Optimizing minBert via Cosine Similarity and Negative Sampling

Stanford CS224N Default Project

Neha Vinjapuri and Ananya Vasireddy
Department of Computer Science
Stanford University
Mentor: Cheng Chang
nehavin@stanford.edu, ananyasv@stanford.edu

Abstract

In recent years, natural language models that are able to generalize to multiple tasks have become increasingly significant. In this report, we investigate several different methods to improve the baseline minBERT model on three different tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. As a result of our experiments, we found that implementing cosine similarity, as examined in (Reimers and Gurevych, 2019), as well as using an unconventional method of negative sampling, inspired by (Henderson et al., 2017) and (IamMrX, 2022), yielded the best improvement on the three tasks. Our final multitask model achieved an overall score of 0.655.

1 Key Information to include

- External collaborators (if you have any): N/A
- External mentor (if you have any): N/A
- Sharing project: No

2 Introduction

With the recent explosion of large language models like OpenAI's ChatGPT, the field of natural language processing has seen an increased demand for models that are able to generalize to different tasks. These models can respond to a variety of prompts, answering questions and producing human-like language in seconds. However, the predecessor of many of these models, BERT (Bidirectional Encoder Representations from Transformers), can also be used to generalize to diverse tasks. In this report, we investigate BERT's multitask potential by presenting an augmented version of minBert, optimized to perform well on several different natural language processing tasks. Specifically, these tasks are 1) sentiment analysis, 2) paraphrase detection, and 3) semantic textual similarity. By incorporating cosine similarity and an unconventional method of negative sampling into our model, we successfully finetuned our baseline model to perform better on these three downstream tasks.

3 Related Work

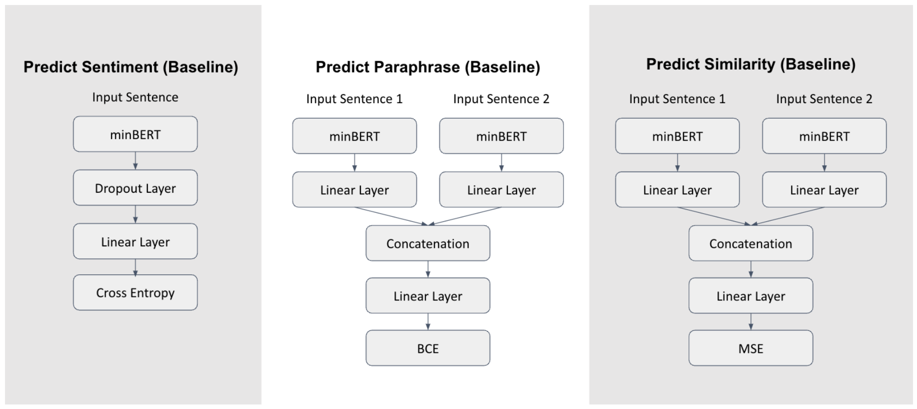
Two papers inspired our approach: "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks", which involves cosine similarity, and "Efficient natural language response suggestion for smart reply", which focuses on negative sampling and multiple negatives ranking loss learning.

In the first paper, the authors introduce a Siamese neural network architecture and use cosine similarity as a measure for the semantic textual similarity task. Since this is one of the tasks we are attempting to improve our model's performance on, we decided to use this cosine similarity measure in our own model when evaluating the similarity between two embeddings.

In the second paper, the authors hope to address the problem of suggesting responses in human conversations. To do this, a dot-product multi-loss scoring model, which involves minimizing the approximated mean negative log probability of the data, was utilized. Inspired by this paper, we decided to build on this approach and incorporate negative samples into our own model - but without using the dot-product multi-loss scoring model. We saw that this had worked for Stable Diffusion, a deep learning text-to-image model (IamMrX, 2022), which improved its image generation by using negative samples (the model specifically made its images very different from corresponding negative prompts). In the context of our project, this is an important exploration because it will illustrate whether simply introducing negative samples, without the corresponding loss function, also improves performance in a wholly natural language processing context.

4 Approach

Our project leverages the baseline minBERT model, as outlined in the Default Project handout. The baseline uses a transformer architecture featuring multi-headed attention mechanisms and a feed-forward network. Each task takes calculates embeddings from our minBERT model and uses linear layers to output prediction values.



Our goal was to refine the model's ability to capture intricate semantic relationships within text data. To advance the performance of our model, we iteratively proposed five new approaches to improve our baseline, including some of our own innovative methodologies.

1. Expanding to All 3 Datasets. Our first improvement to our baseline was expanding our model to train on three datasets over only one, the Stanford Sentiment Treebank (SST) dataset. We included the Quora dataset and SemEval STS Benchmark dataset, which would dramatically increase the quality of our embeddings for the paraphrase detection and semantic textual similarity tasks. For each of the datasets we trained on, we chose to calculate a loss that would best fit the type of data. In the equations below, y_i is the truth value, and $\hat{y}$ is the predicted value.

Semantic Analysis: Cross-Entropy (CE) Loss
Because we output a probability score for each category (1-5) for the given sentence, CE loss will penalize the model when the probabilty of the wrong category is higher than the correct one.

$$CE(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Paraphrase Detection: Binary Cross-Entropy (BCE) Loss
Because we output a binary score (0 or 1) for each sentence pair, BCE loss will penalize the model

when the probabilty of the wrong category is higher than the correct one.

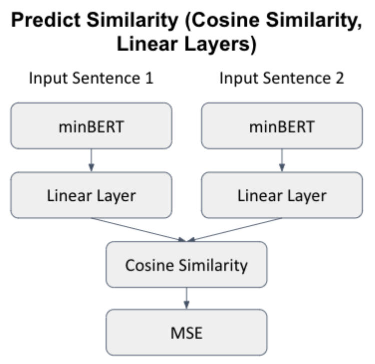
$$BCE(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

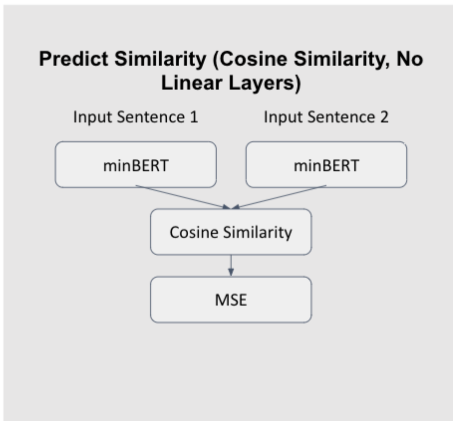
Semantic Textual Similarity: Mean-Squared Error (MSE) Loss
Because we output a float value for similarity between sentence pairs, MSE loss will calculate loss by measuring the distance from the actual score to the predicted value.

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

To decide if our model improved after an epoch, we would update our best model if there was improvement in any single task from the previous state of the model.

2. Cosine Similarity. To improve our semantic textual similarity prediction function, we decided to utilize cosine similarity, inspired by its use in (Reimers and Gurevych, 2019). This extension also aligned well to our decision to use MSE loss, as the researchers in Reimers and Gurevych (2019) used MSE loss with their cosine similarity implementations as well. We tried this via two methods: 1) appending it to our baseline model with linear layers, and 2) only having cosine similarity. We hypothesized that the second method would reduce the required compute needed as there are less parameters and it would better measure the similarity of the between two embeddings, which is the goal of our task.





3. Alternative Optimizer. To improve our model's overall performance, we decided to test a new optimizer, Rectified Adam (RAdam), instead of our implemented AdamW optimizer. RAdam uses the new r_t term to correct the variance, while AdamW uses weight decay to stabilize the training process. We chose this as it doesn't require tuning of the weight decay hyperparameter, making it more efficient to use in practice.

$$r_t = \rho_t - 2\beta_2^t + 1$$
$$\alpha_t = \eta\sqrt{\frac{1 - \beta_2^t}{1 - \beta_1^t}} \cdot \min\left(\frac{r_t}{\sqrt{v_{t-1}}}, 0.001\right)$$
$$\theta_{t+1} = \theta_t - \alpha_t \cdot g_t$$

4. Negative Sampling (Original). Because we are calculating the similarity between sentences using the STS dataset, we brainstormed methods to improve the accuracy scores for the semantic textual similarity task. We were inspired by two applications of introducing negative samples. First, Stable Diffusion (IamMrX, 2022) uses negative prompts in order to improve the quality of their images by making them very different from their negative prompts. In the 'Efficient Natural

Language Response Suggestion for Smart Reply' paper (Henderson et al., 2017), the authors used a set of K possible responses to approximate $P(y|x)$, where y represents possible responses and x represents a given input email. This set of K possible responses was split into 1 correct response and $K - 1$ random negatives. They utilized negative ranking loss, which maximized the distance between the input email and negative responses and minimized the distance between the input email and positive response. (Henderson et al., 2017).

In our case, we are trying to penalize pairings of sentences that are not related. Therefore in our approach, for every sample in the dataset, we create a negative sample using the following method (see Algorithm 1).

Algorithm 1 Negative Sample Generation Function

```
1: function NEGATIVE_SAMPLE_GENERATION(all_data)
2:     negative_samples ← randomly_sample(num_negative_samples, all_data)
3:     pos_data ← all_data
4:     neg_data ← negative_samples
5:     for $i = 1$ to len(negative_samples) do
6:         neg_data[$i$] ← (pos_data[$i$][0], neg_data[$i$][1], 0, pos_data[$i$][3])
7:     end for
8:     all_data ← pos_data + neg_data
9: end function
```

We are now able to pair non-related sentences and assign them to a similarity score of 0, essentially augmenting our dataset by a factor of 2 with the assumption that randomly paired sentences are more likely to be unrelated.

5. Weighted Average to Update Model. As we ran our experiments, we noticed that our highest accuracies were from the paraphrase detection task. This makes logical sense as the number of training examples for this task (17,688) far surpassed the number of examples for semantic analysis (1068) and semantic textual similarity (755). Because of this, we proposed a new method to incentivize the model to improve other tasks over the epochs. Initially, as mentioned in dataset expansion, if there was improvement in any task, we would update the entire model. In this case, we decided to take a weighted average of the accuracies of each task, weighting semantic analysis and semantic textual similarity higher, so that we would update model when we noticed an improvement in the overall model due to the two tasks.

$$\text{weighted\_average\_accuracy} = \alpha \cdot \text{acc\_sst} + \beta \cdot \text{acc\_para} + \gamma \cdot \text{acc\_sts}$$

5 Experiments

Data. For this project, we used the three datasets provided for the Default Project. First, we used the Stanford Sentiment Treebank (SST) for the sentiment analysis task, which contains 11,855 single sentences from movie reviews. After being parsed, these sentences produced 215,154 unique phrases, and each phrase was labeled either negative (0), somewhat negative (1), neutral (2), somewhat positive (3), or positive (4). Next, for the paraphrase detection task, we used the Quora dataset. This dataset was the largest of the three, containing 400,000 question pairs with binary labels (with a label of 1 meaning paraphrase) indicating whether the pairs were paraphrases of each other. Finally, we used the SemEval STS Benchmark dataset for the semantic textual similarity task. This dataset had 8,628 different sentence pairs scored on similarity from 0 (unrelated) to 5 (equivalent meaning).

Evaluation method. We have used the evaluation methods detailed in the Default Project handout. Specifically, we have used accuracies for the sentiment analysis and paraphrase detection tasks and Pearson correlation for the semantic textual similarity task.

Experimental details. For all 8 experiments, we used a finetune learning rate of $1e^{-5}$. For our 1st Milestone Baseline experiment, we used 10 epochs, but for our next 5 experiments we only used 1 epoch to more efficiently finetune and assess improvements after each experiment. For our last 2 experiments, we used 6 epochs, in order to 1) get the best results possible over multiple epochs and 2) compare which conditions to save the model between epochs would be best (as changing those conditions was what was relevant to the experiment).

Results. Below is a table summarizing all of our experiments for our 5 approaches. All the scores, unless otherwise specified as "Test", are what we obtained from the Dev Leaderboard. Our final submission to the Test Leader board is the entry bolded, named "Increased Number of Epochs". Here, we will analyze the progression of our model as a result of these experiments.

Experiment	**SST**	**Paraphrase**	**STS**	**Score**
Milestone Baseline	0.529	0.499	0.016	0.512
Expanding to All 3 Datasets	0.378	0.774	-0.054	0.541
Cosine Similarity with 1 Linear Layer	0.327	0.774	0.078	0.547
Cosine Similarity with No Linear Layer	0.358	0.775	0.152	0.570
RAdam Optimizer (Discarded)	0.349	0.775	0.049	0.549
Negative Sampling	0.382	0.773	0.246	0.593
Increased Number of Epochs	**0.469**	**0.754**	**0.443**	**(Dev) 0.648**
Increased Number of Epochs	**0.488**	**0.764**	**0.428**	**(Test) 0.655**
Weighted Average to Update Model	0.492	0.770	0.382	(Dev) 0.651
Weighted Average to Update Model	0.473	0.772	0.377	(Test) 0.645

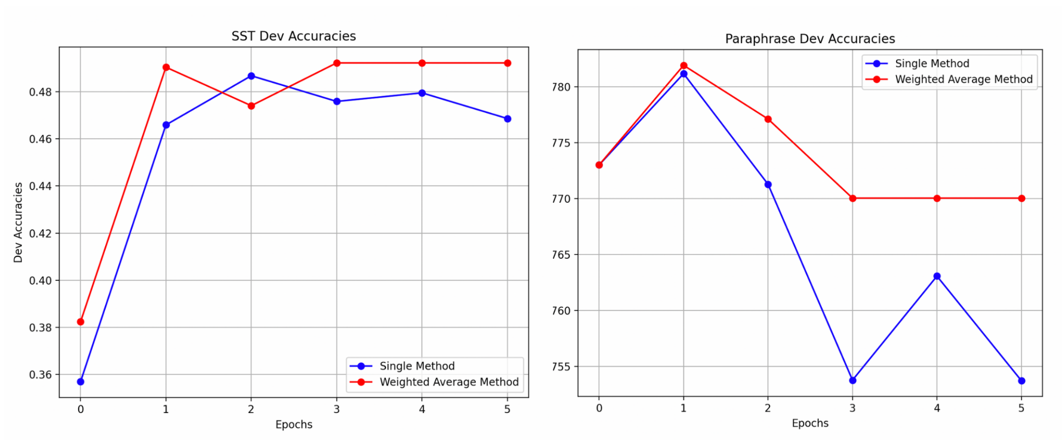
1. Expanding to All 3 Datasets: To begin with, upon expanding to all 3 datasets, we saw a significant increase in overall dev score. This is because the paraphrase accuracy increased significantly, which makes sense due to the large size of the Quora training dataset for this task. We also saw a decrease in SST accuracy, which was also expected because the previous Milestone Baseline only trained on the SST dataset, resulting in better performance for that task. Unexpectedly, we saw a decrease in STS correlation, which motivated us to try our next experiment: cosine similarity.

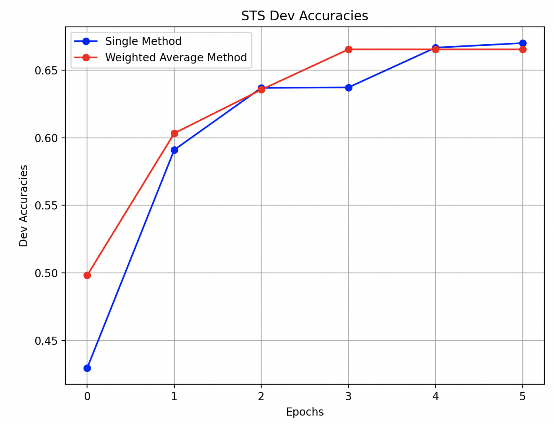
2. Cosine Similarity: For our cosine similarity extension, we specifically chose to focus on the semantic textual similarity task to improve our STS correlation scores. In the $predict\_similarity$ function, our initial approach involved passing both embeddings through the same linear layer (input size of 768, output size of 1) and then returning the cosine similarity of the result. The results were promising, as the STS correlation and overall score improved. However, upon looking at the csv prediction files for the semantic textual similarity task qualitatively, we realized that our model was only outputting logits of -1 or 1. We realized that these binary logits could be limiting our model's ability to scale up to the 0 - 5 labels used in the STS dataset, and decided to try modifying our architecture. Instead of passing the embeddings through the same linear layer before calculating the cosine similarity, we decided to simply compute the cosine similarity on the embeddings themselves and return the result. This gave us floating point logits in the range of [-1, 1] and improved scores across the board, particularly in the STS correlation category.

3. Alternative Optimizer: In this experiment, we wanted to try a different optimizer than the default AdamW, so we chose to use RAdam due to its use of weight decay to stabilize training. However, we found that the RAdam optimizer resulted in significantly worse performance, particularly in terms of STS correlation. This could be due to the lack of a learnable parameter. Thus, we decided to continue using AdamW.

4. Negative Sampling: Although the cosine similarity extension did improve our STS correlation score, it was still much lower than our scores for the other two tasks. As a result, we decided to try our negative sampling approach specifically on the semantic textual similarity task. This was a success; the negative sampling greatly improved our STS correlation score, and also moderately improved the SST accuracy as well.

5. Weighted Average to Update Model: After achieving improvements through our negative sampling approach, we decided to start using the full 6 epochs to train the next 2 experiments. For the first experiment with the full 6 epochs, we simply ran our previous negative sampling model, with its method of saving the model as long as a single task's score was better than the model's previous state. But for the second experiment, we wanted to try a weighted average approach when choosing whether to update the model or not after each epoch. This is because, while our paraphrase accuracy was high, both our SST accuracy and STS correlation scores were comparatively lower. Thus, we kept track of a weighted average of the three task-specific dev scores, increasing the weight of the SST and STS scores by multiplying them by 1.5 to give those tasks more importance in the training process. The results, as visualized below for the three tasks (Increased Number of Epochs = Single Method, Weighted Average to Update Model = Weighted Average Method), seemed to favor the weighted average approach on the Dev Leaderboard, but only by a small amount overall. Although the SST accuracy (as well as the paraphrase accuracy, interestingly) did increase as a result of the weighted average approach, the STS correlation decreased - causing a barely greater score. Thus, since both models were so close in overall score, we chose to submit both to the Test Leaderboard. Upon doing so, we found that the model without the weighted average approach performed better, so we chose that model as our final submission.





6 Analysis

Each task exhibited some positive trends, along with challenges.

Sentiment Analysis: For the Stanford Sentiment Treebank dataset, we noticed that our model performed well at determining relative importance of words based on their ordering. For example, "Great story, bad idea for a movie." seems to be more of a negative connotation, despite having an equal distribution of words for describing the movie as positive and negative, and the predicted score was 1. This would suggest that the model is able to pick up on these subtleties and the fact that this sentence is more negative since the negative context came after the positive.This feature could be exhibited because of our use of attention in minBert.

We also noticed that words that illustrated superlatives had a big influence on the rating of the sentence. For example the three sentences “Extremely dumb.”, “Extremely confusing.”, and “Extremely boring.” are all ranked as 1.

Paraphrase Detection: For the Quora dataset, we noticed that certain question pairs that could be easily mistaken as the same question because of their very similar wording were still correctly classified by the paraphrase detection model. For example, “Which era does ’Game of Thrones’ most resemble from history?” and “Which actors in the Game of Thrones do you think bear a resemblance to a historical figure?” were classified as 0, or not a paraphrase. This is an important and positive feature of the model.

However, another thing we noticed is that sentences missing on specificity were often accidentally be categorized as paraphrases, even when they were not. For example, “What are the folk dances of India?” and “What are some folk dances in your country?” are not paraphrases, but were labelled as so by our model. We can see that India is in the category of countries, which is why they could be mistaken, but the lack of specificity would make this not actual a paraphrase. As we saw in past assignments, categories and items within the category tend to be very close in similarity of embeddings.

Semantic Textual Similarity: For the SemEval STS Benchmark dataset, we noticed that sentences that use very similar wording were given high scores. Because a sentence embedding is a combination of the word embeddings, we could imagine that sentences that directly use the same words are more likely to have a higher cosine similarity than sentences that have similar meaning but do not use the same exact words.

For example, the BLEU score of “Two women are sheering a white sheep inside of a wooden building.” and “Two women shearing a white sheep in a wooden stall.” is 0.7016, relatively high, and correspondingly the predicted score by our final model is high, 0.9799. Similarly, "A man is slicing a bun." and "A man is slicing an onion." has a high BLEU score and high similarity score of 0.9863.

This could be helpful but also harmful in situations where the sentence structures are similar and use similar words (like ‘a’, ‘is’, etc.), while the sentences themselves don’t have as high of a similarity. For example, "The girl is carrying a baby." and "A man is eating a food." have very similar structure, but are not very semantically similar. Their predicted similarity however was 0.9579.

7 Conclusion

In this project, we investigated 5 new methods for improving our minBert model. First, we expanded our model to train on three datasets (Stanford Sentiment Treebank, Quora, SemEval STS Benchmark) corresponding to three tasks. After seeing improved results, we experimented with our predict_similarity function to utilize cosine similarity and have a variable number of linear layers. We found that solely using cosine similarity resulted in enhanced performance. We then created a novel negative sampling methodology that significantly outperformed our previous model. We tested two different optimizers (AdamW and RAdam) and found that AdamW performed better. Finally, we experimented with a new weighted average accuracy to update our best model, and found that there were only minor and variable improvements in performance. Our best and final model utilizes cosine similarity, negative sampling, AdamW, and single accuracy updates. In the future, we aim to better finetune our model by testing with different hyperparameters (ex. different learning rates, weight values for accuracy). We also want to look further into negative sampling, as it provided the largest improvement in our data. Other methods would be to increase the number of negative samples, or find a method for choosing the negative samples rather than random selection.

8 Contributions

Neha worked on the initial milestone classifier BERT model, negative sampling extension, and the milestone optimizer baseline and extension. Ananya worked on the initial multitask milestone submission, three dataset expansion extension, the cosine similarity extension, and the weighted average update model. Both partners collaborated on the final writeup and poster.

References

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. Online. arXiv at Cornell University.

IamMrX. 2022. Negative prompts. Online. Hugging Face.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. Online. arXiv at Cornell University.