Deep Learning for Sequence Analysis

Anndischeh Mostafavi- 220051723

Course: INM706 – 2024–2025

 GitHub Link

 W&B Link

 IMDB 50K Movie Reviews Dataset Link

Typeset using LaTeX

1 Introduction

The exponential rise of user-generated content, particularly online reviews, has made automated sentiment analysis a cornerstone task within Natural Language Processing (NLP). Sentiment analysis seeks to computationally determine the emotional polarity of text, typically classifying it as positive or negative. Its applications span a wide range of domains, including customer feedback analysis, brand reputation monitoring, and market trend tracking. This project focuses on binary sentiment classification of movie reviews using the widely recognized IMDB Large Movie Review Dataset, consisting of 50,000 balanced positive and negative reviews split equally between training and testing sets [1–3].
This dataset is a balanced class distribution. As illustrated in figure 1 , the dataset used for this analysis contains an equal number of positive and negative reviews (approximately 25,000 each). This balance is crucial as it prevents classification models from learning biases towards a majority class, ensuring a fairer evaluation environment.

Fig. 1: Distribution of negative and positive labels[4].

Fig. 2: Word-cloud plot showing frequency of words[5].

The *Word-Cloud* visualizations highlight the most frequent terms within each sentiment category (Figure 2) after basic cleaning for visualization purposes. Common terms related to the domain, such as *"movie"*, *"film"*, and *"character"*, appear prominently in both positive and negative reviews. However, distinct lexical patterns associated with sentiment are clearly visible. The positive review cloud emphasizes words like **"great"**, **"well"**, **"best"**, **"love"**, and **"excellent"**, while the negative review cloud features terms such as **"bad"**, **"waste"**, **"awful"**, **"boring"**, and **"worst"**.
We conduct an empirical comparison of several prominent deep learning architectures, tracing the evolution of modeling approaches. Our investigation begins with sequential models — *Recurrent Neural Networks (RNNs)*, *Long Short-Term Memory networks* [6], and *Gated Recurrent Units* (Figure 5) — designed

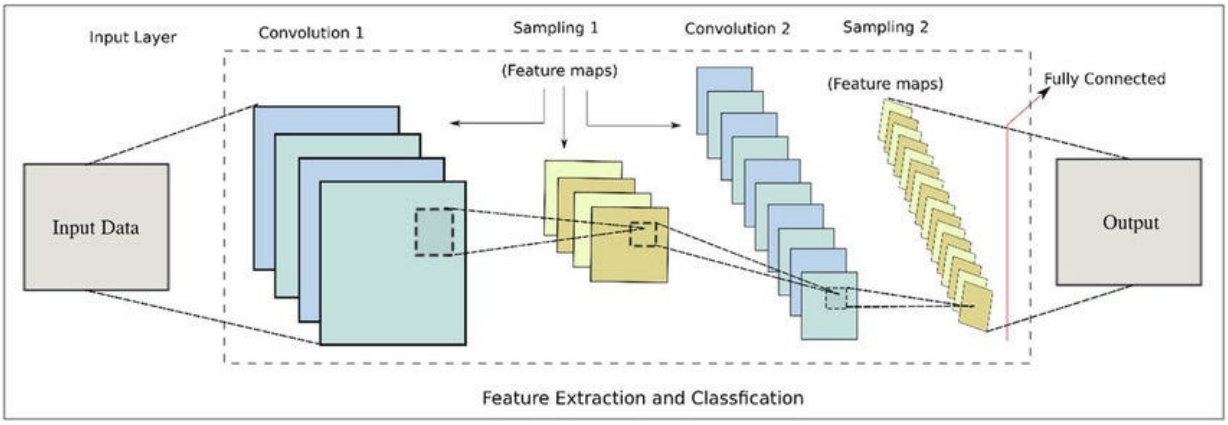


Fig. 3: Convolutional Neural Network Architecture[14].

to capture temporal dependencies in text. We then explore *Convolutional Neural Networks (CNNs)* [7]. Finally, we evaluate a modern Transformer-based model, *DistilBERT* [8], which leverages self-attention mechanisms [9] and large-scale pretraining to model complex language features.
By systematically comparing these architectures within a unified framework, this work aims to provide a comprehensive analysis of their relative strengths and weaknesses on the IMDB sentiment classification task [10].

2 Methodology

In this section, we briefly describe the Steps for training models categorized by base model, explaining their loss functions and architectures.

2.1 Preprocessing

Prior to model training, the raw review texts underwent a series of preprocessing steps to normalize the data and prepare it for ingestion by the different neural network architectures. For the `RNN`, `LSTM`, `GRU`, and `CNN` models, the pipeline involved: converting text to lowercase, removing HTML tags using *BeautifulSoup* [11], stripping URLs via regular expressions, eliminating punctuation marks, and handling emojis (typically by removal). Subsequently, the text was tokenized into individual words using *NLTK* [12]. Common English stop words were removed to reduce noise, and Porter stemming was applied to reduce words to their root form, further consolidating the vocabulary. Finally, these token sequences were converted into numerical indices based on a predefined vocabulary (limited to the most frequent tokens) and padded or truncated to a fixed sequence length (`MAX_LEN`) to facilitate batch processing. For the `DistilBERT` model, preprocessing relied on its dedicated tokenizer from the *Transformers* library [13]. This tokenizer handles lowercasing, subword tokenization (using `WordPiece`), addition of special tokens like `[CLS]` and `[SEP]`, creation of attention masks [9], and padding/truncation to the model's required maximum length, aligning the input with the pretraining paradigm.

2.2 CNN-Based

2.2.1 Convolutional Neural Network (CNN)

This architecture addresses sentiment analysis through a parallel convolutional approach, using word embeddings to represent words as dense vectors. Multiple convolutional layers, figure 3, with different kernel sizes extract various n-gram features, followed by max pooling to highlight salient information. The pooled features are concatenated and passed into a fully connected layer for sentiment prediction. Dropout is applied to prevent overfitting.
Word Embeddings: Each word w is mapped to a vector $\mathbf{e} \in \mathbb{R}^d$, where d is the embedding dimension.
Convolution: For each convolutional layer with kernel W, the output at position i is:

$$\text{ReLU}(W \cdot x_i + b) \tag{1}$$

where x_i is a slice of the embedded text, and b is the bias.
Max Pooling: The maximum value is selected from each region of the feature map.
Concatenation: Max-pooled outputs are concatenated into a single vector.
Fully Connected Layer: A linear transformation and softmax produce the final prediction:

$$y = \text{softmax}(W \cdot x + b) \tag{2}$$

Loss Function: Cross-entropy loss measures the difference between predicted y and true labels.

$$L = -\sum_i y_i \log(\hat{y}_i) \tag{3}$$

2.2.2 Dynamic Pooling CNN (DCNN)

This model enhances CNN with Dynamic k-Max Pooling, [15]. Text is embedded into vectors, followed by a single convolutional layer. Instead of fixed-size pooling, it dynamically selects top k features based on input length, improving flexibility across varying sentence lengths figure4. The selected features are flattened and passed to a fully connected layer for sentiment prediction, with dropout to reduce overfitting.

Adaptability to Sequence Length: Dynamic pooling adjusts k based on sentence length.
Improved Feature Selection: Top k most active features are emphasized.
Word Embeddings: Each word w is mapped to $\mathbf{e} \in \mathbb{R}^d$, forming embedding matrix E.
Convolution: A convolution with filter W followed by ReLU activation:

$$C = \text{ReLU}(W \cdot E + b) \tag{4}$$

Dynamic k-Max Pooling: From C, top k values are selected:

$$k = \min(k_{\text{dynamic}}, \text{sequence}_{\text{length}}) \tag{5}$$

Flattening: The pooled feature map is flattened into vector F.
Fully Connected Layer: A linear transformation and sigmoid activation predict the probability:

$$P = \sigma(W'F + b') \tag{6}$$

Loss Function: Binary cross-entropy loss is used:

$$L = -[y \log(P) + (1 - y) \log(1 - P)] \tag{7}$$

2.3 RNN-Based

2.3.1 Recurrent Neural Network (RNN)

The RNN-Classifier uses a *Recurrent Neural Network (RNN)*, see Figure 5, to capture the sequential nature of text for sentiment classification. Each word is embedded into a dense vector. These embeddings are fed into a multi-layer vanilla `RNN` which processes the sequence step-by-step, maintaining a hidden state that evolves with the input. The final hidden state of the last `RNN` layer is used as the summary representation of the text. This representation is passed through a fully connected layer to predict the sentiment label. A dropout layer is applied after embedding and before classification to prevent overfitting.

Word Embeddings: Each word w is mapped to a vector $\mathbf{e} \in \mathbb{R}^d$, forming an embedding matrix for the sequence.

Recurrent Processing: At each time step t, the RNN updates its hidden state as follows:

$$\mathbf{h}_t = \text{ReLU}(W_{\text{ih}}\mathbf{x}_t + W_{\text{hh}}\mathbf{h}_{t-1} + \mathbf{b}) \tag{8}$$

where $\mathbf{x}_t$ is the embedded input at time t, $\mathbf{h}_{t-1}$ is the previous hidden state, and $W_{\text{ih}}, W_{\text{hh}}$ are weight matrices.

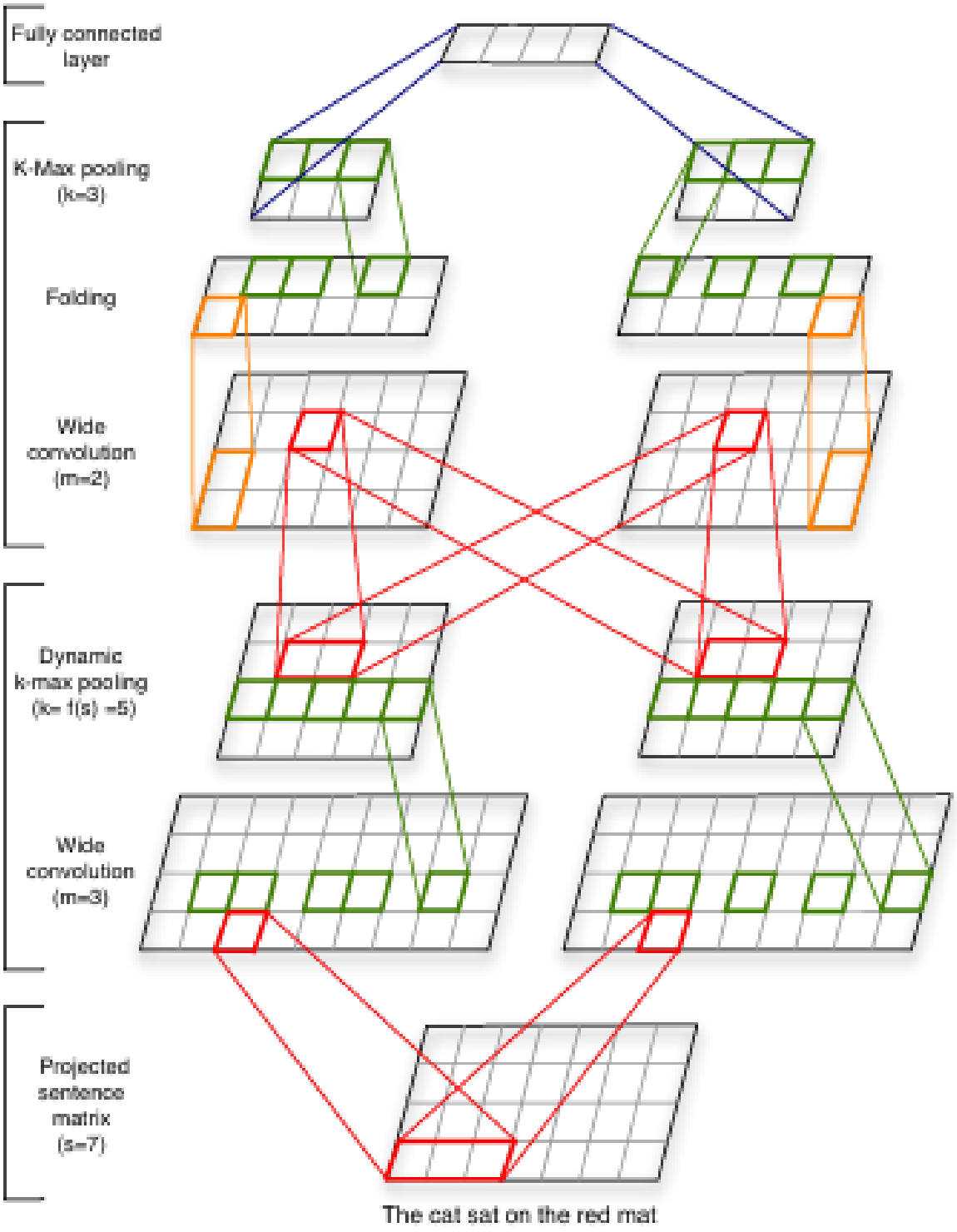


Fig. 4: Dynamic CNN using dynamic k-max pooling[16].

Final Hidden State: The last hidden state $\mathbf{h}_T$ (where T is the sequence length) summarizes the entire input sequence.

Dropout: A dropout layer is applied to $\mathbf{h}_T$ to encourage generalization and reduce overfitting.

Fully Connected Layer: The hidden representation is passed through a fully connected layer with softmax activation:

$$\mathbf{y} = \text{softmax}(W_{\text{fc}}\mathbf{h}_T + \mathbf{b}_{\text{fc}}) \tag{9}$$

Loss Function: The model is trained using cross-entropy loss between predicted labels and ground truth.

2.3.2 Long short term Memory (LSTM)

The LSTM-Classifier utilizes a *Long Short-Term Memory (LSTM)* network, see Figure 5, which is an advanced RNN capable of capturing long-range dependencies through its gating mechanisms. After embedding words, the`LSTM` processes the sequence and outputs hidden states. The model uses either the last hidden state (for unidirectional `LSTM`) or the concatenation of the last forward and backward hidden states (for bidirectional `LSTM`) as the text representation. This is followed by dropout and a fully connected layer for classification.

Word Embeddings. Each word w is mapped to a dense vector $e \in \mathbb{R}^d$.

LSTM Processing. At each time step t, the LSTM updates its hidden and cell states as:

$$(h_t, c_t) = \text{LSTMCell}(x_t, (h_{t-1}, c_{t-1})) \tag{10}$$

where x_t is the embedded input, h_t is the hidden state, and c_t is the memory cell that helps preserve long-term dependencies.

Bidirectional Processing. If the LSTM is bidirectional, the final hidden representation is formed by concatenating the last forward and backward hidden states:

$$h = \text{concat}(h_{\text{forward}}, h_{\text{backward}}) \tag{11}$$

Dropout. Dropout is applied to the final hidden representation h to prevent overfitting.

Fully Connected Layer. A linear transformation maps the hidden representation to sentiment classes:

$$y = \text{softmax}(W_{\text{fc}} \cdot h + b_{\text{fc}}) \tag{12}$$

Loss Function. Cross-entropy loss is used to compare the predicted probabilities with the ground-truth sentiment labels during training.

2.3.3 Gated Recurrent Unit (GRU)

The GRU-Classifier [17] employs a *Gated Recurrent Unit (GRU)*, see Figure 5, a simplified variant of `LSTM` that uses fewer gates but retains the ability to model long-term dependencies effectively. The input text is embedded, and then passed through the `GRU` network. Similar to the `LSTM` setup, the final hidden states are used for classification, with dropout applied for regularization.

Word Embeddings. Each word w is mapped to a dense vector $e \in \mathbb{R}^d$.

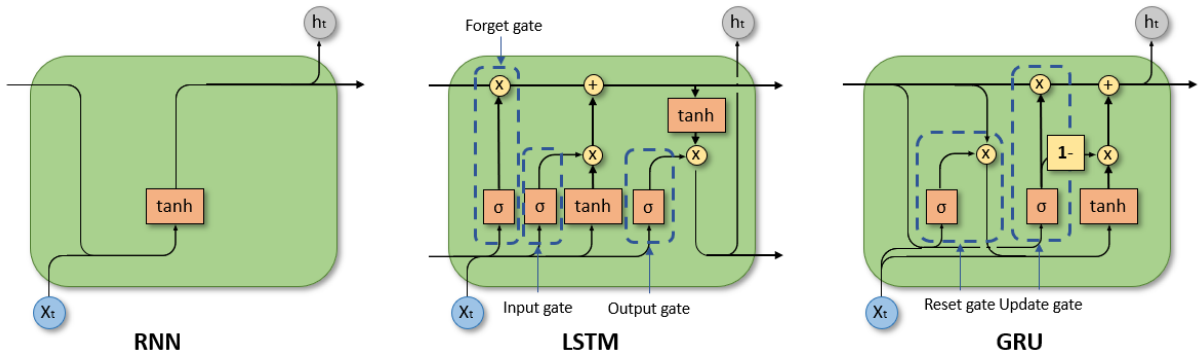


Fig. 5: RNN, LSTM and GRU Architecture for Sentiment Classification[18].

GRU Processing. At each time step t, the `GRU` updates its hidden state using reset and update gates as follows:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{13}$$

where z_t is the update gate, h_{t-1} is the previous hidden state, $\tilde{h}_t$ is the candidate hidden state, and $\odot$ denotes element-wise multiplication.

Bidirectional Processing. For bidirectional `GRUs`, the final forward and backward hidden states are concatenated:

$$h = \text{concat}(h_{\text{forward}}, h_{\text{backward}}) \tag{14}$$

Dropout. Dropout is applied to the final hidden state h to reduce overfitting.

Fully Connected Layer. The hidden state is mapped to sentiment output via a fully connected layer and softmax activation:

$$y = \text{softmax}(W_{\text{fc}} \cdot h + b_{\text{fc}}) \tag{15}$$

Loss Function. Cross-entropy loss is used to optimize the predicted probabilities against the ground-truth labels.

2.4 Transformer-Based

2.4.1 DistilBERT-Classifier

The DistilBERT-Classifier [19] is based on `DistilBERT`, a compact Transformer model designed to retain most of `BERT`'s language understanding capabilities while being faster and lighter. It is particularly suitable for tasks like sentiment classification.
This model processes tokenized input sequences through a pretrained `DistilBERT` encoder, extracts the representation of the `[CLS]` token from the final hidden layer, and passes it through a fully connected layer for sentiment prediction (see Figure 6).
Input Representation: Each input sentence is tokenized into two components:

- ▷ `input_ids`: A sequence of token indices from the tokenizer vocabulary.
- ▷ `attention_mask`: A binary sequence marking real tokens (1) and padding tokens (0).

Formally, given a sequence of words $S = (w_1, w_2, \ldots, w_n)$, we define:

$$\texttt{input\_ids} \in \mathbb{N}^n \tag{16}$$

$$\texttt{attention\_mask} \in \{0, 1\}^n \tag{17}$$

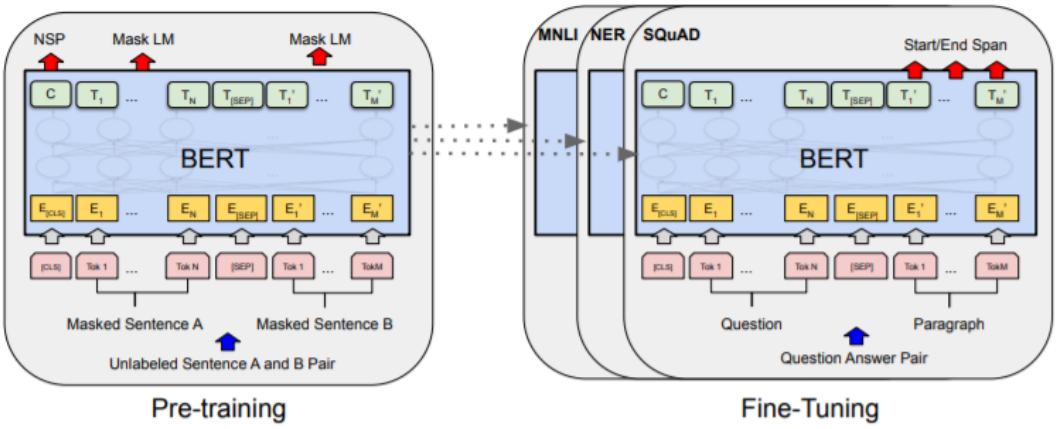


Fig. 6: DistilBERT architecture for Text Classification [20].

DistilBERT Encoding: The input is processed through DistilBERT's transformer layers, where each layer applies multi-head self-attention followed by a position-wise feed-forward network. The output is a sequence of contextualized hidden states:

$$H = (h_1, h_2, \ldots, h_n), \quad h_i \in \mathbb{R}^d \tag{18}$$

where d is the hidden dimension (typically 768).

[CLS] Token Representation: The first hidden state h_1, corresponding to the `[CLS]` token, is used to summarize the sequence:

$$h_{\texttt{[CLS]}} = h_1 \tag{19}$$

Dropout Regularization: To prevent overfitting, a dropout layer is applied:

$$h_{\texttt{[CLS]}} = \text{Dropout}(h_{\texttt{[CLS]}}) \tag{20}$$

Classification Layer: The regularized embedding is passed through a linear layer to obtain logits:

$$\text{logits} = W_{\text{fc}} \cdot h_{\texttt{[CLS]}} + b_{\text{fc}} \tag{21}$$

where $W_{\text{fc}} \in \mathbb{R}^{d_{\text{out}} \times d}$, $b_{\text{fc}} \in \mathbb{R}^{d_{\text{out}}}$, and d_{out} is the number of output classes.

Depending on the task:

- ▷ For binary classification: $\hat{y} = \text{sigmoid}(\text{logits})$
- ▷ For multi-class classification: $\hat{y} = \text{softmax}(\text{logits})$

Loss Function: Model training is done via cross-entropy loss:

$$\mathcal{L} = -\sum_i y_i \log \hat{y}_i \tag{22}$$

where y is the one-hot true label vector, and $\hat{y}$ is the predicted probability distribution.

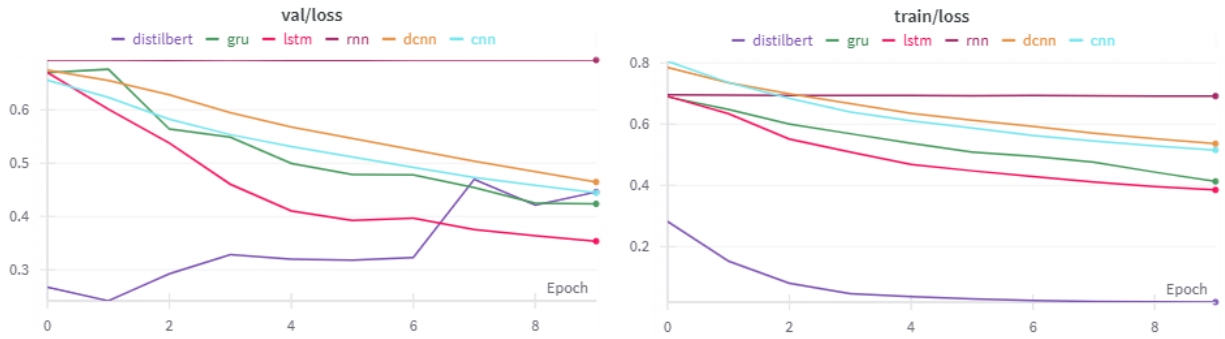
3 Result

The comparative analysis of deep learning architectures for sentiment classification on the `IMDB` dataset [1] reveals notable differences in performance across models. As shown in Table 1, `DistilBERT` clearly

(a) Validation accuracy across epochs.

(b) Training accuracy across epochs.

Fig. 7: Training and validation accuracy curves for all evaluated models on the `IMDB` dataset. `DistilBERT` exhibits rapid convergence and maintains the highest accuracy throughout, while the `RNN` fails to improve beyond random-level performance [4].



(a) Validation loss across epochs.

(b) Training loss across epochs.

Fig. 8: Training and validation loss curves for different models on the `IMDB` dataset, illustrating convergence behaviors and generalization performance [5].

outperforms all other architectures, achieving a test accuracy of 91.63%, a test loss of 0.2218, and an F1-score of 0.9195. These results highlight its strong capability for accurately classifying movie reviews. This is further supported by the training and validation accuracy curves in Figures 7(a) and 7(b), where `DistilBERT` demonstrates rapid convergence and consistently achieves the highest accuracy throughout training. In contrast, the RNN model stagnates around 50% validation accuracy—essentially at chance level. Its poor performance is reflected in a high test loss of 0.6937 and an extremely low recall of 0.0496 (Table 1), suggesting that vanilla `RNN` fails to capture the long-range dependencies present in text data. This limitation is also evident in its flat training loss curve (Figure 8(b)).

Tab. 1: Summary of Models' Evaluation[4]

Model	Test Loss	Accuracy	Precision	Recall	F1-Score
CNN	0.4354	0.8040	0.7783	0.8501	0.8126
DCNN	0.4583	0.7988	0.7765	0.8392	0.8066
RNN	0.6937	0.5041	0.5455	0.0496	0.0909
LSTM	0.3476	0.8591	0.8725	0.8411	0.8565
GRU	0.4025	0.8489	0.8644	0.8277	0.8457
DistilBERT	0.2218	0.9163	0.8851	0.9568	0.9195

The `LSTM` and `GRU` models achieve reasonably high performance, with accuracy scores of 85.91% and 84.89% respectively. Their validation accuracy and loss curves 7(a) and 8(a) demonstrate a more stable learning process compared to the `RNN`. However, they remain significantly lower than `DistilBERT`, potentially due to DistilBERT's pre-training on a massive corpus, enabling it to learn more generalizable language representations. The `CNN` and `DCNN` models show moderate performance with accuracy scores of

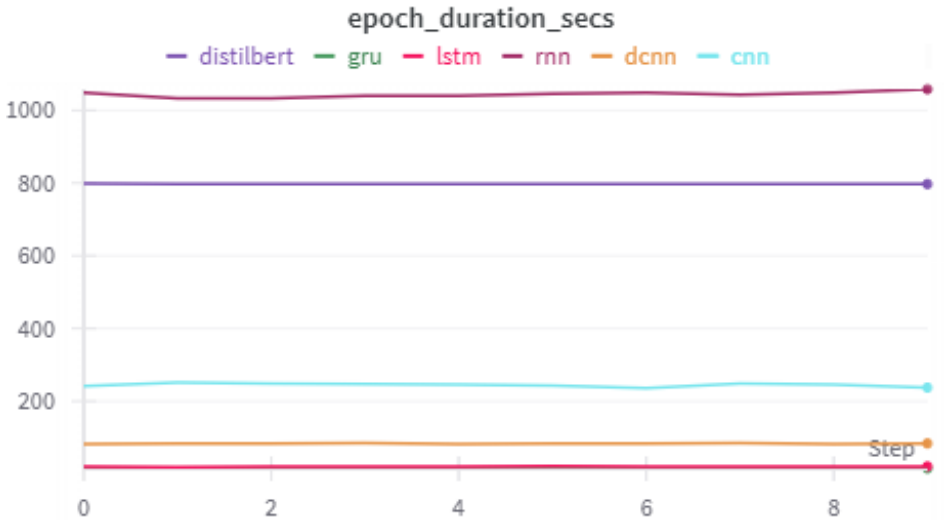


Fig. 9: Training time per epoch for each model. More complex architectures like `DistilBERT` and `LSTM/GRU` require significantly longer computation times[4].

Tab. 2: Probability of predicted sentence's sentiment[4].

Model	Predicted Sentiment (Probability)
CNN	positive (0.5060)
DCNN	negative (0.4754)
RNN	negative (0.4989)
LSTM	negative (0.3993)
GRU	negative (0.4438)
DistilBERT	negative (0.0018)

80.40% and 79.88% respectively. The dynamic pooling strategy employed by `DCNN` appears to offer only a marginal improvement over the standard `CNN` architecture, potentially due to the relatively simple nature of the sentiment classification task.

It is also important to consider the computational cost of these models. Figure 9 highlights the epoch duration, revealing that `DistilBERT` and `LSTM/GRU` have the highest processing time per epoch due to their architectural complexity and, in`DistilBERT`'s case, the larger model size. This needs to be balanced against the performance gains when choosing a model for deployment, particularly in resource-constrained environments. The prediction example in Table 2 further illustrates the varying capabilities of these models. For the specific review tested,"I fell asleep halfway through.", `DistilBERT` correctly predicts a negative sentiment with a high degree of confidence (0.0018 probability of being positive), whereas the other models, except `CNN`, incorrectly predicted a negative sentiment. This single instance underscores `DistilBERT`'s potentially improved ability to discern nuanced sentiment cues compared to the other models. Ultimately, while `DistilBERT` offers the most accurate sentiment classification, the simpler models like `CNN`, `RNN`, `LSTM`, and `GRU` may provide a reasonable trade-off between performance and computational cost in applications where real-time processing or resource constraints are paramount.

As visually represented in the confusion matrices figure 10 and quantified in Table 1; `DistilBERT` demonstrates superior performance in sentiment classification on the `IMDB` dataset [1]. The confusion matrix (figure 10(a)) shows that `DistilBERT` correctly classifies 3284 negative reviews and 3588 positive reviews, with relatively few misclassifications (466 false positives and 162 false negatives), corroborating its high accuracy and F1-score (91.63% and 0.9195, respectively). Conversely, the `RNN` (figure 10(d)) shows very poor performance, with a large number of both false positives (155) and false negatives (3564). The `RNN` is essentially predicting the majority class (negative) for most inputs, consistent with the near-zero recall score in Table 1. This highlights the `RNN`'s inability to effectively learn from the data and discriminate between positive and negative sentiments. The `LSTM` and `GRU` models (figure 10(c)) exhibit more balanced performance, with a clear diagonal dominance indicating accurate classification, although they still have a noticeable number of misclassifications (461 and 487 false positives, respectively, for `LSTM` and `GRU`).

Their relatively higher accuracy and F1-scores compared to the CNN and DCNN are also reflected in the confusion matrices. The CNN and DCNN (10(f)) have more misclassifications than LSTM/GRU or DistilBERT, with the number of false positives (908 and 906, respectively) larger.

4 Conclusions

This project investigated the efficacy of various deep learning architectures for binary sentiment classification of movie reviews using the IMDB dataset. Starting with fundamental sequential models like RNNs, LSTMs, and GRUs, we explored their ability to capture temporal dependencies within text. We then moved to convolutional approaches with CNNs and DCNNs, evaluating their capacity to extract salient n-gram features. Finally, we examined the performance of a state-of-the-art Transformer-based model, DistilBERT, which leverages self-attention mechanisms [9] and pre-training to achieve superior language understanding.

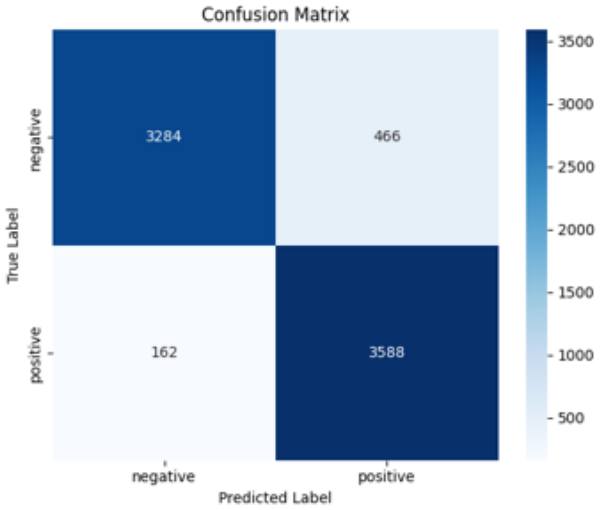
Our findings demonstrate a clear hierarchy in model performance. The vanilla RNN proved largely ineffective, struggling to learn relevant features and essentially performing at chance level. LSTM and GRU models, while more successful in capturing long-range dependencies than the RNN, were outperformed by the CNN and DCNN architectures. Ultimately, DistilBERT emerged as the most accurate model, achieving a test accuracy of 91.63%. This improvement can be attributed to its pre-training on a massive corpus, enabling it to learn more generalizable language representations, and its sophisticated self-attention mechanism. However, this performance comes at a computational cost, with DistilBERT requiring significantly more processing time per epoch compared to the simpler models. Therefore, the choice of model depends on the specific application's requirements, balancing accuracy with computational constraints. While DistilBERT offers the best accuracy, simpler models like CNN, LSTM, and GRU may provide a more practical trade-off for resource-limited or real-time processing scenarios. The comprehensive comparison of these architectures provides valuable insights into their relative strengths and weaknesses for sentiment classification, informing future model selection and development efforts.

5 Reflections

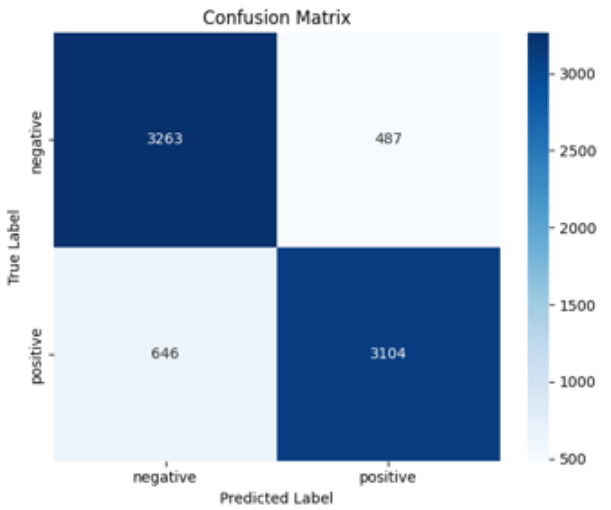
This coursework provided a valuable opportunity to delve into the intricacies of *sentiment analysis* using *deep learning*. The core learning outcome was gaining a practical understanding of different neural network architectures and their suitability for *NLP* tasks. Working with the IMDB dataset allowed for hands-on experience in data preprocessing, model implementation, and performance evaluation. Specifically, the effort involved in cleaning and preparing the raw text data highlighted the crucial role of preprocessing in achieving optimal model performance.

One of the main challenges encountered was the poor performance of the vanilla RNN. Initially, the expectation was that RNNs would be capable of capturing sequential information, but their performance was surprisingly weak. This led to a deeper investigation into the limitations of simple RNNs, particularly their vanishing gradient problem [21], and motivated the exploration of more advanced recurrent architectures like LSTMs and GRUs. Another challenge involved optimizing the hyperparameters for each model. While a systematic grid search was initially planned, time constraints led to a more iterative, experiment-driven approach.

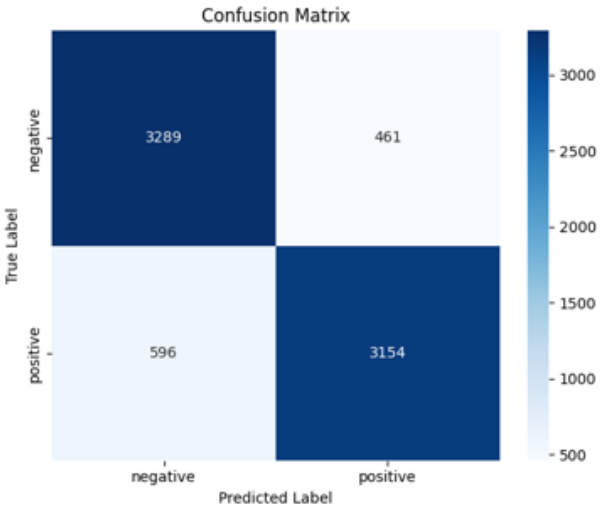
If the project were to be repeated, several changes would be implemented. First, more time would be dedicated to *hyperparameter tuning*, perhaps using a more structured approach like *Bayesian optimization* [22] or implementing an automated hyperparameter optimization framework such as *Ray Tune* [23]. Second, a more extensive exploration of different *word embedding* techniques, such as *GloVe* [24] or *FastText* [25], would be conducted to determine if they could further improve model performance. Finally, given DistilBERT's superior results, further experimentation with fine-tuning strategies and different pre-trained transformer models (e.g., RoBERTa [26], smaller BERT variants) would be prioritized to potentially achieve even higher accuracy. Furthermore, a more robust error analysis would have been beneficial to identify specific types of reviews that were consistently misclassified and to inform potential data augmentation or model refinement strategies. The experience also reinforced the importance of carefully considering the trade-offs between model complexity, performance, and computational cost when selecting an architecture for a specific application.



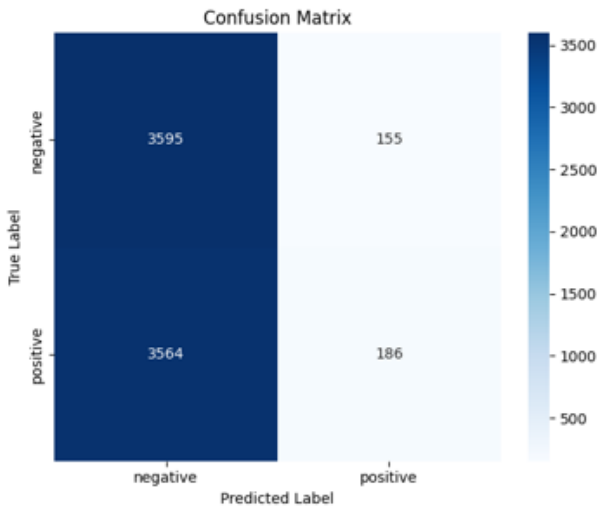
(a) `DistilBERT`: Confusion matrix showing excellent performance with high true positives (3588) and true negatives (3284), and few misclassifications.



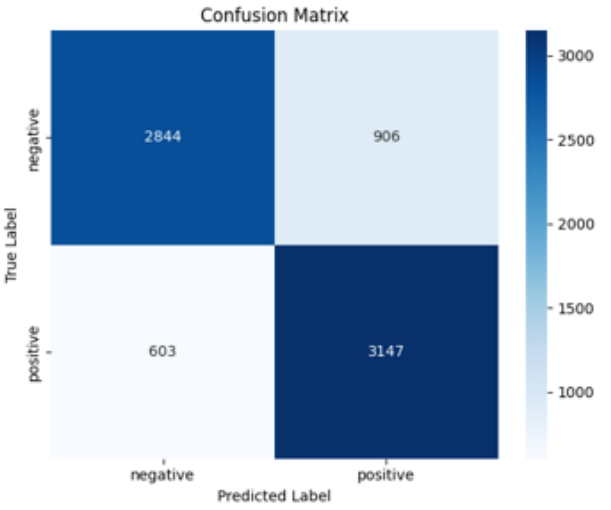
(b) `GRU`: Balanced classification with 487 false positives and 646 false negatives. Performance is acceptable but not optimal.



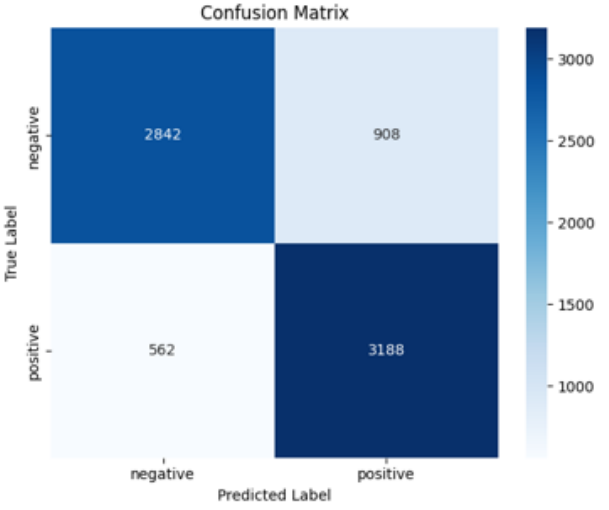
(c) `LSTM`: Solid accuracy with 461 false positives and 596 false negatives. Better performance than CNN-based models.



(d) `RNN`: Poor classification with extremely high false negatives (3564) and low true positives. Indicates failure in sentiment distinction.



(e) `DCNN`: Noticeable misclassifications, especially 906 false positives. Lower accuracy compared to recurrent models.



(f) `CNN`: Performs better than DCNN and RNN, but still suffers from 908 false positives. Less effective than transformer or recurrent models.

Fig. 10: Confusion matrices of different models on the `IMDB` sentiment classification task. `DistilBERT` outperforms other models, while `RNN` performs the worst with highly imbalanced predictions. `LSTM` and `GRU` achieve relatively balanced and accurate results, and CNN-based models show higher misclassification rates[4, 5].

References

[1] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Large movie review dataset. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*, 2011. URL `http://ai.stanford.edu/~amaas/data/sentiment/`.

[2] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

[3] Lakshmi Narayan Pathi. Imdb dataset of 50k movie reviews, 2020. URL `https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`. Accessed: 2025-05-16.

[4] Anndischeh Mostafavi. *wandb project for deep learning for sequence analysis coursework*, 2025.

[5] Anndischeh Mostafavi. *github repository for deep learning for sequence analysis coursework*, 2025.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[8] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[10] Michael E. Akintunde. Deep learning for sequence analysis. Lecture notes, INM706, City University of London, 2025. Accessed in course materials, Spring 2025.

[11] Leonard Richardson. Beautiful soup: We called him tortoise because he taught us. *Beautiful Soup: We called him Tortoise because he taught us*, 2015.

[12] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[13] 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.

[14] Bilal Jan, Haleem Farman, Murad Khan, Muhammad Imran, Ihtesham Ul Islam, Awais Ahmad, Shaukat Ali, and Gwanggil Jeon. Deep learning in big data analytics: A comparative study. *Computers & Electrical Engineering*, 75:275–287, 2019. ISSN 0045-7906. doi: https://doi.org/10.1016/j.compeleceng.2017.12.009. URL `https://www.sciencedirect.com/science/article/pii/S0045790617315835`.

[15] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[16] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In Kristina Toutanova and Hua Wu, editors, *Proceedings of the 52nd Annual*

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1062. URL https://aclanthology.org/P14-1062/.

[17] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the eighth workshop on syntax, semantics and structure in statistical translation*, pages 103–111, 2014.

[18] Zerong Rong, Wei Sun, Yutong Xie, Zexi Huang, and Xinlin Chen. Mixture of experts leveraging informer and lstm variants for enhanced daily streamflow forecasting. *Journal of Hydrology*, 653: 132737, 2025. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2025.132737. URL https://www.sciencedirect.com/science/article/pii/S0022169425000757.

[19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

[21] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[22] Bobak Shahriari, Kevin Wilson, and ... (etc.). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[23] Richard Liaw, Eric Liang, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training, 2018.

[24] 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.

[25] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[26] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, and ... (etc.). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.