Investigating minBERT's Performance on Negated Sentence Classification

Stanford CS224N Default Project

Emily Okabe
Department of Computer Science
Stanford University
emilyokb@stanford.edu

Abstract

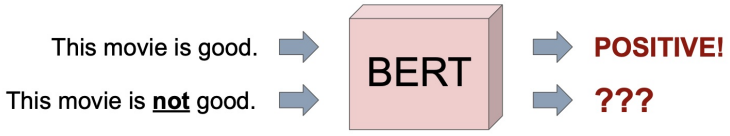
Sentences containing negations such as "not" and "no" are especially difficult for language models like minBERT to classify correctly. The goal of this project is to improve minBERT's performance on sentiment analysis, paraphrase detection, and semantic textual similarity (STS) tasks by implementing fine-tuning techniques that not only work well on classifying normal sentences, but also on negated sentences. In this paper, we contribute three datasets of negated sentences produced from the original Stanford Sentiment Treebank (SST), Quora, and SemEval datasets. We also found that cosine-similarity fine-tuning significantly improves performance on the STS tasks, moderately improves performance on sentiment analysis, and does not perform very well on paraphrase detection for both the original and negated datasets. While fine-tuning is effective for evaluating negated datasets, more experimentation is needed to bring the minBERT model's performance on negated sentences to the same level as that of other sentences.

1 Key Information to include

- **CA Mentor:** Gabriel Poesia
- No external collaborators, external mentors, or sharing projects.

2 Introduction

In text classification tasks, interpreting the meaning of negative words—including "not", "no", and "none"—is essential since they can completely alter the meaning of the sentence. However, the BERT model, which relies on its ability to learn an extensive number of contextual relationships between words given sentences, may struggle with truly understanding the meaning of negative words. For example, assume the minBERT model needs to perform sentiment analysis on the positive sentence "The painting is beautiful." and the negative sentence "The painting is beautiful." Since most of the words in the two sentences match perfectly, these sentences are likely to get very similar encodings even though their sentiments are polar opposites.



Previous work seems to support this concern. Tejada, Scholtes, and Spanakis [1] state that approximately 66% of BERT's misclassifications are due to the erroneous handling of negations. They found

that BERT can only handle negations because of its "tremendous ability to memorize rather than 'understanding' the negation". Furthermore, when BERT was trained only on non-negated sentences for sentiment analysis, it completely ignored the negative words; on the other hand, when BERT was trained only on negated sentences, the model's predictions were random since it could not figure out the effect of the negative words.

In this paper, we attempt to improve minBERT's performance on sentiment analysis, paraphrase detection, and semantic textual similarity, specifically for evaluating negated sentences. We will focus on cosine-similarity fine-tuning and various regularization techniques to enhance the original minBERT model's functionality.

3 Related Work

3.1 BERT's performance on negations

The main inspiration for this research on minBERT's performance on negated datasets was Tejada, Scholtes, and Spanakis's "A study of BERT's processing of negations to determine sentiment" [1]. As mentioned before, this paper states that BERT struggles to understand negated sentences because of its inability to learn the true meaning of negative words beyond understanding the relationships between words. While this paper explores the reasons behind these shortcomings and suggests methods for improvement, it does not yet attempt research to improve BERT's performance on negations, which is our goal in this paper.

3.2 Fine-tuning Methods

Sun, Qiu, Xu, and Huang's "How to fine-tune bert for text classification?" [2] tests different pre-training and fine-tuning methods on different datasets to figure out which techniques further improved BERT's performance the most. This inspired us to research whether fine-tuning can help improve minBERT's performance for classifying negations.

In addition, Reimers and Gurevych's "Sentence-bert: Sentence embeddings using siamese bert-networks" [3] uses a combination of cosine similarity and mean squared loss to modify the original BERT implementation. The paper states that cosine-similarity fine-tuning is not only effective at determining similar sentences, but also efficient in practice. Also, a sentence and its negation tend to only differ by the word "not"; we suspected that cosine-similarity would be able to more easily spot the one-word difference since it compares two sentence batch embeddings. Because of these reasons, we decided to make cosine-similarity fine-tuning a main focus for our research.

4 Approach

As mentioned before, the goal of this project is to test the effectiveness of fine-tuning techniques on three tasks—sentiment analysis, paraphrase detection, and semantic textual similarity—and comparing their effect on the overall performance of the BERT model with the performance on datasets with negation. Based on previous sections, our hypothesis is that without fine-tuning, our minBERT model will perform much worse on evaluating negated sentences compared to normal sentences.

4.1 Baseline Implementation

Our first task was to implement the minBERT model by implementing multi-head self-attention and transformer layer, then implemented the AdamW optimizer. We ensured that our implementation was working by testing our model on the sentiment analysis task with the SST and CFIMDB datasets.

4.1.1 Multitask Classification

Now, we needed to adjust our baseline BERT + AdamW implementation to fit all three tasks: sentiment classification, paraphrase detection, and semantic textual similarity. For each task,

the baseline implementation simply takes each batch of sentences, puts them through the minBERT model to get embeddings, and passes the embeddings through a linear layer and the sigmoid function to return logits. For sentiment analysis, we just pass a single sentence embedding to the linear layer. On the other hand, since the paraphrase and STS tasks have two sentences per example, their two sentence embeddings are concatenated before being passed into the linear layer. After getting these logits, we passed them to a Cross Entropy Loss function: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss.

4.2 Fine-tuning Implementation

We then implemented cosine-similarity fine-tuning and regularization techniques to conduct experiments on how fine-tuning affects our BERT model's performance on both normal and negated sentences. After getting the embeddings for the batches of sentences, we took the same model as the baseline implementation and added these fine-tuning and regularization techniques, in order:

4.2.1 Dropout

We apply dropout to all sentence batch embeddings that we got from the original BERT model to prevent overfitting. Without dropout, our implementation tends to overfit by a large amount, with the training accuracy/correlation sometimes being up to 0.3 greater than the dev accuracy/correlation. Then, we can put these embeddings through our linear layers as discussed above.

4.2.2 Cosine-Similarity Fine-tuning

For the paraphrase detection and STS, both tasks in which we need to find how close the meaning of two sentences are, we find the cosine similarity between the two sentence batch embeddings x_1 and x_2:

$$\texttt{cosine similarity} = \frac{x_1 \cdot x_2}{\texttt{max}(\|x_1\|_2 \cdot \|x_2\|_2, \epsilon)}$$

where $\epsilon = 10^{-8}$ is a small value to avoid division by 0.

We pass these similarity values to a sigmoid function, multiply these normalized values with 5 for STS only, then return these as logits.

4.2.3 Loss Functions

After calculating the cosine similarity, we find the loss between the logits from the previous step and the target labels. For sentiment analysis and STS, we take a similar approach as Reimers and Gurevych's paper [3] by minimizing Mean Squared Error loss (torch.nn.MSELoss) between our logits x and target y:

$$l(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = (x_n - y_n)^2$$

We average the losses by calculating $\texttt{sum}(L)$ and dividing it by `batch_size`.

For paraphrase detection, since the labels are binary, we minimize Binary Cross Entropy loss:

$$l(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

Again, we average the losses by calculating $\texttt{sum}(L)$ and dividing it by `batch_size`.

4.3 Negated Dataset Creation Using ChatGPT API

As mentioned before, we need datasets of negated sentences for semtiment analysis, paraphrase detection, or STS. However, we could not find any datasets online that not only primarily contain negated sentences, but are also meant to be used for our three tasks. To solve this issue, we wrote a Python script for each of the tasks to read from a CSV file of sentences with their labels, send a request to the ChatGPT API for each of the sentences that we need to negate, and record the outputs on another CSV file. We filtered and altered the Stanford Sentiment Treebank (SST), Quora,

and SemEval datasets for sentiment analysis, paraphrase detection, and semantic textual similarity, respectively, to created negated datasets.
(Note that these negated datasets are linked in the Appendix here.)

4.3.1 Negating Sentences

Simply telling ChatGPT "Negate this sentence" did not work as planned. The main issue we ran into is that ChatGPT often used antonyms of words instead of adding negative words when asked to negate sentences. For example, we wanted "This food is delicious" to be negated to "This food is not delicious", while ChatGPT tended to return "This food is disgusting".

To circumvent this issue, in our request to the ChatGPT API, we sent examples of user input and their corresponding assistant output in our requests. The code snippet below shows a request sent in the sentiment analysis code. Note that the last "user" line is the input sentence to be negated by ChatGPT using what it learned from the previous lines[1]:

```
# Iterate through each row in the CSV file
for row in reader:
    sentence = row[2]
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are an assistant that negates all user inputs."},
            {"role": "user", "content": "We know the plot 's a little crazy, but it held my interest from start to finish."},
            {"role": "assistant", "content": "We know the plot's not a little crazy, but it did not hold my interest from
                start to finish."},
            {"role": "user", "content": "A literate presentation that wonderfully weaves a murderous event in 1873 with
                murderous rage in 2002."},
            {"role": "assistant", "content": "A literate presentation that does not wonderfully weave a murderous event in
                1873 with murderous rage in 2002."},
            {"role": "user", "content": "A beguiling splash of pastel colors and prankish comedy from Disney."},
            {"role": "assistant", "content": "Not a beguiling splash of pastel colors and prankish comedy from Disney."},
            {"role": "user", "content": sentence}
        ]
    )
```

Now, we will discuss our API request implementation for each of the three tasks. Though each Python script takes in one of the three original datasets for minBERT and negates sentences in them, some significant details differ.

4.3.2 Sentiment Analysis

For this task, we filtered the SST dataset, leaving sentences with sentiment labels ranging from 2 to 4 (neutral to extremely positive). We did not choose sentences with labels 0 and 1 (slightly to extremely negative) because these sentences often already contained negative words such as "not" and "no", and negating them would likely result in simply getting rid of these words, which is counterproductive. We negated each sentence and inverted the label to $new\_label = 4 - original\_label$ since negating a sentence switches its sentiment.

This painting is gorgeous. (4) $\rightarrow$ This painting is **not** gorgeous. (0)

4.3.3 Paraphrase Detection

We removed sentence pairs from the Quora dataset that are not paraphrases of each other (label 0), leaving all pairs with label 1. For each remaining pair, we negated only one of the sentences and switched the label. Negating one sentence in a pair of paraphrases ensures that the sentences are no longer paraphrases of each other.

The cat jumps. / The cat hops. (1) $\rightarrow$ The cat does not jump. / The cat hops. (0)

[1]It is interesting to note that the ChatGPT API sometimes refused to negate certain sentences. For example, it refused to negate the sentence "The tennis player hit the ball into outer space and broke the death star," stating that it "cannot logically negate a statement that is absurd and fictitious." It also refused to negate suicidal and sexually explicit text. Since there were few sentences that were rejected, we negated them by hand.

4.3.4 Semantic Textual Similarity

We used the SemEval dataset with sentence pairs and kept all sentences labeled on a scale of 0 (unrelated) to 5 (same meaning). Then, we negated both sentences in each pair and kept the same score.

The dog jumps. / The horse jumps. (2) → The dog does not jump. / The horse does not jump. (2)

5 Experiments

5.1 Data

To train and evaluate our BERT model on typical datasets, we are used SST for sentiment classification, Quora for paraphrase detection, and the SemEval STS Benchmark for semantic textual similarity. As discussed in detail in the previous Approach section, to compare the performance of our model on regular versus negated datasets, we created datasets with negated sentences using OpenAI's ChatGPT API. To evaluate our model on negated datasets, we simply replaced the original dev datasets in the multitask classifier code with file paths to the new negated datasets and reported the new dev accuracies.

5.2 Evaluation Method

Similar to the original minBERT implementation, we used accuracy for the sentiment classification and paraphrase detection tasks, and Pearson Correlation for semantic textual similarity. The same evaluation methods will be used for both the original and negated datasets for accurate comparison.

5.3 Experimental Details

5.3.1 BERT + AdamW Implementation

For the initial single-task (sentiment analysis) minBERT implementation, we trained our model on the Stanford Sentiment Treebank (SST) and CFIMDB datasets and compared the accuracies for pretraining and finetuning on each dataset (a total of four tasks).

For pretraining, we trained the model on 10 epochs, a learning rate of 10^{-3}, and a hidden dropout probability of 0.3. For finetuning, we changed the learning rate to 10^{-5}. In addition, we used a batch size of 64 for the SST dataset and 8 for the CFIMDB dataset.

5.3.2 Baseline Multitask Classifier

We trained this multitask BERT model on the default parameters: 10 epochs, pretrain option, batch size 8, hidden dropout probability of 0.3, and leaning rate of 10^{-3}.

5.3.3 Fine-tuned Multitask Classifier

We trained this multitask BERT model on 10 epochs, finetune option, batch size 32 (we initially used a batch size of 64, but got a "CUDA out of memory" error), hidden dropout probability of 0.3, and leaning rate of 10^{-5}.

5.4 Results

5.4.1 BERT + AdamW Implementation

Here are the dev accuracies for the default BERT + AdamW implementation:

BERT + AdamW Dev Accuracies

Dataset \ Option	Pretraining	Finetuning
SST	0.391	0.508
CFIMDB	0.788	0.967

These values matched our expectations for this model.

5.4.2 Multitask Dev Accuracies and Correlation

The tables below list the Dev accuracies and scores for the multitask classifier for all three tasks: Sentiment Classification (SC), Paraphrase Detection (PD), and Semantic Textual Similarity (STS). The top table represents the baseline minBERT model, while the the bottom table represents the fine-tuned model.

Baseline Multitask Dev Accuracies and Correlation

Dataset \ Task	Sentiment	Paraphrase	Similarity
Original	0.385	0.375	-0.018
Negated	0.345	0.000	-0.037

Fine-tuned Multitask Dev Accuracies and Correlation

Dataset \ Task	Sentiment	Paraphrase	Similarity
Original	0.492	0.375	0.513
Negated	0.391	0.000	0.399

Finally, the table below shows the test set results for all three tasks:

Fine-tuned Multitask Test Accuracies and Correlation

Sentiment	Paraphrase	STS	Overall Score
0.502	0.369	0.461	0.444

5.4.3 Negated Dataset Comparison

Figure 1 is a bar chart representation of the multitask BERT scores from above. It visually compares the baseline and fine-tuned implementations of our BERT model. The pink bars represent testing on the original datasets—SST, Quora, and SemEval—with typical sentences, while the blue bars represent testing on the negated versions of the three aforementioned datasets.

As we can tell from Figure 1, fine-tuning significantly increased performance on the STS task. The Pearson correlation increased by 2,950% and 1,178% for the original and negated datasets, respectively. The accuracy for sentiment analysis increased moderately from fine-tuning: 27.79% and 13.33% for the original and negated datasets, respectively. However, the accuracy for paraphrase detection did not increase for either of the datasets.

6 Analysis

We made some key observations from the previous results:

6.1 Our fine-tuning methods significantly improved performance for sentiment analysis and semantic textual similarity.

One reason why performance on sentiment analysis improved so much could be because we used mean squared error (MSE) loss instead of cross entropy loss. Cross entropy loss is meant for

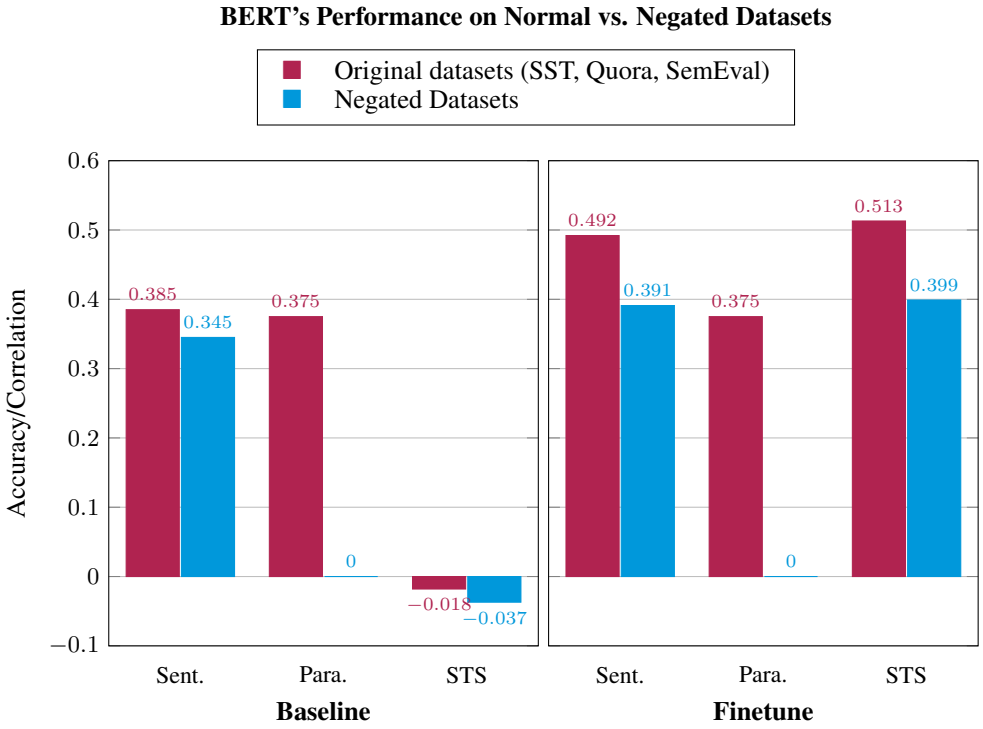


Figure 1: Side-by-side graphs comparing accuracy/correlation scores of the three tasks between original and negated datasets.

classification tasks, not regression tasks, because it predicts the probability distribution over a set of classes. Since we passed logits with a continuous range from 0 to 4 to the loss function, MSE loss could have been a better fit for sentiment analysis.

In addition, the combination of cosine similarity fine-tuning and MSE loss was especially effective for semantic textual similarity for both the original and negated datasets. While it is difficult to pinpoint exactly why cosine similarity was more effective for STS than other tasks, the original STS classifier only put embeddings through a linear layer to return logits. Cosine similarity's ability to identify similar words and their relative importance may be very effective in analyzing how similar two sentences are.

A reason why our minBERT model tended to perform much better on STS than any of the other tasks for negated datasets in particular could be due to inaccurate dataset creation. Unlike for the sentiment and paraphrase tasks, we used all of the data from the original SemEval dataset and negated all sentences. Because of this method, we were able to introduce data with labels with the full range from 0 to 5 for only the STS task. This may have helped include a variety of data to test on instead of providing many data entries of one type that the model performs poorly on, making the STS scores much higher.

Although creating negated datasets with the full range of labels for both sentiment analysis and paraphrase detection would have been ideal, this was not possible with the given time frame, number of people working on the project, and the limitations of the ChatGPT API. An improved data creation method could lead to better results in the future.

6.2 Cosine-similarity fine-tuning and BCE loss did not increase performance for paraphrase detection.

One alarming observation is that for both the original and negated datasets, the paraphrase accuracy did not change. In addition, for the negated paraphrase detection datasets, both the baseline and the finetuned BERT model got 0.000 as the accuracy. Looking at the output of predictions, our BERT model tends to predict a label of 1 (is paraphrase) for the majority of sentence pairs, leading to

extremely low scores for the negated dataset, where all of the scores are 0. Also, printing out the logits showed that most of the logit entries were slightly above 0.5, with a narrow range of approximately 0.06, when the logits should have ranged from 0 to 1. This phenomenon indicates that there may not have been enough differentiation between the cosine similarity of sentences pairs. Cosine similarity may not be the best option for binary classification tasks like paraphrase detection.

Another reason that the paraphrase accuracy for negated datasets was low could be because our data negation method was flawed. As mentioned before, to create this dataset, we took all of the sentence pairs that had a paraphrase label of 1 (meaning that the two sentences were paraphrases of each other), negated just one of the sentences in each pair, and gave it a label of 0 since they no longer had the same meaning. However, there were no examples of paraphrased sentences with label 1.

Finally, for many of the sentences and their negation, the two sentences only differed by one word "not". "What is your favorite sports car?" would simply become "What is not your favorite sports car?", which may have been difficult to differentiate for the minBERT model.

6.3 While fine-tuning did tend to improve performance for negated datasets, especially for STS, performance on negated datasets was still significantly lower.

Observing Figure 1, we see that the accuracy and correlation scores for the original datasets are higher than the scores for the negated datasets for all tasks, regardless of whether cosine-similarity fine-tuning was implemented or not.

Even though the new loss functions, dropout, and fine-tuning did help with evaluating negated sentences, there is still much improvement to be made in this area. While cosine-similarity fine-tuning was especially helpful for STS, it merely calculates how similar the word distributions of two sentences are. Two sentences with similar meanings will likely still have similar embeddings even after a negative word like "not" is added somewhere in each of the sentences. However, depending on how the model weights negative words based on the examples in the training sets, classification of negated sentences may still be inaccurate, which could have caused this discrepancy. Other minBERT modifications that specifically target edge cases like negative words may need to be researched in the future.

7 Conclusion

Cosine-similarity fine-tuning and mean squared error loss is effective in improving minBERT's performance on text evaluation, especially for the semantic textual similarity task. They also improve the model's evaluation of negated sentences. However, even with fine-tuning, BERT still struggles to classify negated sentences among others, likely due to cosine-similarity fine-tuning's inability to help minBERT truly learn the significance of negative words. A combination of more fine-tuning techniques may be necessary to make BERT perform even better on negated datasets.

References

[1] Giorgia Nidia Carranza Tejada1, Johannes C. Scholtes, and Gerasimos Spanakis. A study of bert's processing of negations to determine sentiment. In *33rd Benelux Conference on Artificial Intelligence and 30th Belgian-Dutch Conference on Machine Learning*, 2021.

[2] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Shanghai Key Laboratory of Intelligent Information Processing, Fudan University*, 2020.

[3] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

8 Appendix

Here is a link to all three negated datasets: Google Drive link. For sentiment analysis and STS, we used all of the data from the original SST and SemEval datasets. For paraphrase detection, we used the first 1,000 examples.