

**SC4002 Natural Language Processing | 2024-2025 Assignment**

**Group No. 57**

**Nov 10th, 2024**

| **Name** | **Matric No** | **Contributions** | **Sign** |
| --- | --- | --- | --- |
| Hendy | U2122559J | Backoff and Interpolation, Strategy in handling OOV, Report Writing |  |
| Lee Zheng Xuan | U2120607F | Byte-pair Encoding Strategy,  Tuning of Hyperparameters,  Modified GloVe Embeddings with Sentiment Scores |  |
| Selvaganapathy Arun Esvaran | U2222148F | Implemented Average / Max Pooling for Sentiment Classification, Experimented with OOV, Report Writing |  |
| Ng I-Shen Samuel | U2121307G | Code for 3.1-3.4 |  |
| Oh Jing Hong, Daryl | U2221946H | 1a, 1b, Random Embeddings, BERT |  |
| Tan Jin Shan | U2120742G | Simple & Self Attention Strategy for Sentence Representation, Report Writing |  |

# Table of Contents

[**Table of Contents 2**](#_96hfu3ysprmi)

[**Introduction 3**](#_3kzsgn6e2eqv)

[**Word Embeddings Preparation 5**](#_yaz9qr878gfy)

[**RNN Model Training and Evaluation 5**](#_c0sua6n79s4f)

[**Enhancement 5**](#_razkua4zj3wr)

# 

# 

# 

# Introduction

In recent years, the field of Natural Language Processing (NLP) has made strides in allowing machines to understand and interact with human language more accurately than ever before. Sentiment classification is highlighted as an aspect of NLP that focuses on analysing sentiments conveyed in text data, which has applications in understanding opinions expressed by individuals. This report outlines methods for developing a sentiment classification system that leverages pre-trained word embeddings such as GloVe (Global Vectors, for Word Representation) to improve the effectiveness of learning models.

In our project, we seek to showcase our understanding of NLP by using existing embeddings and assessing the model’s effectiveness. This report outlines our approaches taken, the results obtained, and enhancements implemented during the project. It concludes with an evaluation of how the model performed on the test data set.

# **Question 1. Word Embedding**

## 1a. What is the size of the vocabulary formed from your training data?



*Figure 1a.1: Size of Vocabulary Screenshot*

The size of vocabulary from training data is 18029.

## 1b. We use OOV (out-of-vocabulary) to refer to those words appeared in the training data but not in the Word2vec (or Glove) dictionary. How many OOV words exist in your training data?



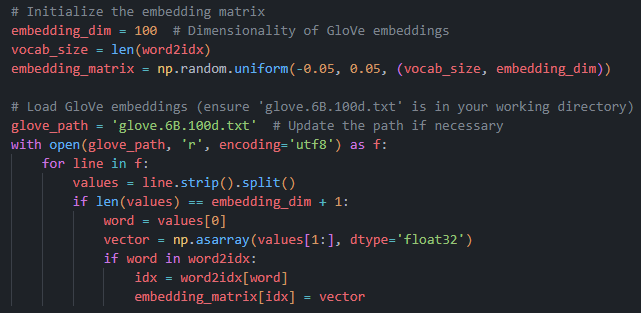
*Figure 1b.1: Number of OOV Words Screenshot*

The number of OOV in training data is 1865.

## 1c. Without using any transformer-based language models (e.g., BERT, GPT, T5), what do you think is the best strategy to mitigate such limitation? Implement your solution in your source code. Show the corresponding code snippet.

Below, we explored three strategies to address out-of-vocabulary (OOV) words.

**Strategy 1: Random Embeddings**

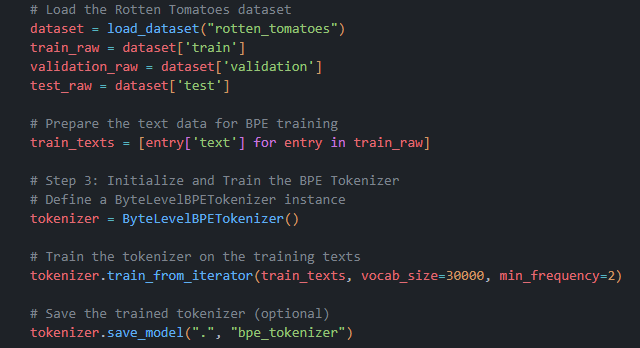
****

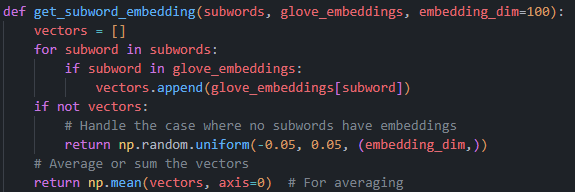
*Figure 1c.1: Random Embedding for OOV words Code Snippet*

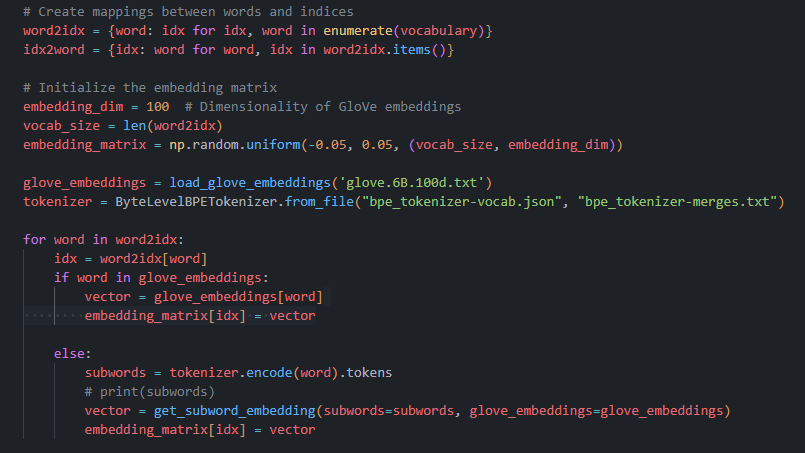
This initial approach to address OOV words is to assign random embeddings to all OOV words. In this strategy, each OOV word is initialised with a randomly generated vector within a fixed range. This method provides the model with representations for all vocabulary terms including those not present in the pre-trained GloVe embeddings without the need for additional steps.

While simple, this approach has notable limitations. Since the embeddings for OOV words are randomly assigned, they do not carry any semantic or structural meaning related to the words themselves. As a result, these random embeddings may not capture meaningful patterns within the data. This can affect the ability of the model to generalise and handle new or rare terms. However, using random embeddings provides a straightforward solution when other methods are unavailable or when computational simplicity is prioritised. This approach acts as a baseline to compare with other strategies that we explored.

**Strategy 2: Byte Pair Encoding (BPE)**

****

****

****

*Figure 1c.2: Byte Pair Encoding Code Snippets*

To handle out-of-vocabulary (OOV) words, another approach is to use Byte Pair Encoding (BPE) which breaks down words into smaller subword units. This method enables better coverage and provides meaningful vector representations for words that do not have full embeddings in GloVe. By representing words as combinations of subwords, BPE allows the model to handle OOV words more effectively. This improves the robustness of embeddings without relying on transformer-based models.

In this approach, text data was extracted from the Rotten Tomatoes training dataset to train the BPE tokenizer. The tokenizer was configured with a vocabulary size of 30,000 and a minimum frequency threshold of 2, ensuring that it captures frequently occurring subwords while keeping the overall vocabulary manageable. This balance is crucial for generating an efficient subword vocabulary that accurately represents common word segments.

Once trained, the BPE tokenizer was applied to OOV words to decompose them into subword units. For each subword, its embedding was retrieved from GloVe if available. If a subword was not in the GloVe vocabulary, a random embedding was initialised to maintain consistency. The embeddings of the subwords were then averaged to form a cohesive and meaningful vector for each OOV word. This provides a practical representation that reflects the word's structure and approximate meaning. This strategy allows the model to leverage existing pretrained embeddings more effectively through handling unknown words by building upon the embeddings of their recognisable subcomponents.

**Strategy 3: Backoff and Interpolation**

The backoff and interpolation technique allows the model to fall back to lower-order n-grams each time when higher-order n-grams are not available. For instance, if a trigram (three consecutive words) is not found, the model can back off to using bigrams (two consecutive words), and if bigrams are also unavailable, it can finally resort to unigrams (individual words). This hierarchical structure enables the model to utilise available data effectively, thereby improving the handling of OOV words.

In our implementation, the function get\_embedding(word) is designed to handle OOV words by first checking if the word exists in the vocabulary (word2idx). If the word is not found, the function constructs bigram and trigram contexts from the input word, The function will then check for the presence of the n-grams. If a n-gram is found, the model computes an interpolated embedding by averaging the embeddings of the words in the respective n-gram context.

The output of this step is then weighted by a predefined parameter (lambda1, lambda2, or lambda3) that determines the contribution of the different n-gram levels to the final embedding.

The sum of the weights should equal 1 to ensure proper normalisation. In this project, we are using the weights shown below.



*Figure 1c.3: Weights for Normalisation Code Snippet*



*Figure 1c.4: Accuracy using Backoff and Interpolation*

When an OOV word is encountered, the get\_embedding function first tries to retrieve an embedding based on the trigram context. If successful, the weighted embedding is returned. If not, it falls back to the bigram context and retrieves the embedding similarly. If neither context is available, the function defaults to a zero vector, signifying that no useful embedding could be derived for the OOV word.

This results in more robust sentiment classification, as the model can leverage contextual information to fill in gaps where direct embeddings are unavailable. Through the structured implementation of these techniques in our code, we ensure that the model maintains a high level of accuracy even in the presence of novel vocabulary.

# Question 2. RNN

## **2a. Report the final configuration of your best model, namely the number of training epochs, learning rate, optimizer, batch size.**

| Learning Rate | 0.0001 | 0.0005 | 0.001 | 0.005 | 0.01 |
| --- | --- | --- | --- | --- | --- |
| Best Validation Loss | 0.4909 | 0.4873 | 0.4724 | 0.4777 | 0.4923 |

| Weight Decay | 0 | 1e-6 | 1e-5 | 1e-4 | 1e-3 |
| --- | --- | --- | --- | --- | --- |
| Best Validation Loss | 0.4804 | 0.4769 | 0.4737 | 0.4699 | 0.478 |

| Batch Size | 16 | 32 | 64 | 128 | 256 |
| --- | --- | --- | --- | --- | --- |
| Best Validation Loss | 0.4732 | 0.4672 | 0.4699 | 0.4666 | 0.4919 |

| Hidden Size | 64 | 128 | 256 | 512 |
| --- | --- | --- | --- | --- |
| Best Validation Loss | 0.4848 | 0.4697 | 0.4666 | 0.4828 |

| Number of Layers | 1 | 2 | 3 |
| --- | --- | --- | --- |
| Best Validation Loss | 0.4686 | 0.456 | 0.4755 |

| Dropout Rate | 0.2 | 0.3 | 0.5 | 0.7 |
| --- | --- | --- | --- | --- |
| Best Validation Loss | 0.4596 | 0.4583 | 0.4622 | 0.4629 |

**Final Configuration**

| Learning Rate | Weight Decay | Batch Size | Hidden size | Number of Layers | Dropout Rate | Number of Epochs | Patience | Optimizer |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0.001 | 1e-4 | 128 | 256 | 1 | 0.3 | 9 | 5 | Adam |

## **2b. Report the accuracy score on the test set, as well as the accuracy score on the validation set for each epoch during training.**

**Validation Accuracy for Each Epoch**

| Epoch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Validation  Accuracy (%) | 69.04 | 73.83 | 75.70 | 77.20 | 76.83 | 78.88 | 77.86 | 78.71 | 78.52 |

| Epoch | 10 | 11 | 12 | 13 |
| --- | --- | --- | --- | --- |
| Validation Accuracy (%) | 77.49 | 78.71 | 77.30 | 78.42 |

**Accuracy Score on Test Set**

****

*Figure 2b.1: Accuracy on Test Dataset Screenshot*

## **2c. RNNs produce a hidden vector for each word, instead of the entire sentence. Which methods have you tried in deriving the final sentence representation to perform sentiment classification? Describe all the strategies you have implemented, together with their accuracy scores on the test set.**

**Strategy 1: Max-Pooling**

Max pooling takes the element-wise maximum across the hidden vectors for each word until the current token during analysis. This approach generally gravitates towards classifying the sentiment of the overall sentence depending on the strongest classifiers available as context.



*Figure 2c.1: Accuracy of max pooling strategy*

**Strategy 2: Mean-Pooling**

An alternative to max-pooling, mean-pooling averages the results of analysed LSTM outputs to produce the corresponding outcome. Instead of predominantly relying on the most indicative classifiers within the sentence, this approach provides a holistic overview of all words within a given sentence, albeit disproportionately assigning weights to words with a neutral sentiment.



*Figure 2c.1: Accuracy of mean pooling strategy*

**Strategy 3: Simple Attention**

This strategy computes a weighted average of the hidden vectors produced by the RNN. The weights are determined by a trainable attention mechanism and assigned to each hidden vector based on its importance. This allows the model to focus on the parts of the sentence that strongly indicate the sentiment when making predictions.



*Figure 2c.1: Accuracy of simple attention strategy*

**Strategy 4: Self-Attention**

Self-attention is different from simple attention as it allows the model to capture the relationship between different words within the sequence itself. It enables the model to understand the connections between words, regardless of their position. This strategy helps capture long-range dependencies and context, within the entire sentence.



*Figure 2c.1: Accuracy of self-attention strategy*

# **Question 3. Enhancement**

## **3a. Report the accuracy score on the test set when the word embeddings are updated (Part 3.1).**



*Figure 3a.1: Accuracy after Word Embeddings Updated*

The accuracy score on the test set when word embeddings are updated is 79.64%.

## **3b. Report the accuracy score on the test set when applying your method to deal with OOV words in Part 3.2.**

**Accuracy Score with OOV Strategy 1: Random Embeddings**



*Figure 3b.1: Accuracy after applying Random Embeddings Strategy*

The accuracy score on the test set after applying Random Embedding OOV Strategy is 80.39%.

**Accuracy Score with OOV Strategy 2: Byte Pair Encoding (BPE)**



*Figure 3b.2: Accuracy after applying BPE Strategy*

The accuracy score on the test set after applying Random Embedding OOV Strategy is 79.83%.

As the Random Embeddings OOV Strategy performs better than BPE strategy, Random Embeddings strategy will be used for subsequent experiments.

## **3c. Report the accuracy scores of biLSTM and biGRU on the test set (Part 3.3).**

**Accuracy Score with of BiLSTM**

****

*Figure 3c.1: Accuracy on Test set with BiLSTM Model*

The accuracy score on the Test set for the BiLSTM model is 79.92%.

**Accuracy Score with of BiGRU**

****

*Figure 3c.2: Accuracy on Test set with BiGRU Model*

The accuracy score on the Test set for the BiGRU model is 79.27%.

## **3d. Report the accuracy scores of CNN on the test set (Part 3.4).**



*Figure 3d.1: Accuracy on Test set with CNN Model*

The accuracy score on the Test set for the CNN model is 78.89%.

## **3e. Describe your final improvement strategy in Part 3.5. Report the accuracy on the test set using your improved model.**

**Strategy 1: BERT**

In 'BERT.ipynb', we utilised BERT (Bidirectional Encoder Representations from Transformers) for sentiment classification. The model architecture employs the 'bert-base-uncased' pre-trained model, which was fine-tuned for binary sentiment classification. The implementation leverages the Hugging Face transformers library, specifically using BertForSequenceClassification with a single output label, configured for binary classification of movie reviews. The model's architecture maintains BERT's core attention mechanism while adapting the output layer for sentiment prediction through a binary classification head.

The preprocessing pipeline incorporates BERT-specific tokenization using BertTokenizer, which handles the conversion of raw text into BERT's expected input format. Each text input is encoded with special tokens ([CLS] and [SEP]), padded to a maximum sequence length of 128 tokens, and accompanied by attention masks to properly handle variable-length sequences. The custom BERTSentimentDataset class manages this preprocessing workflow, ensuring that each batch contains properly formatted input\_ids, attention\_masks, and labels. This approach preserves BERT's ability to capture contextual relationships through its bidirectional attention mechanism while maintaining computational efficiency.

The training process implements several optimization techniques crucial for the effective fine-tuning of BERT models. The AdamW optimizer is utilised with a learning rate of 2e-5, combined with a linear learning rate scheduler that includes warm-up steps. Gradient clipping is applied with a maximum norm of 1.0 to prevent gradient explosion, while the BCEWithLogitsLoss function serves as the training objective. The implementation also features early stopping based on validation F1 score with a patience of 3 epochs, which helps prevent overfitting. This configuration achieved strong performance metrics, with a test accuracy of 83.49% and an F1 score of 84.14% (F1 score is particularly useful in imbalanced datasets as it considers both precision and recall), demonstrating BERT's effectiveness in capturing semantic nuances in sentiment classification tasks.

*Figure 3e.1: Accuracy on Test Set using BERT*

**Strategy 2: Embedding with Sentiment Score**



*Figure 3e.2: Accuracy On Test Set with Modified GloVe Embedding with Sentiment Scores*

We implemented an improvement strategy by incorporating sentiment lexicons into the sentiment classification model. Specifically, we utilised AFINN sentiment lexicon which provides sentiment scores for a wide range of words. The scores indicate the degree of positivity or negativity associated with each word. This lexicon-based approach is aimed at enhancing the ability of the model to capture sentiment-related nuances in the text, especially for words that may have distinct emotional weight.

To integrate these sentiment scores, we modified the word embeddings we originally used. We extended the GloVe embeddings with a dimensionality of 100 by an additional dimension to accommodate the sentiment score from AFINN. For each word in the vocabulary, we added the corresponding sentiment score to this final dimension. If a word does not have a sentiment score in the lexicon, we set this additional dimension to zero. This augmented embedding matrix allowed the model to leverage both semantic information from GloVe embeddings and sentiment information from AFINN. This enriches the input features available for sentiment classification.

We evaluated this new strategy on the BiLSTM model combining random embeddings OOV strategy and without freezing the embeddings. The result can be seen from Figure 3e.2 with an accuracy of 81.05% on the test dataset. This proves that integrating the sentiment scores as an additional dimension in the embeddings helped the model better distinguish between sentiments and especially in cases where word context alone might not have provided sufficient information.

## **3f. Compare the results across different solutions above and describe your observations with possible discussions. (**Tentative, will rewrite after rest of parts is done**)**

**OOV Handling Strategies**

* **Random Embeddings** demonstrated slightly higher accuracy (80.39%) compared to **Byte Pair Encoding (BPE)** (79.83%). This suggests that while BPE is effective in generating subword-level representations, random embeddings provide a simpler yet sufficiently robust solution for handling out-of-vocabulary words. However, random embeddings lack semantic grounding, which might limit generalizability compared to BPE.
* **Backoff and Interpolation** was implemented to enrich embeddings by leveraging n-grams for OOVs, which can enhance the model’s adaptability to diverse vocabulary. While this strategy theoretically supports contextual handling of OOVs better than the random embeddings approach, it may not have been as efficient in capturing relevant semantic meaning for sentiment classification. Random embeddings still perform slightly better in our sentiment classification.

**Enhanced Models**

* **BiLSTM and BiGRU Models**: The BiLSTM achieved 79.92% accuracy. It outperformed BiGRU which reached 79.27%. This indicates that the BiLSTM’s capacity to handle sequential dependencies likely contributed to a slight performance boost in sentiment classification. This aligns with the findings that recurrent models, especially those with bidirectional layers, effectively capture complex language patterns needed for nuanced sentiment interpretation.
* **CNN Model**: With an accuracy of 78.89%, CNN underperformed compared to BiLSTM and BiGRU. While CNNs are effective for capturing local dependencies such as image processing, they may miss out on the sequential context essential for sentiment classification tasks.

**Final Improvement Strategies**

* **BERT:** BERT's pre-trained architecture with fine-tuning for binary sentiment classification, utilising specialised preprocessing, optimisation techniques, and early stopping mechanisms, achieved an 84.14% F1 score. BERT outperformed all previous methods. This result underscores BERT's ability to capture bidirectional context and nuanced semantics. Hence, this makes it the most robust solution in this comparison. This result suggests that transformer-based models like BERT have distinct advantages over conventional RNN and CNN-based models for sentiment tasks.
* **Modified GloVe with Sentiment Scores:** The final model incorporating sentiment scores from the AFINN lexicon alongside GloVe embeddings achieved an accuracy of 81.05%. This approach leveraged additional sentiment context, allowing the model to discern sentiment more effectively than using GloVe embeddings alone. This strategy is a cost-effective yet impactful enhancement, especially when transformer models like BERT have some computational overhead.