EEE 486 - Assignment 2 Report

Emine İrem Esendemir

Department of Computer Engineering Bilkent University

Ankara, Turkey

Nisan 7, 2025

Abstract

This report presents the finetuning of pretrained BERT models for text classification on the Recognizing Textual Entailment (RTE) dataset from the GLUE benchmark. Two different document representation approaches were explored: the standard method using the `CLS` token representation with BertForSequenceClassification in Part 1, and an enhanced approach combining mean pooling with `CLS` token representations in Part 2 . Hyperparameter optimization was performed for both approaches to identify optimal results. The custom mean pooling approach achieved improved accuracy (71.53%) compared to the baseline `CLS` token method (61.73%), demonstrating the effectiveness of combining different BERT representations for the RTE task.

1 Introduction

In this assignment, I conducted experiments with BERT (Bidirectional Encoder Representations from Transformers) [2]. BERT's bidirectional approach allows it to process text from both directions simultaneously, resulting a more comprehensive understanding. For this assignment, I fine-tuned the `bert-base-uncased` model on the Recognizing Textual Entailment (RTE) task from the GLUE benchmark [4]. The RTE task involves determining whether a given hypothesis can be inferred from another statement. The task is a binary classification task resulting in entailment or non-entailment.

The assignment consists of two distinct parts:

In Part 1, I fine-tuned thee standard 'bert-base-uncased' model using the RTE dataset without making any architectural modifications. This approach acts as the baseline for comparison and sticks to the original BERT implementation. Finetuned model is pushed to the Hugginface [3].

For Part 2, I developed an enhanced approach that goes beyond of using only the `CLS` token representation. I implemented a custom architecture called MeanPoolingBertForClassification that combines different BERT outputs to create a contextually richer document representation. Specifically, this model computes a mean-pooled representation across all token embeddings and concatenates it with the `CLS` token representation, followed by a multi-layer classification head. This method utilizes information at both the token and sentence levels, which may help uncover more subtle semantic connections between the sentences.

Both models were evaluated using accuracy as the performance metric. To find optimal results, hyperparameter optimization was conducted by the help of Optuna library for both approaches, exploring different learning rates, training epochs, sequence lengths, and dropout rates [1].

The remaining sections of this report provide detailed explanations of the architecture implementations and present experimental results.

2 Architecture: Enhanced BERT with Mean Pooling

In Part 2, I implemented a custom `MeanPoolingBertForClassification` model by using mean-pooled representation across all token embeddings and concatenates it with the `CLS` token representation

The custom model architecture consists of:

- **BERT Base Encoder [2]**: I used the pre-trained `bert-base-uncased` model with `output_hidden_states=True` to access all hidden representations.
- **Document Representation Strategy**: Instead of relying solely on the `CLS` token, my approach:
  - Extracts the sequence output for all tokens
  - Creating a mean-pooled vector by averaging all token vectors
  - Retrieves the standard pooled output (`CLS` token representation)
  - Concatenates these two representations into a combined feature vector
- **Classification Head**: A multi-layer classifier with:
  - Input dimension of 768×2 (concatenated representations)

– Hidden layer with 512 units and ReLU activation
– Dropout regularization
– Output layer with 2 units (for binary classification)

The key equations for the mean pooling operation are:

$$M_{expanded} = A \otimes \mathbf{1}_h \tag{1}$$

$$S_{emb} = \sum_{i=1}^{L} H_i \odot M_{expanded,i} \tag{2}$$

$$S_{mask} = \sum_{i=1}^{L} M_{expanded,i} \tag{3}$$

$$h_{mean} = \frac{S_{emb}}{max(S_{mask}, \epsilon)} \tag{4}$$

where A is the attention mask, H is the last hidden state, $\odot$ denotes element-wise multiplication, $\otimes$ represents tensor expansion, L is the sequence length.

The final representation is formed by concatenation:

$$h_{combined} = [h_{mean}; h_{[CLS]}] \tag{5}$$

And the classification is performed as:

$$h_{hidden} = \text{ReLU}(W_1 \cdot \text{dropout}(h_{combined}) + b_1) \tag{6}$$

$$y = \text{softmax}(W_2 \cdot \text{dropout}(h_{hidden}) + b_2) \tag{7}$$

This architecture leverages both token-level information (through mean pooling) and sentence-level information (through the `CLS` token), providing a better representation for the classification task.

3 Results

3.1 Part 1: `CLS` Token Approach

In Part 1, I fine-tuned a standard `BertForSequenceClassification` model that uses the `CLS` token representation. After hyperparameter optimization with Optuna (10 trials), the best configuration achieved:

Figure 1 shows the training loss and validation accuracy curves for the Part 1 model.

Table 1: Best Hyperparameters for Part 1

Hyperparameter	**Value**
Learning rate	1.845×10^{-5}
Number of epochs	3
Batch size	64
Dropout rate	0.1
Max sequence length	16

Table 2: Performance Metrics for Part 1

Metric	**Value**
Validation Loss	0.6625
Validation Accuracy	61.73%

The results show that the model reaches a validation accuracy of 61.73% after 3 epochs. The loss curves indicate that the model is learning effectively, as training loss decreases during training.

3.2 Part 2: Mean Pooling Approach

For Part 2, I implemented the custom `MeanPoolingBertForClassification` model and performed hyperparameter optimization with the following search space:

Table 3: Hyperparameter Search Space for Part 2

Hyperparameter	**Range**
Learning rate	$[5 \times 10^{-6}, 3 \times 10^{-5}]$
Number of epochs	[3, 4, 5, 6, 7]
Max sequence length	[48, 64, 128]
Batch size	[16, 32]
Dropout rate	[0.1, 0.2, 0.3]

After optimization, the best configuration achieved:

Figure 2 shows the training and validation metrics for the Part 2 model.

My model achieved its best validation accuracy of 71.53% at epoch 2, followed by a slight decrease to 71.18% at epoch 3. I observed that the training loss steadily decreases from 0.676 to 0.297 across the three

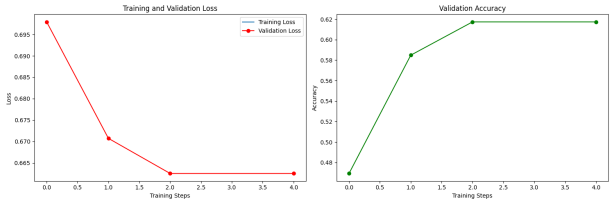

Figure 1: Training and validation metrics for Part 1 (`CLS` token approach). Left: Loss curve. Right: Validation accuracy.

Table 4: Best Hyperparameters for Part 2

Hyperparameter	**Value**
Learning rate	2.693×10^{-5}
Number of epochs	3
Max sequence length	64
Batch size	16
Dropout rate	0.108

epochs, while the validation loss initially decreases from epoch 1 to 2 (0.621 to 0.611) before increasing in epoch 3 (0.680). This pattern suggests that the model begins to overfit slightly after the second epoch, which might happen in fine-tuning tasks. I implemented early stopping in my training loop to save the model weights from the best-performing epoch to avoid this effect.

3.3 Comparison of Approaches

My mean pooling approach outperformed the standard `CLS` token method by 9.80%, demonstrating the effectiveness of combining token-level information through mean pooling with the sentence-level information from the CLS token. In particular, I found that the optimal sequence length for the mean pooling approach (64) was significantly longer than for the `CLS` approach (16), indicating that the mean pooling method better utilizes contextual information from longer sequences.

Table 5: Performance Metrics for Part 2

Metric	**Value**
Validation Loss	0.680
Validation Accuracy	71.53%

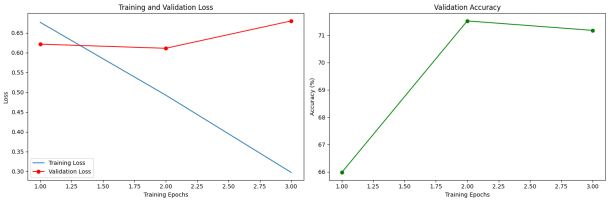

Figure 2: Training and validation metrics for Part 2 (Mean Pooling approach). Left: Loss curves. Right: Validation accuracy.

4 Discussion and Conclusion

In this assignment, I explored two different approaches for fine-tuning BERT on the RTE task, and my results clearly demonstrate that combining different types of representations from BERT can significantly improve performance compared to using only the `CLS` token. My custom mean pooling approach, which combined the `CLS` token representation with a mean-pooled representation of all tokens, achieved a 9.80% improvement in accuracy over the standard approach.

Based on my experiments, I've identified several key factors that contributed to this improvement:

- **Representation Quality:** I found that the way BERT's outputs are extracted and utilized significantly impacts classification performance. The standard approach of using only the `CLS` token, while effective as a baseline, doesn't fully get use of the contextual information captured by BERT across all tokens. By using mean pooling to incorporate information from all tokens, I created a more comprehensive representation of the input text that captured nuances missed by the `CLS` token alone.

Table 6: Comparison of Part 1 and Part 2 Models

Metric	Part 1	Part 2
Validation Accuracy	61.73%	71.53%
Improvement	-	+9.80%
Optimal Sequence Length	16	64
Optimal Epochs	3	2
Optimal Batch Size	64	16

- **Sequence Length Sensitivity:** My experiments revealed that the optimal sequence length differed between the two approaches (16 for `CLS` vs. 64 for mean pooling). This suggests that the mean pooling approach can make better use of longer sequences, likely because it considers all tokens rather than just focusing on the `CLS` token. This ability to process longer contexts is particularly beneficial for the RTE task, which requires understanding relationships between potentially complex sentences.

I also observed a notable between training loss and validation performance. The training loss for my mean pooling model decreased consistently across epochs (from 0.676 to 0.297), while validation accuracy peaked at epoch 2 (71.53%) before slightly declining. This pattern is normal in fine-tuning scenarios and shows the importance of early stopping and model checkpointing strategies, which I implemented to ensure I captured the best-performing model.

While my mean pooling approach showed clear advantages, I recognize some limitations in my study:

- The combined representation requires additional computation due to its larger dimensionality (768*2)
- The approach introduces more hyperparameters to tune, potentially increasing the complexity of the optimization process
- There's evidence of potential overfitting after epoch 2, suggesting that regularization might be beneficial for longer training runs

For future work, I would consider exploring:

- More sophisticated pooling strategies, such as attention-weighted pooling
- Additional regularization techniques
- Ensemble methods that might further improve performance by combining predictions from models using different representation strategies

In conclusion, my experiments demonstrate that moving beyond the standard `CLS` token representation can result in performance improvements for the RTE text classification task. By combining token-level information (through mean pooling) with sentence-level information (from the `CLS` token), I created a more comprehensive document representation that better captured the semantic relationships needed for accurate entailment classification. These results taught me that the default approaches provided by standard libraries, while convenient, may not always result in optimal performance for specific NLP tasks.

References

[1] Takuya Akiba, Shotaro Sano, Takeru Yanase, Toshihiko Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Emine İrem Esendemir. bert-finetuned-rte. `https://huggingface.co/iremEsendemir/bert-base-uncased-finetuned-rte-run_3`, 2025. Accessed: April 2025.

[4] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.