SC4002: Natural Language Processing Assignment

Academic Year 2024-2025 Semester 1

Group 11

Group 11's Members & Contributions:

Name	Matriculation Number	Contributions
Lew Jian Ben Timothy	U2121855L	Part 1, 3, Report Writing
John Paul Lam Wei Keong	U2120120C	Part 1, 2, Report Writing
Lim Qing Chuan	U2021199C	Part 2, Report Writing
Derrick Ng Choon Seng	U2122873F	Part 3, Report Writing
Dexter Voon Kai Xian	U2120267F	Part 3, Report Writing
Lim Jia Earn	U2122747A	Part 3, Report Writing

Table of Contents

1. Word Embedding 1
2. RNN Model Training and Evaluation 2
3. Enhancements to Improve Performance 3
- **3.1. Effect of Updating Word Embeddings** 3
- **3.2. Mitigating OOV Words** 3
- **3.3. biLSTM & biGRU Models** 4
  - **a. biLSTM Results** 4
  - **b. BiGRU Results** 4
- **3.4. CNN Model** 5
  - **a. Base Model** 5
  - **b. Enhanced Model** 5
- **3.5. Strategy Enhancements** 7
- **3.6. Best Performing Strategy** 8

4. Conclusion 8
5. Appendix 9
6. References 16

1. Word Embedding

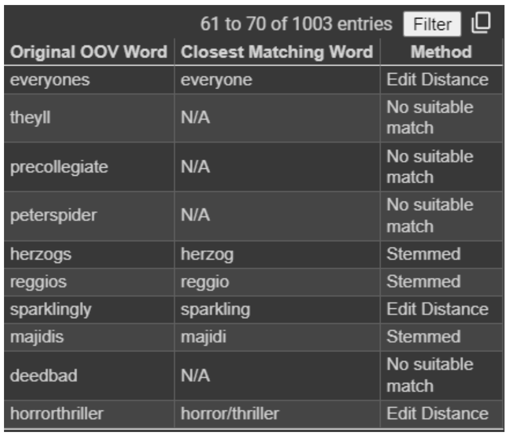
Before preparing word embeddings for the dataset, the dataset must first be normalized and then tokenized. All words in the corpus had non-printable characters removed, special characters including punctuation were removed, with the exception of hyphens, which were converted to a space with regex. They were then converted to lowercase and tokenized using NLTK. After tokenization the size of the vocabulary of the train dataset is 17,016.

Next, we considered different pre-trained embeddings from either GloVe or Word2Vec models. The different pre-trained GloVe and Word2Vec models resulted in varying numbers of out-of-vocabulary (OOV) words, as shown in Table 1.1. We ultimately decided to proceed with the GloVe 6B.300d model because it was significantly smaller in file size compared to the 840b.300d model, yet with only a slightly higher OOV count. Additionally, Glove 6b.300d had roughly half as many OOV words compared to the Word2Vec variant, word2vec-google-news-300. The word2vec-google-news-300 model showed noticeably more OOV words than the GloVe models, likely because news articles tend to use more formal language, whereas our movie review dataset contains informal language, such as abbreviations and slang, that may not appear in news content.

To handle OOV words, we applied the Porter stemmer to the OOV words and checked if the stemmed form existed in the pretrained embeddings. If no match was found, we then proceeded to find the closest word using edit distance, restricted to words that shared the same initial letter. By restricting candidate words to those with the same first letter as the OOV word, it prevented unintentional semantic inversions (e.g., OOV: "irreversible," candidate: "reversible") and also reduced the search space.

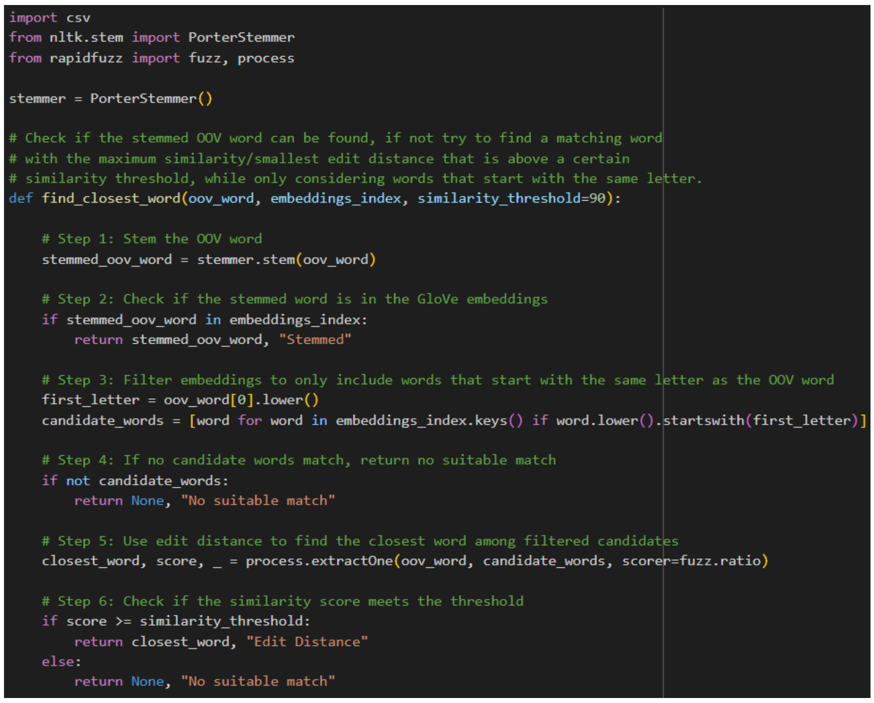
We further ensured that the similarity score between the OOV word and candidate words exceeded a predefined threshold of a fuzz ratio of 0.9, using the RapidFuzz library. The fuzz ratio calculates the normalized indel similarity, similar to the Levenshtein distance except that the ratio will be restricted to the range of [0, 1].

This method of handling OOV was computationally efficient, taking 1-2 minutes in total to find close matches for the OOV. It effectively resolved OOV words caused by typos, with 615/1003 (61.3%) of OOV words able to be resolved using this method, as shown in Figure 1.1. Of the 615 words that were able to be resolved, 336 were resolved using stemming and 279 by edit distance.



61 to 70 of 1003 entries Filter

Original OOV Word	Closest Matching Word	Method
everyones	everyone	Edit Distance
theyll	N/A	No suitable match
precollegiate	N/A	No suitable match
peterspider	N/A	No suitable match
herzogs	herzog	Stemmed
reggios	reggio	Stemmed
sparklingly	sparkling	Edit Distance
majidis	majidi	Stemmed
deedbad	N/A	No suitable match
horrorthriller	horror/thriller	Edit Distance



```python
import csv
from nltk.stem import PorterStemmer
from rapidfuzz import fuzz, process

stemmer = PorterStemmer()

# Check if the stemmed OOV word can be found, if not try to find a matching word
# with the maximum similarity/smallest edit distance that is above a certain
# similarity threshold, while only considering words that start with the same letter.
def find_closest_word(oov_word, embeddings_index, similarity_threshold=90):

    # Step 1: Stem the OOV word
    stemmed_oov_word = stemmer.stem(oov_word)

    # Step 2: Check if the stemmed word is in the GloVe embeddings
    if stemmed_oov_word in embeddings_index:
        return stemmed_oov_word, "Stemmed"

    # Step 3: Filter embeddings to only include words that start with the same letter as the OOV word
    first_letter = oov_word[0].lower()
    candidate_words = [word for word in embeddings_index.keys() if word.lower().startswith(first_letter)]

    # Step 4: If no candidate words match, return no suitable match
    if not candidate_words:
        return None, "No suitable match"

    # Step 5: Use edit distance to find the closest word among filtered candidates
    closest_word, score, _ = process.extractOne(oov_word, candidate_words, scorer=fuzz.ratio)

    # Step 6: Check if the similarity score meets the threshold
    if score >= similarity_threshold:
        return closest_word, "Edit Distance"
    else:
        return None, "No suitable match"
```

Figure 1.1: Code snippet to resolve OOV words with examples of it working and failing.

Table 1.1: Comparison between number of OOV words for different word embedding models. GloVe 6B.300d was chosen for our *project* (highlighted in gray).

Model	**Variant**	**File Size (MB)**	**Trained On**	**Number of OOV**
GloVe	6B.300d	1014	Wikipedia 2014 + Gigaword 5	1003
GloVe	840B.300d	5513	Common Crawl	923
Word2Vec	word2vec-google-news-300	3600	News Sources	2059

2. RNN Model Training and Evaluation

We designed and trained an RNN using Keras and Tensorflow to predict a sentiment label (positive - 1 or negative - 0) for each sentence. For each sentence in the training dataset, a word to index mapping was created, mapping each word in the vocabulary to a unique index. As the number of unique words in the training vocabulary was 17,016 and not inordinately large, all words in the training set are kept in the mapping, even if their frequency was low. 2 extra mappings were created, with index 0 being a zero vector for padding, and index 1 is reserved for OOV words, the vector assigned for OOV words was a fixed randomly generated vector.

The rationale behind this was that OOV words should not be treated the same as padding, as padding just serves to standardize input lengths into the model while OOV words can carry meaningful information.

Next, using the word to index mapping, each review is converted into a sequence of fixed-length (51 tokens long) word indices, and sequences shorter than the max_sequence_length were padded with 0. The fixed sequence length was determined by the longest sequence in the train dataset which was 51 tokens long.

Next, the embedding matrix is created. Each row in the embedding matrix corresponds to a word index, and contains the word embedding vector.

Our initial simple RNN model has the following structure:

1. Embedding layer - maps each word index in sequence to GloVe embedding vector (this layer is not trainable for part 2)
2. A simple RNN layer
3. A dropout layer
4. A simple RNN layer
5. A dropout layer
6. An output dense layer with a sigmoid activation function to produce a probability indicating the likelihood of positive or negative sentiment

The model was compiled with the Adam optimizer, early stopping callback and the loss function used was binary cross entropy loss, and the accuracy metric was being tracked. The best hyperparameters for this model is shown in Table 2.1.1. The test accuracy of this model was 71.85%.

The decision to use two RNN layers in the model architecture was aimed at enhancing the model's ability to capture more complex and nuanced patterns within the data. The first RNN layer primarily identifies basic dependencies in the sequence data, such as short-term relationships between adjacent words. The second RNN layer processes the output of the first, which consists of preliminary extracted features, allowing it to capture higher-order dependencies, such as longer-term relationships and more intricate combinations of features indicative of sentiment.

This two-layered approach increases the model's capacity to generalize, particularly with complex language data, where sentiment can depend on subtle, long-distance dependencies across words. By introducing a second RNN layer, the model achieves an expanded ability to detect nuanced sentiment features and relationships within sentences.

Additionally, the layered design allows for a hierarchical feature representation. The first layer captures fundamental sentiment indicators, while the second layer focuses on refining these into more abstract, high-level features. This hierarchical learning structure enhances the model's capability to capture both short and long-term dependencies, ultimately contributing to improved performance in sentiment prediction.

Table 2.1.1: Hyperparameter settings for the RNN model.

Hyperparameter	**Best Value**	**Remarks**
Number of training epochs	32	Early stopping triggered. Original value was 160 epochs
Learning Rate	$6e^{-05}$	
Optimizer	Adam	
Batch Size	32	
SimpleRNN Units	90	For both SimpleRNN layers
Dropout	0.2	For both dropout layers

The previous simpleRNN model used the last hidden state to represent the sentence. We then tried to use max pooling and average pooling from all the hidden states for the representation. From our training/validation training curve *(Figure 2.1.2)*, we can observe the max pooling model obtained an accuracy of 75.14% on the test set and the average pooling method achieved an accuracy of 73.35% on the test set. The final best configuration (max pooling) for the RNN model is shown in Table 2.1.1 and *(Figure 2.1.3)* and *(Figure 2.1.4)*.

3. Enhancements to Improve Performance

3.1. Effect of Updating Word Embeddings

Upon enabling the embeddings to be trainable, the test accuracy has increased from 75.14% to 76.92% (*Figure 3.1*). The improvement in test accuracy can be attributed to the model's ability to fine-tune the word embeddings specifically for the sentiment analysis task.

3.2. Mitigating OOV Words

Upon implementing our solution to mitigate the effects of OOV words, the test accuracy improved marginally from 76.92% to 77.57% (*Figure 3.2*). This could be because our OOV mitigation strategy is to find the closest word that exists in the pretrained GloVe embeddings, but it is improbable that all possible words that can appear in the review also appear in the pretrained GloVe embeddings, which could be why this strategy was not very successful.

3.3. biLSTM & biGRU Models

a. biLSTM Results

Referencing Part 2, each simple RNN layer was replaced with a biLSTM layer. Layer normalization has been added between each biLSTM and dropout layer to normalize activations within each layer and stabilize the training process. An attention layer after both biLSTM layers was also added to improve the model by enabling it to learn context and focus on more important sequences.

After hyperparameters tuning, the model with structure shown in *Figure 3.3.1* was trained based on the following hyperparameters:

Table 3.3.1: Hyperparameters for biLSTM model

Training epochs: 21 (original 160)	Learning rate: $7e^{-4}$	Batch size = 32
biLSTM units: 9	L2 regularization: 0.02	Dropout: 0.2

Upon training the biLSTM model, the test accuracy improved from 77.57% to 78.14% (*Figure 3.3.2*). This is because Long-Short Term Memory (LSTM) models are able to retain long-term dependencies in data better than simple RNN models. Along with the additional layers to improve the training and performance of the LSTM model, a better performing model was produced compared to the simple RNN model.

b. BiGRU Results

The Bidirectional Gated Recurrent Unit (BiGRU) is a form of the standard GRU, which is a type of Recurrent Neural Network (RNN) designed to address the vanishing gradient problem and improve the learning of long-range dependencies in sequential data. The main feature is its bidirectional architecture which processes input data in both forward and backward directions to learn context from both past and future sequences, providing the model with more extensive context.

From *Figure 3.3.4*, we used 1 bidirectional layer in our model with 140544 parameters and output shape of (None, 128).

Our BiGRU model has the following structure:

1. An Input layer
2. A Bidirectional layer
3. A Dropout layer
4. A Dense layer

Upon training the biGRU model, the test accuracy improved from 77.57% to 79.08% (*Figure 3.3.4*). Similar to LSTM, GRU is able to handle long-term dependencies better than a simple RNN, thus performing better. The biGRU model also performed better than the biLSTM model. This may be because GRUs have fewer parameters than LSTMs, allowing them to train faster and more efficiently compared to LSTM.

3.4. CNN Model

a. Base Model

Building upon the sequential modeling approaches of previous parts, our group also implemented a CNN-based model which leverages spatial features in our dataset. This implementation focuses on identifying local patterns and n-gram features that are relevant for sentiment classification.

Our base CNN architecture *(Figure 3.4.1)* consists of several key components:

1. Parallel Convolutional Layers
    - filter sizes of 2, 3, 4, 5 to capture various n-gram patterns
    - 256 filters / convolutional layer for comprehensive feature extraction

2. Feature Aggregation
    - Global max pooling to capture the most salient features from each filter
    - Dense layers for feature combination and final classification

3. Regularization Strategy
    - Dropout rates of 0.4 and 0.3 in dense layers
    - L2 regularization (0.0001) on convolutional layers
    - This regularization approach builds upon the successful dropout strategies used in our biLSTM/biGRU implementations

This base model achieved a <u>test accuracy of 75.33%</u> and a <u>test loss of 0.5894.</u>

b. Enhanced Model

We can further enhance our base model by tackling 4 main limitations which can be observed from the training result plots *(Figure 3.4.2)* of the base CNN model.

Observation	Problem	Solution
Our training accuracy approached very close to 1 by epoch 8 while validation accuracy plateaued at around 0.77	Potential for overspecialisation to training data	● Added dual convolutional blocks with skip connections to enable hierarchical feature learning ● Introduced balanced pooling strategy (combining average and max pooling) to capture both dominant and overall patterns ● Reduced filter count from to prevent excessive feature memorization ● Optimized filter sizes to [3, 4, 5] based on feature importance analysis ● Simplified dense layer architecture while maintaining model capacity

1. Divergence of training and validation curves 2. Training loss continuously decreasing while validation loss increasing after epoch 4	Signs of overfitting	● Introduced SpatialDropout1D(0.3) specifically for embedding regularization ● Increased L2 regularization from 0.0001 to 0.001 ● Added BatchNormalization layers with momentum for more stable training ● Increased dense layer dropout to 0.5 ● Reduced total parameters to 5.87M (from 6.77M)
Fluctuating validation accuracy despite increasing training accuracy	Potential class imbalances and training dynamics	● Introduced class weights to balance training ● Implemented AdamW optimizer with weight decay ● Added ReduceLROnPlateau callback for adaptive learning rate ● Increased batch size for more stable gradient estimates

The new CNN architecture *(Figure 3.4.3)* achieved a test accuracy of 78.71% and a test loss of 1.3699.

The new training result plots *(Figure 3.4.4)* demonstrates:

1. Smaller metric difference in training and validation
2. Higher test accuracy
3. Better generalization through our higher validation loss

These enhancements resulted in a more robust model with better feature learning capacity and generalization abilities.

3.5. Strategy Enhancements

The attention (keras.layers.attention) used in the previous biLSTM models used the dot product between query and keys, known as Luong-style attention. However, we found that using Bahdanau attention or Additive attention which uses a one-hidden layer feed-forward network to calculate the attention alignment score between queries and keys improved the accuracy of the biLSTM model, as shown in Table 3.5.1. This may be because Bahdanau attention with its feed-forward neural network can capture more nuanced relationships compared to the dot-product approach, albeit with more parameters to be trained.

To further improve the performance of the previous models (BiLSTM, GRU, CNN) and subsequent transformer models, we performed data augmentation with MarianMT to increase the size of the train dataset. This was achieved by translating each review from the source language (English) to a pivot language and then translating it back to the source language. We generated six augmented samples for each original text with 6 pivot languages (French, German, Spanish, Italian, Danish, Chinese), increasing linguistic diversity and providing alternate phrasings of the original train dataset. With the augmented train set, the models can learn to recognize multiple phrasings of the same content, making it more resilient to variations in input. This strategy yielded improvement to both train and valid accuracy for most models, as shown in Table 3.5.1.

We also attempted other data augmentation strategies, such as noise insertion, synonym replacement and text shuffling, but all strategies unfortunately resulted in a significant decrease in validation and test accuracy. This could be because noise insertion may have introduced irrelevant or misleading information into the training data, making it harder for the model to learn meaningful patterns and potentially obscuring the underlying sentiment signals. Similarly, synonym replacement may have inadvertently altered the semantic nuances or sentiment conveyed in the original texts. For instance, certain synonyms may carry different connotations or intensities, leading to inconsistencies between the input data and their corresponding labels. Text shuffling might have caused sentence structures to be broken and loss in context between words. This might cause a decrease in performance, especially since the biLSTM model relies on previous and next words to understand context.

Table 3.5.1: Improvements to Test Accuracy After training with data augmentation using back translation

Model	**Test Accuracy Before Data Augmentation**	**New Test Accuracy After Data Augmentation**
BiLSTM (with Luong Attention)	78.14%	77.7%
BiLSTM (with Bahdanau Attention)	79.74%	81.3%
GRU	79.08%	79.8%
CNN	78.99%	79.5%

However, all the aforementioned strategies have the same downfall even with attention layers added, because GloVe was used as the embedding to begin with.

As compared to BERT, GloVe and Word2Vec embeddings are unable to capture the contextual nuances of a word, for example, words like ‘bank’ would have the same vector representation whether it refers to a financial institution or a river. Moreover, although we implemented OOV handling, it is not as effective as the WordPiece tokenizer used by BERT, which breaks down words into smaller subunits and can handle out-of-vocabulary words by combining embeddings of subwords, thus offering greater flexibility with new or rare words. The advantages of BERT will be further discussed in Chapter 3.6.

Therefore, our final strategy involves using a variant of BERT, an encoder-only transformer, to leverage its contextual embedding capabilities for improved model performance.

3.6. Best Performing Strategy

Our final strategy employs RoBERTa, an optimized variant of BERT developed by Facebook AI, which builds upon BERT with several key improvements, which include:

1. Enhanced Training Regimen: RoBERTa was trained on more data and with longer sequences, enhancing its ability to understand and model complex language patterns.
2. Dynamic Masking: Unlike BERT's static masking, RoBERTa employs dynamic masking, which provides the model with a more diverse range of masked language modeling tasks during training.
3. Byte-Level Byte-Pair Encoding (BPE) Tokenizer: This tokenizer reduces the incidence of OOV tokens and better handles rare words, improving the model's robustness. RoBERTa achieves this by using BPE with bytes as a subunit instead of characters, accommodating Unicode characters.

Using the augmented training dataset from Chapter 3.5, the following preprocessing steps were performed:

1. Tokenization: The tokenization process included padding and truncation to a maximum sequence length of 128 tokens for uniform input size from the augmented data file. Tokenized data was mapped to include input_ids, attention_mask, and labels.
2. TensorFlow Conversion: The tokenized datasets were converted to TensorFlow-compatible format with input features (input_ids, attention_mask) and integer-type labels.
3. Batching: These datasets were batched with a batch size of 16 for efficient training and evaluation.

Using the create_optimizer function from the transformers library, we configured an optimizer with an initial learning rate of 2e-5. The setup considered the total number of training steps across 2 epochs and included a warmup phase of 500 steps, followed by a linear decay schedule for the learning rate. The model was compiled with the Sparse Categorical Cross Entropy loss function and accuracy as the evaluation metric. Training was performed over 2 epochs.

BERT's ability to read text bidirectionally allows it to simultaneously consider both the left and right contexts of each word. This bidirectional reading approach helps BERT capture word meanings based on their surroundings, enhancing its ability to discern nuanced meanings compared to unidirectional models. BERT's WordPiece tokenizer further strengthens this flexibility by breaking words into smaller subunits, which aids in handling OOV words while preserving semantic integrity.

In comparison to BiLSTM, which processes text sequentially and can struggle with long dependencies, RoBERTa leverages self-attention mechanisms to capture complex dependencies and contextual relationships more effectively. The transformer architecture of RoBERTa, with its approximately 125M parameters, surpasses the parameter count of BiLSTM and GRU models (~9-10M), enabling it to model intricate language patterns and deliver superior performance in sentiment analysis tasks.

Our final test accuracy using RoBERTa was 88.9%, which can be seen in Figure 3.6.1.

4. Conclusion

This project has allowed us to learn more and apply our knowledge of NLP for sentiment analysis, and appreciate the advancements made in NLP from the era of GloVe (2014) to modern approaches in the era of Transformers, through the use of BERT (2018) and RoBERTa (2019).

5. Appendix

Figure 2.1.2: Training and Validation Accuracy of the SimpleRNN model (last hidden state)

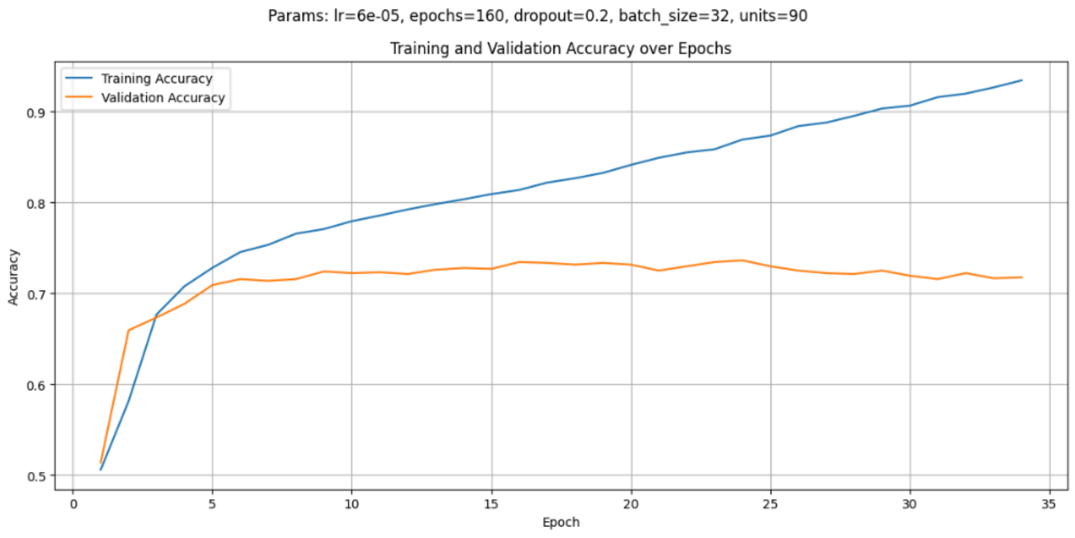
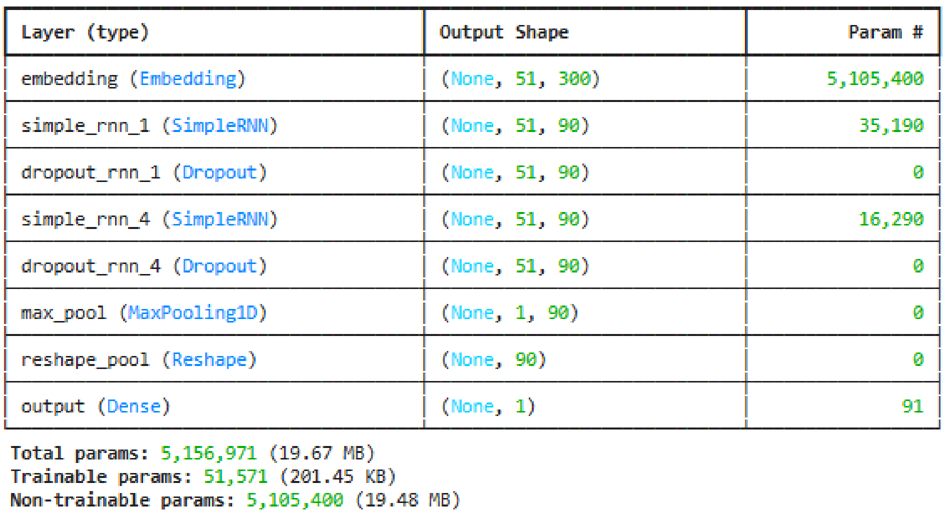


Figure 2.1.3: Model Summary of the SimpleRNN model (max pooling)



Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 51, 300)	5,105,400
simple_rnn_1 (SimpleRNN)	(None, 51, 90)	35,190
dropout_rnn_1 (Dropout)	(None, 51, 90)	0
simple_rnn_4 (SimpleRNN)	(None, 51, 90)	16,290
dropout_rnn_4 (Dropout)	(None, 51, 90)	0
max_pool (MaxPooling1D)	(None, 1, 90)	0
reshape_pool (Reshape)	(None, 90)	0
output (Dense)	(None, 1)	91

```
Total params: 5,156,971 (19.67 MB)
Trainable params: 51,571 (201.45 KB)
Non-trainable params: 5,105,400 (19.48 MB)
```

Figure 2.1.4: Training and Validation Accuracy of the SimpleRNN model (max pooling)

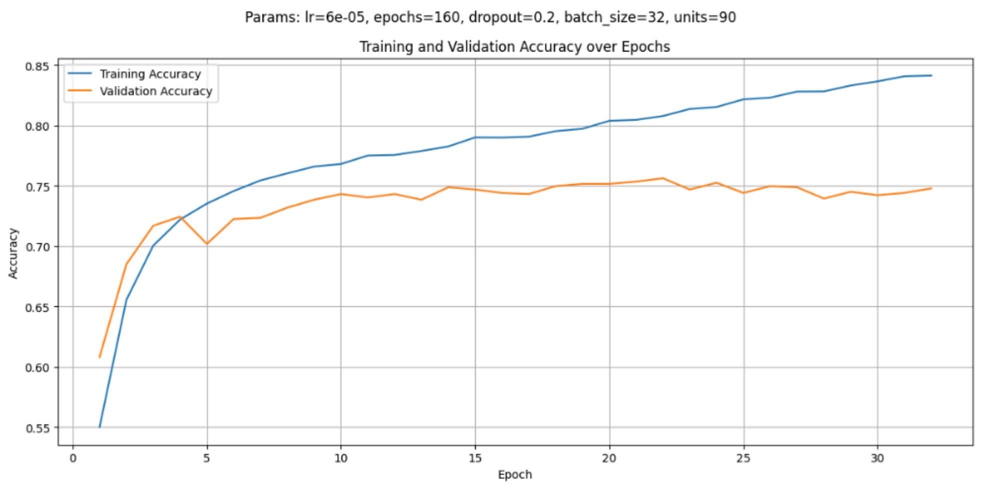
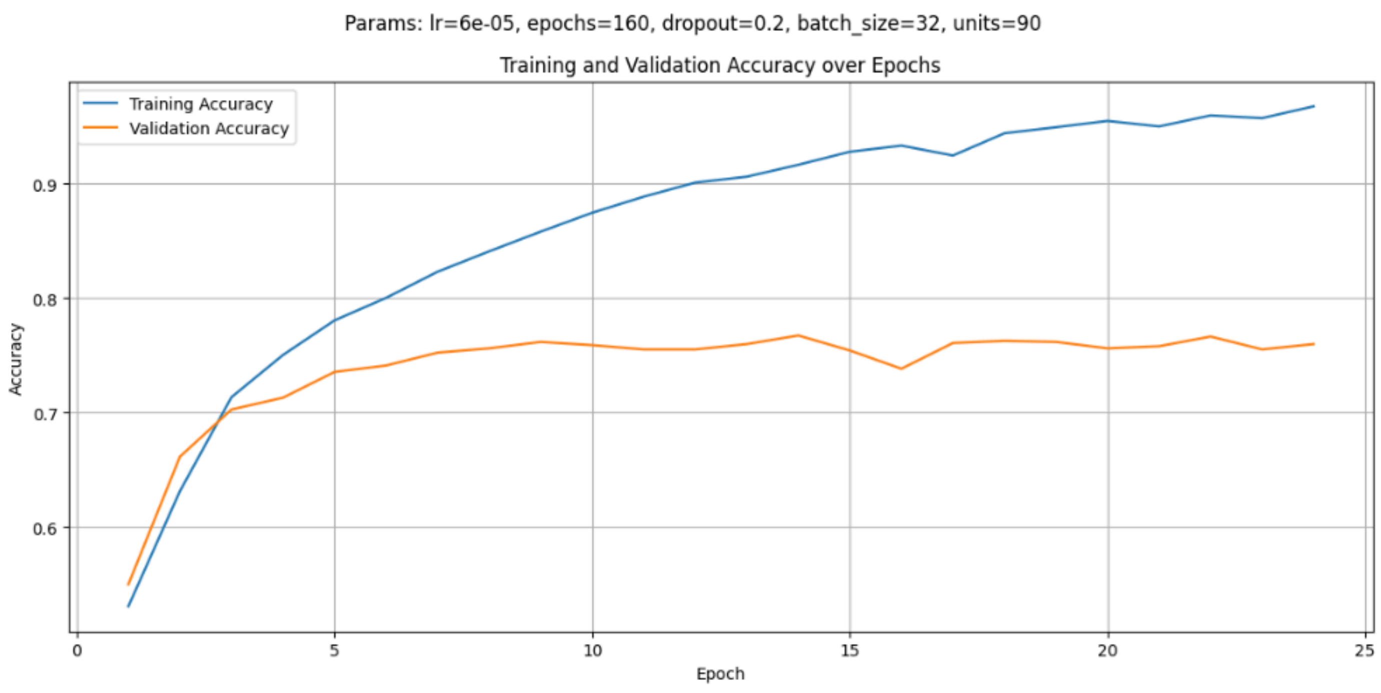


Figure 3.1 : Embeddings Trainable Test Results

```
Test Loss: 1.1867082118988037
Test Accuracy: 0.7692307829856873
```



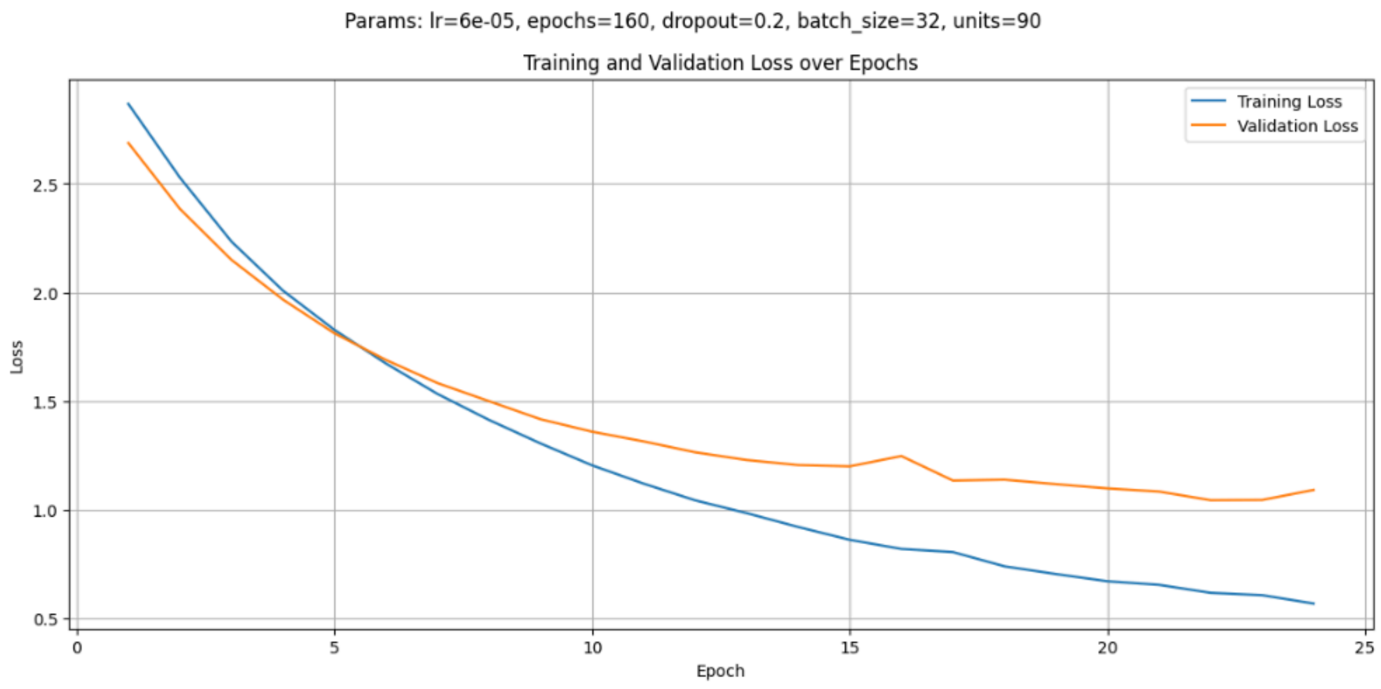
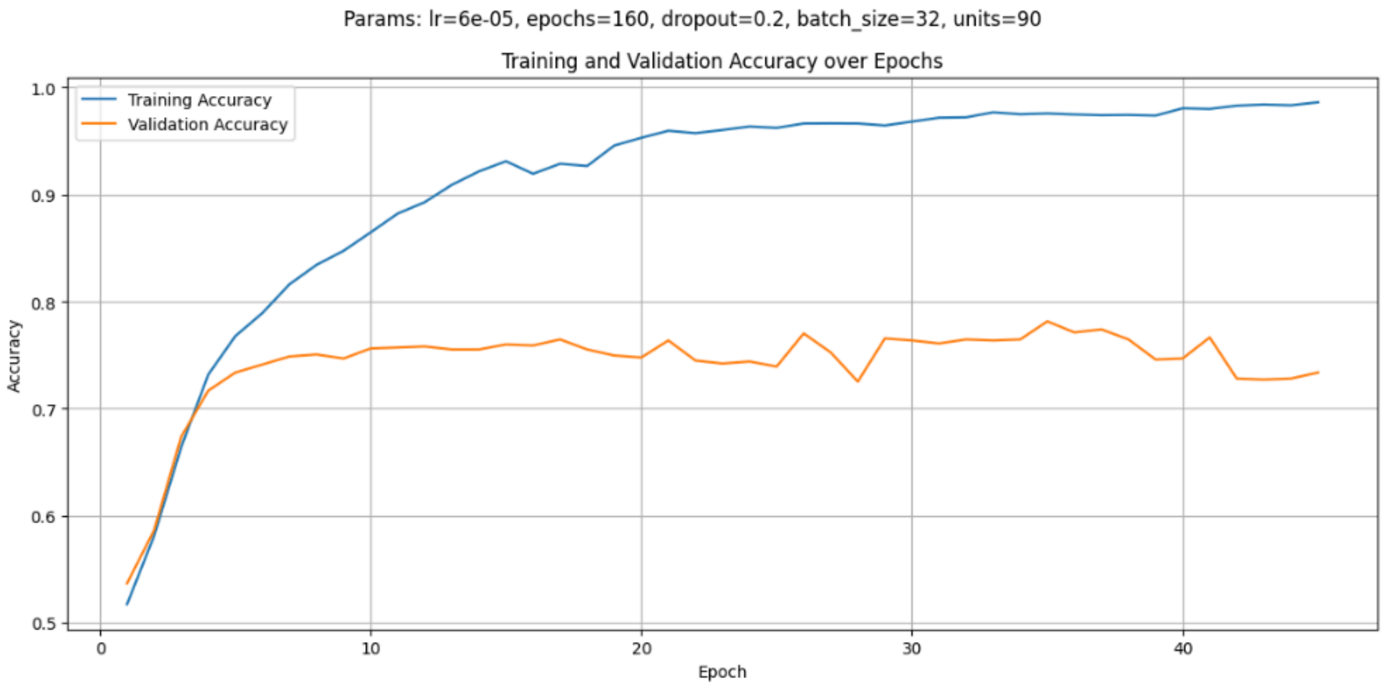


Figure 3.2 : OOV Mitigation Test Results

```
Test Loss: 0.9669904112815857
Test Accuracy: 0.7757973670959473
```



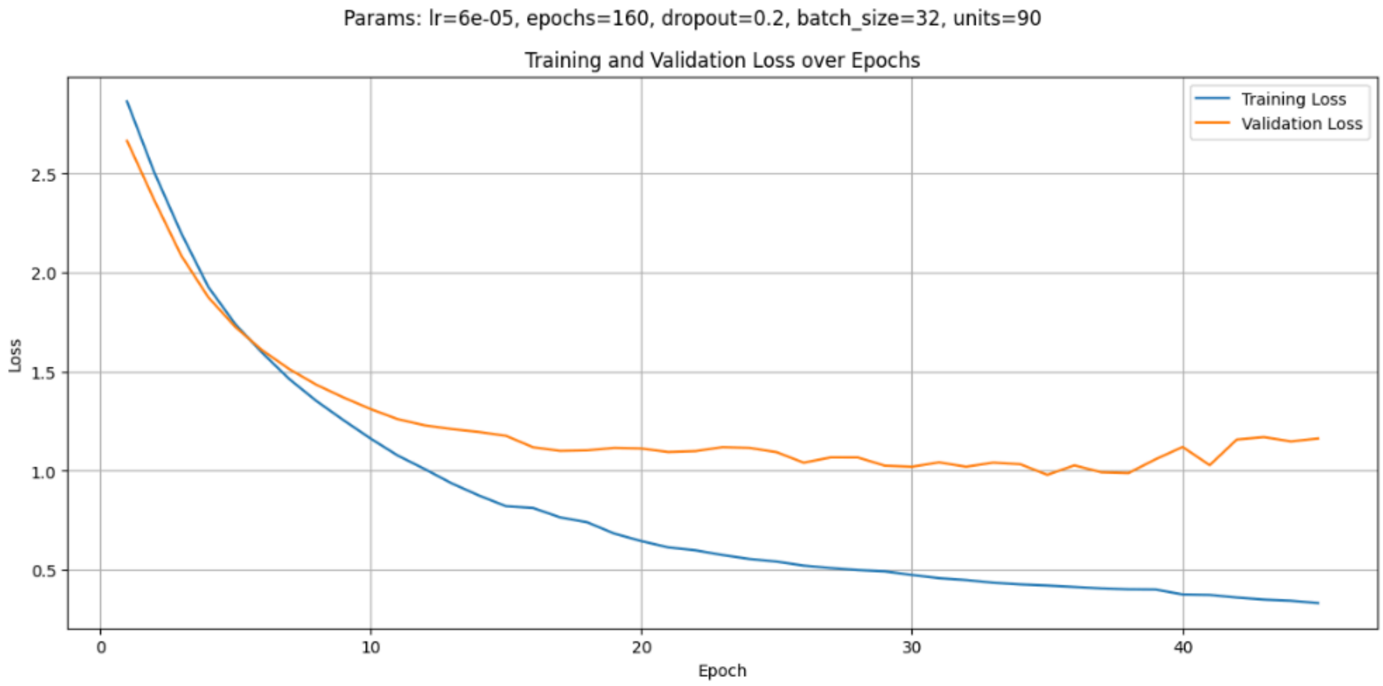
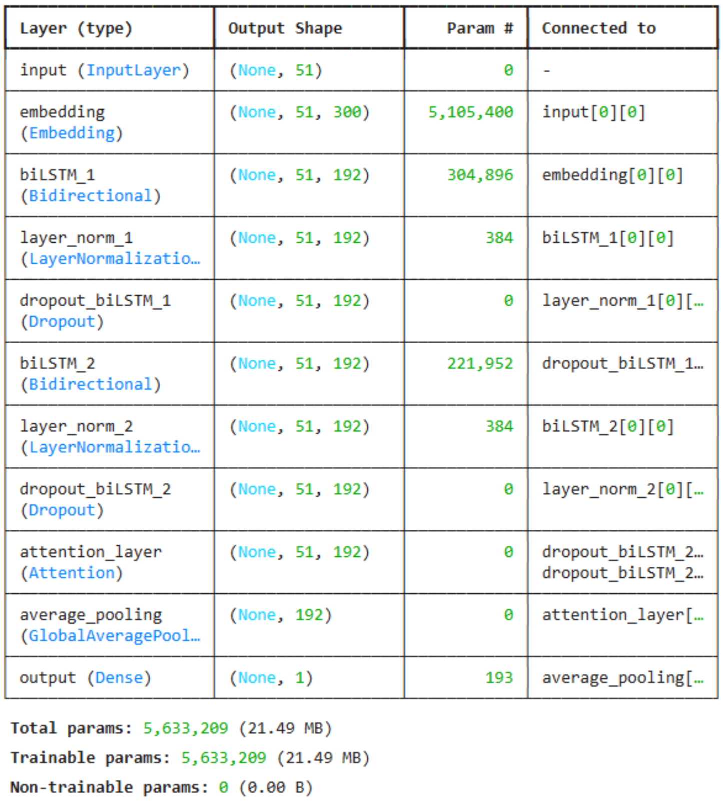


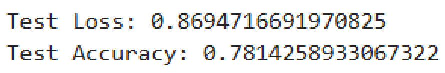
Figure 3.3.1 : biLSTM Model Architecture



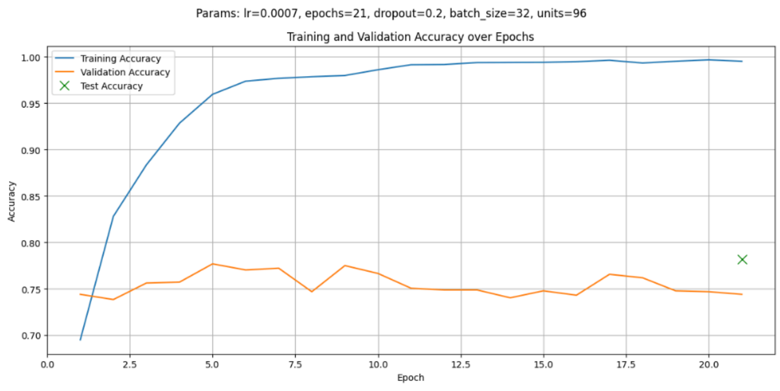
Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 51)	0	-
embedding (Embedding)	(None, 51, 300)	5,105,400	input[0][0]
biLSTM_1 (Bidirectional)	(None, 51, 192)	304,896	embedding[0][0]
layer_norm_1 (LayerNormalizatio…	(None, 51, 192)	384	biLSTM_1[0][0]
dropout_biLSTM_1 (Dropout)	(None, 51, 192)	0	layer_norm_1[0][…
biLSTM_2 (Bidirectional)	(None, 51, 192)	221,952	dropout_biLSTM_1…
layer_norm_2 (LayerNormalizatio…	(None, 51, 192)	384	biLSTM_2[0][0]
dropout_biLSTM_2 (Dropout)	(None, 51, 192)	0	layer_norm_2[0][…
attention_layer (Attention)	(None, 51, 192)	0	dropout_biLSTM_2… dropout_biLSTM_2…
average_pooling (GlobalAveragePool…	(None, 192)	0	attention_layer[…
output (Dense)	(None, 1)	193	average_pooling[…

```
Total params: 5,633,209 (21.49 MB)
Trainable params: 5,633,209 (21.49 MB)
Non-trainable params: 0 (0.00 B)
```

Figure 3.3.2 : biLSTM Test Results



```
Test Loss: 0.8694716691970825
Test Accuracy: 0.7814258933067322
```



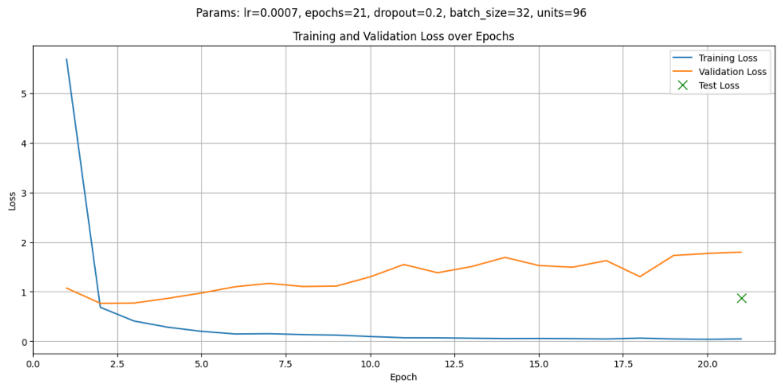
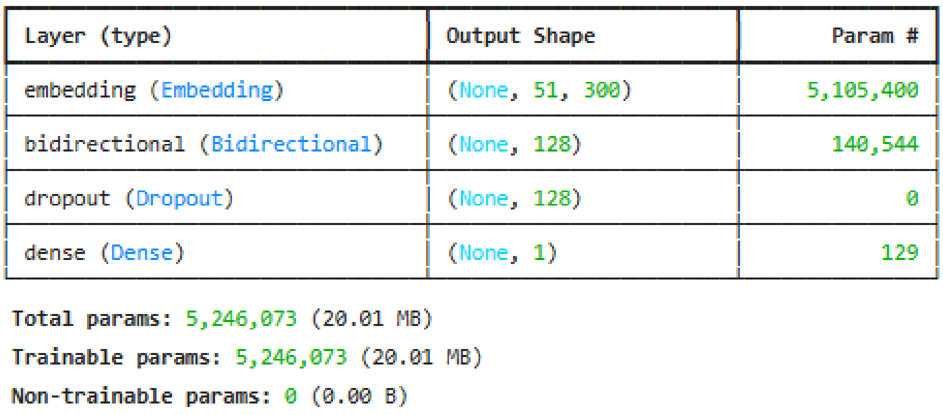


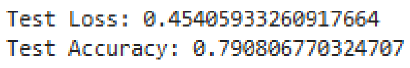
Figure 3.3.3 : biGRU Model Architecture



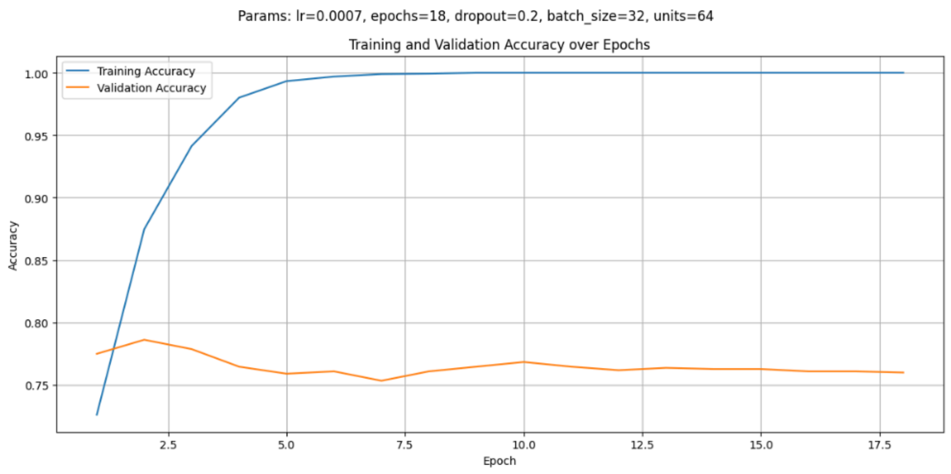
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 51, 300)	5,105,400
bidirectional (Bidirectional)	(None, 128)	140,544
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 1)	129

```
Total params: 5,246,073 (20.01 MB)
Trainable params: 5,246,073 (20.01 MB)
Non-trainable params: 0 (0.00 B)
```

Figure 3.3.4 : biGRU Test Results



```
Test Loss: 0.45405933260917664
Test Accuracy: 0.790806770324707
```



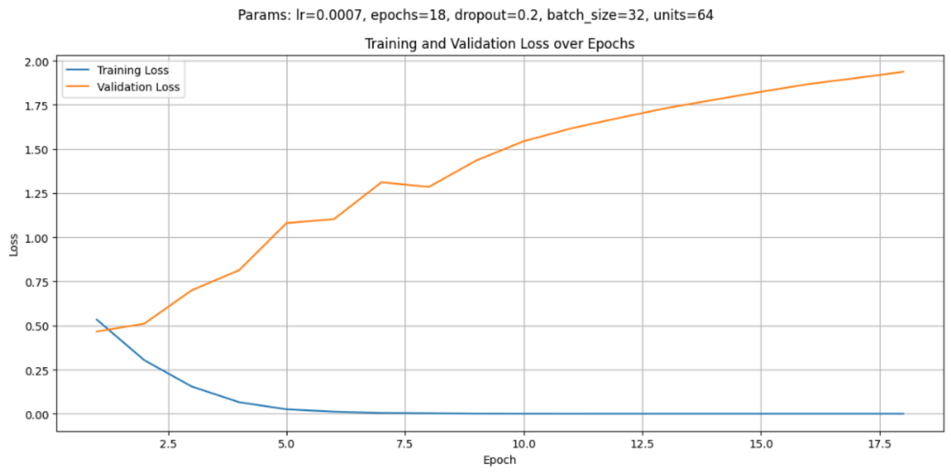
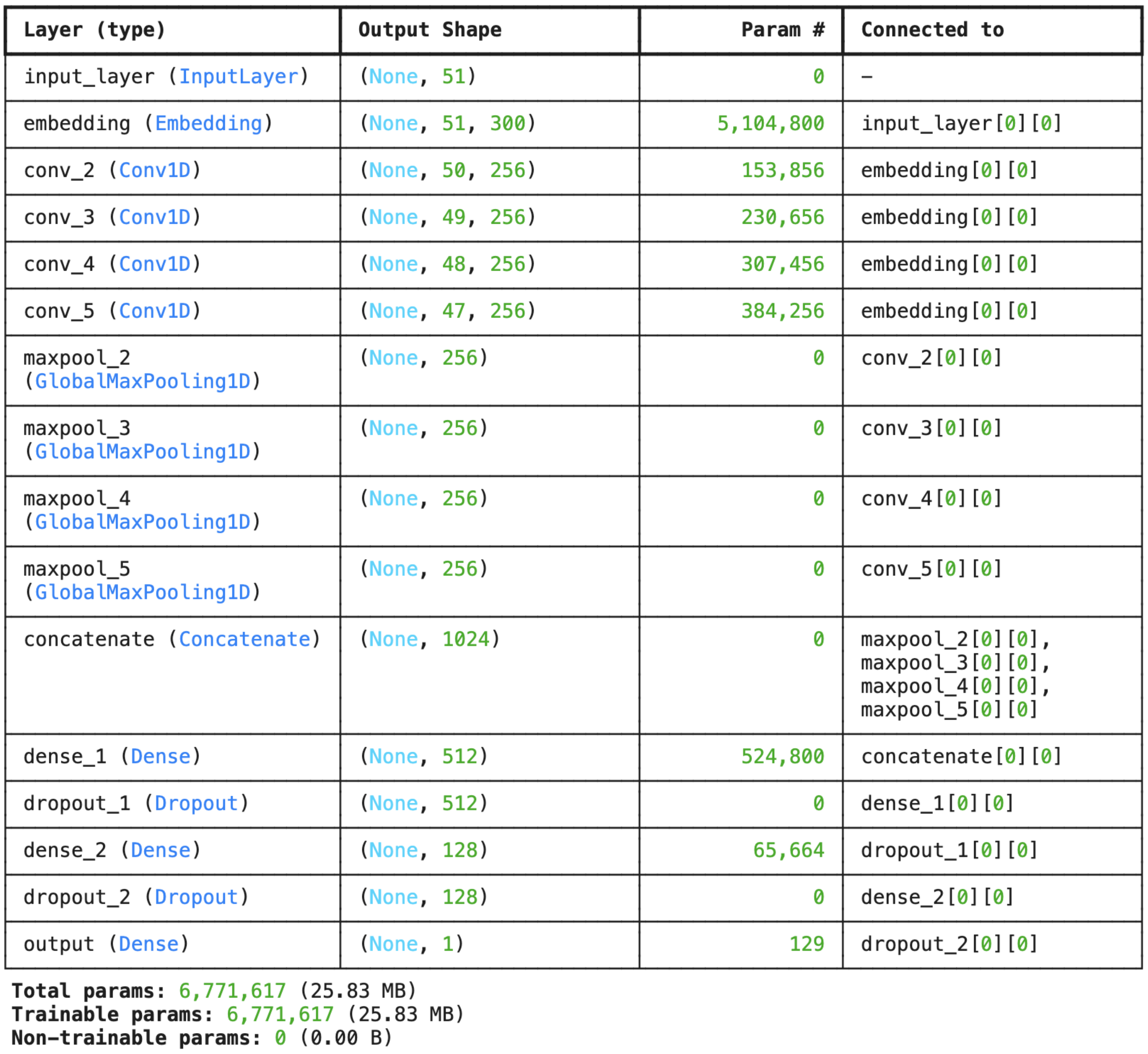


Figure 3.4.1 : Base CNN Model Architecture



Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 51)	0	-
embedding (Embedding)	(None, 51, 300)	5,104,800	input_layer[0][0]
conv_2 (Conv1D)	(None, 50, 256)	153,856	embedding[0][0]
conv_3 (Conv1D)	(None, 49, 256)	230,656	embedding[0][0]
conv_4 (Conv1D)	(None, 48, 256)	307,456	embedding[0][0]
conv_5 (Conv1D)	(None, 47, 256)	384,256	embedding[0][0]
maxpool_2 (GlobalMaxPooling1D)	(None, 256)	0	conv_2[0][0]
maxpool_3 (GlobalMaxPooling1D)	(None, 256)	0	conv_3[0][0]
maxpool_4 (GlobalMaxPooling1D)	(None, 256)	0	conv_4[0][0]
maxpool_5 (GlobalMaxPooling1D)	(None, 256)	0	conv_5[0][0]
concatenate (Concatenate)	(None, 1024)	0	maxpool_2[0][0], maxpool_3[0][0], maxpool_4[0][0], maxpool_5[0][0]
dense_1 (Dense)	(None, 512)	524,800	concatenate[0][0]
dropout_1 (Dropout)	(None, 512)	0	dense_1[0][0]
dense_2 (Dense)	(None, 128)	65,664	dropout_1[0][0]
dropout_2 (Dropout)	(None, 128)	0	dense_2[0][0]
output (Dense)	(None, 1)	129	dropout_2[0][0]

Total params: 6,771,617 (25.83 MB)
Trainable params: 6,771,617 (25.83 MB)
Non-trainable params: 0 (0.00 B)

Figure 3.4.2 : Base CNN Model Train/Validation Accuracy & Loss Plot

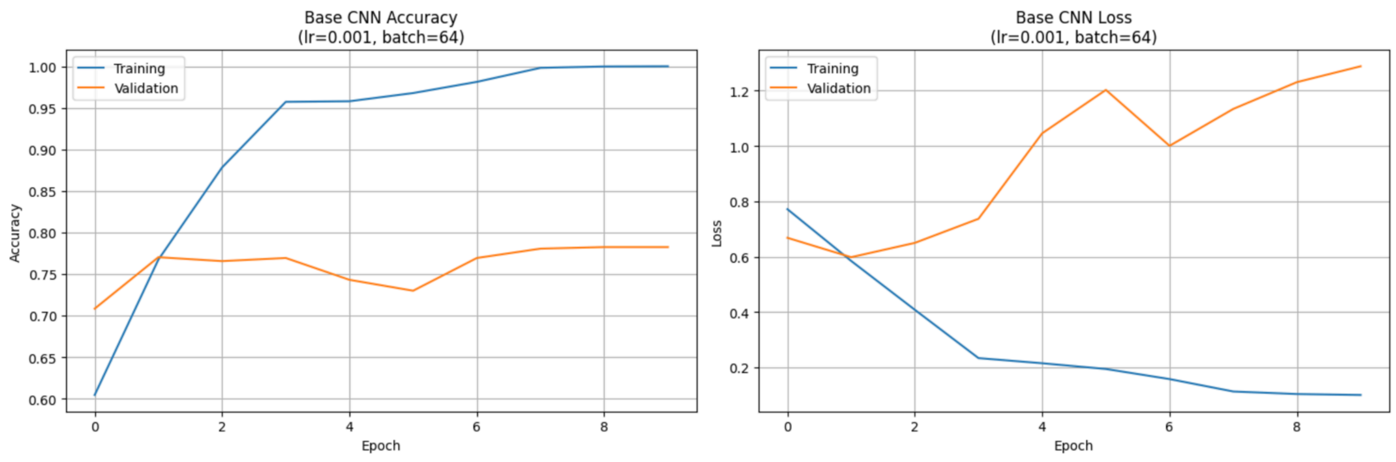
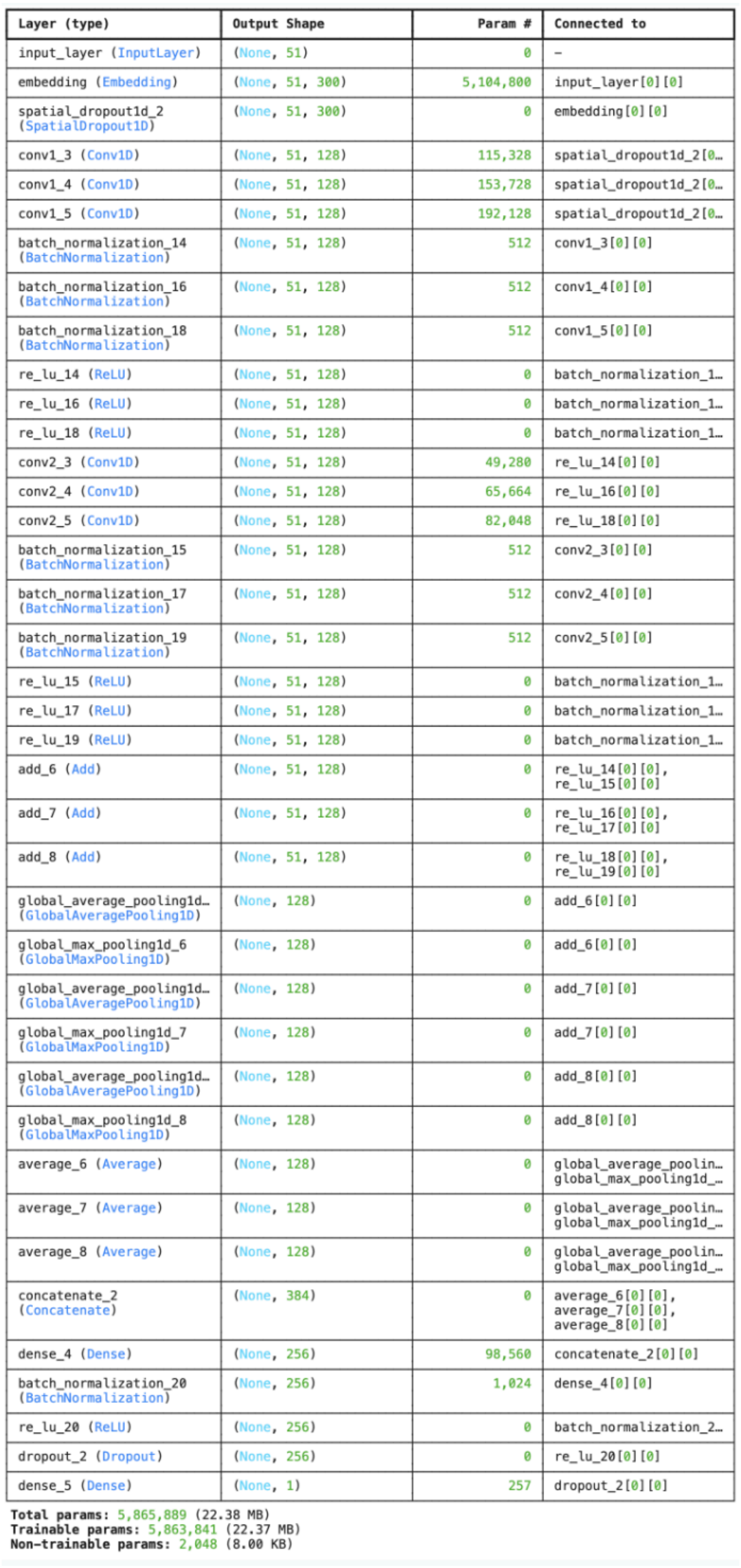


Figure 3.4.3 : Enhanced CNN Model Architecture



Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 51)	0	-
embedding (Embedding)	(None, 51, 300)	5,104,800	input_layer[0][0]
spatial_dropout1d_2 (SpatialDropout1D)	(None, 51, 300)	0	embedding[0][0]
conv1_3 (Conv1D)	(None, 51, 128)	115,328	spatial_dropout1d_2[0…
conv1_4 (Conv1D)	(None, 51, 128)	153,728	spatial_dropout1d_2[0…
conv1_5 (Conv1D)	(None, 51, 128)	192,128	spatial_dropout1d_2[0…
batch_normalization_14 (BatchNormalization)	(None, 51, 128)	512	conv1_3[0][0]
batch_normalization_16 (BatchNormalization)	(None, 51, 128)	512	conv1_4[0][0]
batch_normalization_18 (BatchNormalization)	(None, 51, 128)	512	conv1_5[0][0]
re_lu_14 (ReLU)	(None, 51, 128)	0	batch_normalization_1…
re_lu_16 (ReLU)	(None, 51, 128)	0	batch_normalization_1…
re_lu_18 (ReLU)	(None, 51, 128)	0	batch_normalization_1…
conv2_3 (Conv1D)	(None, 51, 128)	49,280	re_lu_14[0][0]
conv2_4 (Conv1D)	(None, 51, 128)	65,664	re_lu_16[0][0]
conv2_5 (Conv1D)	(None, 51, 128)	82,048	re_lu_18[0][0]
batch_normalization_15 (BatchNormalization)	(None, 51, 128)	512	conv2_3[0][0]
batch_normalization_17 (BatchNormalization)	(None, 51, 128)	512	conv2_4[0][0]
batch_normalization_19 (BatchNormalization)	(None, 51, 128)	512	conv2_5[0][0]
re_lu_15 (ReLU)	(None, 51, 128)	0	batch_normalization_1…
re_lu_17 (ReLU)	(None, 51, 128)	0	batch_normalization_1…
re_lu_19 (ReLU)	(None, 51, 128)	0	batch_normalization_1…
add_6 (Add)	(None, 51, 128)	0	re_lu_14[0][0], re_lu_15[0][0]
add_7 (Add)	(None, 51, 128)	0	re_lu_16[0][0], re_lu_17[0][0]
add_8 (Add)	(None, 51, 128)	0	re_lu_18[0][0], re_lu_19[0][0]
global_average_pooling1d… (GlobalAveragePooling1D)	(None, 128)	0	add_6[0][0]
global_max_pooling1d_6 (GlobalMaxPooling1D)	(None, 128)	0	add_6[0][0]
global_average_pooling1d… (GlobalAveragePooling1D)	(None, 128)	0	add_7[0][0]
global_max_pooling1d_7 (GlobalMaxPooling1D)	(None, 128)	0	add_7[0][0]
global_average_pooling1d… (GlobalAveragePooling1D)	(None, 128)	0	add_8[0][0]
global_max_pooling1d_8 (GlobalMaxPooling1D)	(None, 128)	0	add_8[0][0]
average_6 (Average)	(None, 128)	0	global_average_poolin… global_max_pooling1d_…
average_7 (Average)	(None, 128)	0	global_average_poolin… global_max_pooling1d_…
average_8 (Average)	(None, 128)	0	global_average_poolin… global_max_pooling1d_…
concatenate_2 (Concatenate)	(None, 384)	0	average_6[0][0], average_7[0][0], average_8[0][0]
dense_4 (Dense)	(None, 256)	98,560	concatenate_2[0][0]
batch_normalization_20 (BatchNormalization)	(None, 256)	1,024	dense_4[0][0]
re_lu_20 (ReLU)	(None, 256)	0	batch_normalization_2…
dropout_2 (Dropout)	(None, 256)	0	re_lu_20[0][0]
dense_5 (Dense)	(None, 1)	257	dropout_2[0][0]

```
Total params: 5,865,889 (22.38 MB)
Trainable params: 5,863,841 (22.37 MB)
Non-trainable params: 2,048 (8.00 KB)
```

Figure 3.4.4 : Enhanced CNN Model Train/Validation Accuracy & Loss Plot

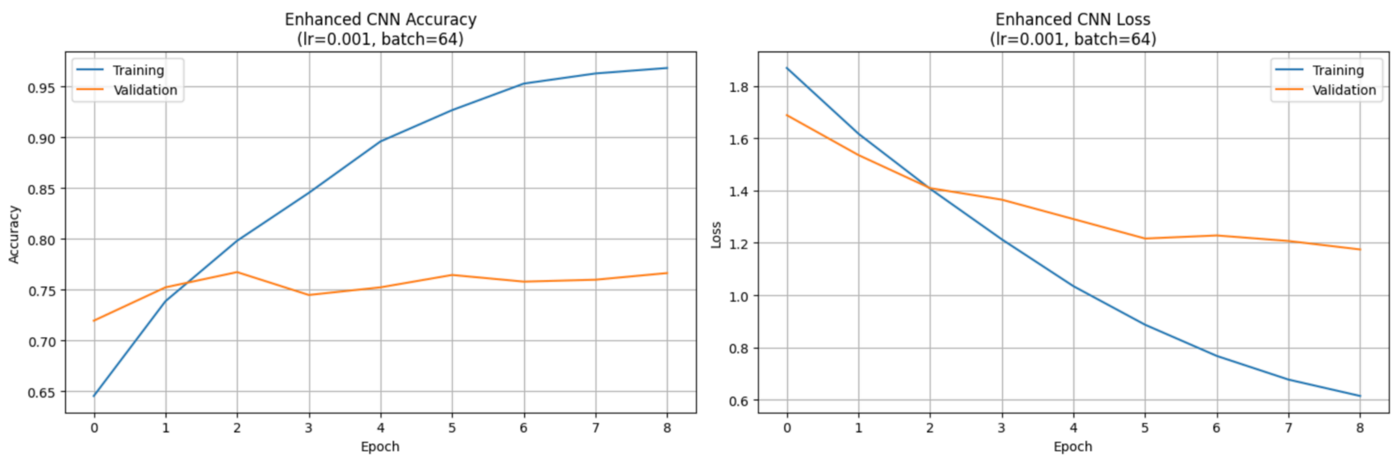
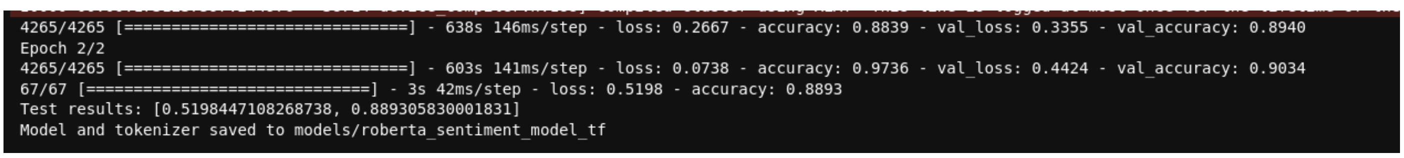


Figure 3.6.1 : RoBERTa results on train, valid and test set



```
4265/4265 [==============================] - 638s 146ms/step - loss: 0.2667 - accuracy: 0.8839 - val_loss: 0.3355 - val_accuracy: 0.8940
Epoch 2/2
4265/4265 [==============================] - 603s 141ms/step - loss: 0.0738 - accuracy: 0.9736 - val_loss: 0.4424 - val_accuracy: 0.9034
67/67 [==============================] - 3s 42ms/step - loss: 0.5198 - accuracy: 0.8893
Test results: [0.5198447108268738, 0.889305830001831]
Model and tokenizer saved to models/roberta_sentiment_model_tf
```

6. References

Mudadla, S. (2023, December 13). Layer normalization. Medium. https://medium.com/@sujathamudadla1213/layer-normalization-48ee115a14a4

Hong, Z. (2023, October 13). Attention mechanisms in deep learning: Enhancing model performance. Medium. https://medium.com/@zhonghong9998/attention-mechanisms-in-deep-learning-enhancing-model-performance-32a91006092a

Harsha, A. (2023, May 22). The ultimate showdown: Rnn Vs LSTM vs gru – which is the best? - shiksha online. Study in India. https://www.shiksha.com/online-courses/articles/rnn-vs-gru-vs-lstm/#:~:text=RNNs%2C%20LSTMs%2C%20and%20GRUs%20are,of%20memory%20cell%20and%20gates

Roberta. (2018). https://huggingface.co/docs/transformers/en/model_doc/roberta

Google-Bert/Bert-base-uncased · hugging face. google-bert/bert-base-uncased · Hugging Face. (n.d.). https://huggingface.co/google-bert/bert-base-uncased

Kacprzak, K. (2024, January 26). Roberta vs. Bert: Exploring the evolution of Transformer models. Data Engineering, MlOps and Databricks services. https://dsstream.com/roberta-vs-bert-exploring-the-evolution-of-transformer-models/

Facebookai/Roberta-base · hugging face. FacebookAI/roberta-base · Hugging Face. (n.d.). https://huggingface.co/FacebookAI/roberta-base