Pretraining, Feature Engineering, and More:
Fine-tuning BERT for Multitask Capabilities

Stanford CS224N Default Project
Mentor: Drew Kaul

Dawson Sproul  
Department of Computer Science  
Stanford University  
sproulda@stanford.edu

Hanna Lee  
Department of Computer Science  
Stanford University  
hylee719@stanford.edu

Abstract

The development of large pre-trained language models, such as BERT, has led to significant advancements in Natural Language Processing (NLP) tasks such as Sentiment Analysis (SA), Paraphrase Detection (PD), and Semantic Textual Similarity (STS). However, these tasks are often addressed independently, resulting in models that are task-specific and not efficient in multitasking scenarios. In this project, we propose three extensions to the BERT model to create a multitask model that can handle the tasks of SA, PD, and STS simultaneously. These extensions include incorporating a cosine embedding loss for STS, additional pre-training of Stanford Sentiment Treebank (SST) data with a Masked Language Modeling (MLM) objective, and adding Part-of-Speech (POS) tags to the word embeddings. Our experimental results demonstrate that the proposed extensions slightly improve the multitasking ability of the BERT model, leading to improved performance on SA, PD, and STS tasks.

1 Introduction

Natural Language Processing (NLP) has seen significant advancements in recent years, thanks to the development of large pre-trained language models such as BERT. These models have shown remarkable performance in various NLP tasks, including Sentiment Analysis (SA), Paraphrase Detection (PD), and Semantic Textual Similarity (STS). However, these tasks are often addressed independently, resulting in models that are task-specific and not efficient in multitasking scenarios. To address this issue, researchers have explored multitask learning approaches, which enable models to perform multiple tasks simultaneously by sharing some of the parameters. Multitask learning has been shown to improve model performance and reduce the need for task-specific models, making it a promising approach for NLP. The investigation into which model extensions/modifications exactly can best allow for this is an area for exploration.

In this project, we aim to explore the extensions to the BERT model to create a multitask model that can handle the tasks of SA, PD, and STS simultaneously. Specifically, we propose three extensions to the base model to improve its performance on these tasks.

The first extension involves incorporating a cosine embedding loss for STS. The cosine embedding loss is a similarity measure that compares the embeddings of two sentences and calculates the cosine distance between them. By adding this loss to the model, we aim to improve its ability to capture the semantic similarity between sentences.

The second extension involves additional pre-training of Stanford Sentiment Treebank (SST) data with a Masked Language Modeling (MLM) objective. Pre-training with an MLM objective involves masking out some of the words in a sentence and training the model to predict them based on the context. Since the BERT model was trained on a general corpus that is different from our task-based datasets, we aim to enhance the model’s ability to perform on our three tasks by pre-training on the
SST data with an MLM objective. This approach has been shown to be effective in improving models’ ability to capture the syntactic and semantic information in text.

The third extension involves adding Part-of-Speech (POS) tags to the word embeddings. POS tags provide additional syntactic information to the model, enabling it to better understand the structure of the sentence. When approaching sentence comparison tasks specifically, the context of a word is crucial in determining its meaning. For instance, the word *hand* can be both a noun and a verb; the knowledge of which role a word plays in a given sentence allows the model to better understand the nuances behind each word and phrase for better prediction. By incorporating POS tags into the model, we aim to improve its performance on tasks that require a deeper understanding of sentence structure, such as STS.

Overall, our goal is to demonstrate that the proposed extensions can improve the multitasking ability of the BERT model, leading to improved performance on SA, PD, and STS tasks. In the following sections, we describe these extensions in detail and evaluate their abilities to enhance multitasking in various combinations.

## 2 Related Work

### 2.1 Cosine Embedding Loss (CL)

Cosine loss utilizes cosine similarity (scale of -1 to 1) to determine the degree of similarity between inputs. Within deep learning, this entails mapping the vector representations of two inputs into high-dimensional space and computing the cosine angle between them, such that those with a score near −1 are highly dissimilar and scores near 1 are similar. Sentence comparison tasks lend themselves to using loss functions such as cosine loss to find a degree of similarity between a sentence pair. Ideally, our embeddings would minimize and maximize distances between similar and dissimilar vectors, respectively. The usage of cosine loss within a Siamese BERT model, for example, outperformed BERT on multiple tasks, one of which being STS (Reimers and Gurevych, 2019).

### 2.2 Masked Language Modeling

The method of masked language modeling for a pre-training objective is described in the original BERT model (Devlin et al., 2018) and is intended to extract deep bidirectional representations, unlike previous work, such as ELMo (Matthew Peters and Zettlemoyer, 2018) and OpenAI GPT (Alec Radford and Sutskever, 2018), which use unidirectional language models. The BERT authors argue that for token-level tasks such as question answering, it is crucial to incorporate context from both directions and not just from left to right. The MLM objective enables the representation to combine the left and the right context, which allows pre-training of a deep bidirectional Transformer. Results showed that among the five tasks tested, using MLM versus only a left-to-right language model yielded higher accuracies for all five tasks (Devlin et al., 2018).

A novel modification to BERT, RoBERTa, explored the effects of additional training data on the BERT model for enhanced performance. When utilizing an identical MLM objective and model architecture but modifying the extent and time of training, the authors found increased performance across all nine single tasks which they analyzed, SST and STS being two (Liu et al., 2019). RoBERTa shows that MLM pre-training is beneficial and motivates its application in tandem with other extensions for downstream task performance.

### 2.3 Part-of-speech tagging

Sentence pair tasks such as STS have been shown to benefit from the addition of part of speech tags to BERT output embeddings. He et al. (2023) discovered that feature engineering in this regard leads to an average F1 increase of nearly 10% over alternative BERT model approaches.
3 Approach

3.1 Baseline

We began by creating a miniature BERT model that closely follows the original BERT architecture (Devlin et al., 2018). We implemented this model from the project’s skeleton code. Our contributions were implementations of the multi-head self-attention layer, the full transformer layer, a sentiment classifier model, and AdamW stochastic optimization (Kingma and Ba, 2017; Vaswani et al., 2017; CS224N, 2023).

Our baseline model was a BERT model with pre-trained weights on the tasks of Masked Language Modeling and Next Sentence Prediction. We implemented this as a single-task model that utilizes the SST data for SA, PD, and STS tasks. It involves the addition of a general dropout, linear layer, and cosine similarity function. The model loss was solely influenced by cross-entropy loss on the SA task, though cosine similarity was used as a loss metric for model evaluation on PD and STS.

3.2 Model Architecture

We expanded upon the BERT architecture by adding layers for each of our three downstream tasks. Figure 1 provides visualizations of the additional layers for each task that yield logits from embeddings. This architecture was utilized throughout all of our extensions and corresponding experiments. Additionally, we implemented a round-robin training approach for iterating through the datasets for fine-tuning. This involved zipping all three datasets together and iteratively taking a batch from each of the datasets. Our loss calculation was an aggregation of loss per task: cross-entropy loss, binary cross-entropy loss, and cosine loss were utilized as metrics for the SA, PD, and STS tasks respectively.

3.3 Extensions

3.3.1 Cosine Embedding Loss

We implemented a version of cosine loss when fine-tuning the SemEval dataset and STS tasks (1). Our approach begins by feeding each sentence through our BERT to attain the embeddings. We then feed each of the pooler outputs through our semantic layers which include a dropout layer and a linear layer to obtain logits for each and calculate their associated cosine similarity. We approached scaling these losses in two ways: 1) scaling each factor to a true representative scale of 0 to 5 and 2) scaling by a factor of 5. Cosine similarity scores were then utilized alongside labels within a mean-squared error loss to produce our final prediction logits for STS.

\[ \text{cossim} = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \times ||\mathbf{v}||} \]  

(1)

Figure 1: Representations of our model’s architecture in terms of layers for each task. A: sequential layers for sentiment analysis utilized to produce (batch_size, sentiment classes) logits. B: sequential layers for paraphrase detection utilized to produce (batch_size, 2) logits. C: sequential layers for semantic textual similarity task to produce (batch_size, hidden_size) embeddings.
3.3.2 Masked Language Modeling

Our BERT model was trained on a general dataset that is not entirely representative of the datasets utilized in our fine-tuning. With our objective of creating robust embeddings for downstream tasks, we investigated the task of additional pre-training on our model. We utilized the SST training dataset to pre-train on the masked language modeling objective, as the BERT model does for the general data. Our approach follows that outlined in the original BERT model (Devlin et al., 2018). We masked 15% of the word tokens – 80% of masking instances were replaced with the [MASK] token, 10% were replaced with a random token, and the remaining were unchanged. We created a new MLMDataset class that contains this masking logic and utilize it in our multitask classifier pre-training function on our loaded SST data. We have a separate forward function for the pre-training that examines the last hidden state and applies a linear layer, where the output is the size of the BERT vocab (30522). Once we have our predictions, we perform cross-entropy loss between the true labels given by the MLMDataset class and the predictions from the MLM head to pre-train for 10 epochs. Once the pre-training is complete, we perform our multitask training on the current state of the model.

3.3.3 Part of speech Features

BERT embeddings capture semantic relationships well without part of speech tags. Still, POS tags provide crucial insight into word positioning nuances that could further inform the classification of sentiment. To incorporate these tags and investigate their additional semantic insight, we implemented a part of speech feature addition to our BERT model output embeddings.

The process began by collecting the word embeddings for a given set of input_ids tokens. To determine a representation for the parts of speech present in a given sentence, we then created a count vector to determine the occurrences of parts of speech in the sentence. This process involved the usage of the nltk package's averaged perceptron tagger and universal tagset data (Bird et al., 2009). Each list of input_ids was converted into its corresponding sentence, fed through the tagger, and then translated into one of twelve universal tag representations (i.e. 'NOUN', 'VERB', 'ADJ', etc.) and the count vector was updated as follows at the respective index of each POS. These count vectors were then concatenated with the pooler output from the sentence embedding and returned as the representative embedding for a given input.

4 Experiments

4.1 Data

Our project utilizes the datasets described in the handout: Stanford Sentiment Treebank (SST) Socher et al. (2013), Quora (Csernai 2017), and SemEval (Agirre et al., 2013).

SST data contains 11,855 single sentences from movie reviews, which were parsed to include 215,154 unique phrases labeled by humans as either negative, somewhat negative, neutral, somewhat positive, or positive and is utilized for sentiment analysis the data was split 8,544/1,101/2,210 for train/dev/test sets. Quora is used for testing paraphrase detection, and it contains 400,000 question pairs and labels indicating whether two queries are paraphrases of each other. The data was separated into 141,506/20,215/40,431 size sets for train/dev/test sets. SemEval is used for testing STS and contains 8,628 sentence pairs labeled on a continuous scale of 0-5 for the degree of similarity. The data was separated into 6,401/864/1,726 for train/dev/test sets.

4.2 Evaluation method

Evaluation of SA and PD tasks utilizes accuracy metrics for classifying sentences according to the five labels for SA and as binary labels for PD. For STS evaluation, we utilized Pearson correlations of the true labels versus our predicted labels (Agirre et al., 2013).

4.3 Experimental Details

We began by implementing our cosine loss approach for the STS task to extend upon our baseline model, utilizing the first metric from 3.3.1. From there, we tested our additional extensions on the model one by one. All models utilized AdamW optimization, similar to the baseline.
We implemented our dependency parser tools to add part of speech tags to our model on top of cosine loss to generate a \textit{BERT} + \textit{CL} + \textit{POS} model. We fine-tuned this model for a total of 10 epochs with a learning rate of 0.00001 and a batch size of eight. We also attempted fine-tuning the number of epochs for this model, changing it to 20, but this hyperparameter change did not yield any model improvements, so we did not continue with it. The overall time to completion for this model was around 1.5 hours.

Our next approach was to implement the MLM modeling on top of the cosine loss to give us a \textit{BERT} + \textit{CL} + \textit{MLM} model. We performed the pre-training on exclusively the train and dev SST data, which contains 8,544 and 1,101 sentences, respectively. We finetuned this model with a learning rate of 0.0001 and ran this for 10 epochs with a batch size of eight. Pre-training took about 30 minutes, and the multitask training took about 1.5 hours to complete. We found that during the pre-training process, the loss for MLM gradually decreased from about 30 to 26 over the 10 epochs.

After seeing the results of the above experiments, we shifted to testing out our second approach to cosine loss by using the latter scaling method mentioned in 3.3.1. We utilized the same existing extensions for this model to create \textit{BERT} + \textit{CL}** + \textit{POS} and \textit{BERT} + \textit{CL}** + \textit{MLM}. We finetuned both of these models with learning rates of 0.0001 for ten epochs with a batch size of eight. The training time for these models was approximately 1.5 hours each.

### 4.4 Results

The results of all of our model experiments are outlined in Table 1 and accuracy/Pearson correlation plots are shown in Figure 2. Our baseline model reported 0.523, 0.412, and 0.262 accuracy/Pearson correlation metrics per task for an average of 0.399.

#### 4.4.1 \textit{BERT} + \textit{CL} + \textit{POS} and \textit{BERT} + \textit{CL} + \textit{MLM}

Our \textit{BERT}+\textit{CL}+\textit{POS} model performed worse than our baseline model on SA and STS, reporting accuracies and Pearson correlations of 0.296 and 0.111, respectively, compared to the baseline’s 0.523 and 0.262. This model did lead to improved performance on PD, increasing our accuracy metric by 0.113. The average accuracy across tasks was lower than the baseline and our test accuracy was 0.261.

Our \textit{BERT}+\textit{CL}+\textit{MLM} model performed worse than our baseline model and \textit{BERT}+\textit{CL}+\textit{POS} model on SA and STS, reporting accuracies and Pearson correlations of 0.296 and -0.014, respectively, compared to the baseline’s 0.523 and 0.262. This model did lead to improved good performance on PD, increasing our accuracy metric by 0.201. The average accuracy across tasks, 0.287, was lower than the baseline and our test accuracy was 0.261.

These results were much worse than we had expected. Given the literature and existing models that utilize similar tactics individually, we expected our accuracy metrics to improve across all tasks; the MLM and POS extensions should enhance the word embedding robustness for downstream task performance ability, but this is not the case given our results. This data served as a notion that perhaps our method for cosine loss was not scaling correctly for our STS task, as we saw the largest performance drop for that task.

#### 4.4.2 \textit{BERT} + \textit{CL}** + \textit{POS} and \textit{BERT} + \textit{CL}** + \textit{MLM}

Our \textit{BERT}+\textit{CL}**+\textit{POS} model performed the best of all our experiments and outperformed the baseline on PD. We saw accuracy metrics of 0.362 and 0.665 for SA and PD and a Pearson correlation of 0.216 for STS, giving us an average dev accuracy of 0.414 and a test accuracy of 0.407. This was the largest improvement above baseline PD (by 0.253). The model performed worse on SA and STS than the baseline.

Our \textit{BERT}+\textit{CL}**+\textit{MLM} model saw accuracies of 0.322, 0.665, and 0.226 across SA, PD, and STS respectively for an average dev accuracy of 0.404 and test accuracy of 0.388.

Similar to our previous approaches, these results were also not what we had expected. While they outperformed our baseline models on average, their per-task accuracies were far from our hopes. Paraphrase accuracy did increase as we had expected, however, we expected the SA and STS tasks to show at least some increase from baseline approaches. Additional pre-training on an MLM objective
as well as POS tags should allow for better prediction abilities for these tasks which makes us believe that perhaps there was an implementation issue preventing us from seeing accuracy increases.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>SA</th>
<th>PD</th>
<th>STS*</th>
<th>Avg. Dev Acc</th>
<th>Avg. Test Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT Baseline</td>
<td>0.523</td>
<td>0.412</td>
<td>0.262</td>
<td>0.399</td>
<td>-</td>
</tr>
<tr>
<td>BERT+CL+MLM</td>
<td>0.262</td>
<td>0.613</td>
<td>0.014</td>
<td>0.287</td>
<td>0.261</td>
</tr>
<tr>
<td>BERT+CL+POS</td>
<td>0.296</td>
<td>0.525</td>
<td>0.111</td>
<td>0.310</td>
<td>0.261</td>
</tr>
<tr>
<td>BERT+CL++MLM</td>
<td>0.322</td>
<td>0.665</td>
<td>0.226</td>
<td>0.404</td>
<td>0.388</td>
</tr>
<tr>
<td>BERT+CL++POS</td>
<td>0.362</td>
<td>0.665</td>
<td>0.216</td>
<td>0.414</td>
<td>0.407</td>
</tr>
</tbody>
</table>

Table 1: Dev set and test set accuracies (average and per-task) on baseline and extension models. *Metrics reported are Pearson correlations. **Cosine loss implemented as \((\text{logit}) \times 5\) rather than \(0.5 \times (\text{logit} + 1) \times 5\). SA=Sentiment Analysis, PD=Paraphrase Detection, STS=Semantic Textual Similarity, CL=Cosine Loss, POS=Part-of-speech, MLM=Masked Language Modeling.

5 Analysis

The accuracies across all models show that the baseline BERT model has the highest SA performance by far. However, this may be because the baseline model is only pre-trained on SST data and does not include multitask learning. It would make sense that it is the most optimized for SA performance.

Generally, our extension models seem to succeed when we scale for cosine similarity values by solely multiplying by five: from a -1 to 1 scale to a -5 to 5 scale. This is likely because our first implementation treated the loss of any sentence pairs that result in a cosine similarity value from -1 to 0 as equally dissimilar, though our scale from 0 to 5 indicates that this should not be the case. Therefore, by scaling this over a larger range, we can extrapolate the relations and receive comparable difference-scaled logits to pass through our MSE function for our final cosine embedding loss.
The overall negligible improvements on multitask ability when using cosine loss for STS could be explained by the nature of BERT embeddings. The representations of sentences produced by BERT may capture the semantic relationships well enough that computing cosine similarity is not needed, meaning one could forego this step and proceed directly with a mean-squared error loss function instead.

The limited improvement when using MLM could be a result of only using the SST data during the prediction pre-training objective. We hypothesize that MLM may have been more effective if we had more and a greater diversity of data. Pre-training only on the SST data may not have been enough to capture diverse language patterns or nuances in other datasets. We also consider that the impact of adding MLM to a BERT model can be highly dependent on the specifics of the task and the data being used. Therefore, more investigation into our MLM approach and its interaction with our three specific tasks is necessary to identify points of possible improvement.

Intuitively, we hypothesized that the concatenation of part of speech tags to our word embeddings would provide further semantic/contextual information about an input that could assist the machine’s learning throughout fine-tuning and, according to our results, this is the case to a large extent. It is possible that our approach to feature engineering was unnecessary, as our fine-tuning tasks require semantic information about the input, but potentially not more than the already insightful BERT embeddings provide us; would our task have been next sentence prediction, this may have been different. End-to-end engineering in tasks like ours is more appropriate to save computation time – avoiding the individual tagging of all inputs – especially when our reported accuracy increases are not substantial. It is also possible that our model does not perform well with occurrence vectors appended to pooler output. An implementation that utilizes the last hidden state from the embeddings and concatenates POS tags then, or prior to the feed-forward layer would be interesting to attempt.

Figure 2 provides valuable insight into the potential missteps of our model. Analyzing the accuracy metrics across the epochs, we see that no model truly improves its accuracy in a seemingly linear fashion, but rather experiences oscillations from regression to improvement. Our model may fail because of the multitask loss implementation during our round-robin fine-tuning approach. Our losses might be weighted inappropriately when aggregated, such that some have a higher effect on the total loss than others despite differences in dataset sizes and number of total batches. Techniques like gradient surgery could sanity-check whether the losses are reported correctly.

Overall, our results indicate that the extensions we proposed in tandem lead to slight performance increases for SA, PD, and STS tasks. However, the magnitude of this increase was quite lower than we had expected. Given the previous research these techniques to extend BERT, we expected accuracies in the range of 0.6 or 0.7, especially for sentence pair tasks like SA and STS. The data, along with our notes above, indicate that further investigation into the direct implementation of these extensions is necessary. These results could indicate some misstep in our direct model implementation that is not allowing for proper updating, propagation, or learning for our downstream tasks.

6 Conclusion

Our investigation into BERT model extensions for enhanced multitask ability shows that the usage of cosine embedding loss in conjunction with MLM and POS tagging leads to slight performance improvements compared to an SST single-task BERT model.

Our data showcase the combination of using cosine loss for the STS task and POS tagging works to create the best-performing model, with an improvement of over 25 percent in the PD task over the baseline model. Notably, both MLM and POS extensions increased PD accuracy by over 20 percent.

Comparing our two methods of cosine scaling and their respective model accuracies, we learned that the scaling method for cosine loss to match that of the data labels can significantly impact results. Both of our model iterations that used $5 \times (\text{logit})$ scaling rather than $0.5 \times (\text{logit} + 1) \times 5$ increased test accuracy by over 10 percent. Still, despite our increases in multitask accuracy, performance in the STS task was slightly worse than the baseline across all of our models, which prompts us to take a deeper look into our cosine loss implementation and possible single-task fine-tuning STS.

We discovered that the inclusion of MLM and POS tagging does not lead to significant improvements over the baseline that prove these extensions’ efficacy for multitask learning. However, these
extensions have enhanced downstream task performance in prior research, so we are hesitant to draw definitive conclusions.

There exist limitations to our work. One is the certainty for why we saw increases in accuracy for some tasks (e.g. PD), while we saw decreases in others (especially STS) for some implementations. Out of our three primary extensions, MLM and POS were both meant to ameliorate all tasks, and cosine loss was the only addition created for one task specifically; SST improvement was expected but not observed. Other limitations involve the small data size utilized for MLM pre-training and our uncertainty of loss calculation approach throughout our round-robin style fine-tuning.

In the future, we plan to determine if the small magnitude of performance increase is due to our specific implementation or the need for further extension. We would like to explore more MLM pre-training on the Quora and SemEval datasets, test out alternative optimization algorithms, as well as implement gradient surgery to determine if gradient projections from one task onto another could improve loss calculation and performance.

References


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

CS224N. 2023. Cs 224n: Default final project: minbert and downstream tasks

Kornél Csernai. 2017. First quora dataset release: Question pairs


