# Experiments in Improving NLP Multitask Performance

Stanford CS224N Default Project

**Carl Shan**
Department of Computer Science
Stanford University
carlshan@stanford.edu

## Abstract

In my project, I explore multiple ways to improve downstream NLP tasks performance by leveraging a variety of techniques. In this report, I describe several of the techniques employed: dynamically batch adjustments based on learning loss, weight initialization, increasing epoch size, increasing prediction heads' hidden size, doubling the batch size, equally weighting tasks during training and more. I also used ensemble methods (Random Forest) with embeddings for downstream classification. I end with a discussion of learnings and potential further ideas.

## 1 Key Information to include

- Mentor: NA
- External Collaborators (if you have any): NA
- Sharing project: NA

## 2 Introduction

Bidirectional Encoder Representations from Transformers ("BERT") is a transformer-based model that generates contextual word embeddings introduced by Devlin et al. (2019). These contextual representations can be used in downstream tasks, including sentiment classification, paraphrase detection, semantic textual similarity and more.. In this report, I assess the performance using BERT embeddings on these tasks and how performance changes in response to augmenting different aspects of my approach, including batch size, batch number, prediction head architecture, the number of epochs and other aspects of my system.

I summarize these results below in the Data section, and conclude by discussing learnings and reflections, as well as potential further augmentations and ideas.

## 3 Related Work

The Transformer architecture, introduced by Vaswani et al. (2023), marked a significant moment the field of natural language processing (NLP). This neural network architecture eschews the traditional recurrent neural networks (RNNs) in favor of a self-attention mechanism, enabling more efficient parallel processing and capturing long-range dependencies in text. The Transformer's encoder-decoder structure, originally designed for machine translation, has been adapted and extended for a wide array of NLP tasks, including several of the ones I investigated in my project.

Building upon the Transformer architecture, Devlin et al. (2019) developed BERT (Bidirectional Encoder Representations from Transformers), a pre-trained language model that has achieved state-of-the-art performance on numerous NLP benchmarks. BERT leverages the bidirectional nature

of the Transformer encoder to learn deep contextual representations of words, allowing it to better understand the nuances and ambiguities of language.

These embeddings can be used for a variety of tasks. The tasks I investigated were the following:

- **Sentiment Classification:** A 5-way classification task in assessing the sentiment of a piece of text.
- **Paraphrase Detection**: A binary classification task in understand whether two sentences are paraphrases, or "restatements of text giving the meaning in another form." Fernando and Stevenson (2009)
- **Semantic Textual Similarity (STS):** A regression task to measure the degree of semantic equivalance between two sentences.

## 4 Approach

In my experiments, I aimed to explore the effectiveness of BERT embeddings for various downstream NLP tasks, including sentiment analysis, paraphrase detection, and semantic textual similarity assessment. To begin, I focused on the architecture of the prediction heads for each task. I experimented with different Multi-layered Perceptrons (MLPs) using GELU activation, Dropout, and a linear layer. I varied the hidden size of these MLPs, testing values of 128 and 384, to determine the optimal configuration for each task.

### 4.1 Prediction Head Architectures:

The architecture of the prediction heads I used are described below:

- **Sentiment Classification:** A linear layer, with hidden size half the BERT hidden size, GELU activation, dropout ($p = 0.3$), and finally a Softmax. I use Cross-Entropy loss, where I also tried weighting the loss by the inverse frequency of the class occurrences.
- **Quora Paraphrase:** A linear layer, with hidden size half the BERT hidden size, GELU activation, dropout ($p = 0.3$), and finally a sigmoid to output a value in $[0, 1]$. I use Binary Cross-Entropy loss.
- **Sentence Similarity** A linear layer, with hidden size half the BERT hidden size, GELU activation, dropout ($p = 0.3$), and finally a sigmoid to output a value in $[0, 1]$. I use Mean-Squared Error loss for this task.

### 4.2 Dynamic Batch Sizes and Varying Epochs

To further optimize the training process, I implemented a dynamic batch size adjustment strategy first introduced by (Devarakonda et al., 2017). I monitored the loss for each task during training and compared the increase in loss between the tasks. If a particular task exhibited a larger increase in loss compared to the others, I scaled up its batch size by a factor of 1.5 for the subsequent epoch. This approach allowed me to allocate more computational resources to the tasks that were experiencing greater difficulty in learning, potentially improving their performance.

Furthermore, I investigated the impact of increasing the number of epochs on the model's performance. I conducted experiments with both 10 and 20 epochs to observe how the models behaved over longer training periods. This allowed me to assess whether the models continued to improve with additional training or if they reached a plateau.

### 4.3 Prediction Head Inputs

In the paraphrase detection and semantic textual similarity tasks, I investigated different input representations. Instead of using the embeddings of individual sentences, I experimented with concatenating the embeddings of both sentences in a pair. Additionally, I explored the use of Cosine Similarity as a measure of similarity between the sentence embeddings. These variations aimed to capture the relationship between the sentences more effectively.

### 4.4 Loss Weighting

To address the class imbalance problem in the sentiment analysis task, I employed Loss Weighting in the Cross Entropy loss function. By incorporating the inverse frequency of the class labels, I aimed to prevent the model from completely ignoring the minority class during training. This approach sought to ensure that the model paid sufficient attention to all classes, even those with fewer instances.

### 4.5 Ensemble Methods

Finally, I explored the use of an ensemble method, specifically the Random Forest classifier, for the paraphrase detection and semantic textual similarity tasks. Instead of relying solely on the BERT embeddings, I concatenated the embeddings of both sentences in a pair and used this representation as input to the Random Forest classifier. This approach aimed to leverage the strengths of both BERT embeddings and the Random Forest algorithm to enhance the performance of these tasks.

### 4.6 Challenges and Limitations

In conducting my experiments, I had trouble securing access to a T4 GPU in the zones I had my Google Cloud Provider server in. Beyond that, my iteration speed during some experiments was quite slow as the Pytorch installed in the conda environment was not compiled with the GCP installed version of CUDA. This limited the number of different configurations of experiments I was able to successfully run for all 10 epochs. Some of the below reported results are for fewer than 10 epochs, as I was attempting to conserve my GCP credits.

## 5 Experiments

Below I describe my experimental setup, evaluation method and results.

### 5.1 Data

Since I am reporting on my results for the Default Final Project, the data I used to conduct my experiments were the provided datasets from the Stanford Sentiment Treebank, the Quora Paraphrase dataset, and the Semantic Textual Similarity dataset.

### 5.2 Evaluation method

- **Sentiment Classification:** To assess the result of my experiments on this task, I use accuracy, computed as the percentage of predicted predictions matching the true labels.
- **Quora Paraphrase:** I similarly use accuracy as my metric on this binary classification problem.
- **Sentence Similarity:** I use Pearson correlation with the ground truth labels to assess the results of my regression prediction head.

### 5.3 Experimental details

Below, I describe the variety of configurations I used in my results and analyses on the provided task dev sets.

### 5.4 Results

Results from my experiments are in Table 1. Due to constraints in training (i.e., difficulty in getting access to GPUs during peak hours) and long training times (especially with finetuning) I was not always able to try running the training loop with only one of the above experiments to isolate the single effect of changing just one attribute of my model. Instead I had to combine multiple changes in one experiment, so I am not able to fully isolate the effect of just a single change. I plan on doing so, however, for the final report. See below for results in Table 1. In Table 2 I describe my hyperparameters configurations for the ensemble models prediction models.

| Configuration | Dev Sentiment Acc | Dev Paraphrase Acc | Dev STS Corr |
|---|---|---|---|
| Pre-training (LR: 1e-3) | 0.395 | 0.650 | 0.194 |
| Pre-training on Dynamic Batch Sizes | 0.334 | 0.625 | 0.160 |
| Fine-tuning (LR: 1e-5) | 0.415 | 0.747 | **0.454** |
| Fine-tuning, 15 Epochs, 384 Hidden, Cosine | 0.380 | **0.748** | 0.370 |
| RF Pre-Finetuning, 128 Hidden, Decay=0.01 | **0.468** | 0.679 | 0.349 |
| RF Post-Finetuning, 128 Hidden, Decay=0.01 | **0.468** | 0.672 | 0.403 |

Table 1: Results of different training configurations. Best results in **bold**.

| Type | Num Estimators | Max Depth | Max Features/Split |
|---|---|---|---|
| RF Classifier | 100 | 10 | Square Root |
| RF Classifier | 100 | 10 | Square Root |

Table 2: Random Forest classifier and regressor hyperparameters.

# 6 Analysis

## 6.1 Prediction Head Hyperparameters

The most significant improvements for model configurations that did not use ensemble methods (the first four rows of Table 1) occurred in the jump from using the pre-trained (e.g., frozen) BERT model weights, to finetuning the BERT weights while training on all three tasks.

Specifically, while all tasks saw significant improvements, I saw the largest improvement in the Semantic Textual Similarity correlation results. I believe further investigation is needed to understand why this task in particular saw large gains, whereas the other tasks achieved more modest improvements. One hypothesis could be that semantic similarity is a much harder task to assess with naive BERT embeddings that created from a model trained on masked token prediction, and that finetuning the BERT model parameters helped calibrate the embeddings to be able to more clearly distinguish semantic meaning in sentences.

In addition, I had also tried initiating my training loop by only sampling $10\%$ of the available data for each task, during each epoch. Then, I computed the loss on subsequent epochs. The task that saw the small improvement in loss would have the samples it is trained on be scaled up by $1.5$ each time.

I tried this as a way to balance the distribution of data between tasks, where the paraphrase dataset was over an order of magnitude larger than other tasks. During multitask training, I was concerned that the parameters would see more significant updates towards minimizing the loss function of paraphrase detection (Mean-Squared Error) versus other tasks.

However, this actually led to the decline in results even after 10 epochs, likely due to the fact that even after 10 epochs the model may not have seen $100\%$ of the data across all tasks.

Lastly, I computed the Cosine Similarity of the two embeddings and attempted to use that as input to a simple linear layer for the Paraphrase and STS tasks, but did not see any marked improvements there.

## 6.2 Ensemble Methods

The last two rows of Table 1 correspond to runs in which I used an ensemble method, namely a Random Forest Classifier for the Paraphrase task and a Random Forest Regressor for the Semantic Textual Similarity Task with the following difference: I used the native BERT embeddings as input to the Random Forest for the two tasks and report these results as "RF Pre-Finetuning".

Then, I fine-tune the BERT layers using the Sentiment Classification task for 10 epochs, and feed in the new BERT embeddings to train new Random Forest ensembles and report the subsequent results. I describe these results as "RF Post-Finetuning".

I also add a regularization term to the AdamW optimizer, introduced by Loshchilov and Hutter (2019), of $0.01$ for weight decay . In our custom implementation of AdamW, the default regularization term

is $0$. However, the PyTorch implementation of AdamW defaults weight decay to $0.01$, which I wanted to test as well.

#### 6.2.1 Hypothesis and Discussion

The guiding hypothesis I had here was that fine-tuning the BERT layers on the Sentiment data would enable richer embeddings, and potentially lead to higher performance on the downstream tasks.

In addition, I wanted to combine classical machine learning techniques, like the Random Forest, with innovations in the recent decade, namely high-dimension representations of data generated by deep neural networks.

Lastly, since training the Random Forest classifier and regressor on the concatenated embeddings ($d = 1536$) was a very computationally expensive operation on the limited CPU provided by GCP, I set the max depth to $10$ and the maximum number of input features to be floor($\sqrt{d}$) $= 39$ to reduce the computational burden and model training time. This could also potentially help prevent the Random Forest from overfitting on the training set, as each split for each estimator now has access to a small fraction of all possible input features.

#### 6.2.2 Analysis of Performance Using Ensemble Methods

The results surprised me: Sentiment classification performance improved significantly, which I had expected as the BERT embeddings were now fine tuned solely to improve a single downstream task, both the paraphrase accuracy and STS correlation were less than just naive finetuning BERT embeddings while training on all three tasks together.

That being said, the Random Forest's performance on STS correlation did improve significantly as well as a result of finetuning the BERT model embeddings on a different task, which was still a positive result.

Beyond that, I suspect that limiting the each estimator's split to only consider $39$ out of $1536$ features in the input space was a constraint that limited the performance of the Random Forest. This may happen if there are only a few key dimensions in the representation space that matter significantly. If so, then it's likely that the splits would not pick up on these dimensions and would instead be splitting on noisy features that have little bearing on the downstream task.

## 7 Conclusion

While I saw many of my attempted experiments fail at improving downstream task performance across all tasks, and in many cases not even improve performance on a single task, I found all the experiments to be highly educational and informative for my own learning.

In particular, I am eager to continue exploring several other potentially promising avenues of research. Specifically I would like to implement masked token prediction, and fine tune the BERT layers on an additional text corpus. In addition, I would also like to first fine tune the model on the on the IMDB dataset before using it on the sentiment analysis task, as these two datasets are approximately in the same "task space" and hence could see mutually beneficial results when trained jointly.

Beyond that, since finetuning was an expensive operation from the perspective of both time and GCP credits used, I did not try finetuning results more than beyond 15 epochs. This finetuning rune was also confounded by the introduction of solely using the Cosine Similarity between embeddings, so the decreases in performance may not be attributable to the larger epoch size.

Instead, I'd like to try exploring feeding the Cosine Similarity as a single additional feature into the MLP prediction heads ($d = 1536 + 1$) rather than using Cosine Similarity by itself. I believe that, at worst, this would lead to no net change in performance, and at best could provide a "shortcut" for the model to more quickly learn this whether this higher order metric could be useful in downstream tasks.

Lastly, I'd like to just briefly share that this report did not cover the extensive learnings and experiments I conducted with using Google Cloud. This project helped me become more familiar with tools like tmux, the gcloud CLI, and also Weights and Biases as an MLOps platform to track training results. While I had initial immense difficulties in configuring my Google Cloud server to allow me to use the

GPU, now that I'm done with this project I feel very grateful to have encountered these obstacles at this point rather than on the job when the situation may be higher stakes.

## References


Aditya Devarakonda, Maxim Naumov, and Michael Garland. 2017. Adabatch: Adaptive batch sizes for training deep neural networks. *CoRR*, abs/1712.02029.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Samuel Fernando and Mark Stevenson. 2009. A semantic similarity approach to paraphrase detection. *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.