neural network models by identifying and labeling fake news collected in the Kaggle dataset implementing like CNN with GlobalMaxpool and CNN with DeepNetwork and LSTM to test their performance in fake news classification. Specifically, the author applied the Word2Vec model in all models to increase the accuracy which also inspired me to try to apply the model to increase the accuracy. All the neural network models in the paper exceed the accuracy of 92%, and my model received an accuracy of 74% comparatively. The difficulties of each dataset and structure of the model lead to a disparate result.

LSTM

In the paper *A C-LSTM neural network for text classification,* it combines the CNN network with the LSTM model on sentiment classification data. Similarly, the dataset applied for this model also has 5 labels from extremely negative to extremely positive. By conducting the regularization padding and word vector initialization, the performance of LSTM results in 93.2% and the C-LSTM results in 94.6%, which outperforme all the other models implemented.

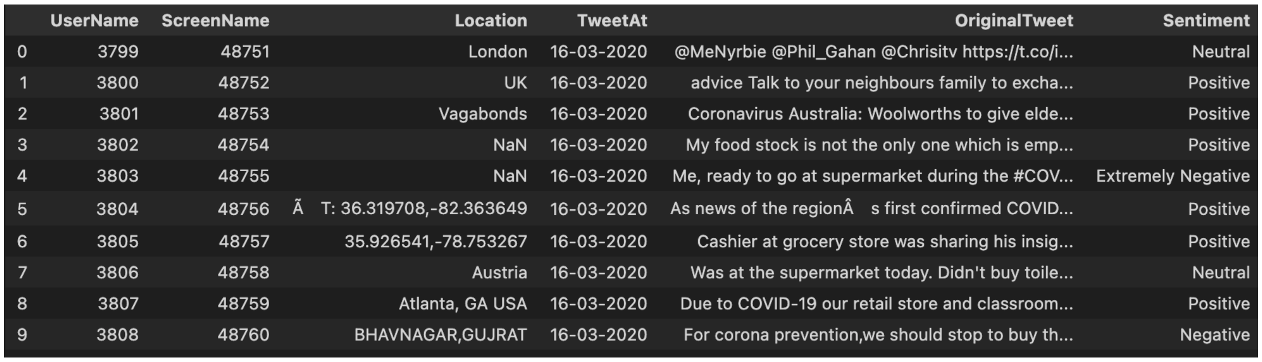
BILSTM

Text classification frameworks include traditional machine learning, such as support vector machines and naive Bayes, which obtain text semantic features and then classify them by using representation methods such as word bag method, Tf-idf or manual feature extraction. Although these methods are simple and easy, they can extract advanced features such as semantics and parts of speech, etc. This will lead to increased computational costs, and another disadvantage is that manually designing features is time consuming and performs poorly in generalization due to the low coverage of different training data sets; In addition, the accuracy of the traditional model is also poor, especially in the multi-text classification level of this project, and the performance is better without deep neural network.
The second is networks such as RNN and CNN. CNN network is suitable for extracting the semantic features of short texts, but it is powerless for the semantics of long distances, so it cannot be used for this project. Although RNN is suitable for long-distance text extraction, it is prone to gradient explosion and gradient disappearance, thus unable to carry out back propagation. However, its variant LSTM network solves this problem well, alleviates the problem of gradient propagation, and can carry out back propagation well. The above work is relevant, but here we use BILSTM+Attention, which is a Bi-directional LSTM network with Attention.

Dataset Used

For the four models implemented in this problem, we used the same dataset and preprocessing illustrated in the following.

We use the tweets that were pulled from Twitter and tag them with sentiment. And we pulled the data set from Kaggle. The data set contains 6 columns and 3798 different tweets, as shown in the figure 1 below is an example of the original data.



	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative
5	3804	48756	Ã T: 36.319708,-82.363649	16-03-2020	As news of the regionÂ s first confirmed COVID...	Positive
6	3805	48757	35.926541,-78.753267	16-03-2020	Cashier at grocery store was sharing his insig...	Positive
7	3806	48758	Austria	16-03-2020	Was at the supermarket today. Didn't buy toile...	Neutral
8	3807	48759	Atlanta, GA USA	16-03-2020	Due to COVID-19 our retail store and classroom...	Positive
9	3808	48760	BHAVNAGAR,GUJRAT	16-03-2020	For corona prevention,we should stop to buy th...	Negative

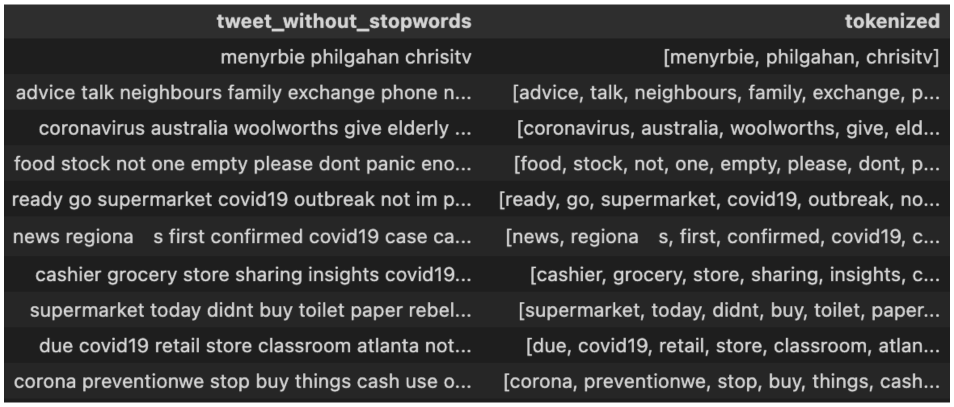
Figure 1: Data Exploration

Missing value and columns dropping

To further analyze the data we performed some data cleaning. First we dropped the duplicated data by calling the built-in function drop_duplicates. As this project is only using the tweets to predict the sentiment we decided to drop all the other columns although they will increase the accuracy of the prediction. To check the missing value we find that there is no value missing.

Text cleaning

To clean the text we changed all the tweets to lower case, removed all the URLs, removed all the punctuation and special words. For stop words, we decided to keep ‘no’ and ‘not’, since they might be important to determine the sentiment. Lastly we applied tokenization to the data set for easier modeling.



tweet_without_stopwords	tokenized
menyrbie philgahan chrisitv	[menyrbie, philgahan, chrisitv]
advice talk neighbours family exchange phone n...	[advice, talk, neighbours, family, exchange, p...
coronavirus australia woolworths give elderly ...	[coronavirus, australia, woolworths, give, eld...
food stock not one empty please dont panic eno...	[food, stock, not, one, empty, please, dont, p...
ready go supermarket covid19 outbreak not im p...	[ready, go, supermarket, covid19, outbreak, no...
news regiona s first confirmed covid19 case ca...	[news, regiona s, first, confirmed, covid19, c...
cashier grocery store sharing insights covid19...	[cashier, grocery, store, sharing, insights, c...
supermarket today didnt buy toilet paper rebel...	[supermarket, today, didnt, buy, toilet, paper...
due covid19 retail store classroom atlanta not...	[due, covid19, retail, store, classroom, atlan...
corona preventionwe stop buy things cash use o...	[corona, preventionwe, stop, buy, things, cash...

Figure 2: Data after cleaning and tokenized

Sentiment Transform

Lastly we transform the sentiment value from categorical to numerical for better prediction.

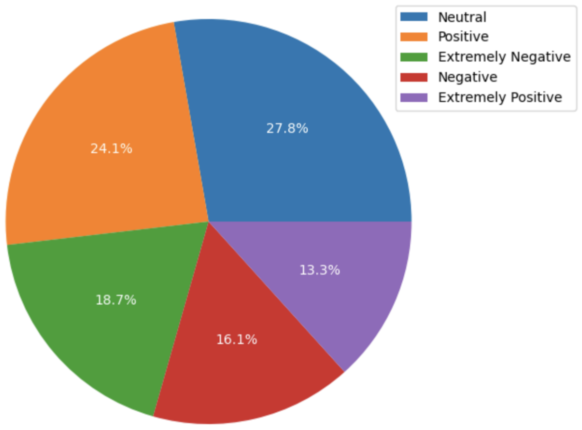
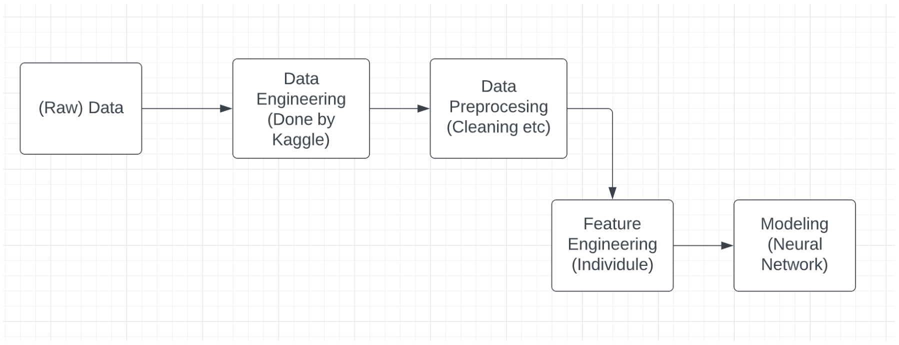


Figure 3: Sentiment Value for Training set

Data Flow

Our data flow is shown in the finger 4.



(Raw) Data
Data Engineering (Done by Kaggle)
Data Preprocesing (Cleaning etc)
Feature Engineering (Individule)
Modeling (Neural Network)

Figure 4: Data Flow

Implementation:

Recurrent Neural Network:

Model Choice:

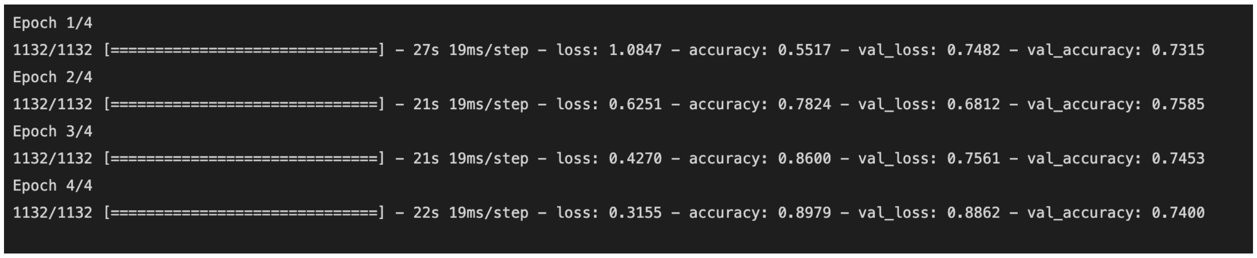
For this project RNN was the first choice as the base model, and to improve the accuracy I tried to import Bi-GRU which has improved the accuracy from 60% to 72%. The following model discussed will be Bi-GRU.

Experimental Setup:

Environment: macOS Monterey 2.3 GHz 8-Core Intel Core i9,16 GB 2667 MHz DDR4
Tensorflow: Use Tensorflow for modeling and test accuracy and loss. To download, use ‘pip install tensorflow’; for import, use ‘import tensorflow’.
Tensorflow.keras.layers: Basic building blocks of neural networks in Keras, installed to perform RNN and other models. To import just ‘import tensorflow.keras.layers’
Sklearn: Use sklearn’s different package for accuracy. This package can be installed by ‘pip install -U scikit-learn’. I am using accuracy and confusion matrix which to import use ‘from sklearn.metrics import accuracy_score’ and ‘from sklearn.metrics import confusion_matrix’

Training/Testing/Validation

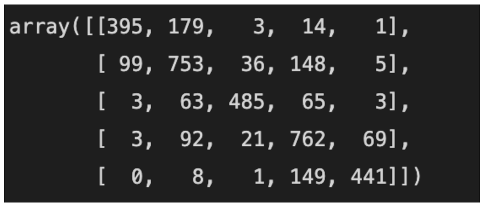
I am using 2 epochs as the final iteration, since I saw significant overtraining after 2 epochs – the training accuracy keeps going up but the testing accuracy is dropping. Shows in the figure below.



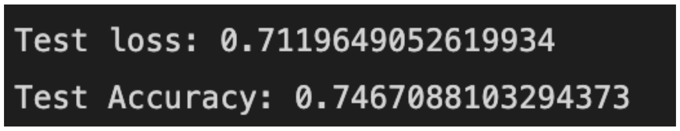
```
Epoch 1/4
1132/1132 [==============================] - 27s 19ms/step - loss: 1.0847 - accuracy: 0.5517 - val_loss: 0.7482 - val_accuracy: 0.7315
Epoch 2/4
1132/1132 [==============================] - 21s 19ms/step - loss: 0.6251 - accuracy: 0.7824 - val_loss: 0.6812 - val_accuracy: 0.7585
Epoch 3/4
1132/1132 [==============================] - 21s 19ms/step - loss: 0.4270 - accuracy: 0.8600 - val_loss: 0.7561 - val_accuracy: 0.7453
Epoch 4/4
1132/1132 [==============================] - 22s 19ms/step - loss: 0.3155 - accuracy: 0.8979 - val_loss: 0.8862 - val_accuracy: 0.7400
```

Figure 5: Training Epoch

In order to perform the Bi GRU I am using sequential keras from Tensorflow - which stacks a bounce of layer to modeling. The first step to text classification is embedding, which is an alternator to one hot encoding and can be used to reduce the dimension. For this particular model, I am setting the variable to the vocabulary size of all the tweets, and the input size to the length of the text after it has been tokenized. And for the embedding dimension I choose the standard 16 dimension. For Bi-GRU I am using 250 as its unit which will be the output shape, because with a small unit the model could run faster but the accuracy will be lower, and higher will be the opposite. After testing 250 will perform the best. In order to make the model have less chance of overfitting I added a dropout layer which will randomly drop an input with a frequency at a certain rate. To test the accuracy, I have implemented loss, accuracy and confusion matrix.



```
array([[395, 179,   3,  14,   1],
       [ 99, 753,  36, 148,   5],
       [  3,  63, 485,  65,   3],
       [  3,  92,  21, 762,  69],
       [  0,   8,   1, 149, 441]])
```



```
Test loss: 0.7119649052619934
Test Accuracy: 0.7467088103294373
```

Figure 6: Test Accuracy

LSTM

Experimental Setup

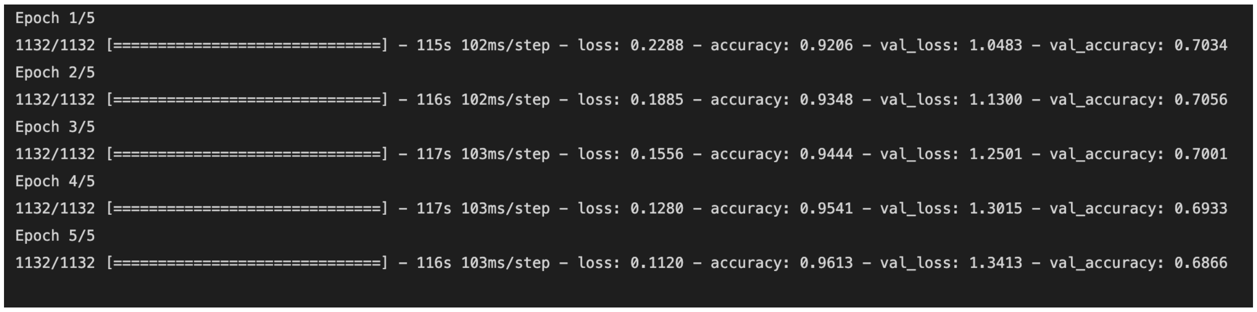
The system configuration of mine is Windows 11 version 21H2 with the processor Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz and 32.0 GB RAM. I import tensorflow to build the LSTM model, in which 'tensorflow.keras.layers' are used to add layers into the model. I also import the confusion matrix and accuracy score from sklearn for evaluating the performance.

Model Building

The LSTM model consists of 4 parts. First, we start with embedding the sequence input which converts words into vectors. Then we pass them on to the LSTM layer to convert them into smaller features. After this, we go through the flatten layer to reduce features. Finally, we deliver the dense for classification and use a softmax activation function to filter the final output.

Training/Testing/Validation

Four epochs are run for training since the fifth epoch starts to overtrain with training accuracy increased but validation accuracy dropped as shown in the below figure. Also, the dropout is set as 0.4 to prevent overfitting.

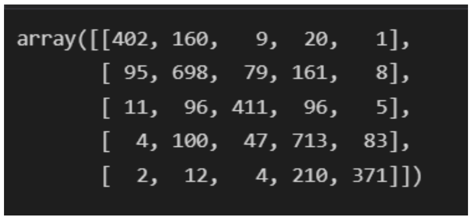


```
Epoch 1/5
1132/1132 [==============================] - 115s 102ms/step - loss: 0.2288 - accuracy: 0.9206 - val_loss: 1.0483 - val_accuracy: 0.7034
Epoch 2/5
1132/1132 [==============================] - 116s 102ms/step - loss: 0.1885 - accuracy: 0.9348 - val_loss: 1.1300 - val_accuracy: 0.7056
Epoch 3/5
1132/1132 [==============================] - 117s 103ms/step - loss: 0.1556 - accuracy: 0.9444 - val_loss: 1.2501 - val_accuracy: 0.7001
Epoch 4/5
1132/1132 [==============================] - 117s 103ms/step - loss: 0.1280 - accuracy: 0.9541 - val_loss: 1.3015 - val_accuracy: 0.6933
Epoch 5/5
1132/1132 [==============================] - 116s 103ms/step - loss: 0.1120 - accuracy: 0.9613 - val_loss: 1.3413 - val_accuracy: 0.6866
```

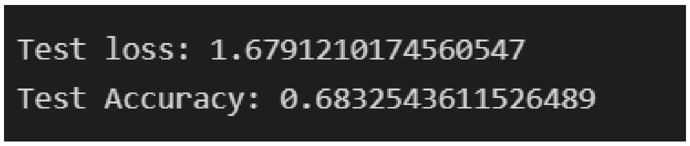
Figure 7: Five epochs for training

Based on the number of total parameters in the dataset, I have also set the embedding dimension as 16 and the hidden layer as 256. Through observation of the performance with different settings of parameters, I set the batch size as 32, validation split as 0.12 to generate optimal test accuracy.

With all parameters set, the final test accuracy is 68.3% as shown below , which is 8.3% more accurate than the base model RNN.



```
array([[402, 160,   9,  20,   1],
       [ 95, 698,  79, 161,   8],
       [ 11,  96, 411,  96,   5],
       [  4, 100,  47, 713,  83],
       [  2,  12,   4, 210, 371]])
```



```
Test loss: 1.6791210174560547
Test Accuracy: 0.6832543611526489
```

Figure 8: Confusion Matrix and Test Accuracy

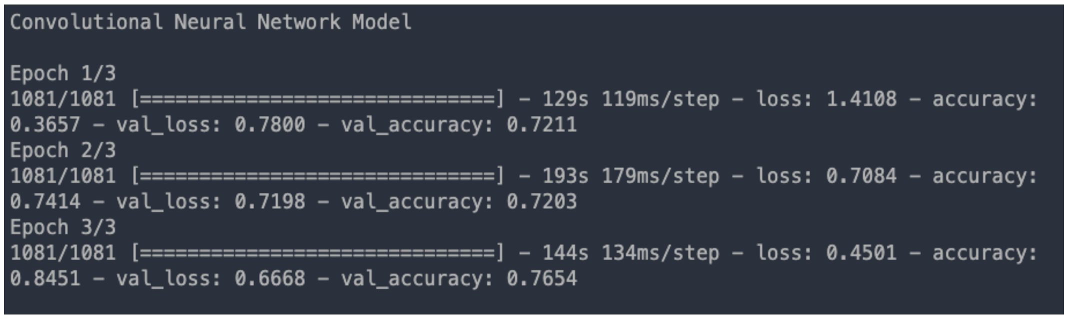
CNN

Experimental Setup

My experiment is based on the macOS Monterey system version 12.5 and an Apple M1 chip with 8 GB of memory. The environment where I installed the libraries is built by Anaconda, and the compiler is VS Code with the extension of Jupyter Notebook. The major implementation environment is based on Python 3.8.15 and TensorFlow 2.4.0.

Training/Testing/Validation

In my own model, which is based on SOTA, I used three epochs with 1081 steps each and a small batch size of 32 to encourage the network to explore more and perform better during training. After some experiments, the learning rate is set to default since 0.01 will produce the best result. The basic structure of my convolutional neural network contains an Embedding layer that processes the tokenized text into a vector with a vector dimension of 15 and the dimension of the input data, two convolutional layers with size of 512 and a size of 256 as standard in which it will extract the features by performing a dot product between the kernel and the receptive field with the Relu activation function, and two Max pooling layers after each convolutional layer downsampling the output of the network to reduce the spatial size of the representation and decrease computation and weights. I also add two dropout layers with a frequency of 0.5 and 0.4 rate at each step during training, which helps prevent overfitting. After several experiments, dropping a neuron with a probability of 0.4 gets the highest variance for the distribution, which can effectively prevent overtraining. Finally, after many trials and errors, two dense layers with a size of 64 are set up and the data are activated by Relu function, in which it maps the representation between the input data and the output.Meanwhile, the size of the last dense layer should have the same dimension as the sentiment label.



```
Convolutional Neural Network Model

Epoch 1/3
1081/1081 [==============================] - 129s 119ms/step - loss: 1.4108 - accuracy:
0.3657 - val_loss: 0.7800 - val_accuracy: 0.7211
Epoch 2/3
1081/1081 [==============================] - 193s 179ms/step - loss: 0.7084 - accuracy:
0.7414 - val_loss: 0.7198 - val_accuracy: 0.7203
Epoch 3/3
1081/1081 [==============================] - 144s 134ms/step - loss: 0.4501 - accuracy:
0.8451 - val_loss: 0.6668 - val_accuracy: 0.7654
```

Figure 9: Training Epoch & Accuracy

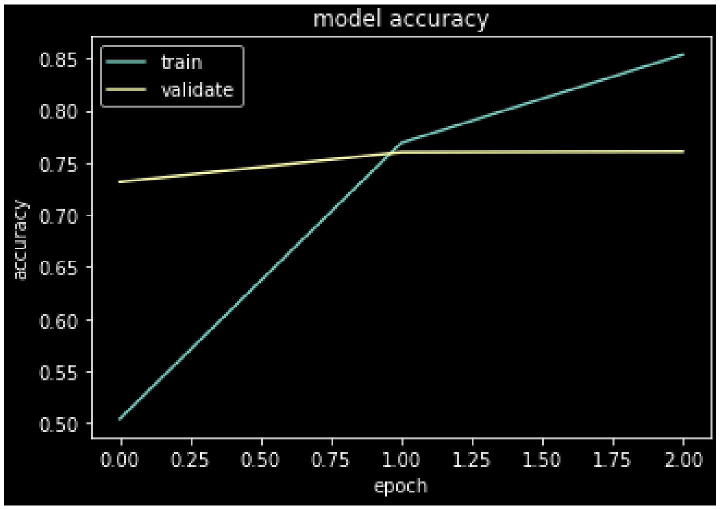
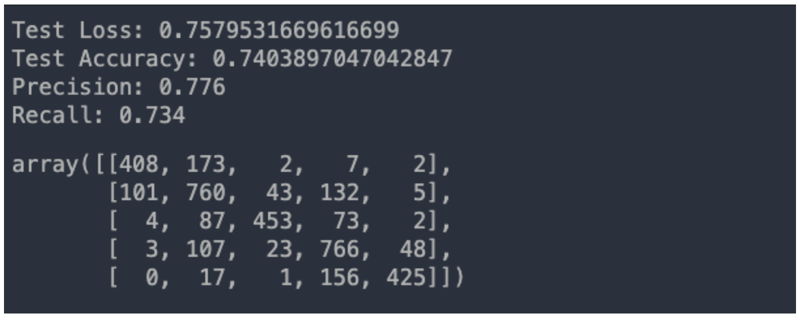


Figure 10: Training Accuracy Charts

The validation is set to 0.15, which means that 0.15 percent of the training data is to be used as validation data, and the accuracy of the training data and validation data is shown above. The training accuracy increased continuously with a dramatically decreased training loss, but the validation accuracy did not show a good result.



```
Test Loss: 0.7579531669616699
Test Accuracy: 0.7403897047042847
Precision: 0.776
Recall: 0.734

array([[408, 173,   2,   7,   2],
       [101, 760,  43, 132,   5],
       [  4,  87, 453,  73,   2],
       [  3, 107,  23, 766,  48],
       [  0,  17,   1, 156, 425]])
```

Figure 11: Testing Result & Evaluation

I evaluate my own model, which is based on SOTA, by calculating the loss and accuracy. I also include a precision, recall, and confusion matrix to assess the performance of the model on a five sentiment text classification task.

BI-LSTM+Attention

Model Building

The model I chose is as follows: Bi-LSTM + Attention refers to the bidirectional LSTM model plus the Attention mechanism. This model uses the attention mechanism of neural network and the bidirectional long and short term memory network to capture important semantic information in sentences and aggregate it for processing, so as to obtain the emotional tendency of the text and complete the task of text classification.
Our final model adds the Attention mechanism to the LSTM network, which is essentially inspired by the human visual attention mechanism. The main idea is that when our vision perceives things, it doesn't usually look at a scene from end to end all at once, but rather it tends to look at specific parts as needed. And when we find that a scene frequently shows something we want to observe in a certain part, we learn from it so that we can focus on that part in the future when similar scenes occur. From the perspective of the model, Attention is essentially a series of attention allocation coefficients. Assigning different attention weights to each token will be combined and finally applied to text classification. The model structure of Bi-LSTM + Attention is as follows:
Bi-LSTM + Attention adds the Attention layer to the Bi-LSTM model. In Bi-LSTM, we will use the output vector of the last time sequence as the feature vector, and then conduct softmax classification. The added Attention is to calculate the weight of each time sequence first. Then, the vectors of all time sequences are weighted and added as feature vectors, and then softmax classification is performed.
The role of each layer of the model is as follows:

1. Input layer: input sentence to this model;
2. Embedding layer: map each word into a low dimension vector;
3. LSTM layer: utilize BLSTM to get high level features from step
4. Attention layer: produce a weight vector, and merge word-level features from each time step into a sentence-level feature vector, by multiplying the weight vector;
5. Output layer: the sentence-level feature vector is finally used for relation classification.

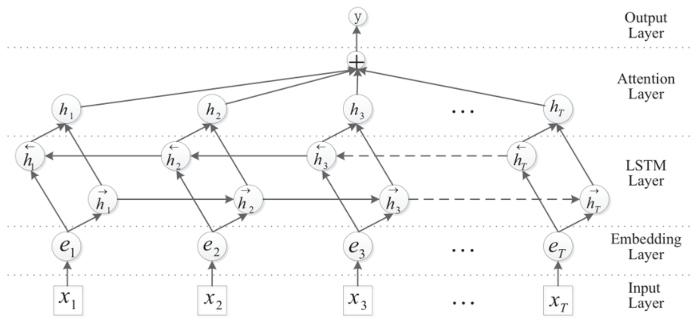

y
Output Layer
Attention Layer
h_1 h_2 h_3 … h_T
LSTM Layer
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
Embedding Layer
e_1 e_2 e_3 … e_T
Input Layer
x_1 x_2 x_3 … x_T

Figure 1: Bidirectional LSTM model with Attention

Experimental Setup

This section starts with the configuration of the running environment. In this task, I used Pytorch deep learning framework. PyTorch is an open source deep learning framework developed by the team of Facebook Artificial Intelligence Research Institute. Its bottom layer is based on Torch, but its implementation and application are all completed by python. In addition, I also used many auxiliary libraries, including Scikit-learn, NLTK, Pandas, Numpy, etc., to complete the development of this task.

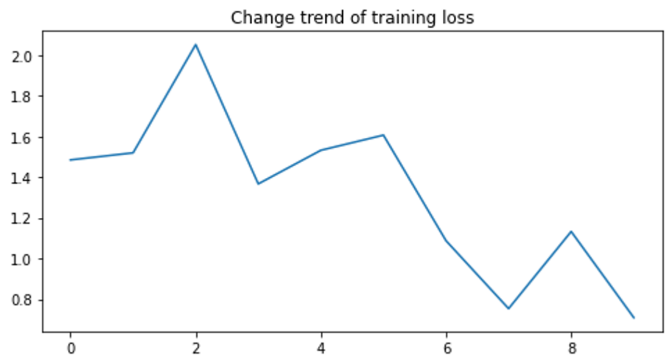
In addition, during the task development, I conducted experiments and training on the Python jupyter notebook.

Training of model

In this section, we'll train BILSTM-Attention on the twitter dataset. The data set is first divided into a training set and a test set. The training is performed on the training set, and then the performance of the model is tested on the test set. We used the built-in functions in Pytorch to complete the batch training, reducing training time and improving efficiency.
The parameters in model training are shown in the table below:

Parameter	**Value**
Embedding dim	50
Hidden size	64
Batch size	32
Criterion	Cross Entropy Loss
Optimizer	Adam algorithm
Epoch	30

The following is the decline trend of loss in training:

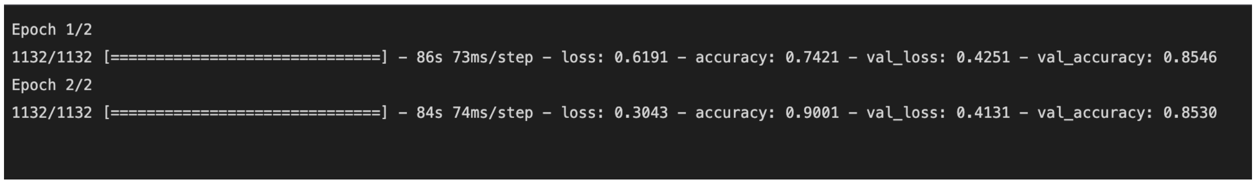


Results and Discussion

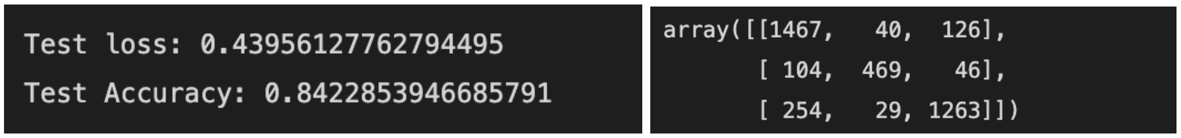
	Group 17	SOTA
RNN (Bi- GRU)	74.7%	84.1%
CNN	74.0%	84.4%
LSTM	68.3%	84.1%
BI-LSTM+Attention	78.2%	84.0%

RNN

First cause of the lower accuracy of our RNN is due to the sentiment difference. The SOTA is using three different sentiment values where we are using five. To show the accuracy of our tuning we have changed the sentiment to match and get the following result.



```
Epoch 1/2
1132/1132 [==============================] - 86s 73ms/step - loss: 0.6191 - accuracy: 0.7421 - val_loss: 0.4251 - val_accuracy: 0.8546
Epoch 2/2
1132/1132 [==============================] - 84s 74ms/step - loss: 0.3043 - accuracy: 0.9001 - val_loss: 0.4131 - val_accuracy: 0.8530
```



```
Test loss: 0.43956127762794495
Test Accuracy: 0.8422853946685791
```

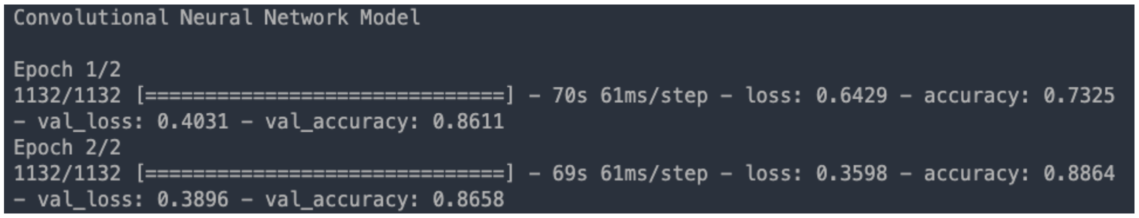
```
array([[1467,   40,  126],
       [ 104,  469,   46],
       [ 254,   29, 1263]])
```

Figure 12: Three sentiment value accuracy

We are keeping the five sentiment values as one of the originality of the project. And in the SOTA the writer has overtrain, as we can see in the history although his testing accuracy is higher his validation accuracy is dropping. The low accuracy also could be caused by the model is a newly published model that there are not many implementations of it.

CNN

The figures following depict SOTA's performance on three sentiment value classification tasks. The training, validation, and testing accuracy of its model are respectively 88.6%, 86.5%, and 84.4%, whereas my modified model based on SOTA on sentiment value classification is respectively 85.6%, 76.0%, and 74.0%.



```
Convolutional Neural Network Model

Epoch 1/2
1132/1132 [==============================] - 70s 61ms/step - loss: 0.6429 - accuracy: 0.7325
- val_loss: 0.4031 - val_accuracy: 0.8611
Epoch 2/2
1132/1132 [==============================] - 69s 61ms/step - loss: 0.3598 - accuracy: 0.8864
- val_loss: 0.3896 - val_accuracy: 0.8658
```

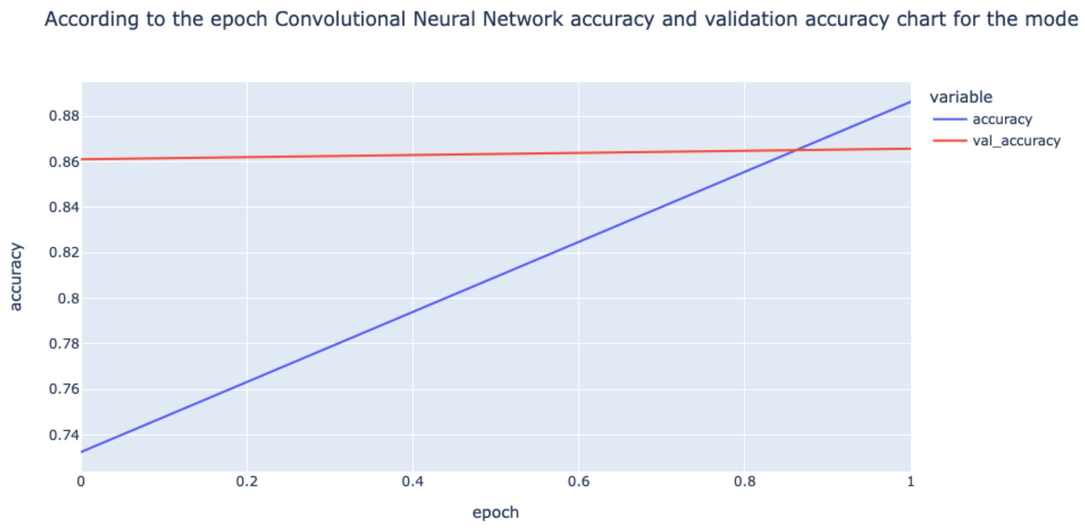
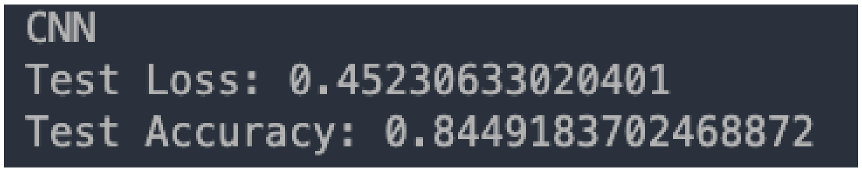


Figure 13: Three sentiment Training & Validation accuracy



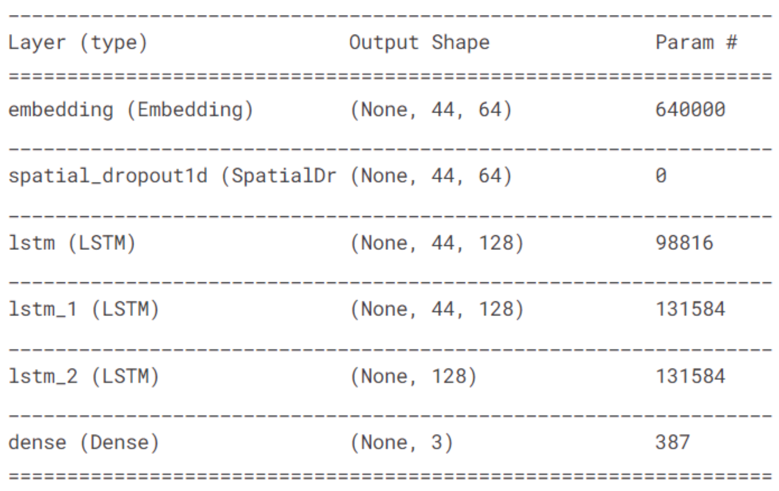
```
CNN
Test Loss: 0.45230633020401
Test Accuracy: 0.8449183702468872
```

Figure 14: Three sentiment Testing accuracy

As the summary made by Steven in RNN, the sentiment difference can make it more difficult for the base mode to train, which leads to low accuracy. By comparing the accuracy with other members of the group, the accuracy of everyone was very similar. After critical thinking about the cause of low accuracy, I generate two points. Firstly, data processing may need some enhancements or modifications since it may not be well applicable to our model. Secondly, the base model may not be good at this dataset or need more improvements since its accuracy is also around 85%, which is comparatively lower than the current CNN classification task. In my code, I explained and demonstrated my intention to increase the overall accuracy by implementing the Word2Vec model in my CNN model. Although plenty of time was spent trying to apply the Word2Vec model to my Embedding layer, it finally failed since the results were less than my expectations. Meanwhile, Only after lots of trials and errors and numerous waits for the model training to be completed, I obtain the optimal results of my model, which are around 74%.

LSTM

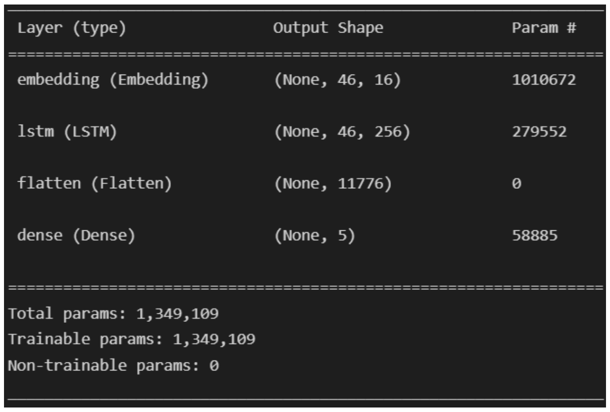
By comparing my model with the SOTA LSTM model, I realize that it generates higher accuracy with three more layers. First, the SpatialDropout1D layer after the embedding improves the independence among features compared with the regular dropout. Then it adds two more LSTM layers which is the main difference that promotes its accuracy.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 44, 64)            640000
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 44, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 44, 128)           98816
_________________________________________________________________
lstm_1 (LSTM)                (None, 44, 128)           131584
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense (Dense)                (None, 3)                 387
=================================================================
```

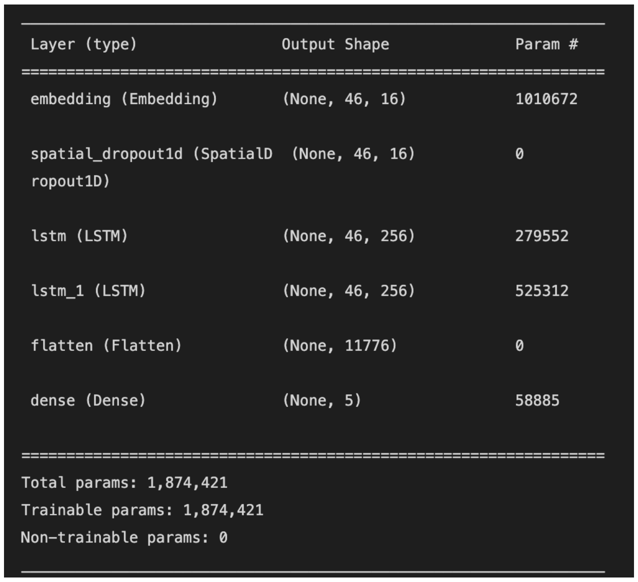
Figure 15:. SOTA LSTM Model

Learned from the SOTA LSTM model, by applying the SpatialDropout1D and one more LSTM layer to my original model, the accuracy improves from 68.3% to 70.2% with the tradeoff of greater time consumption.



```
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 46, 16)            1010672

 lstm (LSTM)                 (None, 46, 256)           279552

 flatten (Flatten)           (None, 11776)             0

 dense (Dense)               (None, 5)                 58885

=================================================================
Total params: 1,349,109
Trainable params: 1,349,109
Non-trainable params: 0
_________________________________________________________________
```

Figure16: Original LSTM Model



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 46, 16)            1010672

 spatial_dropout1d (SpatialD  (None, 46, 16)           0
 ropout1D)

 lstm (LSTM)                 (None, 46, 256)           279552

 lstm_1 (LSTM)               (None, 46, 256)           525312

 flatten (Flatten)           (None, 11776)             0

 dense (Dense)               (None, 5)                 58885

=================================================================
Total params: 1,874,421
Trainable params: 1,874,421
Non-trainable params: 0
_________________________________________________________________
```

Figure 17: Modified LSTM Model

BI-LSTM+Attention

In this part, evaluation indexes of the algorithm on the test set are mainly carried out, as shown in the table below. Since there are multiple classification indexes, Accuracy Score and F1 Score are only used as evaluation indexes. Accuracy of evaluation. The higher the value is, the more accurate the prediction is. However, sample imbalance is not taken into account. F1 score is an evaluation index with unbalanced comprehensive sample distribution, which can effectively judge the overall generalization ability of the model.

Metric	Value
Accuracy	0.782
F1 score	0.714

Conclusion and future work

This project proposes a Tweet text sentiment analysis method that uses different models. The RNN(Bi-GRU) model was one of the best performing models due to its nature of being easier to modify. This can make the tuning easier. BiGRU can capture the context information of the input sequence and establish an effective connection between each word segment of the sentence. The LSTM model gives out the worst performance of 68.3% accuracy, due to being sensitive to different random weight initializations and harder to train since it requires a lot of memory. Also it is easy to overtrain in LSTM – hard to implement dropout. For the CNN model, the structure of the neural network, hyperparameter tuning, and training time also consumed most of the time during this project. Also CNNs are "feed-forward neural networks' ' that use filters and pooling layers, whereas RNNs feed results back into the network. It would perform worse than the RNN model on text classification. Overall the BILSTM performed the best full use of sequence information as it can process both the forward and backward information of a sequence. This is useful for sequential data analysis tasks such as natural language processing. Also it solves the problem of gradient disappearance, as its bi-directional propagation improves the stability of the model and avoids the decay of gradients during backward propagation. Reduces the risk of overfitting, as its bi-directional propagation reduces the complexity of the model and improves its generalization ability
In the future, it is the goal of the next step to study how to improve the classification accuracy and reduce the time cost or a better algorithm such as Adam and RMSProp, can be used to improve the convergence speed of the model. For the model we could apply different numbers of layers and compare the performance to choose the optimal length of the layer. We also intend to investigate the optimal filter length for this environment.

Reference:

1. Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021, January 4). Deep Learning Based Text Classification: A comprehensive review. arXiv.org. Retrieved September 16, 2022, from https://doi.org/10.48550/arXiv.2004.03705
2. Miglani, A. (2020, September 8). Coronavirus tweets NLP - text classification. Kaggle. Retrieved December 6, 2022, from https://www.kaggle.com/datasets/datatattle/covid-19-nlp-text-classification/code

3. wafafaisal7. (2021, November 21). Text classification: Recurrent neural network RNN. Kaggle. Retrieved December 6, 2022, from https://www.kaggle.com/code/wafafaisal7/text-classification-recurrent-neural-network-rnn#Deep-Learning-Modeling---Recurrent-Neural-Network
4. Caraffa, B. (2022, September 1). *Twitter sentiment analysis using LSTM*. Medium. Retrieved December 6, 2022, from https://towardsdatascience.com/using-lstm-in-twitter-sentiment-analysis-a5d9013b523b
5. Zhou, C., Sun, C., Liu, Z., & Lau, F. C. M. (2015, November 30). *A C-LSTM neural network for text classification*. arXiv.org. Retrieved December 6, 2022, from https://arxiv.org/abs/1511.08630
6. İLKER GALIP ATAK. (2022, June 16). NLP with NN comparison on covid-19 tweet dataset. Kaggle. Retrieved December 6, 2022, from https://www.kaggle.com/code/lkergalipatak/nlp-with-nn-comparison-on-covid-19-tweet-dataset
7. Choubey, V. (2021b, December 15). Text classification using CNN - Voice Tech Podcast. Medium. https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9
8. Jianfeng Deng, Lianglun Cheng, & Zhuowei Wang (2021). Attention-based BiLSTM fused CNN with gating mechanism model for Chinese long text classification Computer Speech & Language.
9. Ovi Sarkar, Faysal Ahamed, Tahsin Tasnia Khan, Moloy Kumar Ghosh, & Md. Robiul Islam (2021). An Experimental Framework of Bangla Text Classification for Analyzing Sentiment Applying CNN & BiLSTM 2021 2nd International Conference for Emerging Technology (INCET).
10. Zhaoyang Niu, Guoqiang Zhong, & Hui Yu (2021). A review on the attention mechanism of deep learning Neurocomputing.