Speedy SBERT

Stanford CS224N Default Project

Mentor: Anirudh Sriram

Renee White
Department of Computer Science
Stanford University
reneedw@stanford.edu

Leyth Toubassy
Department of Computer Science
Stanford University
leytht@stanford.edu

Abstract

We have implemented a single enhanced BERT model called Speedy SBERT which can outperform a finetuned standard BERT model on three tasks: sentiment classification, paraphrase detection, and semantic textual similarity. Our model uses mean pooling to generate meaningful sentence embeddings that can be used to improve all three tasks. We use these embeddings in our linear layers for sentiment classification and paraphrase detection and calculate cosine similarity for semantic textual similarity. Along with these accuracy improvements, we significantly increase the time efficiency of our model compared to the baseline by a factor of about 5 (hence the speediness of the SBERT). To do this, we use a novel training optimization which we created ourselves by exploiting the hybrid model design. Overall, we achieved improvements in all three tasks, though with the most improvement in similarity textual similarity, as that was the principal goal of using an SBERT-based model.

1 Introduction

In this paper, we present our take on a multi-purpose Sentence-Bert (SBERT) model built for sentiment classification (SST), paraphrase detection, and semantic textual similarity (STS). BERT, introduced in 2018, is a very powerful model that can be easily specialized for use on specific language problems like the three we are examining. More difficult though, is creating a *single* modified model capable of performing multiple tasks using the same underlying embeddings for each. In order to accomplish these, we've made modifications to BERT in order to generate more meaningful embeddings for entire sentences using the approach pioneered by Reimers and Gurevych (2019). The baseline approach when using BERT is to use the embeddings for the start of sequence <CLS> token as the embedding for the entire sentence. By using mean pooling, one can instead generate sentence embeddings which far better capture the content and construction of the sentence. In our early testing on semantic textual similarity, we found that a pre-trained BERT using a linear neural network finetuned on these <CLS> token embeddings achieved a correlation score of 0.101. In our mean pooled embeddings, we were able to achieve a correlation of 0.833, using only the pre-trained BERT model and cosine similarity between the two sentences. Using these more meaningful sentence embeddings, we executed a variety of optimizations to our model to increase accuracy and training efficiency. One such optimization involved freezing BERT Parameters partway through our training method in order to speed up training by a factor of 5x for both sentiment classification and paraphrase accuracy.

2 Related Work

2.1 BERT

Large language models, developed for natural language (NLP) tasks, have experienced drastic changes in recent years. The development of Bidirectional Encoder Representations from Transformers (BERT) sparked great change in the field, setting a new state-of-the-art performance on various NLP tasks (Devlin et al., 2018). Despite its impressive performance, BERT requires massive computational overhead for tasks such as semantic textual similarity, one of the three tasks this model performs. Sentence-BERT (SBERT) remedied this computational limitation without major performance sacrifices.

2.2 SBERT

SBERT is a modification of the pre-trained BERT model that utilizes semantically meaningful sentence embeddings for tasks including semantic textual similarity (Reimers and Gurevych, 2019). By deriving fixed-size sentence embeddings, researchers can use linear algebra techniques to calculate similarity scores. There are several methods for deriving these sentence embeddings. SBERT utilizes an added pooling layer to the BERT output, testing three pooling strategies. They used the CLS-token output, the calculated mean of output vectors, and the maximum of output vectors. Additionally, SBERT used siamese and triplet BERT networks to handle tasks that require input sizes of two and three sentences respectively. To calculate the similarity between sentence embeddings, SBERT uses several objective and loss functions. Results found that the best combination of pooling and concatenation strategies achieved the best performance at the time in 5 out of 7 tasks, including multiple sentiment prediction tasks.

2.3 Further Extensions of SBERT

Since the publication of SBERT, there has been expanded interest in improving sentence embeddings. Some models did not extend and improve SBERT's derivation technique, opting for other methods. Li et al. (2020) explored a smooth and isotropic Gaussian distribution for sentence embeddings that showed significant performance improvements for semantic textual similarity tasks. Gao et al. (2021) utilized a contrastive learning framework that uses an input sentence to predict itself, allowing for more uniform sentence embeddings.

Other research methods have been dedicated to extending and improving SBERT's derivation techniques. One example is (Peng et al., 2021), which incorporated structural information to obtain improved sentence embeddings that enable their model to outperform SBERT and other sentence encoders on semantic textual similarity tasks.

3 Approach

3.1 Baseline Model

We use the 224n minBERT model we developed as a baseline. This model serves as a basic, smaller version of the BERT model. The model consists of 12 encoder transformer layers, each of which has multi-head attention, an additive and normalization layer with a residual connection, a feed-forward layer, and a second additive and normalization with a residual connection. Our training consisted of training over all three tasks, with different loss functions for each task. For sentiment classification, we used the provided cross-entropy loss. For paraphrase detection, we used binary cross entropy. For semantic textual similarity, we used the same loss function as sentiment classification, although with different parameters. For each task, we use a linear layer and dropout to finetune the min-BERT model after pre-training. For more details, see the 2024 CS224n default project handout.

3.2 Speedy SBERT Model

The sentence BERT model is a fairly simple modification on top of our existing minBert model. There are two main differences. The first involves the embeddings we use to feed into the prediction functions of our model. The default minBert model returns a pooled embedding in the form of the

embedding for the CLS token of the sentence. This CLS embedding, performed adequately for the sentiment classification and paraphrase detection tasks but resulted in a very low correlation for semantic similarity. Instead, as in Reimers and Gurevych (2019), we sought to swap to more meaningful embeddings for sentences, and did so by conducting mean pooling when retrieving our embeddings from BERT. Mean pooling involves taking the geometric mean of all of the token embeddings in the sentence to be our semantically meaningful embedding as opposed to simply using the CLS token's embedding for the sentence. Our sentence BERT model then uses these embeddings for all 3 of the tasks as seen in Figure 1. Similar to the baseline model, our model passes the sentence embeddings into two separate linear layers for sentiment classification and paraphrase detection. Our model greatly differs in architecture from BERT for semantic similarity: we leverage the new embeddings by using cosine similarity. Cosine similarity achieves correlation scores of roughly 0.8 on the unaltered pre-trained BERT model we implemented.

Additionally, we attempted to use cosine similarity for the paraphrase task but found that it performed worse than our baseline linear layer (when run on the new embeddings). As in the baseline model, we use cross entropy as our loss function for sentiment classification. For paraphrase detection, we use binary cross entropy as our loss function. For semantic similarity, we swapped to mean squared error loss.

We also implemented a new training method. We first train the model on STS, which modifies the BERT parameters as there is no additional linear layer involved. Then, these parameters are frozen before training the other two tasks. Since the other two tasks both have their own linear layers, we skip back-propagating on BERT and instead, solely back-propagate over the linear layers. This approach cuts 80% of the computation when training for these two tasks, with only a small reduction in accuracy.

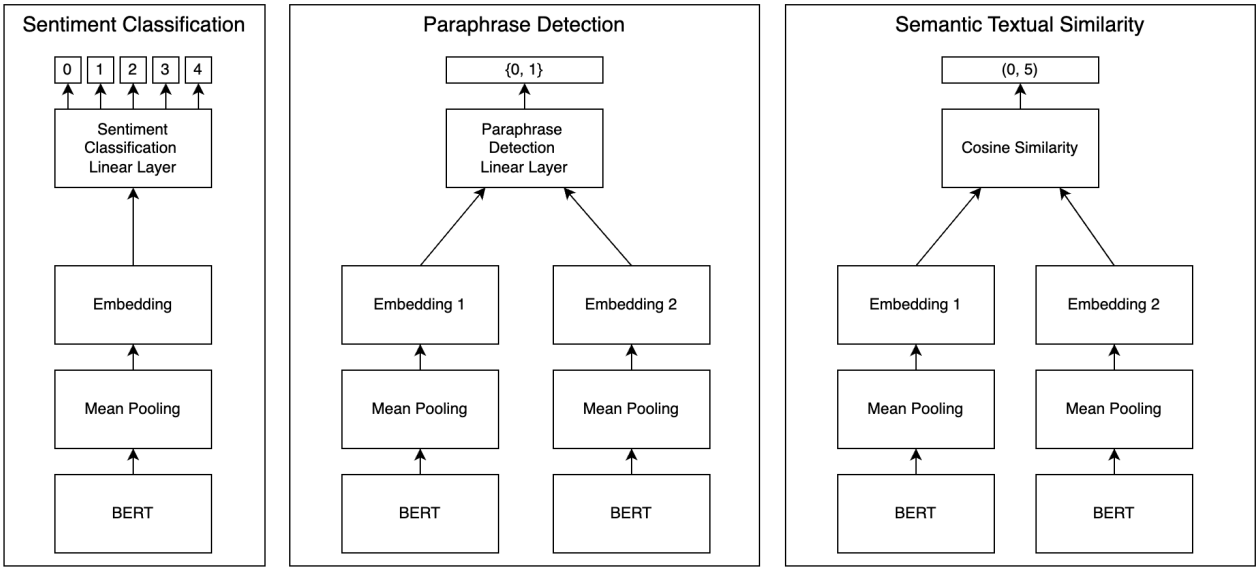

Figure 1: Speedy SBERT's architecture.
Note: the output of cosine similarity computed for semantic textual similarity is then scaled by 5 to match the desired 0, 1, ..., 4 similarity labels.

4 Experiments

4.1 Data

Dataset	Train	Dev	Test	Total	Example Format	Label Range
Stanford Sentiment Treebank	8,544	1,101	2,210	11,855	Single	0, 1, ..., 4
Quora Dataset	141,506	20,215	40,431	202,152	Pairs	0 or 1
SemEval STS Benchmark	6,041	864	1,726	8,631	Pairs	0, 1, ..., 5

Table 1: Datasets used for training and testing the model developed in this project.

We are using three datasets: the Stanford Sentiment Treebank (SST) dataset, the Quora Dataset, and the SemEval STS Benchmark Dataset. Table **??** is a visual representation of key information about each dataset used for the development of this model.

- **4.1.1 Sentiment Analysis**

  The SST dataset, provided to us in the project spec, will be used for the sentiment analysis task. In sentiment analysis, the model classifies how positive, negative, or neutral a sentence is. This dataset classifies sentences as negative, somewhat negative, neutral, somewhat positive, or positive, with labels ranging from 0 to 4.

- **4.1.2 Paraphrase Detection**

  The Quora dataset, provided to us in the project spec, will be used for the paraphrase detection task. In paraphrase detection, the model outputs whether or not two sentences convey the same semantic meaning. In this model, a pair of sentences is input, and the model outputs 0 or 1, the former signifying not paraphrased, and the latter paraphrased.

- **4.1.3 Semantic Textual Similarity**

  The SemEval STS Benchmark dataset, provided to us in the project spec, will be used for the semantic textual similarity (STS) task. In semantic textual similarity, the model outputs how semantically similar two texts are. In this model, a pair of sentences is input, and the model outputs a label ranging from 0 to 5 representing how similar the sentences are (5 being the exact same and 0 completely different).

4.2 Evaluation method

We use the evaluation methods defined in the default project handout. We use the accuracy for sentiment analysis and paraphrase detection. For semantic textual similarity, we will use Pearson correlation of the true similarity values against the predicted similarity values. Additionally, we examined the iterations per second during each training task as a way of analyzing training efficiency.

4.3 Experimental details

4.3.1 Model Architecture Testing

When we set out on this research, we decided we wanted to use cosine similarity instead of neural networks in our multitask classifier to optimize performance. To that end, we decided to run some experiments to determine the increase in performance for using cosine similarity instead of linear layers. Our baseline uses linear layers for all three tasks, for SST we pass the mean pooled embedding into our SST linear layer, and for paraphrase and STS we concatenated our mean pooled embeddings and passed them into the STS and paraphrase linear layers respectively. Then, in our experimental cases for paraphrase and STS, we instead return the cosine similarity of the two mean pooled embeddings. (SST remains the same during this experiment because it only takes one sentence as input)

4.3.2 Dropout Testing

In our preliminary testing, our train loss for sentiment classification was low and our training accuracy was very high, while our development accuracy was mediocre. In order to counteract

over-fitting to the training data, we decided to test for an optimal dropout probability for our model. Due to the time required to finetune using the full Quora dataset, for each epoch we used a random selection of 1,000 batches from the dataset shuffled separately for both training and evaluating the model with the different dropout probabilities.

4.3.3 Efficiency Testing

Once we had our general model architecture and our dropout set appropriately, we noticed a significant difference in the training speed of the model when running the `multitask_classifier` with the `pretrain` flag versus with the `finetune` flag. With the `pretrain` flag set, the model freezes the BERT parameters. Alternatively, when the `finetune` flag is set, the training back-propagates over BERT as well, modifying the whole model. Because we trained our model with the `finetune` flag on SST, Paraphrase, and STS every epoch, this increased computation time caused our model to take an extremely long time to train (roughly 11 hours). We knew that this back-propagation was important for our method of training semantic textual similarity. This is because our using cosine similarity resulted in no additional neural network whose parameters would be adjusted based on the loss function when training STS. This means that changing the BERT parameters was the only way to increase its performance. Because of this, we reworked our training function to fully train our STS task at finetune's slower speed, followed by freezing the BERT parameters and finally training SST and paraphrase detections. We called this more optimized, speedier BERT model, Speedy Sentence BERT, or Speedy SBERT for short.

4.4 Results

4.4.1 Model Architecture Results

Using cosine similarity leads to overall performance gains when compared to the baseline as seen in Table 2, resulting in an overall score of 0.614, higher than the baseline overall score of 0.540. We also noticed though that the Paraphrase accuracy when using cosine similarity was quite a bit lower than the baseline paraphrase accuracy, so we reran the test swapping back to the baseline implementation of paraphrase. In this new text, we found that using the baseline neural network implementation of paraphrase, and the cosine similarity implementation of semantic textual similarity raised the overall score to 0.696, giving us our finalized model architecture.

	Sentiment (SST)	**Paraphrase (Quora)**	**Similarity (STS)**	
Model	Test Accuracy	Test Accuracy	Test Correlation	Overall Score
Baseline	0.397	0.672	0.101	0.540
Paraphrase (Cosine Similarity)	0.457	0.472	0.824	0.614
Paraphrase (Neural Network)	**0.507**	0.668	0.824	**0.696**
Speedy SBERT	0.463	**0.708**	**0.833**	**0.696**

Table 2: Results of our models and the baseline on the test set.

4.4.2 Dropout Results

Table 3 depicts our preliminary testing on the effect of various dropout rates applied before the linear layer for sentiment classification and paraphrase detection. This combination may potentially indicate overfitting due to a too-low dropout rate, motivating these tests to identify an ideal dropout rate.

Due to the time required to finetune using the full Quora dataset, each epoch we used a random selection of 1,000 of the dataset pairs for both training and evaluating the model. From these tests, we found that the ideal dropout rate for the linear layers used in the sentiment classification and paraphrase detection tasks is 0.3. Although the accuracy for paraphrase detection with a dropout of 0.3 is not the highest of the tests, our inclusion of the paraphrase accuracy was to ensure that finding an ideal dropout rate for sentiment analysis doesn't significantly affect our paraphrase accuracy. Moreover, these paraphrase accuracies are not exact due to the subset they are trained and evaluated on, further motivating their insignificance in this experiment.

	Training Accuracies		Development Accuracies	
Dropout Level	Sentiment (SST)	Paraphrase (Quora)	Sentiment (SST)	Paraphrase (Quora)
0.2	0.971	0.752	0.490	0.722
0.3	0.984	0.761	**0.512**	0.737
0.4	0.988	0.760	0.476	0.738
0.5	0.986	0.763	0.469	0.739

Table 3: Results on dev datasets, using 1,000 sentence pairs from the Quora dataset for paraphrase detection at a time.

4.4.3 Efficiency Results

As seen in Table 4, our efficiency experiment showed a massive increase in the number of iterations per second for Sentiment Classification and Paraphrase Detection by freezing the BERT model after STS. Sentiment classification and paraphrase detection trained 5x faster, allowing us to increase the model's epochs, and get an overall better model. Although the model trained much faster, the accuracy improvement for each epoch was less. When run with the BERT freeze training method for 15 epochs, the overall accuracy is on par with the accuracy of our standard training approach after 10 epochs as can be seen in the data in Table 5 (where by some miracle they got the exact same overall score). Nevertheless, the BERT freeze method completed 15 epochs in 4 hours, while the standard model that runs 10 epochs takes 11 hours. Additionally, we found that the standard approach of calculating train accuracy every epoch was taking up more time than actually training for the epoch due to the size of the paraphrase training set. In order to remedy this, when calculating our train accuracy in each epoch, we randomly selected 500 batches from the training dataset to evaluate, decreasing each epoch duration from 30 minutes to only 10.

Task	Standard Train	Freeze BERT after STS
Sentiment (SST)	$\sim$ 8 it/s	$\sim$ 40 it/s
Paraphrase	$\sim$ 6 it/s	$\sim$ 30 it/s
Similarity (STS)	$\sim$ 6 it/s	$\sim$ 6 it/s

Table 4: Iterations per second of classic SBERT compared to our novel Speedy SBERT

	Sentiment (SST)	**Paraphrase (Quora)**	**Similarity (STS)**	
Training Method	Test Accuracy	Test Accuracy	Test Correlation	Overall Score
Standard Train	**0.507**	0.668	0.824	**0.696**
Speedy SBERT	0.463	**0.708**	**0.833**	**0.696**

Table 5: Results of our model with both training configurations.

5 Analysis

Our results for our model architecture testing were quite surprising. We expected to see substantial improvements in accuracy for both the paraphrase and STS tasks after implementing cosine similarity, but instead only saw meaningful improvement for STS. Cosine similarity resulted in a pretty significant decrease in the paraphrase accuracy. Despite our initial assumption that paraphrase detection was a less granular version of semantic similarity, our results showed that these tasks were more different than we had anticipated. The results of this experiment caused us to change our model architecture to use a linear layer for the paraphrase task, only utilizing cosine similarity for the STS task.

We also noticed anecdotally that even with no modification of BERT's parameters (i.e. no finetuning), our STS predictions using cosine similarity and our new mean pooled embeddings achieved a correlation score of 0.833. This was 8 times higher than the trained neural network used with the `<CLS>` tokens, showing that the `<CLS>` tokens do a poor job of capturing meaningful information about the sentences.

The Speedy SBERT model also tended to have much smaller differences between the training accuracy and the development accuracy, making it less prone to over-fitting, theoretically allowing it to continue improving after many epochs. Despite attempting to improve sentiment classification's overfitting by testing various dropout rates on the sentiment classification linear layer, We saw only minor improvement from the baseline.

6 Conclusion

Overall, we achieved improvements in all three tasks, though with the most improvement in similarity textual similarity, as that was the principal goal of using an SBERT-based model. Our most significant novel finding was our Speedy SBERT's speed optimization by freezing the BERT model once STS training was completed. This optimization allowed us to run code less prone to over-fitting without conducting unnecessary back-propagation calculations, instead only performing these calculations when necessary to improve model accuracy.

Our main takeaway as researchers is that the approaches that should theoretically work the best don't always actually turn out that way. Sometimes instead of doing lots of theoretical work to find something that will definitely be better, it's more effective to try outside the box methods that might work. We are especially proud of our modified, more efficient training approach, because this isn't something we read about in a paper, but instead something novel that we came up with without outside influence. We did however, get this training method working rather close to the deadline and were not able to test its limits when it comes to its slow accuracy improvements. Based on the way the data was trending, we think it had the potential to score even higher if run for many many epochs, as the over-fitting we observed was far less than in our other models.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Conference on Empirical Methods in Natural Language Processing*.

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. On the sentence embeddings from bert for semantic textual similarity. *ArXiv*, abs/2011.05864.

Qiwei Peng, David Weir, and Julie Weeds. 2021. Structure-aware sentence encoder in bert-based siamese network. In *Workshop on Representation Learning for NLP*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.