Problem Statement

To design and implement multiple deep learning models: BERT, LSTM, GRU, and RNN in order to perform sentiment analysis on a given dataset containing textual data and corresponding sentiment labels. Additionally, a comparative analysis based on their performance metrics is to be conducted.

Summary

This report evaluates the performance of four deep learning models—BERT, LSTM, GRU, and RNN—on a Twitter sentiment classification task. The goal was to classify tweets into "Negative" or "Positive" sentiment categories. The models were trained and evaluated based on accuracy, precision, recall, F1-score, and ROC-AUC metrics.

Key observations :

- ➢ LSTM performed best in terms of accuracy (87.17%) and ROC-AUC (0.788).
- ➢ GRU was slightly worse than LSTM but still competitive.
- ➢ RNN had the lowest performance, likely due to its difficulty in capturing long-term dependencies.
- ➢ BERT underperformed (validation accuracy: 85.63%), possibly due to insufficient fine-tuning, lower no. of epochs or dataset mismatch.
- ➢ All models performed better on negative class (higher recall)
- ➢ RNN trained 5-7times faster than GRU/LSTM

Data Description & Preprocessing

The dataset consists of ~ 1.05million rows and 6 fields:

1. Polarity (Column 0):
  - ○ Sentiment label for the tweet:
    - ■ 0: Negative sentiment
    - ■ 2: Neutral sentiment
    - ■ 4: Positive sentiment

2. Tweet ID (Column 1):
  - ○ A unique identifier for each tweet.

3. Date (Column 2):
  - ○ The timestamp of when the tweet was posted, in the format Day Month Date HH:MM:SS UTC Year (e.g., Sat May 16 23:58:44 UTC 2009).

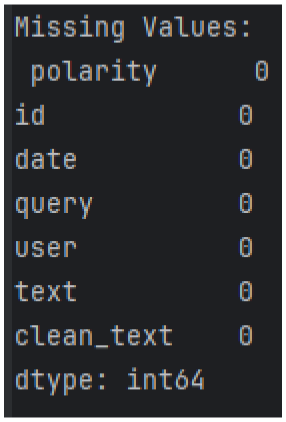
4. Query (Column 3):
  - ○ The search query used to retrieve the tweet. If no query was used, the value is NO_QUERY.

5. User (Column 4):
  - ○ The username of the account that posted the tweet

6. Text (Column 5):

- ○ The content of the tweet, consisting of raw text after emoticons have been removed (e.g., Lyx is cool).

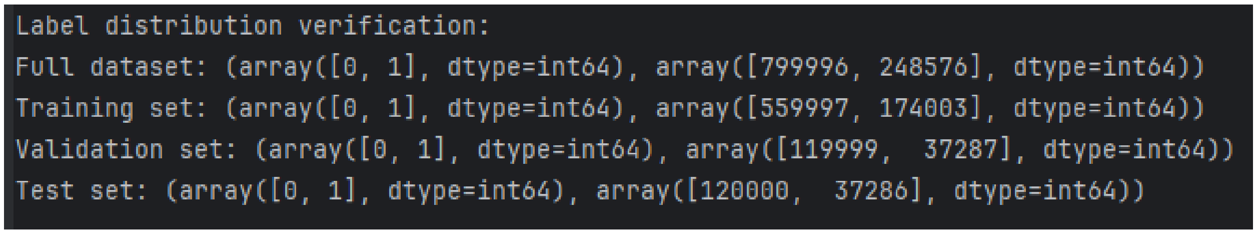
There are no missing values in any of the columns -



```
Missing Values:
 polarity      0
id            0
date          0
query         0
user          0
text          0
clean_text    0
dtype: int64
```

Most of the initial cleaning of the data remains almost the same as what we had seen for the TF-IDF vectorizer project which includes cleaning the column, “text of the tweet”, to remove any html tags, URLs, special characters, numbers, and extra spaces and finally converting the text to lowercase.

In the “polarity of tweet” column, there were only 0s and 4s and hence no “neutral” comments. I replaced the 4s with 1s to keep it binary, i.e. 0 = negative, 1 = positive. I split the data into 70% train and 15% each into validation and test data and checked the label distribution to verify the splits are as expected:



```
Label distribution verification:
Full dataset: (array([0, 1], dtype=int64), array([799996, 248576], dtype=int64))
Training set: (array([0, 1], dtype=int64), array([559997, 174003], dtype=int64))
Validation set: (array([0, 1], dtype=int64), array([119999,  37287], dtype=int64))
Test set: (array([0, 1], dtype=int64), array([120000,  37286], dtype=int64))
```

<u>Feature Engineering</u>

I used a RoBERTa tokenizer from Hugging Face library pretrained on Twitter data for BERT training using a bert_tokenize() function that tokenizes text (from train, validation and test datasets) with special tokens, pads/truncates to fixed length of 64 tokens and returns input IDs and attention masks.

For the RNN-based models, I have tokenized the text data using NLTK to convert the split texts in each set into tokens. Then trained the Word2Vec embeddings (100-dim) on training data, created an embedding matrix and converted text to sequences and padded to length 100 for uniformity.

Model Implementation

The BERTSentimentClassifier class loads a pretrained RoBERTa model from Hugging Face library which has been specifically fine-tuned for Twitter sentiment analysis. It sets up the device configuration to automatically use GPU if available. The training process includes implementing a loop with AdamW optimizer (a variant of Adam with weight decay fix) and setting the model to training mode which processes batches from DataLoader while performing forward pass with attention masks, computing loss using built-in model loss(CrossEntropy), back-propagating and updating the weights. This also includes gradient clipping for stability by preventing exploding gradients in transformer training and finally tracks and prints the training/validation metrics. The BERT training is done for 4 epochs.

In the evaluation mode, it disables gradient calculation and computes validation loss and accuracy and uses argmax on logits for predictions. In the predict() method, it returns raw predictions, collects both predictions and true labels and moves results back to CPU for further processing.

Note: I had earlier used the pretrained bert-base-uncased earlier with only 2 epochs, however, reverted to the above due to very poor results (31% accuracy) as can be seen from the snippet from the console below:

```
“*** Training BERT ***
Some weights of BertForSequenceClassification were not initialized from the model checkpoint
at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
Epoch 1/2
Train Loss: 0.0940 | Val Loss: 0.6716 | Val Acc: 0.5751
Epoch 2/2
Train Loss: 0.0800 | Val Loss: 0.6743 | Val Acc: 0.3146”
```

We will see in the results section how the pretrained Twitter dataset was better in terms of performance.

For the RNN-based models, the function creates sequential Keras model by embedding layers initialised with pre-trained Word2Vec weights and adds 3 different options including bidirectional LSTM, GRU (to provide context from both directions) or a simple RNN with additional dense layers and dropout for regularization and a final sigmoid activation for binary classification. It trains the RNN-based models (LSTM, GRU, simple RNN) for up to 10 epochs with early stopping.

Dropout regularization method is employed to address overfitting issues by starting with standard dropout of 0.2 between RNN layers, recurrent dropout of 0.2 within RNN cells and an additional dropout by adjusting upwards to 0.5 before final dense layer. This prevents overfitting as by disabling neurons randomly, the network cannot rely too much on specific connections between them, although droput does increase training duration.

I have also used progressive downsampling with 128 units in 1st layer to 64 units in 2nd layer which enhances system performance and efficiency by reducing processing needs.

The function finally compiles accuracy, precision, recall, and AUC to track model performance.

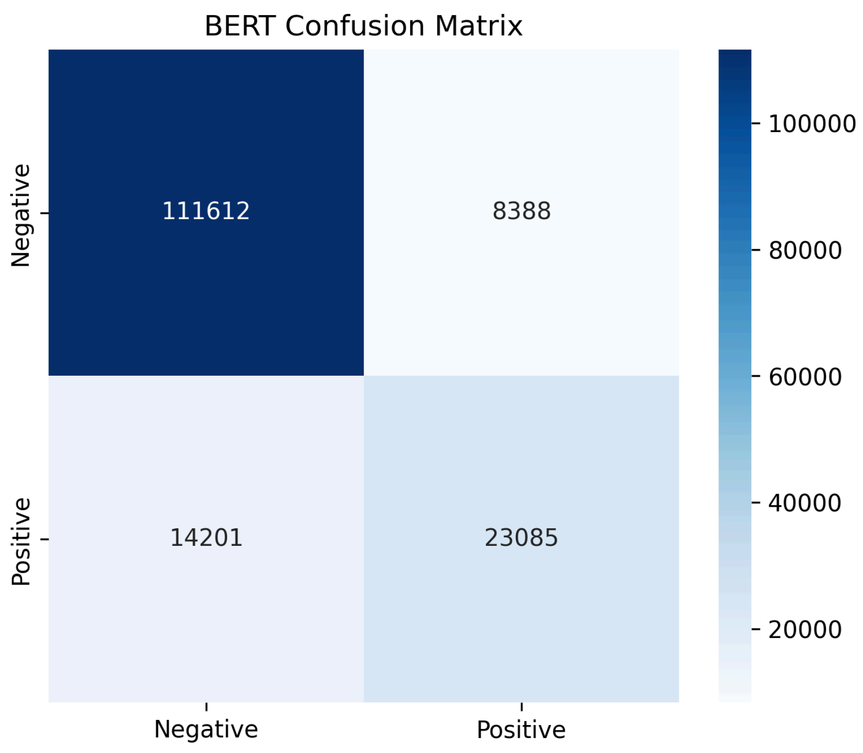
Results & Comparative Analysis

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**	**Training time (s)**	**Parameters**
BERT	85.64%	0.85	0.86	0.85	0.775	N/A	124,647,170
LSTM	87.17%	0.87	0.87	0.87	0.788	63,297.6	24,176,969
GRU	86.72%	0.86	0.87	0.86	0.779	50,132.1	24,078,409
RNN	84.62%	0.84	0.85	0.84	0.766	9,273.9	23,815,689

- **BERT**

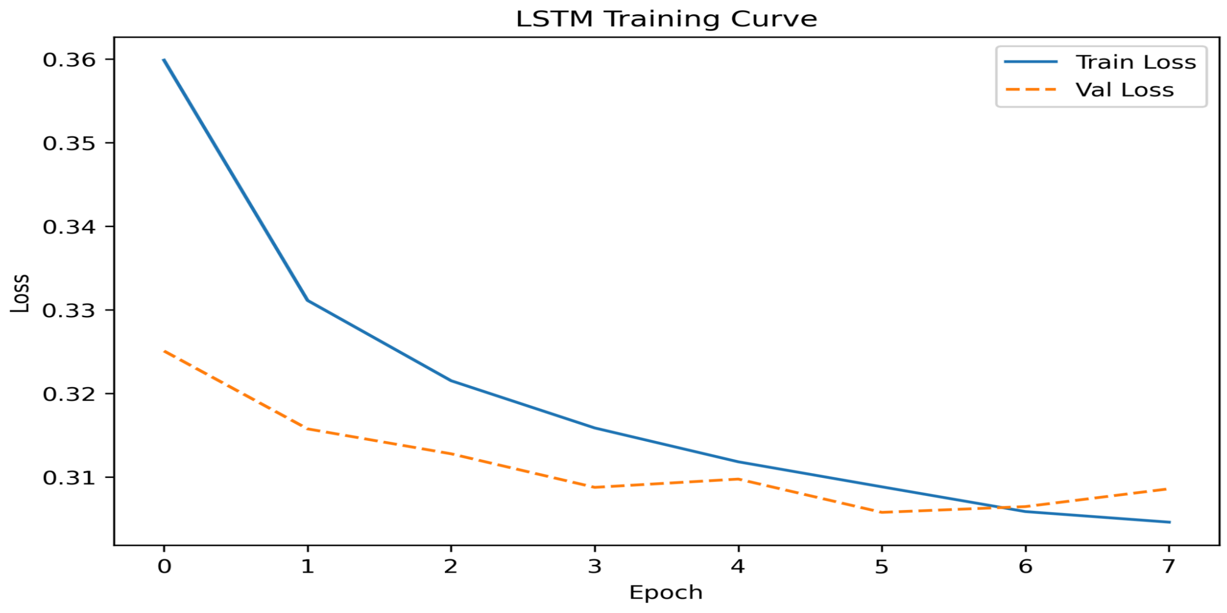
BERT achieved a 85.64% accuracy compared to 87.17% accuracy in the case of bidirectional LSTM even though the expectation was for BERT to have a better overall performance. This could be due to the model requiring more epochs and learning rate adjustments. Although as noted earlier, I had used the bert-base-uncased pretrained model with much lower accuracy in 2 epochs and felt the dataset preprocessing might not have aligned well with BERT's tokenization. And even in that case, the accuracy had actually dropped from 57% to 31%, possibly implying overfitting.

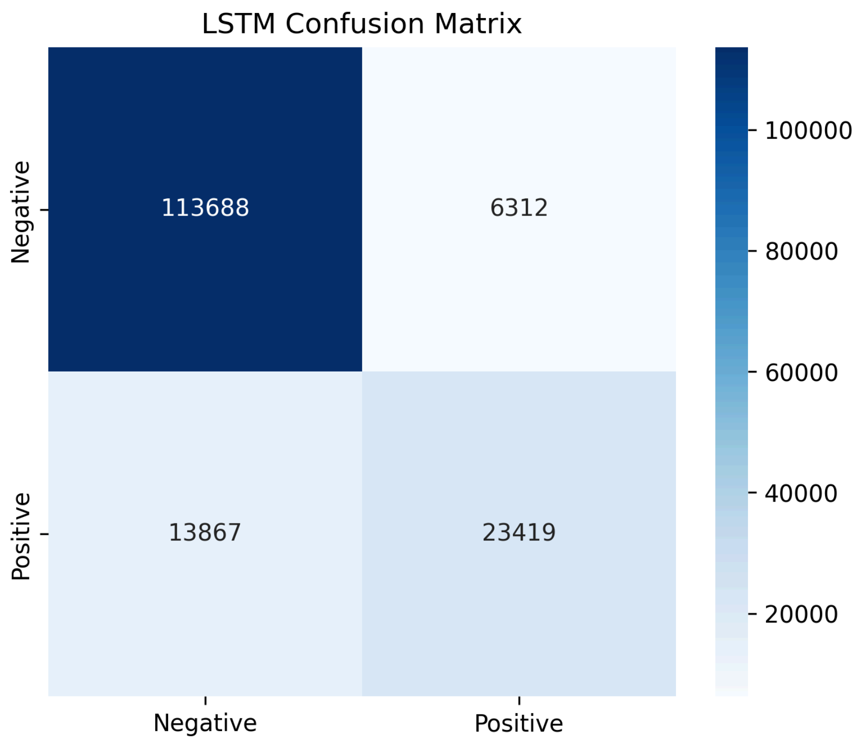
BERT does have a balanced precision (0.85) and recall (0.86). While the confusion matrix shows strong negative class identification (111,612 correct), the positive class identification could definitely improve as seen from the 23k correct vs 14k incorrect classifications. The training time data wasn't available (or at least I couldn't reproduce that) for BERT, however, we can see it has the largest model parameter count (124.65 million) and definitely took the longest time to run (for almost a week if I recollect).



- **LSTM (Long Short-Term Memory)**

This is the best performing model in all the runs I performed on this dataset with the highest accuracy at 87.17% and F1-score of 0.87. The balance across precision and recall is also good (0.87 for both). The training curve also shows stable convergence with the loss decreasing from 0.38 to 0.31 over 7 epochs (refer to appendix showing actual runs)





LSTM also has a strong negative class performance (113,688 correct), while the positive class identification with 23k correct vs ~14k incorrect is the best performance so far among other models.

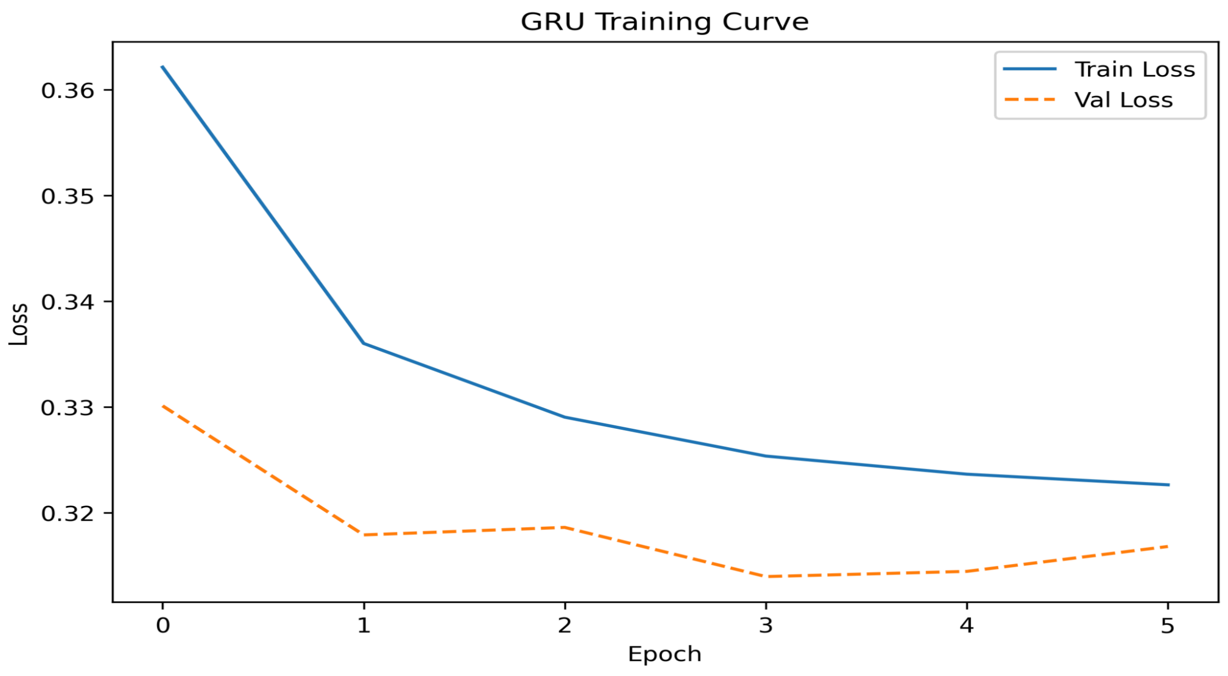
The ROC-AUC at 78.8% is also the best amongst all models suggesting decent separability between positive and negative classes.

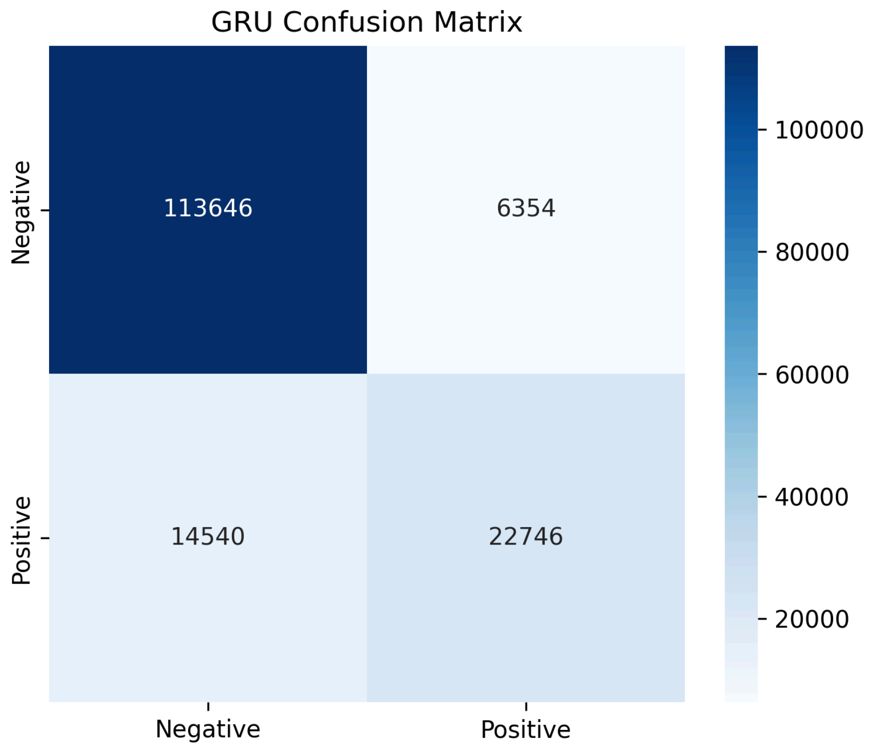
- **GRU (Gated Recurrent Unit)**

A Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that enhances the speed performance of LSTM networks by simplifying the structure with only two gates: the update gate and the reset gate

In terms of performance, GRU nearly matches that of LSTM with 86.72% accuracy with a similar model parameter count, while the training time was much faster for reasons mentioned earlier. The loss decreased from 0.38 to 0.32 over 8 epochs (refer to appendix showing actual runs).

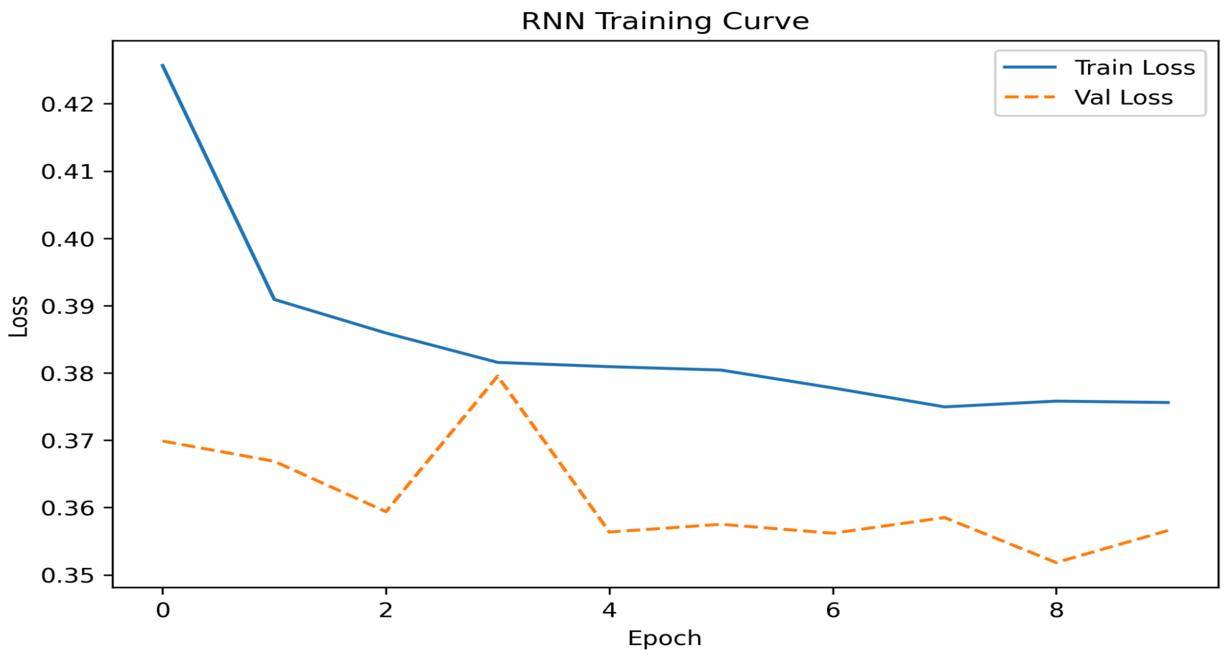
GRU has slightly lower positive class recall than LSTM (with 22,746 correct) while the negative class identification is almost as good as LSTM.

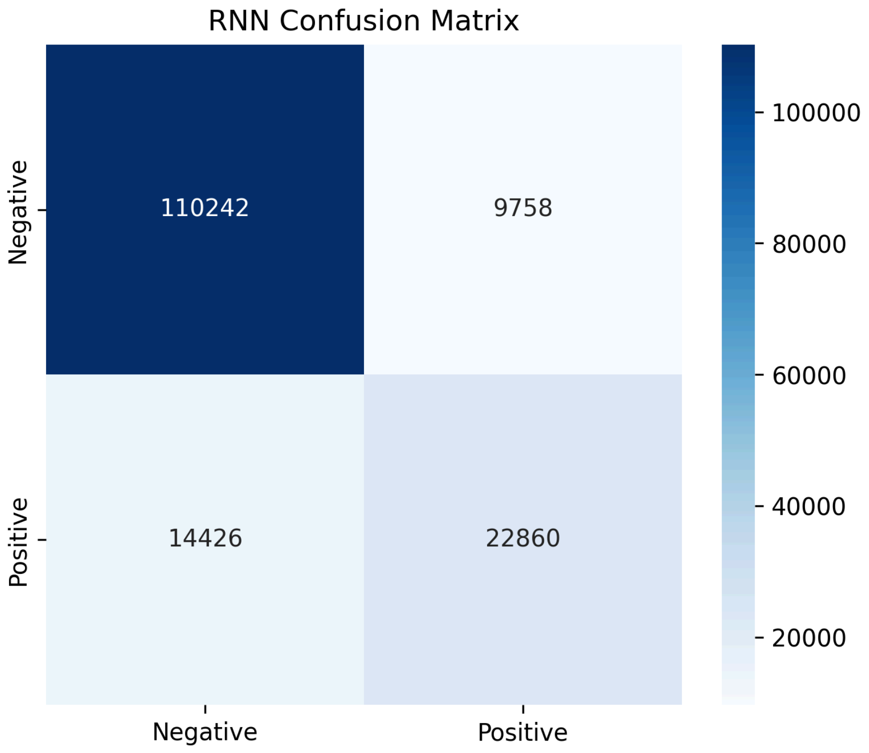




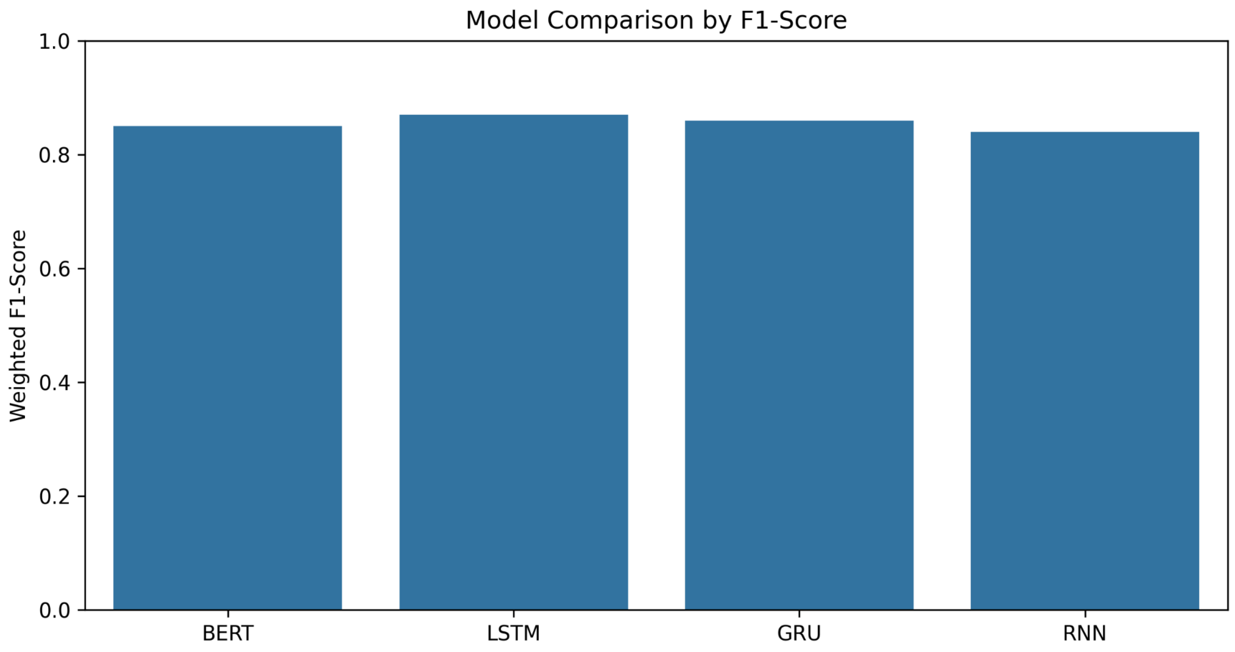
- **Simple RNN**

The simple RNN model implementation has the fastest training time and the lowest model parameter count. Given that, the accuracy is very reasonable at 84.62% although it has the lowest accuracy across the 4 model implementations.





The training curve shows more volatility with loss decreasing from 0.45 to 0.38 over 4 epochs while the validation curve also shows unstable performance. The negative class identification is the weakest among all models, while when it comes to positive class identification, it does slightly better than GRU actually. Having said that, RNN performs consistently lower than all other models across almost all metrics.



Conclusion & Recommendations

Based on the results we have seen here, LSTM would be the recommended model for deployment where accuracy is the highest priority, although the expectation would have been that BERT would perform better than the other models. Hence one would be expected to investigate the BERT training process further to have potentially improved results. For time-critical tasks with almost equal results, GRU could be used as well since it offers better performance/efficiency balance. RNN would only be recommended as a baseline case, or when training speed is very critical.

<u>Appendix</u>

***** Training BERT *****

```
You are using a model of type roberta to instantiate a model of type bert. This is not supported
for all configurations of models and can yield errors.
Some weights of BertForSequenceClassification were not initialized from the model checkpoint
at cardiffnlp/twitter-roberta-base-sentiment and are newly initialized:
Epoch 1/4
Train Loss: 0.3811 | Val Loss: 0.3563 | Val Acc: 0.8490
Epoch 2/4
Train Loss: 0.3468 | Val Loss: 0.3555 | Val Acc: 0.8553
Epoch 3/4
Train Loss: 0.3348 | Val Loss: 0.3425 | Val Acc: 0.8573
Epoch 4/4
Train Loss: 0.3274 | Val Loss: 0.3580 | Val Acc: 0.8563

accuracy_score(true_labels, pred_labels)
0.8563826405401626
classification_report(true_labels, pred_labels, target_names=['Negative', 'Positive'])
'            precision    recall  f1-score   support\n\n    Negative       0.89      0.93      0.91
120000\n    Positive       0.73      0.62      0.67     37286\n\n    accuracy                           0.86
157286\n   macro avg       0.81      0.77      0.79    157286\nweighted avg       0.85      0.86
0.85    157286\n'
confusion_matrix(true_labels, pred_labels)
array([[111612,   8388],
       [ 14201,  23085]], dtype=int64)
roc_auc_score(true_labels, pred_labels)
0.7746165933594379
sum(bert_train_metrics['time'])
```

***** Training LSTM *****

```
C:\Users\avina\PyCharmMiscProject\.venv\Lib\site-packages\keras\src\layers\core\embedding.p
y:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Epoch 1/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 11431s 2s/step - accuracy: 0.8336 -
auc_3: 0.8478 - loss: 0.3846 - precision_3: 0.7240 - recall_3: 0.4800 - val_accuracy: 0.8588 -
val_auc_3: 0.8983 - val_loss: 0.3266 - val_precision_3: 0.8077 - val_recall_3: 0.5309
Epoch 2/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 10363s 2s/step - accuracy: 0.8593 -
auc_3: 0.8931 - loss: 0.3320 - precision_3: 0.7719 - recall_3: 0.5787 - val_accuracy: 0.8651 -
val_auc_3: 0.9051 - val_loss: 0.3151 - val_precision_3: 0.7833 - val_recall_3: 0.5959
Epoch 3/10
```

```
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 10800s 2s/step - accuracy: 0.8640 -
auc_3: 0.9016 - loss: 0.3201 - precision_3: 0.7788 - recall_3: 0.5962 - val_accuracy: 0.8673 -
val_auc_3: 0.9078 - val_loss: 0.3126 - val_precision_3: 0.7897 - val_recall_3: 0.6002
Epoch 4/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 13302s 2s/step - accuracy: 0.8666 -
auc_3: 0.9048 - loss: 0.3150 - precision_3: 0.7821 - recall_3: 0.6045 - val_accuracy: 0.8680 -
val_auc_3: 0.9091 - val_loss: 0.3099 - val_precision_3: 0.7954 - val_recall_3: 0.5969
Epoch 5/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 12506s 2s/step - accuracy: 0.8681 -
auc_3: 0.9074 - loss: 0.3113 - precision_3: 0.7871 - recall_3: 0.6084 - val_accuracy: 0.8695 -
val_auc_3: 0.9104 - val_loss: 0.3067 - val_precision_3: 0.7850 - val_recall_3: 0.6190
Epoch 6/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 16237s 3s/step - accuracy: 0.8697 -
auc_3: 0.9094 - loss: 0.3080 - precision_3: 0.7878 - recall_3: 0.6146 - val_accuracy: 0.8694 -
val_auc_3: 0.9108 - val_loss: 0.3092 - val_precision_3: 0.7685 - val_recall_3: 0.6427
Epoch 7/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 12903s 2s/step - accuracy: 0.8709 -
auc_3: 0.9112 - loss: 0.3055 - precision_3: 0.7898 - recall_3: 0.6219 - val_accuracy: 0.8701 -
val_auc_3: 0.9112 - val_loss: 0.3073 - val_precision_3: 0.7773 - val_recall_3: 0.6338
```

***** Training GRU *****

```
Epoch 1/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 13305s 2s/step - accuracy: 0.8331 -
auc_4: 0.8452 - loss: 0.3870 - precision_4: 0.7247 - recall_4: 0.4735 - val_accuracy: 0.8601 -
val_auc_4: 0.8974 - val_loss: 0.3265 - val_precision_4: 0.7597 - val_recall_4: 0.5994
Epoch 2/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 11113s 2s/step - accuracy: 0.8564 -
auc_4: 0.8893 - loss: 0.3371 - precision_4: 0.7692 - recall_4: 0.5613 - val_accuracy: 0.8641 -
val_auc_4: 0.9018 - val_loss: 0.3202 - val_precision_4: 0.7588 - val_recall_4: 0.6254
Epoch 3/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 11556s 2s/step - accuracy: 0.8597 -
auc_4: 0.8963 - loss: 0.3282 - precision_4: 0.7733 - recall_4: 0.5768 - val_accuracy: 0.8654 -
val_auc_4: 0.9047 - val_loss: 0.3172 - val_precision_4: 0.7861 - val_recall_4: 0.5938
Epoch 4/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 15766s 3s/step - accuracy: 0.8620 -
auc_4: 0.8983 - loss: 0.3250 - precision_4: 0.7793 - recall_4: 0.5843 - val_accuracy: 0.8654 -
val_auc_4: 0.9059 - val_loss: 0.3151 - val_precision_4: 0.7800 - val_recall_4: 0.6021
Epoch 5/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 16784s 3s/step - accuracy: 0.8618 -
auc_4: 0.8998 - loss: 0.3237 - precision_4: 0.7793 - recall_4: 0.5842 - val_accuracy: 0.8666 -
val_auc_4: 0.9069 - val_loss: 0.3135 - val_precision_4: 0.7752 - val_recall_4: 0.6155
Epoch 6/10
```

```
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 13680s 2s/step - accuracy: 0.8633 -
auc_4: 0.9009 - loss: 0.3217 - precision_4: 0.7807 - recall_4: 0.5898 - val_accuracy: 0.8673 -
val_auc_4: 0.9075 - val_loss: 0.3127 - val_precision_4: 0.7775 - val_recall_4: 0.6170
Epoch 7/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 15549s 3s/step - accuracy: 0.8639 -
auc_4: 0.9019 - loss: 0.3196 - precision_4: 0.7821 - recall_4: 0.5879 - val_accuracy: 0.8666 -
val_auc_4: 0.9069 - val_loss: 0.3138 - val_precision_4: 0.7951 - val_recall_4: 0.5890
Epoch 8/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 13587s 2s/step - accuracy: 0.8642 -
auc_4: 0.9027 - loss: 0.3188 - precision_4: 0.7798 - recall_4: 0.5927 - val_accuracy: 0.8658 -
val_auc_4: 0.9066 - val_loss: 0.3176 - val_precision_4: 0.7482 - val_recall_4: 0.6539
```

***** Training RNN *****

```
Epoch 1/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 1497s 261ms/step - accuracy: 0.8006
- auc_5: 0.7736 - loss: 0.4512 - precision_5: 0.6494 - recall_5: 0.3303 - val_accuracy: 0.8364 -
val_auc_5: 0.8618 - val_loss: 0.3846 - val_precision_5: 0.7466 - val_recall_5: 0.4693
Epoch 2/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 1487s 259ms/step - accuracy: 0.8286
- auc_5: 0.8427 - loss: 0.3936 - precision_5: 0.7207 - recall_5: 0.4513 - val_accuracy: 0.8432 -
val_auc_5: 0.8693 - val_loss: 0.3642 - val_precision_5: 0.7757 - val_recall_5: 0.4762
Epoch 3/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 1509s 263ms/step - accuracy: 0.8309
- auc_5: 0.8471 - loss: 0.3891 - precision_5: 0.7262 - recall_5: 0.4582 - val_accuracy: 0.8416 -
val_auc_5: 0.8707 - val_loss: 0.3651 - val_precision_5: 0.6952 - val_recall_5: 0.5907
Epoch 4/10
5735/5735 ━━━━━━━━━━━━━━━━━━━━ 1516s 264ms/step - accuracy: 0.8344
- auc_5: 0.8542 - loss: 0.3824 - precision_5: 0.7343 - recall_5: 0.4773 - val_accuracy: 0.8422 -
val_auc_5: 0.8759 - val_loss: 0.3643 - val_precision_5: 0.6755 - val_recall_5: 0.6430
```