Enhancing minBERT for Sentence Similarity with Cosine Similarity and Contrastive Learning

Stanford CS224N Default Project

Yi-Chin Huang, Xiaomiao Zhang
Department of Computer Science
Stanford University
yichinh@stanford.edu zxmeow98@stanford.edu
Mentor: Davey Huang

Abstract

This project aims to first implement minBERT model and subsequently, we utilize the completed BERT model to perform sentiment analysis on the Stanford Sentiment Treebank dataset as well as CFIMDB data set which contains highly polar movie reviews. Finally, in the latter half of this project, we fine-tuned and extended the BERT model to create sentence embeddings that can perform well across a wide range of downstream tasks including sentiment analysis, paraphrase detection and semantics textual similarity.

We explored different ways of fine-tuning the model and tried out different methods to improve the model training accuracy, including using cosine similarity to predict similarity score and mean square error as the loss function as well as contrastive learning with supervised approach.

The main findings of this project are the approaches that can improve the performance of the BERT-based system performing these three tasks: sentiment analysis, paraphrase detection, and semantic textual. We noticed that training the three datasets simultaneously can yield better results then training sequentially. Furthermore, both cosine similarity and contrastive learning improved the performance and accuracy of the model.

1 Introduction

Imagine being able to accurately understand and interpret human language. This is the goal of natural language processing (NLP), a field that has seen tremendous growth in recent years thanks to advancements in deep learning models like BERT. In this project, we delve into the fascinating world of NLP by implementing the key aspects of the original BERT model, which includes multi-head self-attention and a Transformer layer. We then put our completed BERT model to the test by performing sentiment analysis on datasets including . But we didn't stop there. In the latter half of the project, we fine-tuned and extended the BERT model to create sentence embeddings that perform well across a wide range of downstream tasks, including sentiment analysis, paraphrase detection, and semantic textual similarity.

To achieve this, we explored different ways of fine-tuning the model and experimented with various methods to improve training accuracy, such as using cosine similarity to predict similarity scores and mean square loss as the loss function, following similar approach introduced by Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks [1]. We also incorporated contrastive learning loss to better calculate the total loss, with approach introduced by Simple Contrastive Learning of Sentence Embeddings [2]. They both proved to be effective. We also explored different temperatures used for constrastive learning and different ratios of combining the loss. More details can be seen in the Experimental details section. Our findings revealed that training the datasets simultaneously

yielded better results than training them sequentially. Moreover, we observed that both cosine similarity and contrastive learning improved the performance and accuracy of the model.

Our work highlights the potential of BERT-based systems for complex NLP tasks and offers insights into novel techniques for improving text similarity tasks. We aspire to contribute to the advancement of NLP by building on existing models and exploring new approaches.

2 Related Work

The field of natural language processing has seen significant progress in recent years, thanks in part to the introduction of BERT (Bidirectional Encoder Representations from Transformers) by BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [3], a state-of-the-art language model developed by Google in 2018. BERT has since become a popular tool for a wide range of NLP tasks, including sentiment analysis, question answering, and language translation.

The use of cosine similarity and contrastive learning in NLP has been explored in several recent studies. In a paper Sentence-bert: Sentence embeddings using siamese bert- networks (2019) [1], the authors proposed a novel method for training text embeddings using cosine similarity. The authors showed that their method outperformed traditional methods, such as negative sampling and hierarchical softmax, in various downstream NLP tasks. Furthermore, "Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks and Rank Learning" by Yin et al. (2018) [4] also explores the effectiveness of cosine similarity in NLP. The authors show that cosine similarity can be used to effectively rank short text pairs, such as those found in question answering and search engine queries.

Another recent paper that explores the use of contrastive learning in NLP is "SimCSE: Simple Contrastive Learning of Sentence Embeddings" by Gao et al. (2021) [2]. The authors proposed a simple yet effective method for training sentence embeddings using contrastive learning. In SimCSE, the model is trained to maximize the similarity between different views of the same sentence and minimize the similarity between views of different sentences. This approach allows the model to learn representations that capture the semantic meaning of text, while also being robust to variations in style and syntax.

Overall, these papers highlight the growing interest in using BERT for a wide range of NLP tasks, as well as the importance of techniques such as cosine similarity and contrastive learning in improving the performance of NLP models.

3 Approach

Baseline. Based on the pretrained model provided in the default project repo, we developed a multitask classifier for three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. The baseline model structure for the three tasks is described below.

- Sentiment Classification: The model takes in a batch of sentences as input and passes them through a pre-trained BERT model. It then extracts the [CLS] token embedding from the final layer of BERT and passes it through two fully connected layers (with ReLU activation in between) to classify the sentiment of each sentence into one of five classes (negative, somewhat negative, neutral, somewhat positive, positive). The output of the model is a set of 5 logits (unnormalized values) for each sentence.
- Paraphrase Detection: The model takes in a batch of pairs of sentences as input and passes each sentence through the same pre-trained BERT model separately. It then extracts the output of the pooler output of BERT for each sentence and concatenates them together. The concatenated vector is passed through two fully connected layers (with ReLU activation in between) to predict whether the two sentences are paraphrases or not. The output of the model is a single unnormalized logit for each pair of sentences.
- Semantic Textual Similarity: The model takes in a batch of pairs of sentences as input and passes each sentence through the same pre-trained BERT model separately. It then extracts the output of the pooler output of BERT for each sentence and concatenates them together. The concatenated vector is passed through two fully connected layers (with ReLU activation

in between) to predict the similarity between the two sentences. The output of the model is a single unnormalized logit for each pair of sentences.

For training stage, we implemented Adam optimizer and a multitask classifier to simultaneously do classification on the tasks of sentiment analysis, paraphrase detection, and semantic textual. The Adam optimizer is a stochastic optimization algorithm that computes adaptive learning rates for different parameters by estimating the first and second moments of the gradients. It updates exponential moving averages of the gradient and the squared gradient, performs bias correction, and adjusts the learning rate accordingly. In each epoch, we trained through the whole dataset for each task one after the other. We use cross entropy loss for sentiment analysis and semantic textual, and L1 loss for paraphrase detection, since it is a binary classification task.

Extensions. Our approach involved implementing cosine similarity fine-tuning for paraphrase detection and semantic textual tasks, given their shared objective of evaluating sentence similarity. We drew inspiration from the paper Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks [1] and included cosine similarity as a feature in our model. Specifically, we computed cosine similarity between the pooler output of two sentence embeddings and added it right before the classification layer, and utilized mean squared error loss as the objective function. Additionally, we experimented with cosine embedding loss as the objective function, with the input of the pooler output of two sentence embeddings.

Due to the nature of our dataset, we also integrated contrastive learning into the tasks of paraphrase detection and semantic textual. Referring to the SimCSE paper [2], we used the loss in contrastive learning with a supervised approach since the labels for both tasks represent whether two sentences are similar. We obtained the pooler output of sentence embeddings of two sentences, passed them through a linear layer, and then used the contrastive loss function. We used the contrastive loss function from the GitHub repo SupContrast [5] and made slight modifications to it. As we also wanted to continue training the classification layer, we formulated the loss as:

$$total\_loss = \lambda_1 \times contrastive\_loss + \lambda_2 \times loss$$

where the loss was either L1 loss (for paraphrase detection), cross-entropy loss (for sementic textual), or mean squared error loss (for our final model) for our prediction function. We tuned the values of λ_1 and λ_2 to obtain better results, which will be discussed in the experiment section.

After conducting some experiments, we discovered that integrating both cosine similarity with mean squared error loss and contrastive learning in our model improved the results. As a result, we decided to incorporate these techniques and tuned the parameters to train our final model.

4 Experiments

4.1 Data

- **Stanford Sentiment Treebank (SST) dataset**

  The Stanford Sentiment Treebank (SST) dataset comprises 11,855 movie review sentences parsed using the Stanford parser, resulting in 215,154 unique phrases labeled by three human judges as negative, somewhat negative, neutral, somewhat positive, or positive. This project employs BERT embeddings to predict sentiment classification labels for the dataset, which is split into train (8,544 examples), dev (1,101 examples), and test (2,210 examples).

- **Quora Dataset** The Quora dataset is a subset of the 400,000 question pairs in which labels indicate whether the instances are paraphrases. The dataset is split into train (141,506 examples), dev (20,215 examples), and test (40,431 examples), and the metric used to test it is accuracy due to the binary labels.

- **SemEval STS Benchmark Dataset**

  The SemEval STS Benchmark dataset contains 8,628 sentence pairs with varying similarity levels ranging from 0 (unrelated) to 5 (equivalent meaning). The dataset is split into train (6,041 examples), dev (864 examples), and test (1,726 examples).

4.2 Evaluation method

We use data sets from the above to train and test the model we have implemented. The data sets are split into three parts – train, dev and test. We first use the training data to train the model, the call dev and test to get accuracy ratio and Pearson correlation (for semantics similarity). Here is the formulas we use for accuracy calculation and Pearson correlation calculation.

$$Accuracy = \frac{NumberOfCorrectPredictions}{TotalNumberOfPredictions}$$

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where r_{xy} is the Pearson correlation coefficient between variables x and y, n is the number of observations, x_i and y_i are the ith observations of variables x and y, $\bar{x}$ and $\bar{y}$ are the means of variables x and y, respectively.

4.3 Experimental details

We conducted experiments on the following aspects of the model training and fine-tuning process:

- ConsineSimilarity plus entropy loss vs. ConsineSimilarity plus MSE loss vs. CosineEmbeddingLoss – apply cosine similarity to predict function with loss calculated using entropy or MSE loss function vs. apply cosine embedding loss function directly
- Contrastive learning loss and its parameter – Whether to apply contrastive learning loss as well as choosing the most appropriate temperature value
- Training order – whether to train each data set sequentially or to train all data sets at once with each epoch, and in what order should we train the three datasets
- Loss function factors – choosing appropriate training loss factor for MSE loss and contrastive learning loss

4.3.1 ConsineSimilarity with cross entropy loss vs. ConsineSimilarity with MSE loss vs. CosineEmbeddingLoss

We first added ConsineSimilarity on top of baseline with cross entropy loss. Subsequently, we found ConsineSimilarity combined with MSE loss function was present by Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks [1], while CosineEmbeddingLoss was recommended by the handout.

Thus, we decided to set up tests to see which approach is the best. With everything else being baseline setup, we have received the following results with epoch number 3.

Table 1: Experiment result of Baseline, ConsineSimilarity with cross entropy loss (CosSim+CE), ConsineSimilarity with MSE loss (CosSim+MSE) and CosineEmbeddingLoss with baseline setup

Metric	Baseline	CosSim+CE	CosSim+MSE	CosineEmbeddingLoss
SST Dev Accuracy	0.262	0.510	0.494	0.519
Paraphrase Dev Accuracy	0.669	0.625	0.810	0.375
STS Dev Correlation	0.153	0.221	0.374	0.016

From the results shown in Table 1, we can see that SST accuracy was roughly the same for all approaches while the Paraphrase and STS accuracies are significantly improved when we apply cosine similarity to predict function with loss calculated using ConsineSimilarity with MSE loss function.

4.3.2 Contrastive learning loss and its parameter

We also wanted to make sure that using contrastive learning loss can help improve the model performance. We implemented contrastive learning on top of baseline with temperature being 0.5 and got the following result.

Table 2: Experiment result of model with Contrastive learning loss vs. without Contrastive learning loss

Metric	Baseline	Contrastive learning loss
SST Dev Accuracy	0.262	0.407
Paraphrase Dev Accuracy	0.669	0.804
STS Dev Correlation	0.153	0.289

From the result, it seems that using contrastive learning can improve the performance. Furthermore, we are also curious what value we should set the temperature to be to achieve the best result. Here are the tests we tried by changing the temperature values.

Table 3: Experiment result of Contrastive learning loss with different temperature values

Temperature value and Metrics	0.5	0.7	1	1.3
SST Dev Accuracy	0.407	0.400	0.397	0.424
Paraphrase Dev Accuracy	0.804	0.788	0.793	0.792
STS Dev Correlation	0.289	0.235	0.287	0.331

The experimental results indicate that different temperature values have varying effects on the accuracy of contrastive learning loss. For paraphrase detection, a temperature value of 0.5 produced the highest accuracy, while for STS, a temperature value of 1.3 yielded the best accuracy.

4.3.3 Training mechanism and order

After the experiments above, we have integrated with cosine similarity as well as contrastive learning loss. We decided to try out different training mechanisms. The first approach is to train all data sets with each epoch (simultaneous training), and the second approach is to train each data set after another (sequential training). To test this, we used epoch number 3. Result is shown in Table 4.

Table 4: Experiment result of training mechanism

Metric	Simultaneous training	Sequential training
SST Dev Accuracy	0.507	0.351
Paraphrase Dev Accuracy	0.790	0.765
STS Dev Correlation	0.488	0.519

From the obtained dev accuracy, we can see that simultaneous training yields a better result with all other conditions being the same.

Furthermore, we also wondered if changing the training order of datasets would make a difference. To explore this, we tried two datasets training orders and their results are as follows:

Table 5: Experiment result of training order

Metric/Training order	STS-SST-Para	SST-Para-STS
SST Dev Accuracy	0.460	0.507
Paraphrase Dev Accuracy	0.812	0.790
STS Dev Correlation	0.411	0.488

From results above, the order of datasets doesn't seem to yield much difference.

4.3.4 Loss function factors

With all the features we have tested above, we had completed a model that simutaneously trains the data sets, and applies cosine similarity to predict functions with loss calculated using MSE loss function and contrastive learning loss function that has temperature value 0.5 for paraphrase detection and 1.3 for semantic textual similarity. However, the approporiate proportion of MSE loss and contrastive loss that should be considered towards total loss is still unknown. Thus, with the model, we tested out different factors for MSE loss (mse_loss) and contrastive loss (contra_loss) with the function:

$$total\_loss = \lambda_{para} contra\_loss + mse\_loss$$
$$total\_loss = \lambda_{sts} contra\_loss + mse\_loss$$

where the upper function is the loss for paraphrase detection, and the lower one is the loss for STS. Here are the results of all factors we tried:

Table 6: Experiment result of loss function factors

Metrics	$\lambda_{para} = 0.01$ $\lambda_{sts} = 0.01$	$\lambda_{para} = 0.001$ $\lambda_{sts} = 0.005$	$\lambda_1 = 0.001$ $\lambda_2 = 0.01$
SST Dev Accuracy	0.456	0.507	0.485
Paraphrase Dev Accuracy	0.774	0.790	0.771
STS Dev Accuracy	0.514	0.448	0.508

4.4 Results

Table 7: Final Results

Metric	Baseline	Dev set Results	Test set Result
SST Dev Accuracy	0.262	0.481	0.492
Paraphrase Dev Accuracy	0.669	0.808	0.810
STS Dev Correlation	0.153	0.510	0.478

Integrating all of the parameters which perform better in the previous experiments, we built our final model. The result is pretty much what we expected after adding cosine similarity with MSE loss and contrastive learning loss. It suggests that our approach solidly improves the model compared to the baseline.

5 Analysis

We conducted several experiments to evaluate the performance of our model on three different data sets – SST, Paraphrase, and STS. We tested different aspects of the model training and fine-tuning process, including cosine similarity, contrastive learning loss, training order, and loss function factors.

We compared the effectiveness of cosine similarity combined with MSE loss function versus cosine embedding loss. From Table 1, we found that cosine similarity with MSE loss function performances the best for paraphrase and STS datasets across the four approaches. Including cosine similarity improves the performance of paraphrase detection and semantic textual tasks because it is a measure of the similarity between two vectors in a high-dimensional space, which is an effective way to compare the semantic similarity between two texts. By incorporating cosine similarity as a loss function, the model can learn to project texts into a high-dimensional space where similar texts have similar representations. This allows the model to better capture the underlying semantic similarity between texts, leading to improved performance on tasks such as paraphrase detection and semantic textual similarity. Additionally, cosine similarity is less sensitive to the length and magnitude of the vectors being compared, making it a more robust measure of similarity than other distance metrics.

Integrating contrastive learning loss into the model leads to better performance because it encourages the model to learn better feature representations by contrasting similar and dissimilar pairs of data points. This results in a more discriminative feature space that can better capture the underlying similarities and differences between the data points. The results of the experiment show that using contrastive learning loss with a temperature value of 0.5 leads to the best performance for paraphrase detection, while a temperature value of 1.3 leads to the best performance for semantic similarity. By tuning the temperature value, the model can better balance the effect of similar and dissimilar pairs, leading to improved performance. Overall, the use of contrastive learning loss can help improve the performance of multi-task learning models by encouraging the model to learn better feature representations.

Combining cosine similarity and contrastive learning in our model, we also wanted to investigate the impact of training order on model performance. Therefore, we tested two different training

mechanisms: training all datasets with each epoch and training each dataset sequentially. From table 4, we observed that training sequentially yielded better results for STS, it resulted in worse results for SST and paraphrase tasks. This is due to the model forgetting what it had learned from the previous tasks when training sequentially. In contrast, when training simultaneously, the model kept updating with all three tasks in mind in each epoch. Based on these findings, we continued to train our model simultaneously.

We also experimented the training order of the datasets. We compared the performance of two orders: STS-SST-Para and SST-Para-STS. We suspected that order SST-Para-STS could be better, since paraphrase and STS are similar, and the latter task is more complex than the former. We thought the earlier task provides a good initialization point for the subsequent task, allowing the model to build on the learned features and adapt to the new task more efficiently. However, the experiment shows that there is no significant difference on the overall result between the two orders.

Finally, we tested different loss function factors for MSE loss and contrastive learning loss. We found that assigning 0.001 as weight to contrastive learning loss for paraphrase detection and 0.005 as weight to contrastive learning loss for STS produced the best overall result for the three datasets. This is because the scale of contrastive loss is larger, about 1000 times of the value of MSE loss. Therefore, it is reasonable to assign a lower weight to contrastive loss to balance between them.

In conclusion, we tested different aspects of the training process, including loss functions and training order. We found that combining cosine similarity and contrastive learning led to better performance. We also tested two different training mechanisms and found that training simultaneously resulted in better performance.

6 Conclusion

The experiments conducted in this project focused on improving the accuracy of a sentence similarity model by adjusting various aspects of the training and fine-tuning process. The findings revealed that sequential training yielded better results than training all data sets with each epoch. Additionally, combining cosine similarity with MSE loss function was found to be effective for Paraphrase and STS data sets. The use of contrastive learning loss was found to improve the model performance for all data sets, with a temperature value of 1.3 for STS and 0.5 for Paraphrase data set achieving the best result overall. Finally, assigning 0.001 as weight to contrastive learning loss for paraphrase detection and 0.005 as weight to contrastive learning loss for STS produced the best overall result for the three datasets.

In conclusion, this study provides useful insights into how different aspects of the training and fine-tuning process can be adjusted to improve the accuracy of a sentence similarity model. The findings can be applied to various natural language processing tasks where sentence similarity is a crucial component. However, this study has limitations as it was only tested on three datasets, and the epoch number we are using to finetune the data sets is pretty small due to the limitation of cloud computing capacity and large amount of computing time needed. Furthermore, due to our virtual machine's memory limitation, we weren't able to try out different batch sizes. We know that contrastive learning can be more effective when the batch size is large. When the batch size is small, the number of examples that can be compared in each training step is limited. As a result, the contrastive loss signal may be noisy and not representative of the true distribution of the examples. For future research, we are looking forward to expanding to other datasets and models, with larger epoch number and batch sizes to further validate the findings.

References

[1] 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), pages 3982–3992*, 2019.

[2] Xingcheng Yao Tianyu Gao and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

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

[4] Ying Fan Fangzhao Wu and Fei Sun. Learning to rank short text pairs with convolutional deep neural networks and rank learning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[5] Yonglong Tian. Supcontrast: Supervised contrastive learning. In *GitHub*, 2020.