Integrating Cosine Similarity into minBERT for Paraphrase and Semantic Analysis

Stanford CS224N Default Project

Gerald J. Sufleta II
Department of Computer Science
Stanford University
`sufleta@stanford.edu`

Abstract

In Natural Language Processing (NLP), efficient and semantically rich sentence embeddings has been a driving force for innovation. This paper investigates harnesses miniBERT, a scaled down version of the default BERT architecture, while trying to enhance it with sentence embeddings through Siamese BERT-networks (SBERT). The objective was to have a balance between computational efficiency and performance in downstream tasks. Leveraging insights from Reimers and Gurevych (2019), miniBERT inherits the efficiency enhancements of SBERT while striving to maintain high performance. This paper highlights my attempts to harnass the architecture and performance evaluation of miniBERT across various tasks, including sentiment analysis, paraphrase detection, and semantic textual analysis, with the upgrades proposed by the SBERT model. Experimental results discussed herein highlight the challenges of overfitting, performance plateauing, and task antagonism, particularly in the fine-tuning phase. Strategies such as progressive layer unfreezing, task-specific optimization heads, selective backpropagation, and early stopping are discussed as potential avenues for mitigating these challenges. The study provides insights into the nuanced dynamics of model optimization and suggests directions for future research to enhance model robustness and performance across diverse NLP tasks.

1 Key Information

- Mentor: Rohan Taori
- External Collaborators: None
- Sharing project: Yes

2 Introduction

Deep learning has transformed NLP, especially on the development of sophisticated sentence embedding techniques that can capture semantic richness and versatility. The work of Devlin et al. (2019) with BERT introduced a new paradigm in language understanding, using deep bidirectional transformers to extract contextually enriched embeddings. This work brought an understanding of language nuances, paving the way for advanced applications in text analysis, translation, and sentiment analysis.

However, despite the achievements of BERT, its application in real-world scenarios is hamstrung by computational and efficiency constraints, like when processing extensive sets of sentences. Recognizing this limitation, Reimers and Gurevych (2019) proposed Sentence-BERT (SBERT), an adaptation of the BERT architecture, aiming to overcome computational bottlenecks through the use of Siamese and triplet network architectures. This modification significantly reduced the time required for generating sentence embeddings, and enhanced BERT's utility in applications requiring semantic comparison.

While these advancements were significant steps in the field of NLP, they also show the difficulties in balancing computational efficiency with the need for semantic understanding. The introduction of miniBERT, a scaled-down variant of BERT, attempts to reconcile these aspects by offering a compact yet robust solution for generating sentence embeddings. This paper builds upon the foundational work of Devlin et al. (2019) and Reimers and Gurevych (2019), exploring the integration of cosine similarity measures into miniBERT to further enhance its capability for paraphrase detection and semantic analysis, as well as using MEAN-pooling to generate the embeddings.

Drawing inspiration from Reimers and Gurevych (2019), this study seeks to use the architectural efficiencies of SBERT within the miniBERT framework, aiming to create a model that not only has high performance in downstream tasks but also addresses the computational limitations that often hurts the scalability of such models. The challenges of overfitting, performance plateauing, and task antagonism, particularly prevalent during the fine-tuning phase of model training, are critically examined.

Moreover, the empirical investigation discussed in this paper highlights the need for model optimization and performance, offering insights into the balance required to achieve both computational efficiency and semantic richness in NLP models. Through an analysis of experimental results across various NLP tasks, including sentiment analysis, paraphrase detection, and semantic textual similarity, this paper demonstrates the interface between model architecture, optimization strategies, and task-specific performance.

On reflection, the integration of miniBERT with SBERT's efficiency has unveiled a challenging landscape, characterized by a performance plateau during fine-tuning and a nuanced antagonism among tasks such as semantic similarity, paraphrase detection, and sentiment analysis. This experience has illuminated the balancing act required in optimizing language models for multiple NLP tasks, where progress in one area might inadvertently impede another. My findings emphasize the critical need for innovative optimization strategies that navigate the complexities of diverse NLP tasks.

3 Related Work

In NLP, the driving force behind much innovation has been the search for efficient and semantically rich sentence embeddings. BERT stormed the stage when it was introduced by Devlin et al. (2019), by introducing a new way of teaching computers to understand language by looking at words in sentences from both directions at once. Unlike older models that read text sequentially, like RNN's, BERT reads the whole sentence together. This approach helps it get a better idea of what each word means based on the words around it. At the core of BERT is Multi-headed attention, which permits it to focus on different parts of a sentence at the same time to better understand it. Succinctly stated, BERT has a combination of heads that use query, keys, and vectors to simultaneously focus on different parts of a sentence to extract semantic meaning.

This, however, is a computationally intensive way of doing things, especially when comparing large sets of sentences. The seminal work by Reimers and Gurevych (2019) (SBERT) addressed these limitations, and attempted to overcome them by modifying the default BERT architecture. Using the power of Siamese and triplet networks to produce fixed-size sentence embeddings, Reimers and Gurevych (2019) aimed to execute BERT's Multi-headed attention more quickly. Indeed, they were able to take a task that would usually take upwards of 65 hours down to 5 seconds.

In essence, SBERT changes the BERT architecture, allowing it to make comparisons using cosine similarity, and also uses MEAN pooling of output vectors to construct semantically meaningful sentence embeddings. SBERT still remains hamstrung, however, as its performance relies on BERT's pre-trained models and the empirical focus on English language datasets, which raises questions about the transferability of the SBERT's enhancements across various languages and text genres.

This project uses miniBERT as its launching point, a more compact and faster version of BERT, and draws from SBERT's efficient computational framework . The more compact miniBERT tries to maintain high efficiency and performance for downstream tasks.

4 Approach

For my approach to the problem, classifier neural architecture, I used a composition of a dropout layer to prevent overfitting, followed by a first linear layer that maps the hidden states to an intermediate representation. Then, the first linear layer feeds into a ReLU activation function for non-linearity, and finally, outputs to a final linear layer.

For the downstream tasks, the neural architecture was similar to the first. The Sentiment Stack used a dropout layer followed by a linear layer for classification. Meanwhile, the Paraphrase Detection Stack and the Semantic Similarity Stack both utilized similar architectures to one another, including dropout layers, linear transformations, with a BatchNorm1d and GELU activation function sandwiched between the two linear transformations.

The dropout layers were inserted in order to help combat overfitting, although as will be detailed below, significant overfitting was observed. The inclusion of BatchNorm1D layers in the model was intended to stabilize and accelerate training by normalizing the activations of previous layers, attempting to avoid issues like vanishing or exploding gradients. Finally, the GeLU layer was to allow the model to learn non-linearities.

The last layer I included was a cosine similarity layer in both the Paraphrase Detection Stack and the Semantic Similarity Stacks. The inclusion of this layer was premised on the idea it could help the model to discern relationships between text segments. The cosine similarity is computed as follows: $CosineSimilarity(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$

Influenced by Reimers and Gurevych (2019), in conjunction with the cosine similarity layer I used a MEAN Pooling strategy for token embeddings instead of the default `[CLS]` token. The MEAN Pooling approach averages the embeddings across all tokens, yielding a single vector that encapsulates the semantic information of the input sequence:

$MEAN(S) = \frac{1}{|S|} \sum_{i=1}^{|S|} h_i$, where S denotes the number of tokens in each sentence, and h_i denotes the embedding of the i-th token.

5 Experiments

In total, there were two main experiments that I ran. The first was performing sentiment analysis on the Stanford Sentiment Treebank (SST) and the CFIMDB movie review database. The second experiment was to perform senitment analysis, paraphrase detection, and semantic textual analysis on the SST, Quora, and SemEval datasets, respectively.

5.1 Data

In the initial phase of the project, my focus was on two main datasets: the SST (Socher et al., 2013) which contains over 11,000 sentences extracted from movie reviews, and the CFIMDB dataset comprising 2,434 highly polar movie reviews. The primary goal was to assess the performance of the miniBERT model I developed for sentiment analysis on these datasets.

For part two, I used SST dataset (Socher et al., 2013) for sentiment analysis, along with the Quora dataset (Quora, 2017) for paraphrase detection and the SemEval dataset (Fernando and Stevenson, 2009) for semantic textual analysis. My analysis on these datasets utilized the ideas outlined by Reimers and Gurevych (2019); namely, MEAN pooling and cosine similarity.

5.2 Evaluation method

In the initial phase of the project, sentiment analysis was conducted on the SST and CFIMDB datasets. Utilizing the miniBERT model, attempts were made to accurately predict sentiment, which had been labeled by human reviewers. These datasets were partitioned into three subsets: one for training the system, another for developing it, and a third for testing its performance.

In the subsequent phase, the three datasets were similarly divided into train, development, and test sets. This three-fold division facilitated comprehensive evaluation and comparison of the model's performance across diverse datasets.

5.3 Experimental details

In my exploration of optimizing the miniBERT implementation, I conducted a series of experiments, categorized into two distinct phases based on the computing environment: the first group on my personal Laptop NVIDIA 4070 GPU and the second on a Google Cloud Platform NVIDIA T100. This dual-approach allowed for a comparative analysis under differing computational capabilities. Here, I explore the details regarding the hyperparameters, specifically dropout probability, learning rate, weight decay, epochs, and their impact on model performance across both batches.

5.3.1 First Group of Models

For the initial batch trained on my Laptop NVIDIA 4070 GPU, I adhered to a consistent dropout probability of 0.3, which was set as the default across all experiments. This constant allowed me to assess the influence of other hyperparameters more accurately without the confounding effects of varying dropout rates. The training time per epoch ranged 11 minutes and 45 seconds, when the batch size was only 8, to 27 minutes and 38 seconds with a batch size of 16; interestingly, the highest time per epoch at 32 minutes was when the batch size was also 8, suggesting variability in training beyond mere batch size.

Throughout this phase, I experimented with a range of weight decay values and epochs. Starting with a weight decay of 0.001 and 2 epochs for the initial model, I adjusted these parameters in subsequent models to observe their impact. Notably, an increase in weight decay to 0.0075 and epochs to 5 in one of the intermediate models did not substantially ameliorate overfitting issues. The learning rates used ranged from $5e-6$ to $1e-4$.

This overfitting became starkly apparent in the final model of the first group of models, where despite achieving near 98% training accuracies, the development accuracies lagged significantly behind by over 35%. The persistent overfitting underscored the limited efficacy of these hyperparameter adjustments in combating the issue, highlighting the need for a more nuanced approach.

5.3.2 Second Batch of Models: Parameter Exploration

In the second group, conducted on the Google Cloud Platform NVIDIA T100, I aimed to refine the model further, starting with a stronger pretrained model basis in terms of Pearson accuracy. The time to train per epoch ranged from almost 13.5 minutes to, at the high end, 54 minutes.

This batch exhibited a broader exploration of weight decay and epochs. Initially leveraging a model with a weight decay of 0.0075 and 5 epochs, which provided a relatively better Pearson accuracy, I attempted to push the limits of model performance. However, aggressive adjustments, particularly with higher learning rates, resulted in deteriorating Pearson correlation scores, indicating that while accuracy metrics might show improvement, the model's ability to generalize and capture nuanced relationships within the data was compromised.

5.4 Results

5.4.1 Comparison Between both Groups of Models

In comparing the performance of two sets of models—those trained on a Laptop NVIDIA 4070 GPU and those on a Google Cloud Platform NVIDIA T100—distinct trends in development accuracy are evident. The models trained on the laptop GPU, while subjected to various hyperparameter adjustments, failed to overcome overfitting. This was clear as they showed high training accuracies with a considerable drop in development accuracies, suggesting a disparity in learning generalizable features.

On the other hand, the models trained on the T100, which started with a more competent pretrained base, showed a less pronounced improvement in development accuracies. Adjusting the learning rate aggressively improved the accuracies but at the cost of Pearson correlation scores, which plummeted.

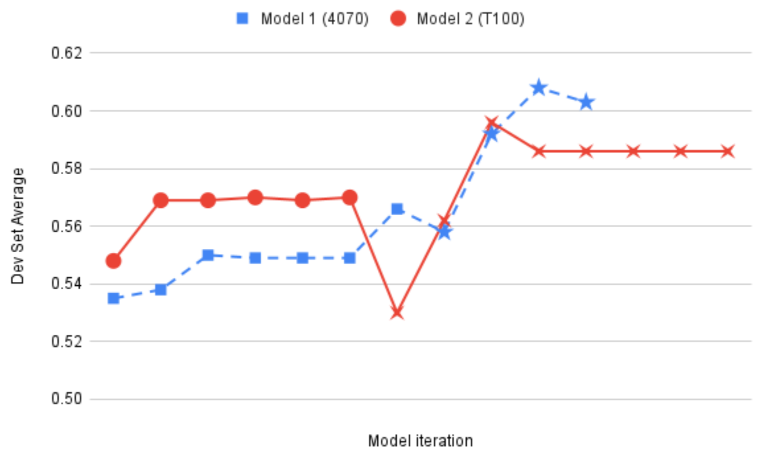

Figure 1: Dev Set Average for Each Group of Models; Square and Circle Marks Denote Prerained Models while Star and X-marks Denote Finetuned Models

On the contrary, a conservative learning rate yielded little improvement from the baseline pretrained model, indicating a potential plateau in learning efficacy.

The line graph representing development average performance in Figure 1, above, further illustrates these findings, with the T100 models displaying a less volatile progression in development accuracy. In particular, while there's an observable increase in performance from the baseline, the models fail to sustain a continuous upward trajectory as seen with the 4070 models. This plateau suggests that further tuning on the T100 may require different strategies than those effective on the 4070 GPU. The graph shows that despite a stronger starting point with the T100 models, the optimization paths taken did not lead to a proportionate increase in development accuracy, reflecting the nuanced challenges in transferring improvements in model pre-training to final performance gains.

5.4.2 Test Set Results: Analysis and Reflections

The empirical results of my experimentation, while informative, did not align with my initial expectations. In the pre-training phase, the models were set on an upward trajectory, with higher learning rates inducing rapid advancements which then reached a plateau. This was particularly evident with the first group of models, where the anticipated continuous improvement in tasks such as paraphrase detection and sentiment analysis was met with early stagnation. The second batch, despite the advantage of an improved pre-training and the implementation of a smaller learning rate, echoed this trend, manifesting in a slight increase in Pearson correlation but an eventual plateau.

Fine-tuning further underscored the pitfalls of my approach. The first batch of models was hampered by overfitting almost immediately, halting any progressive gains. The second group's models, though benefiting from a refined pre-training phase, did not fulfill the heightened expectations I had for it. Despite a strategic reduction in learning rate and an increased dropout rate to prevent overfitting, these models too plateaued without significant progress.

Here are the summarized test results:

Metric	Score
SST test accuracy	0.535
Paraphrase test accuracy	0.596
STS test correlation	0.380
Overall test score	**0.607**

Table 1: Summary of Test Results

This realization suggests that the gradient descent paths for the various tasks were in contention, with one task's optimization process possibly overshadowing the rest. This discordance within the learning objectives might explain why one task's performance enhanced at the expense of another, indicating a need for more nuanced task-specific optimization strategies.

5.4.3 Training Time

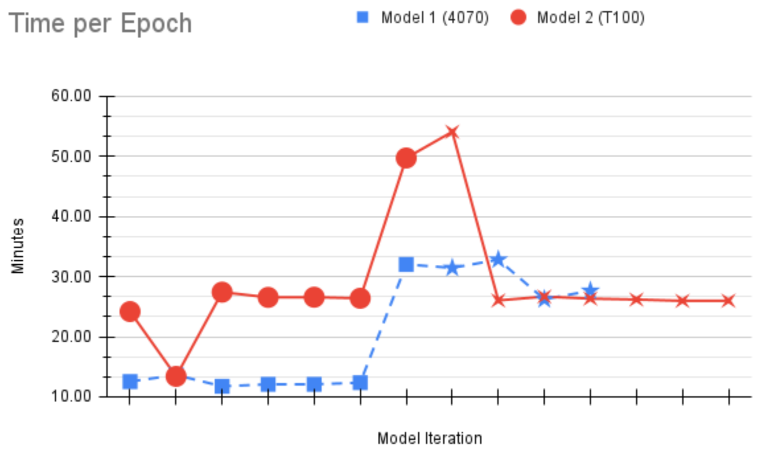

Figure 2: Training time per epoch; Square and Circle Marks Denote Prerained Models while Star and X-marks Denote Finetuned Models

Training time per epoch differed significantly between the two model architectures, as can be seen in Figure 2 (above). Model 1, trained on the NVIDIA 4070, showed fluctuation in training times, with a spike during fine-tuning suggesting periods of intensive computational load or periods of model adjustment. In contrast, Model 2 trained on the NVIDIA T100 on Google Cloud Platform presented a more uniform, though higher, time-per-epoch across iterations.

Several factors can explain these discrepancies in training times. The variable demand on cloud services, such as those provided by Google Cloud Platform, can lead to inconsistent availability of computational resources. This variability can result in fluctuating training times, as the allocated server performance may differ from one training instance to another, especially on shared resources that serve multiple users concurrently.

Furthermore, the underlying architecture of the GPUs plays a role. The NVIDIA T100's architecture is designed for optimal performance with cloud-based services, possibly leading to longer training times per epoch due to differences in processing efficiency compared to the laptop-based NVIDIA 4070.

The use of different hyperparameters across the two models also contributes to variations in training times. For instance, Model 2 utilized larger batch sizes (either 16 or 32), different learning rates that were often orders of magnitude smaller than those of Model 1, and varying dropout rates; whereas, Model 1 used batch sizes of 8, 12, and 16, with a consistent dropout probability of 0.3 (the default). Larger batch sizes can lead to increased computation per epoch, while lower learning rates may require more epochs for the model to converge, affecting overall training duration. Similarly, adjustments in dropout rates can change the complexity of the training process, as they shift the network's ability to generalize from the training data, potentially increasing the training time required to achieve optimal performance.

We see that training on a cloud platform such as Google Cloud requires accounting for fluctuating computational resources, which can introduce variability in training times. The architectural differences between the GPUs must also be considered, as they directly impact the efficiency of model training. Finally, the choice of hyperparameters needs to be considered, as they can significantly alter both the duration and the effectiveness of the training process.

6 Analysis

The plateauing of model performance during pre-training suggests that while the model initially learned useful representations from the data, it quickly exhausted the knowledge it could extract given its architecture and the complexity of the tasks. The model might have gained from more diverse training data, or perhaps the architecture should have been adjusted to capture more nuanced patterns.

Despite the simplicity of the sentiment stack, the model demonstrated a reasonable level of understanding. However, the MEAN pooling technique used and the straightforward nature of the sentiment classification task may have led to a ceiling on the performance that could be achieved. Moreover, the sentiment stack's simplicity, although beneficial for reducing overfitting, might also have limited its ability to capture the necessary complex patterns. This may have caused the plateauing seen during pre-training and the overfitting during fine-tuning.

For the paraphrase detection and semantic textual similarity tasks, the cosine similarity on mean pooled embeddings was a strong approach. The cosine similarity is an effective measure for these tasks due to its focus on the angle between embedding vectors. However, it is possible that the simplicity of the sentiment stack versus the complexity of the other stacks antagonized each other.

7 Conclusion

The main takeway of this project was the realization of how difficult training an NLP model really is. While certain aspects of it were relatively straightforward, such as implementing the forward functions, sentence embeddings, and even the neural architecture, there is an art form to tuning these models that came into relief as I began to train the model.

Particularly, the delicate dance between pretraining and finetuning showed how complex it can be to make these models excel at their given tasks. I was not prepared for the antagonism between different tasks, yet this aspect of it was a revelation and taught me a lot about how models work. This experience showed the importance of a nuanced understanding of model behavior across various domains, pushing me to think critically about the trade-offs involved in model optimization. Balancing performance on one task without detrimentally impacting another became a pivotal focus of my work, shedding light on the interdependencies that exist within the NLP landscape.

Moreover, the challenge of addressing overfitting, while ensuring the model retains its generalization ability, further highlighted the nuanced skill set required in this field. Each adjustment, whether in the learning rate, dropout layers, or architecture tweaks, had a profound impact on outcomes, illustrating the meticulous precision needed in fine-tuning NLP models.

7.1 Avenue for Future Work

In hindsight, a possible different approach might have been to implement progressive layer unfreezing to facilitate finer learning without the problem of competing gradients. Additionally, task-specific optimization heads could have provided more focused learning, potentially allowing for balanced improvements across tasks.

To further address the challenges of overfitting, model plateauing, and antagonistic task performance, I would consider using selective backpropagation to help the model's learning capacity to be better. By focusing updates on underperforming tasks, the model might have better balanced its performance across all tasks. Also, adversarial training could have improved the model, forcing it to learn more general representations and possibly breaking through the plateauing by exposing the model to more difficult training.

Early stopping for each task might have helped with the overfitting when fine-tuning, because tasks that reached their plateau could have been held constant, preserving them while allowing other tasks to continue learning. Lastly, automated hyperparameter optimization could have explored a wider configuration space, potentially finding settings that could lead to better performance across tasks, reducing the likelihood of overfitting, and pushing past performance plateaus.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Samuel Fernando and Mark Stevenson. 2009. A semantic similarity approach to paraphrase detection. *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*.

Quora. 2017. First quora dataset release: Question pairs. `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`. Accessed: February 13, 2024.

Nils Reimers and Iryna Gurevych. 2019. 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.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.