University of
Kwazulu-Natal

**May 2022**

# COMP703 Assignment 2

*Report: Comparison between LSTM and GRU in Sentiment Classification Tasks*

**Prepared by** | Sashen Moodley (219006946)

# TABLE OF CONTENTS

# 1. INTRODUCTION

The aim of this report is to compare 2 Recurrent Neural Network (RNN) architectures, namely the Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU) in sentiment classification. Both architectures were designed to tackle the issues of the traditional RNN namely the exploding and vanishing gradient problem which plagues longer sequence dependencies. They both utilize gated units, but with different internal approaches to tackling these issues, hence why they form an interesting basis for comparison.

The particular sentiment task looked at in this report is around Twitter sentiment classification, based on a publicly available dataset [1]. This report presents the data pre-processing activities in preparing these tweets appropriately for LSTM and GRU classification, and presents the steps taken in training and evaluating the LSTM and GRU models. The remainder of this report goes through the comparison results, personal observations and deductions, and concluding remarks.

# 2. DATA PRE-PROCESSING

Data cleaning in machine learning refers to the technique of preparing (cleaning and organizing), the raw data to make it more suitable for building and training machine learning models [2]. There are six general steps taken in data pre-processing which I applied in conducting my research:

1. Acquire the data set – The "*sentiment_dataset.csv*" file was acquired from an online drive, originating from a publicly available data set [1], consisting of 1.6 million tweets. These tweets are tagged with either their respective sentiment label (0 – negative, 4 – positive) providing plentiful data for our *supervised learning* models. Further characteristics of this raw and processed data and are presented in *Table 2.1* below and *Appendix A*.
2. Import the crucial libraries – For the task of data pre-processing, the popular *Pandas*, *NumPy*, *scikit-learn*, and *Keras* pre-processing libraries were used.
3. Import the dataset – The dataset was loaded into a Pandas Data Frame for easy manipulation, where the unnecessary columns (ID, Date, Flag, and User) were dropped. The remaining 'Target' and 'Text' columns were split into separate lists whereby the tweets would be subject to *denoising*/*cleaning*. The *denoising* process is quite an intensive and crucial task, in ensuring the models are fed only relevant information. This is emphasised in the Twitter dataset which is littered with general online social jargon and semantics. The process involved converting the tweets to lower case, removing URL links, removing mentions ('@' and what follows), removing hashtags ('#' but not what follows as it is a good indicator of sentiment), removing non-alphabets (except characters and digits), representing emojis as a standard symbol from a provided dictionary [3], and replacing 3 or more consecutive letters with only 2 or more (e.g. "foooood" is equivalent to "food"). A summary of this cleaned data is presented in *Appendix A* and *Table 2.1* below. Other popular methods in pre-processing include removing stop words and stemming or lemmatization. However, as pointed out by [4], while this might improve results in other datasets, for Twitter tweets this will have an opposite effect. After testing out this notion on this particular dataset, my findings validate this view as I found an average increase of 3.8% in F-1 score by not removing stop words or stemming/lemmatizing the tweets.
4. Identifying and handling missing values – The 1.6 million data entries are evenly distributed across negative and positive sentiment (see *Table 2.1* and *Table A.1*). There were no missing values from the columns of interest, however positive sentiment was tagged with a value of '4' rather than '1', when there are only positive and negative sentiment present in the dataset. Hence, all positive sentiment was changed to '1' which makes handling the data easier whilst also conforming to the binary classification sigmoid activation output.
5. Splitting the dataset – With the abundance of data available, a 90:10 split was made on the dataset constituting the training and testing set respectively. The split was made with the use of a randomization seed and the same data was fed into the LSTM and GRU during the same runtime to keep results consistent and comparable. The testing set was further split in 90:10 fashion with 10% of the training set being set aside for validation. This resulted in 1,296,000, 144,000, and 160,000 entries for the training, validation, and test set respectively.
6. Encoding of data – With the dataset split being made, the extracted training and testing tweets were tokenized and padded. Tokenization results in a word index that provides the denoised vocabulary size and an index-word mapping to be used for word embedding. Padding ensures the sequence lengths remain consistent (in my case '30' words), and each tweet was represented from the word index via the *Keras* text_*to_sequences* method. *Word embedding* is a document representation scheme (continuous/distributed vectors) that is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc [5]. After performing multiple tests with a trainable Keras embedding layer, the results were not favourable as they resulted in dataset overfitting (constantly increasing validation loss and decreasing accuracy) which seemingly no amount or dropout or regularization could solve. After migrating to the pre-trained *GloVe embedding* from Stanford AI [6] these issues were alleviated and produced desirable results. Although not entirely necessary, the labels were also encoded for binary classification.

***Table 2.1***

| Dataset | Samples | # Positive samples | # Negative samples | # Tokens | Max tweet length | Min tweet length | Average tweet length |
|---|---|---|---|---|---|---|---|
| **Uncleansed** | 1,600,000 | 800,000 | 800,000 | 21,081,841 | 64 | 1 | 13.2 |
| **Cleaned** | 1,600,000 | 800,000 | 800,000 | 20,989,524 | 53 | 0 | 13.1 |

# 3. MODEL TRAINING

Model training in machine language is the process of feeding an ML algorithm with data to help identify and learn good values for all attributes involved [7]. There are several types of machine learning models, of which the most common ones are supervised and unsupervised learning. As I am using a labelled Twitter corpus, the learning takes place in a supervised manner. The data prepared form the data pre-processing stage has been fed into the model architectures described below.

## 3.1 MODEL ARCHITECTURE

Model training for both the LSTM and GRU models are facilitated in TensorFlow Keras architectures. The architectures are setup in a manner that allows for a consistent comparison of results, by only swapping out the LSTM layer for the GRU layer or vice versa. The specific sequential architecture for both models are as follows: **Embedding Layer → Dropout → LSTM/GRU → Dropout → Dense Sigmoid Output**. The specific details of the architectures are presented in *Appendix B* via model summaries. The embedding layer is non-trainable and uses the *GloVe* encoding [6] as described above in the data pre-processing stage. Dropout is introduced as a mechanism to reduce overfitting of the models to training data. With the sentiment classification task being one of binary classification the *sigmoid activation* output layer is the most appropriate.

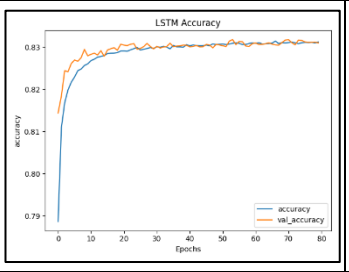
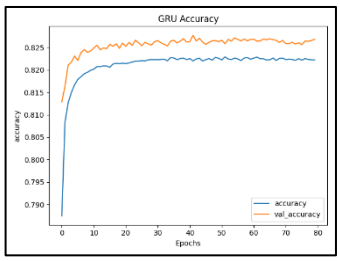
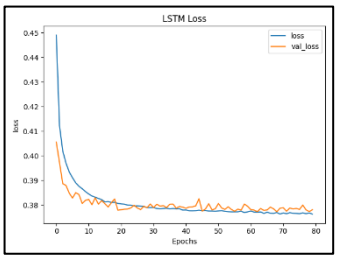
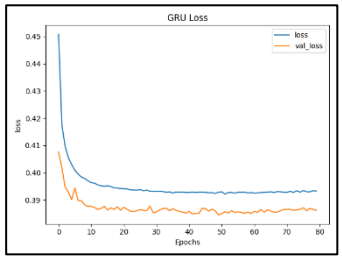
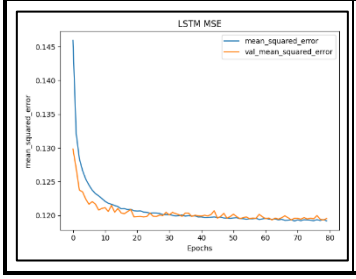
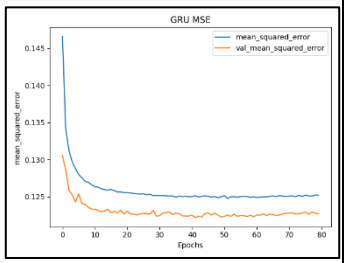
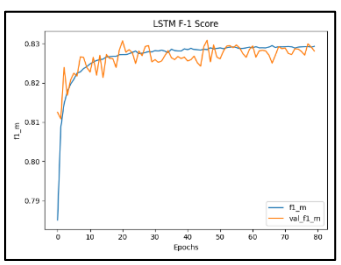
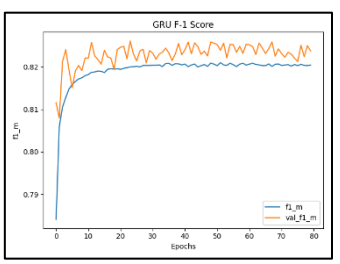
Both models are compiled in a similar manner, once again to allow for a fair comparison of results. The *loss function* used is *Binary Cross-entropy* as it is most suitable for binary classification tasks. *Adam optimization algorithm* is used as it combines the benefit of both *AdaGrad* and *RMSProp* [8]. Additionally *dynamic learning rates* and *model checkpointing* are incorporated into the training via *Keras ReduceLROnPlateau* and *ModelCheckpoint callbacks* respectively, ensuring I have persistent access to the best models for the LSTM and GRU observed in training.

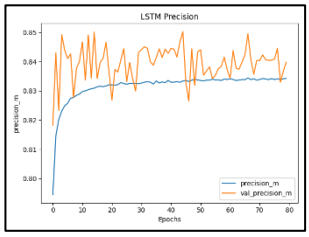
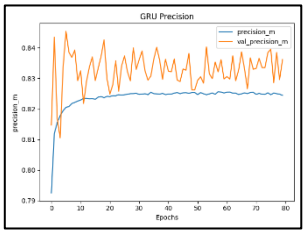
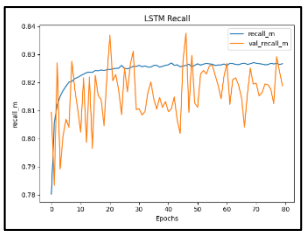
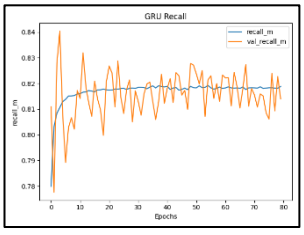
For the matter of sentiment classification, the metrics tracked across *epochs* are: accuracy, loss, mean squared error, F1-score, precision, and recall. Additionally, the time taken to train the model is noted and reported later in model evaluation. With the overarching architecture remaining constant between both the LSTM and GRU, any difference in observed values for these metrics is owing to the internal mechanisms of the LSTM and GRU, thus providing our basis for comparison.

## 3.2 TRAINING AND VALIDATION METRIC PLOTS

The metrics accuracy, loss, mean squared error, F1-score, precision, and recall were noted across 80 epochs, with a *batch size* of '128', during model fitting for both the training and validation sets. These LSTM and GRU metrics are provided as plotted graphs below. Note: These figures are additionally provided in *Appendix C.*

***Table 3.1***

| Accuracy | | Loss | |
|---|---|---|---|
| **LSTM** | **GRU** | **LSTM** | **GRU** |
|  |  |  |  |
| **MSE** | | **F1-Score** | |
| **LSTM** | **GRU** | **LSTM** | **GRU** |
|  |  |  |  |

| Precision | | Recall | |
|---|---|---|---|
| **LSTM** | **GRU** | **LSTM** | **GRU** |
|  |  |  |  |



# 4. MODEL EVALUATION

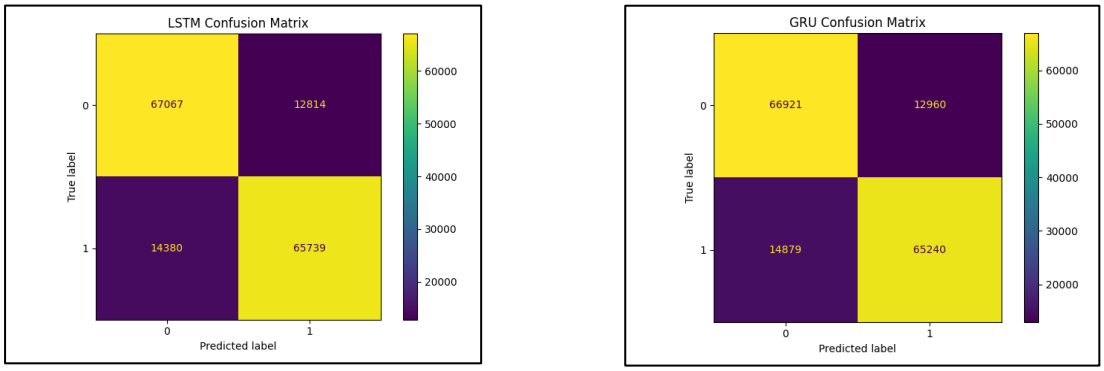
Once trained, both the LSTM and GRU models are evaluated on the same testing data. The evaluation is provided by Keras *evaluate* function where the 160,000 processed sample testing data is passed in. Along with training validation, the training set can provide us an indication of how well the LSTM and GRU models can generalize on unseen data. Keras provides model evaluation functions such as *evaluate* which conveniently provides us with evaluation metrics based on what was used to compile/fit the model. In our case: accuracy, loss, mean squared error, F1-score, precision, and recall. This information is summarised in *Table 4.1* below

***Table 4.1***

| | | Metric | | | | | |
|---|---|---|---|---|---|---|---|
| | | **Accuracy** | **Loss** | **MSE** | **F1-Score** | **Precision** | **Recall** |
| **RNN Architecture** | **LSTM** | 83.18 | 0.3785 | 0.1196 | 82.63 | 83.75 | 82.43 |
| | **GRU** | 82.60 | 0.3832 | 0.1232 | 81.97 | 83.44 | 81.46 |
| | **Metric Delta** | 0.58 | 0.0047 | 0.0036 | 0.66 | 0.31 | 0.97 |

As seen from this summary, the LSTM slightly outperforms the GRU in every metric. A more detailed analysis of this is provided in the observations and deductions section following.

Whilst the *evaluate* function provides us with the metrics for the evaluation it does not provide us with the actual predictions for the samples in a batch for all batches. Therefore, both models were subject to the *predict* function where the following confusion matrix heat maps (*Figure 4.1*) were generated to summarise the results.




***a) LSTM confusion matrix heat map*** ***b) GRU confusion matrix heat map***

***Figure 4.1***

Additionally, the time taken to train and evaluate the models, as well as the time taken to predict the tweet sentiments were noted and presented in *Table 4.2* below:

***Table 4.2***

| | | Stage | | |
|---|---|---|---|---|
| | | **Training (s)** | **Evaluation (s)** | **Prediction (s)** |
| **RNN Architecture** | **LSTM** | 4683 | 15.82 | 2.33 |
| | **GRU** | 4241 | 13.81 | 2.47 |
| | **Time Delta (s)** | 442 | 2.01 | 0.14 |

As indicated by *Table 4.2* the GRU is faster than the LSTM in training and evaluation. These findings are analysed in the following observations and deductions section.

# 5. OBSERVATIONS AND DEDUCTIONS

From the metrics presented above along with the training and validation data plots, we can note the following main observations from this particular Twitter domain:

1. The LSTM produces slightly more accurate results than the GRU
2. The LSTM converges sooner than the GRU
3. The GRU faster to train and evaluate than the LSTM

Diving into this deeper, we can try and gain an understanding of these observations. The underpinning differentiating factor are the gated units employed in the LSTM and GRU. These gates are introduced to help mitigate the issue of vanishing gradient during backwards propagation for longer sequence tasks present in the 'vanilla' RNNs. The gated units can control the flow of information and choose what information is important. The LSTM has 3 gates: an input, output, and forget gate, whilst the GRU has 2 gates: an update, and reset gate. Whereas the LSTM uses the *cell state* to capture and transfer information from earlier steps the GRU uses the *hidden state* itself, i.e., the GRU simply exposes the complete hidden content layers (memory) without the level of control like the LSTM as it does not have a dedicated *memory cell.*

Firstly, from *point 1* above, in understanding why the LSTM produces more accurate results than the GRU. With this particular task being on of binary sentiment classification the minimum accuracy baseline of 50% was set and greatly exceeded by both models achieving above 80% accuracy F1-scores (see *Table 4.1*). This is quite impressive when considering typical human baseline accuracies [9] around sentiment classification, especially twitter sentiment classification where we find quite sporadic human tropes such as sarcasm, and general online social jargon. As noted in the results for each metric (see *Table 4.1, Figure 4.1(a) and Figure 4.2(b)*), the LSTM outperforms the GRU. Due to the GRU having less gates, and no dedicated memory cell state akin to the *cell* state in the LSTM, the GRU works with less training parameters thus giving the advantage to the LSTM which has a greater capacity to learn. However, this more complex structure employed by the LSTM results in only slight improvements in the respective metrics (as indicated by the *metric delta* row in *Table 4.1*). The reason for this similarity could be due to the short nature of tweets. As indicated by *Table 2.1*, the average tweet length in the dataset is '13.1' words long. This is relatively short for a sequence and the LSTM seems to be 'overqualified' for this domain, as it is more suitable for capturing long-term dependencies/relationships in a sequence.

Secondly, whilst the GRU achieves results mostly on par with the LSTM, the LSTM converges on these slightly superior metrics earlier than the GRU. As indicated by *Table 3.1* the LSTM can be seen reaching training and validation crossover convergence at earlier epochs, whereas the GRU training and validation accuracy and loss curves do not intersect over the 80 epochs. The GRU training and validation accuracy and loss curves do converge, but at differing values. In contrast these same curves in the LSTM converge at similar values. The LSTM establishes this earlier on due to its ability to retain information in long-distance relations, which is a result of its learning capabilities. This could be partially explained by the *dropout* layer of the architecture which seems to favour the LSTM. The *dropout* layer randomly sets input units to '0' with a frequency of rate at each step during training time, which helps prevent overfitting [10]. Whilst this has the welcomed effect of preventing overfitting, it has the additive effect of postponing convergence to later epochs.

Thirdly, we find the GRU to be more efficient than the LSTM with regard to training and evaluation times, and memory allocations. Unlike the evaluation metrics, the difference in the training and evaluation times are quite significant. Particularly in training we see the GRU train 9.5% faster than the LSTM (see *Table 4.2*). The GRU is faster to train and evaluate for the same reasons why it is slightly less accurate than the LSTM, that being it only works with 2 gated units and does not need to worry about updating an internal *cell state*. Therefore, the GRU works with less training parameters, in turn requiring fewer gated computations and less memory.

# 6. CONCLUSION

A case could be made for either model as the LSTM achieves slightly better results, but is more complex, whereas GRU is faster and more memory efficient to train but is not as accurate as the LSTM. However, in choosing a single RNN architecture to perform, twitter, sentiment analysis the GRU would have to be the more favourable choice. The GRU boasts a significantly faster training time whilst achieving metrics essentially on par with its more complex LSTM competitor. This is mostly due to this particular sentiment classification task whereby we find tweets taking upon a short sequence of words playing into the particular strengths of the GRU. As this report indicates, the trade-off between minor accuracy lost in favour of a considerable time efficiency boost is well worth it, thus crowning the GRU as the most suitable RNN architecture for Twitter sentiment analysis classifier.

## 7. REFERENCES


[1] ‘Sentiment140 dataset with 1.6 million tweets’. https://www.kaggle.com/kazanova/sentiment140 (accessed April 27, 2022).

[2] ‘Data Preprocessing in Machine Learning: 7 Easy Steps To Follow | upGrad blog’. https://www.upgrad.com/blog/data-preprocessing-in-machine-learning/ (accessed May 2, 2022).

[3] ‘Twitter Sentiment Analysis for Beginners’. https://kaggle.com/stoicstatic/twitter-sentiment-analysis-for-beginners (accessed May 1, 2022).

[4] H. Saif, M. Fernandez, and H. Alani, ‘On stopwords, filtering and data sparsity for sentiment analysis of twitter’, Proceedings of the 9th International Language Resources and Evaluation Conference (LREC’14), pp. 810–817, Jan. 2014.

[5] J. Brownlee, ‘What Are Word Embeddings for Text?’, Machine Learning Mastery, Oct. 10, 2017. https://machinelearningmastery.com/what-are-word-embeddings/ (accessed May 2, 2022).

[6] ‘GloVe: Global Vectors for Word Representation’. https://nlp.stanford.edu/projects/glove/ (accessed May 1, 2022).

[7] ‘What is Model Training’, Oden Technologies. https://oden.io/glossary/model-training/ (accessed May 2, 2022).

[8] J. Brownlee, ‘Gentle Introduction to the Adam Optimization Algorithm for Deep Learning’, Machine Learning Mastery, Jul. 02, 2017. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (accessed May 2, 2022).

[9] ‘Sentiment Accuracy: Explaining the Baseline and How to Test It’, Lexalytics, Jul. 11, 2019. https://www.lexalytics.com/lexablog/sentiment-accuracy-baseline-testing (accessed May 5, 2022).

[10] K. Team, ‘Keras documentation: Dropout layer’. https://keras.io/api/layers/regularization_layers/dropout/ (accessed May 2, 2022).

[11] M. Soni, ‘Word Embeddings and the chamber of secrets| LSTM | GRU | tf.keras’, *Medium*, Nov. 19, 2020. https://towardsdatascience.com/word-embeddings-and-the-chamber-of-secrets-lstm-gru-tf-keras-de3f5c21bf16 (accessed May 1, 2022).

# 8. APPENDIX A – DATASET INFORMATION

***Table A.1*** **– Added row depicting how much information is lost after performing lemmatization and stop-word removal**

| Dataset | Samples | # Positive samples | # Negative samples | # Tokens | Max tweet length | Min tweet length | Average tweet length |
|---|---|---|---|---|---|---|---|
| **Uncleansed** | 1,600,000 | 800,000 | 800,000 | 21,081,841 | 64 | 1 | 13.2 |
| **Cleaned** | 1,600,000 | 800,000 | 800,000 | 20,989,524 | 53 | 0 | 13.1 |
| **Cleaned + Lemmatization + Stop word removal** | 1,600,000 | 800,000 | 800,000 | 11,424,966 | 50 | 0 | 7.1 |

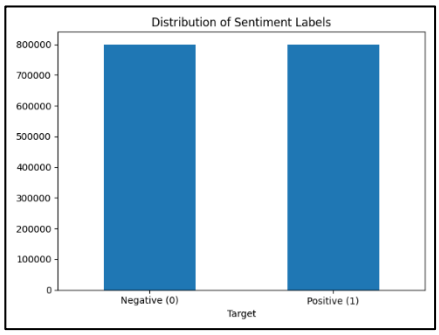
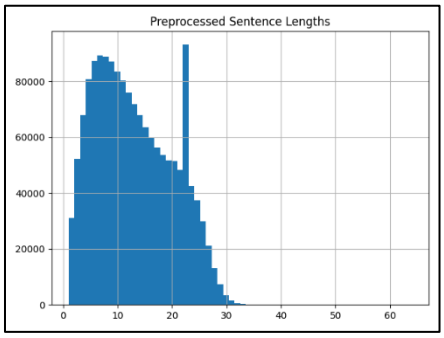
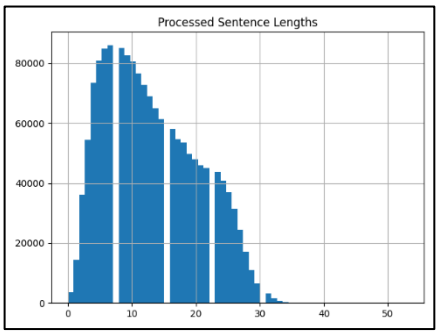



***Figure A.1*** **– Sentiment label distribution**

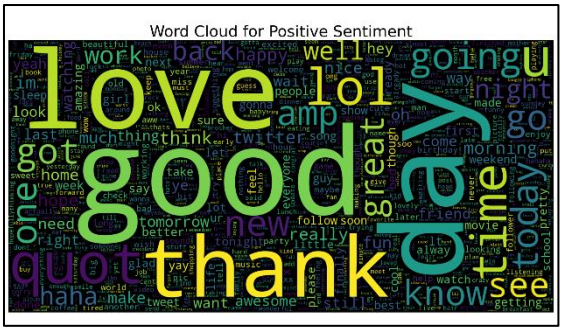



***(a)*** **– Pre-processed Distribution**

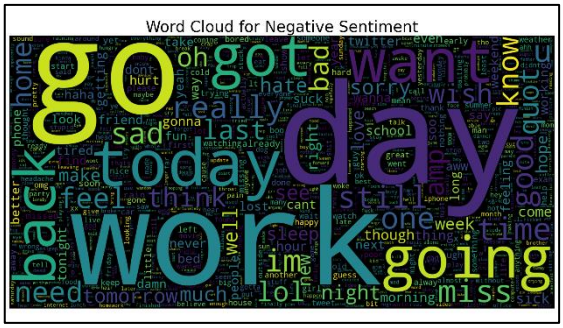



***(b)*** **– Processed Distribution**

***Figure A.2*** **– Sentence Length Distribution**




***(a)*** **–Positive Sentiment Word Cloud**




***(b)*** **–Negative Sentiment Word Cloud**

***Figure A.3*** **– Word Cloud Visualizations**

# 9. APPENDIX B – MODEL ARCHITECTURE

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 30, 300)           73293600

dropout (Dropout)            (None, 30, 300)           0

lstm (LSTM)                  (None, 64)                93440

dropout_1 (Dropout)          (None, 64)                0

dense (Dense)                (None, 1)                 65

=================================================================
Total params: 73,387,105
Trainable params: 93,505
Non-trainable params: 73,293,600
_________________________________________________________________
```

**(a) – LSTM model summary**

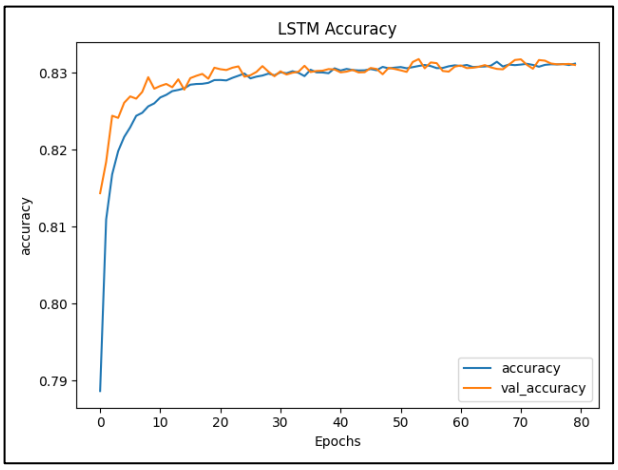
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 30, 300)           73293600

dropout_2 (Dropout)          (None, 30, 300)           0

gru (GRU)                    (None, 64)                70272

dropout_3 (Dropout)          (None, 64)                0

dense_1 (Dense)              (None, 1)                 65

=================================================================
Total params: 73,363,937
Trainable params: 70,337
Non-trainable params: 73,293,600
_________________________________________________________________
```

**(b) – GRU model summary**

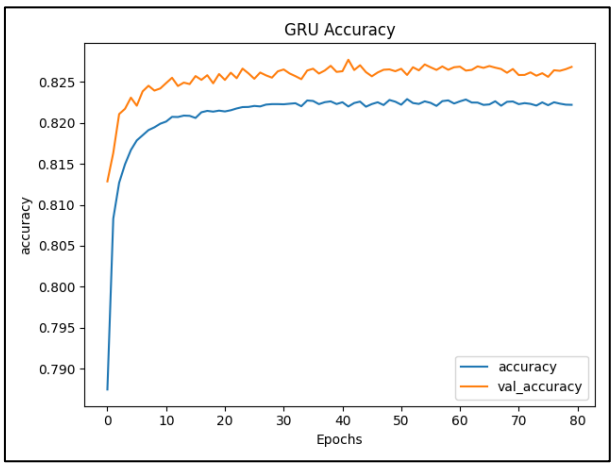


***Figure B.1*** **– Model Architecture Summaries with GloVe Embeddings**

# 10. APPENDIX C – TRAINING AND VALIDATION METRIC PLOTS

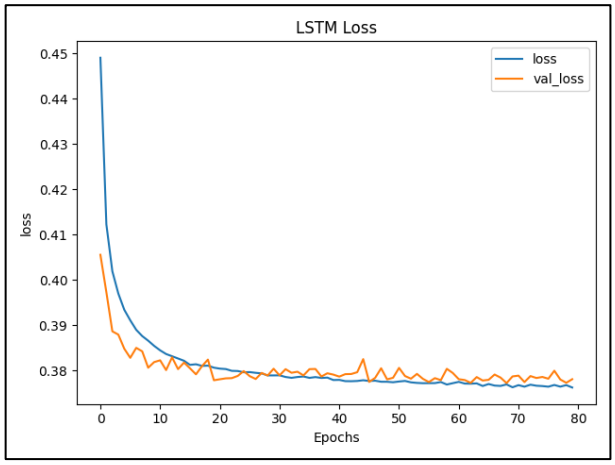



***(a)* – LSTM**

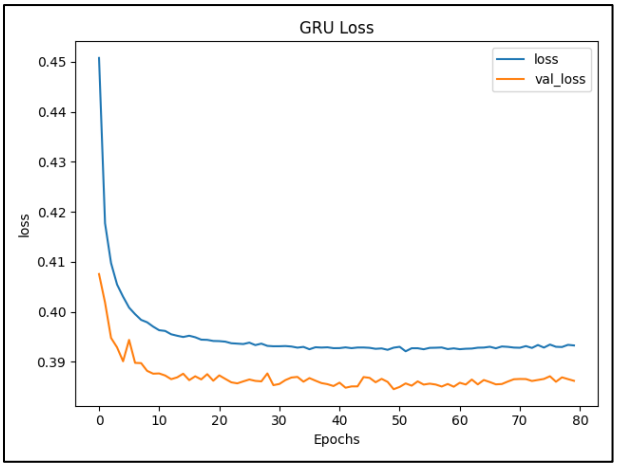



***(b)* – GRU**

***Figure C.1* – Accuracy Plots**

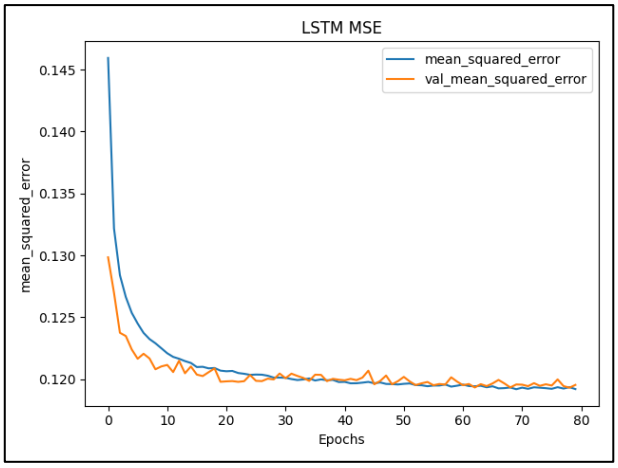



***(a)* – LSTM**

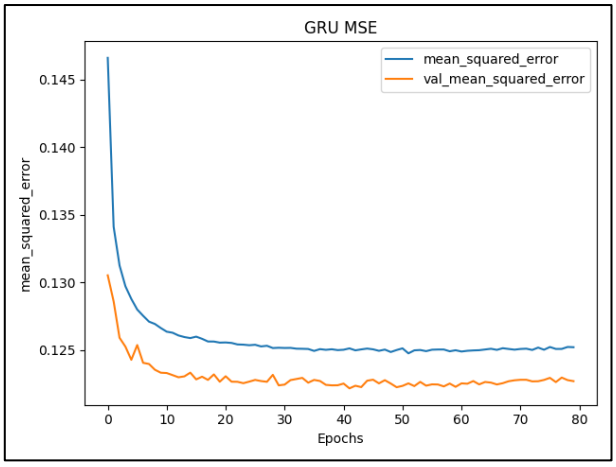



***(b)* – GRU**

***Figure C.2* – Loss Plots**

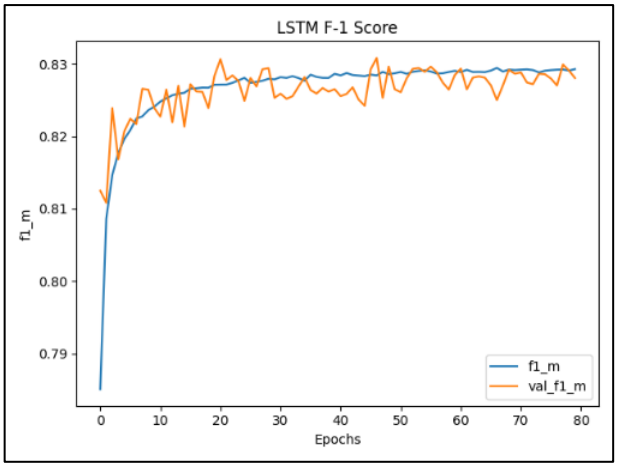



***(a)* – LSTM**

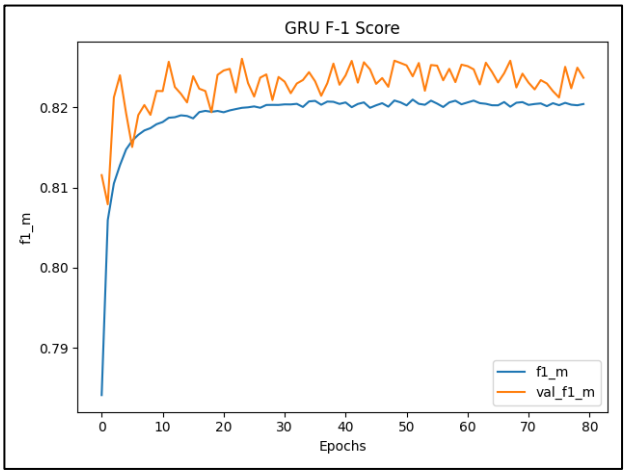



***(b)* – GRU**

***Figure C.3* – MSE Plots**

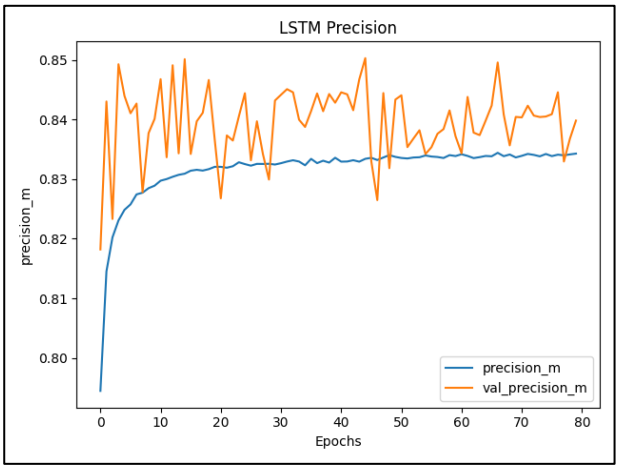



***(a)* – LSTM**

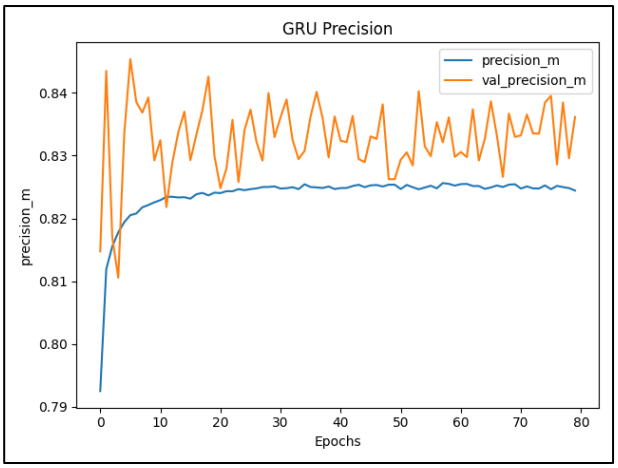



***(b)* – GRU**

***Figure C.4* – F1-Score Plots**

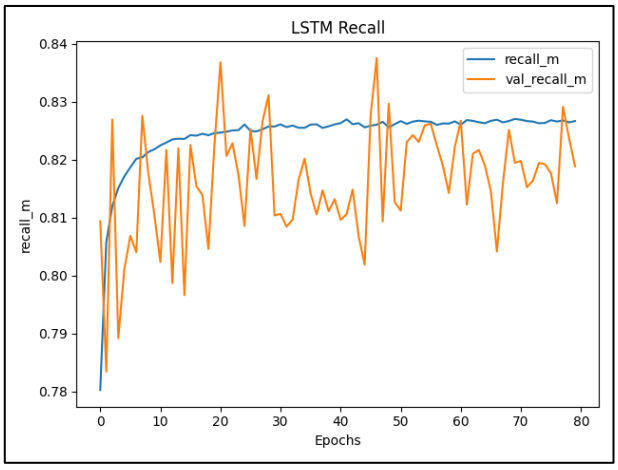



***(a)* – LSTM**




***(b)* – GRU**

***Figure C.5* – Precision Plots**




***(a)* – LSTM**


***(b)* – GRU**

***Figure C.6* – Recall Plots**