# SC4002 Natural Language Processing

## Group Project

| Name | Email | Matric Number |
|---|---|---|
| Pu Fanyi | FPU001@e.ntu.edu.sg | U2220175K |
| Shan Yi | SH0005YI@e.ntu.edu.sg | U2222846C |
| Zhang Kaichen | ZHAN0564@e.ntu.edu.sg | U2123722J |
| Zhang Jiahua | jzhang132@e.ntu.edu.sg | U2221393K |
| Tang Yutong | TANG0513@e.ntu.edu.sg | U2220495H |
| Zeng Ruixiao | ZENG0146@e.ntu.edu.sg | U2220375D |

Nanyang Technological University, Singapore

2024/2025 Semester 1

## 1 Introduction

This report focuses on building and evaluating a sentiment classification system using pretrained word embeddings. Our work is organized into three main parts:

In section 2, we prepare and analyze the word embeddings that form the foundation of our classification system. We use GloVe as our embedding method and address the out-of-vocabulary (OOV) issue through a novel greedy matching approach.

In section 3, we implement and evaluate a basic RNN model for sentiment classification. We explore different strategies for sentence representation and conduct extensive hyperparameter tuning to optimize model performance.

In the last section, we utilized different methods to improve classificat model and proposed our final method.

**Note** Results may fluctuate due to variations in settings such as GPU type and other parameters. However, we ensure consistency within each set of ablations and experiments by maintaining identical settings throughout. Specific settings are provided for each experiment we conduct.

## 2 Preparing Word Embeddings

### 2.1 Tokenizer and Embeddings

GloVe [6] is used as our embedding method. We selected `glove.840B.300d`[*] as our pretrained embedding model. To easier manage the GloVe model, we pack the model to Hugging Face [7].

We tried to take an overview of this dataset. NLTK tokenizer [1] is used to split the word. We report the **out-of-vocabulary (OOV)** issue in table 1 and found about 0.96% tokens are unknown in the vocabulary.

| Words | Count | Percentage |
|---|---|---|
| Known | 182211 | 99.04% |
| Unknown | 1757 | 0.96% |
| Total | 183968 | 100% |

Table 1: Word count for the training dataset [4]

### 2.2 Mitigating Out of Vocabulary Limitations

To address the OOV issue, we employ two approaches for handling various edge cases. **1)** We predefine the `<|UNK|>` token and its ID in the tokenizer class to map any unknown character to this token. The word embedding for `<|UNK|>` is resized and initialized to zeros. **2)** To maximize the pretrained knowledge from Glove, we use a **Greedy Matching** approach, which tokenizes each sentence by breaking down unknown words into recognizable segments. This allows us to reconstruct unknown words using known prefixes, suffixes, and semantic tokens, effectively leveraging the extensive vocabulary and latent space of the word embeddings.

Then we use the greedy matching algorithm to find the longest matching known word. If an unknown word is encountered, replace it with the `<|UNK|>` token ID.

There are 2 methods implementing this greedy idea, *Hash-based Greedy Matching* and *Trie-based Greedy Matching*, we analysed both and selected the best one.

**Hash-based Greedy Matching** All known words are organised in a hash table $\mathcal{H}$. We enumerate all possible substrings and choose the longest one in $\mathcal{H}$. The detailed algorithm is shown in Algorithm

[*]Trained on Common Crawl, 840B tokens, 2.2M vocab, cased and 300d vectors, accessed from https://nlp.stanford.edu/data/glove.840B.300d.zip

1. We can proof that the time complexity of this algorithm is at most $\mathcal{O}\left(\|s\|^2\right)$. If we define the average word length is $\mathcal{W}$, the average time complexity is

$$\begin{aligned} T(s,\mathcal{W}) &= \mathbb{E}\left[\sum_{i=1}^{\|s\|} \mathbb{1}_{s_i \text{ is beginning of a word}} \cdot (\|s\| - i)\right] \\ &= \sum_{i=1}^{\|s\|} (\|s\| - i) \cdot \mathbb{P}(s_i \text{ is beginning of a word}) \\ &= \mathcal{O}\left(\frac{\|s\|^2}{\mathcal{W}}\right) \end{aligned} \tag{1}$$

**Trie-based Greedy Matching** Algorithm 2 illustrates implementing the greedy matching algorithm with Trie [2]. In this algorithm, every character in $s$ will only be visited at once, so the time complexity is $\mathcal{O}(\|s\|)$.

Although Trie-based Greedy Matching shows a better time complexity, we found that in [4], $\|s\|$ is always small as there is no long contexts, while building a Trie costs extra spaces and time. So we finally decided to adopt Hash-based Greedy Matching for our algorithm. The code snip is listed in Appendix A.

**Algorithm 1** TRIEBASEDGREEDYMATCHING

```
$\mathcal{H} \leftarrow$ BUILDHASH(known words)
$s \leftarrow$ input sentense
$L \leftarrow 0$
$\mathcal{M} \leftarrow$ empty list
while $L <$ LENGTH($s$) do
    $R \leftarrow \emptyset$
    for $r \in [L,$ LENGTH($s$)) do
        $t \leftarrow$ STRIP($s_{L:r}$)
        if $t \in \mathcal{H}$ then
            $R \leftarrow r$
        end if
    end for
    $L \leftarrow R + 1$
    if $R = \emptyset$ then
        APPEND($\mathcal{M}$, <|UNK|>)
    else
        APPEND($\mathcal{M}$, $s_{L:R}$)
    end if
end while
return $\mathcal{M}$
```

## 3 RNN

### 3.1 Model Training

We utilised *Weights & Biases Agent*[†] to track and sweep the best model hyperparameter. Bayesian Optimization [3] is used as the strategy to sweep the best hyperparameter. By default, we set the *require_grad* for word embedding to False to freeze the parameters. We use the simple RNN design with PyTorch [5] that takes each token as hidden states and iterative to the final token for each sequences.

[†]Documents available in `https://docs.wandb.ai/guides/sweeps`

**Algorithm 2** TRIEBASEDGREEDYMATCHING

```
$\mathcal{T} \leftarrow$ BUILDTRIE(known words)
$s \leftarrow$ input sentense
$L \leftarrow 0$
$\mathcal{M} \leftarrow$ empty list
while $L <$ LENGTH$(s)$ do
    while $L <$ LENGTH$(s) \wedge \neg$VISIBLE$(s_L)$ do
        $L \leftarrow L + 1$
    end while
    $R \leftarrow \emptyset$
    $n \leftarrow$ ROOT$(\mathcal{T})$
    for $r \in [L,$ LENGTH$(s)) \wedge$ SON$(n, s_r) \neq \emptyset$ do
        $n \leftarrow$ SON$(n, s_r)$
        if $s_{L:r}$ is a known word then
            $R \leftarrow r$
        end if
    end for
    $L \leftarrow R$
    APPEND$(\mathcal{M}, s_{L:R-1})$
    if $R = \emptyset$ then
        APPEND$(\mathcal{M},$ <|UNK|>$)$
    else
        APPEND$(\mathcal{M}, s_{L:R})$
    end if
end while
return $\mathcal{M}$
```

## 3.2 Best model

Due to computational and time limit, we can only run limited experiments. We employ two search strategies, **1)** Grid Search, **2)** Manually tuned. The config and the search space is listed in Table 2. However, we find out that one of the config when manually tuned the parameters reaches the best result and we adopt that setting as our best model config

| **Config** | **Value** | **Grid Search Space** |
|---|---|---|
| Layer | 5 | $\{3, 4, 5, 6\}$ |
| Input Dim | 300 | - |
| Hidden Dim | 512 | - |
| Optimizer | AdamW | - |
| LR Scheduler | `CosineAnnealingLR` | - |
| Warmup Ratio | 0.03 | - |
| Learning Rate | $1.63 \times 10^{-5}$ | $1 \times 10^{-3} \sim 1 \times 10^{-5}$ |
| Weight Decay | 0.4 | - |
| Batch Size | 4 (GPUs) $\times$ 8 | $4 \times \{8, 16, 32\}$ |

Table 2: Best Model Config

We also present our grid search results in fig. 1 for multiple trials and use the best model config that achieve the highest evaluation accuracy.

## 3.3 Model Performance

Our model's performance, as shown in Table 3, is evaluated across the training, validation, and test sets using both loss and accuracy metrics. The loss values are 0.5619, 0.5228, and 0.5416 for the training, validation, and test sets, respectively, indicating consistent performance with slight variations across the datasets. The model achieves an accuracy of 77.86% on the validation set and 76.27% on the test set, suggesting that it generalizes well to unseen data with minimal overfitting.

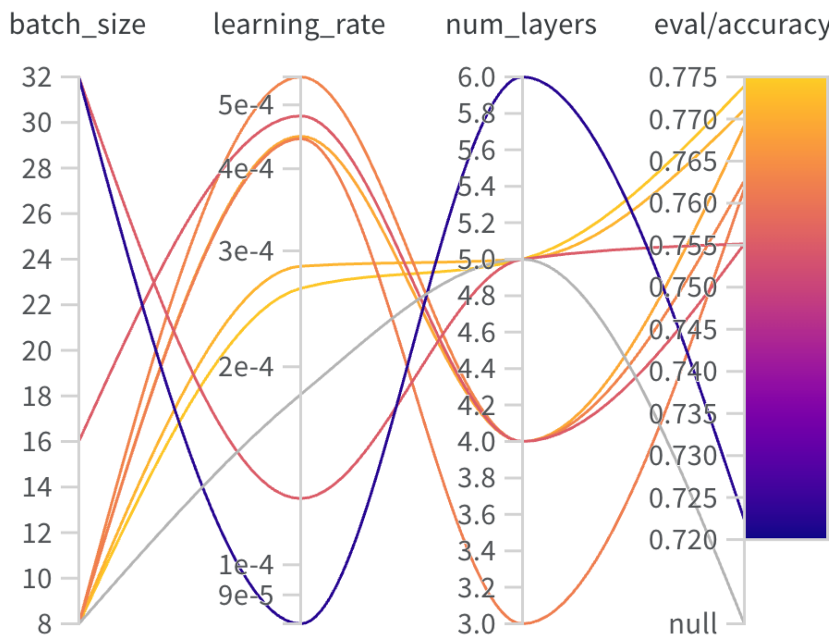

Figure 1: Sweep result

| **Metric** | **Train set** | **Validation set** | **Test set** |
|---|---|---|---|
| **Loss** | 0.5619 | 0.5228 | 0.5416 |
| **Accuracy** | - | 77.86% | 76.27% |

Table 3: Performance table

**Setting** We use the same settings to run all the experiments with batch size equals to 8 and learning rate equals to $1 \times 10^{-5}$. For every run, we train 10 epochs and report the final validation and test result.

### 3.4 Sentence Representation Methods

Table 4: The result of using different sentence representation strategies

| Aggregation | Test/Acc | Test/Loss | Val/Acc | Val/Loss |
|---|---|---|---|---|
| Average | 0.72 | 0.63 | 0.73 | 0.63 |
| Sum | **0.76** | **0.56** | **0.76** | **0.55** |
| Final Token | 0.51 | 0.69 | 0.50 | 0.69 |

We compare three sentence representation methods—average, sum, and final token—using accuracy and loss metrics on both the validation and test sets. As shown in Table 4, the sum aggregation method achieves the best performance, with a test accuracy of 0.76 and a test loss of 0.56. This method also yields a validation accuracy of 0.76 and a validation loss of 0.55, outperforming both the average and final token methods. The average method yields moderate results, with a test accuracy of 0.72 and validation accuracy of 0.73, while the final token method shows the lowest performance across all metrics. These results indicate that summing embeddings captures sentence information most effectively in our RNN model, supporting our decision to use this method as the default aggregation strategy.

## 4 Enhancement

### 4.1 Unfreezing the word embedding

As the models we used above are based on the froze pre-trained word embedding which is trained with general corpus, this word embedding may not perform the best on our current corpus. In this section, we will unfreeze the word embedding so that it can better match our current corpus.

Table 5: Result of the Unfreezing Word Embedding

| | Test/Acc | Test/Loss | Val/Acc | Val/Loss |
|---|---|---|---|---|
| Unfreeze | 77.01 | 53.29 | 78.42 | 51.82 |

### 4.2 Mitigating the OOV word influence

Table 6: Result of the OOV strategy

| | Test/Acc | Test/Loss | Val/Acc | Val/Loss |
|---|---|---|---|---|
| Simple Ignore | 0.71 | 0.61 | **0.74** | **0.60** |
| Greedy Matching | **0.73** | **0.60** | 0.73 | 0.61 |

Our approach to mitigating the out-of-vocabulary (OOV) issue is evaluated using two strategies: simple ignore (replacing OOV tokens with a padding token) and a hash-based greedy matching method, implemented in our code as described in algorithm 1. Both strategies were evaluated on accuracy and loss for the validation and test sets under the same experimental conditions, with the word embeddings unfrozen and trained for 3 epochs.

As shown in Table 6, the greedy matching strategy slightly outperforms the simple ignore method on test accuracy (0.73 vs. 0.71) and test loss (0.60 vs. 0.61). However, the simple ignore method achieves marginally better results on the validation set, with a validation accuracy of 0.74 and validation loss of 0.60. These results suggest that while the greedy matching method is generally more effective on the test set. Though simple ignore performs better on the validation set, it is shows trend to overfit on the test set and may perform even worse on more complex dataset.

### 4.3 biLSTM and biGRU

Table 7: Result of the biLSTM and biGRU

| | Test/Acc | Test/Loss | Val/Acc | Val/Loss |
|---|---|---|---|---|
| biLSTM | 0.77 | 0.53 | 0.79 | 0.52 |
| biGRU | 0.78 | 0.53 | 0.78 | 0.52 |

Table 7 presents the performance results for the biLSTM and biGRU models. The biLSTM achieves a test accuracy of 0.77 with a test loss of 0.53, while its validation accuracy and loss are 0.79 and 0.52, respectively. The biGRU model shows slightly higher test accuracy at 0.78 with the same test loss of 0.53, and its validation accuracy and loss are 0.78 and 0.52, respectively.

### 4.4 CNN

Our CNN model achieves a test accuracy of 0.77 and a test loss of 0.54, with a validation accuracy of 0.77 and validation loss of 0.52, as shown in Table 8. The hyperparameters were optimized using a WandB agent to ensure the model's best performance under our experimental conditions.

### 4.5 Model enhancement

We incorporate a self-attention mechanism prior to the aggregation step to leverage global information from each token, aiming to enhance the model's ability to capture contextual dependencies. The

Table 8: Result of the CNN

| | Test/Acc | Test/Loss | Val/Acc | Val/Loss |
|---|---|---|---|---|
| CNN | 0.77 | 0.54 | 0.77 | 0.52 |

Table 9: The result of our enhancement method

| | Test/Acc | Test/Loss | Val/Acc | Val/Loss |
|---|---|---|---|---|
| Baseline | 0.72 | 0.59 | 0.75 | 0.58 |
| w. Attn | **0.74** | **0.57** | **0.76** | **0.55** |

impact of this approach is shown in Table 9, where we compare the baseline model with and without self-attention.

The model with self-attention (denoted as 'w. Attn') outperforms the baseline across both validation and test sets. Specifically, it achieves a test accuracy of 0.74 and a test loss of 0.57, compared to 0.72 and 0.59 for the baseline. On the validation set, the self-attention model yields an accuracy of 0.76 and a loss of 0.55, surpassing the baseline's accuracy of 0.75 and loss of 0.58. These results suggest that integrating self-attention enhances the model's performance by effectively capturing token-level interactions, which improves generalization on unseen data.

## 4.6 Discussion

In this subsection, we summarize our findings from previous experiments and discuss factors that may have influenced the model's performance.

**Vocabulary Size** We believe vocabulary size significantly impacts model performance. In our ablation studies, using a smaller GloVe embedding yielded worse results compared to a larger version under the same settings. With only around 1% of tokens in the test set facing out-of-vocabulary (OOV) issues, the model can still extract substantial information from the pretrained GloVe vocabulary. Consequently, our strategy for addressing OOV issues provided only minimal improvements.

**Dataset Size** The dataset contains approximately 8,000 rows, which limits the distribution from which the model can learn. Even if the model is scaled, without leveraging domain-specific knowledge from large pretrained models such as BERT, training from scratch remains challenging on such a small dataset.

**Parameter Tuning** Parameter tuning proved to be particularly impactful with this small dataset. We suspect that due to the limited data distribution, the loss landscape is sharp, making convergence to a stable minimum challenging. An example of the gradient norm using the RNN is shown in fig. 2. As illustrated, the gradient norm fluctuates significantly, indicating large steps taken by the optimizer and highlighting the difficulty of achieving stability

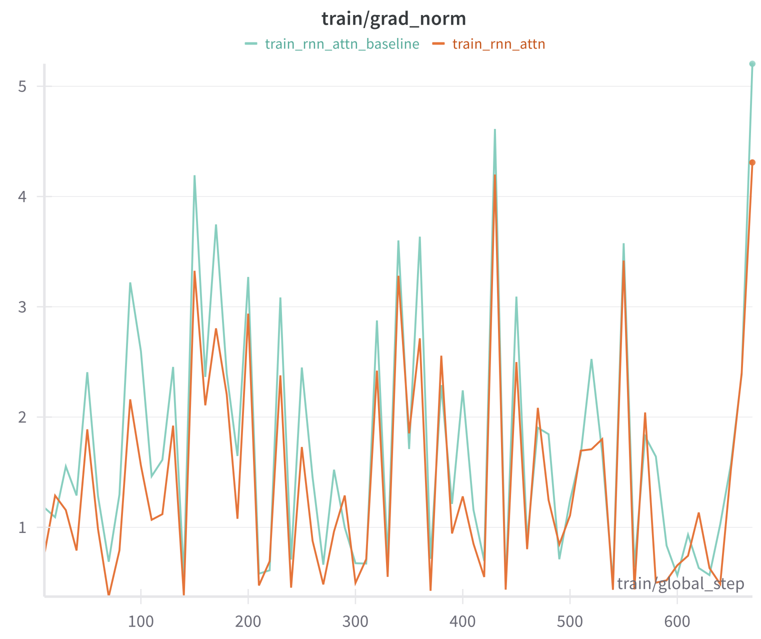

Figure 2: A demonstration of the gradient norm in our training process

## References

[1] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, 2006.

[2] Rene De La Briandais. File searching using variable length keys. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 295–298, 1959.

[3] Ian Dewancker, Michael McCourt, and Scott Clark. Bayesian optimization for machine learning: A practical guidebook. *arXiv preprint arXiv:1612.04858*, 2016.

[4] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the ACL*, 2005.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[7] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

## A Code for Greedy Matching

```python
def greedy_match(self, s: str):
    tokens = []
    i = 0
    while i < len(s):
        longest_match = None
        for j in range(i + 1, len(s) + 1):
            substring = s[i:j]
            if substring.strip() in self.tokenizer_dict:
                if longest_match is None or len(substring) > len(longest_match):
                    longest_match = substring

        if longest_match:
            tokens.append(self.tokenizer_dict[longest_match.strip()])
            i += len(longest_match)
        else:
            i += 1
            tokens.append(len(self.tokenizer_dict))

    return tokens
```