Implementing and Fine-Tuning BERT for Sentiment Classification, Paraphrase Detection and Semantic Similarity Analysis

Stanford CS224N Default Project

Anqi Zhu
Department of Civil and Environmental Engineering
Stanford University
anqizhu@stanford.edu

Antonio Torres Skillicorn
Department of Civil and Environmental Engineering
Stanford University
ats2258@stanford.edu

Kyra Sophie Kraft
Department of Sustainability Science and Practice
Stanford University
kskraft@stanford.edu

Abstract

Transformer-based large language models have the potential to aid in automating numerous previously laborious tasks. We implement a version of the original BERT model and apply this to the tasks of sentiment analysis, paraphrase detection, and evaluation of semantic textual similarity across several datasets. Our model exceeds baseline accuracies for sentiment classification on the Stanford Sentiment Treebank (SST) and CFIMB datasets. For additional tasks, we observe large improvements in performance with the optimization of model hyper-parameters, including an increase in the learning rate, a decrease in batch size, an increase in the number of epochs, and a fine-tuning of the full model as opposed to the last-linear layer. We also successfully implement regularized fine-tuning and cosine similarity fine tuning achieving an overall accuracy of .594 for our multi-classifier, and a correlation for our STS task of .479. These improvements demonstrate the power small changes in hyper-parameters and model architecture on predictive performance.

1 Key Information to include

- TA mentor: Josh Singh
- External collaborators (if no, indicate “No”): No
- External mentor (if no, indicate “No”): No
- Sharing project (if no, indicate “No”): No

2 Approach

2.1 Introduction/Overview

The CS224N default project consists of three main tasks, including an implementation of certain layers of the original Bert Model, an application of this BERT model to sentiment analysis, and finally, an improvement of BERT's performance on additional tasks. The additional tasks in question include sentiment analysis, paraphrase detection and evaluation of semantic textual similarity. The following subsections explore these different aspects of the project in more depth, outlining our approaches, baselines, and original contributions.

2.2 Implementing minBERT

In the context of this project, the implementation of minBERT primarily includes creating transformer layers of the model, including the multi-head self-attention mechanism. Figure 1 demonstrates the importance of multi-head self attention as the backbone of the BERT architecture.

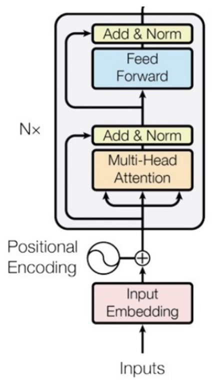

Figure 1[1]: Encoder Layer of Transformer used in BERT, with Multi-Head Self Attention as a Key Mechanism

In order to implement the multi-head attention layer, which is responsible for transforming hidden states for each element while considering all others, we leverage equations 1 - 5. These equations demonstrate how multiple attention heads perform dot-product attention, therefore representing a sequence element as a weighted sum of all sequence elements' hidden states, allowing for different amounts of focus in specific spaces. These head outputs are concatenated together, linearly transformed, normalized, and fed into a feed-forward network before having drop-out applied.

Attention Equations[2]

[1]Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2023). Attention is all you need. arXiv. https://arxiv.org/abs/1706.03762v7

[2]Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171–4186). Association for Computational Linguistics.

$$\text{Attention}_i(\mathbf{h}_j) = \sum_t \text{softmax}\left(\frac{W_i^q \mathbf{h}_j \cdot W_i^k \mathbf{h}_t}{\sqrt{d/n}}\right) W_i^v \mathbf{h}_t \tag{1}$$

$$\text{MH}(\mathbf{h}) = W^o \left[\text{Attention}_1(\mathbf{h}), ..., \text{Attention}_n(\mathbf{h})\right] \tag{2}$$

$$\text{SA}(\mathbf{h}) = \text{FFN}(\text{LN}(\mathbf{h} + \text{MH}(\mathbf{h}))), \tag{3}$$

$$\text{FFN}(\mathbf{h}) = W_2 f(W_1 \mathbf{h} + b_1) + b_2, \tag{4}$$

$$\text{BL}(\mathbf{h}) = \text{LN}(\mathbf{h} + \text{SA}(\mathbf{h})) \tag{5}$$

Implementation of these layers is fairly standard, and did not allow for unique contributions on our part. Furthermore, beyond the provided sanity check, there is not a concrete metric we used to evaluate success of our minBERT implementation. It should be noted that the following sentiment analysis we discuss would not possible without a functioning minBERT, indicating our successful completion of this portion of the project.

2.3 Sentiment Analysis

The second portion of our project consists of a sentiment analysis on two different datasets leveraging the previously constructed minBERT. While skeleton code is provided, our contribution mostly consists of 1) the implementation of Adam Optimizer[3][4] and 2) a sentiment classification task. To complete the Adam Optimizer we first update gradient moments, then apply bias correction and finally adjust weights using decoupled weight decay. For the sentiment classifier, we leverage BERT's pre-trained embeddings, while also adding a classification layer.

Several mean reference accuracy values calculated over 10 random seeds are considered for the different datasets in question as baselines. These are explored in more depth in section 3.1.

2.4 Improved Performance

The last portion of this project is a multi-classifier implemented for the three specific tasks: sentiment classification, paraphrase detection, and semantic textual similarity. In executing our approach we start with the implementation of the task-specific layers, including a linear layer that can create the logits for five different sentiment classes. Next, we implement a linear layer that processes the concatenated BERT embeddings which are derived from sentence pairs. Finally, we implement a linear layer for the prediction of the similarity score. This approach still relies on the pre-trained representations from BERT, but allows for fine-tuning on specific tasks of interest.

Our baseline for the multi- classifier consists of the predictive accuracy values for each task/dataset using default model parameters. These parameters are explored in more depth in section 3.2.

2.5 Extensions

We implemented three improvements to our multi-classifier: 1) fine-tuning with regularized optimization, 2) systematic hyper-parameter optimization, and 3) cosine similarity fine-tuning.

[3]Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv:1711.05101, 2017. arXiv preprint

[4]Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

2.5.1 Regularized Optimization

We refer to SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization[5] for approaches to reduce overfitting, and create more generalizable embeddings. More specifically, we implement smoothness-inducing adversarial regularization. This is a method that ensures our model is robust to small perturbations in the input data. Exact equations can be found in the aforementioned publication, but we achieve this effect by including a smoothness-inducing regularisation term that creates noise in the input data. We then run the model on both the original and noisy inputs, calculate the symmetric KL-divergence between the outputs and incorporate this with a regularization coefficient into the loss expression.

2.5.2 Hyper-Parameter Optimization

Our hyper-parameter optimization includes manipulation of batch size, learning rate, full model vs. single layer fine-tuning, number of epochs, and hidden dropout probability. A larger batch size can lead to more stable convergence but lower batch size can sometimes act as regularization. Similarly there are trade offs with learning rate where a higher learning rate can increase training speed but a lower learning rate can make more precise updates to parameters. Finally, higher dropout probability acts as a form of regularization but too high of a dropout rate can result in under fitting.

2.5.3 Cosine-Similarity Fine Tuning

Our cosine-similarly fine tuning extension, given two sets of input sentences, computes the cosine similarity of the embeddings, to be used as the similarity metric for the loss function. [6]

3 Experiments

In the following subsections, we outline the datasets we deploy for each portion of the project, our evaluation metrics, and experimental results to date.

3.1 Sentiment Analysis

3.1.1 Data

Our sentiment classifier is fine-tuned and tested on the Stanford Sentiment Treebank[7] and the CFIMNB movie review dataset. The train, dev and test holdout is as follows:

Table 1: Number of Samples: Train, Dev, and Test

	SST	CFIMDB
Train	8544	1701
Dev	1101	245
Test	2210	488

3.1.2 Evaluation Method

Classification accuracy is deployed as the evaluation metric for the fine-tuned models and is derived using a train, dev, test split for the model's performance on each dataset.

[5]Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. arXiv preprint arXiv:1911.03437, 2019.

[6]PyTorch Contributors. (2023).CosineEmbeddingLoss. PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.CosineEmbeddingLoss.html

[7]Stanford NLP Group. Sentiment Analysis. Retrieved from https://nlp.stanford.edu/sentiment/treebank.html

3.1.3 Experimental Details

3.1.4 Results

Table 2 demonstrates that our model accuracy exceeds baseline performance for both datasets, including cases where only the last linear layer is fine-tuned! Notably, performance is much better on the CFIMNB data, likely due to the more clearly polarized and emotional nature of those movie reviews.

Table 2: Development Accuracies for minBert Sentiment Classification

Dataset	Baseline Accuracy	Our Accuracy
SST Last Linear Layer	.35	.474
SSt Full Finetuning Dev Accuracy	.45	.527
CFIMB Last Linear Layer Dev Accuracy	.70	.886
CFIMDB Finetune Dev Accuracy	.90	.959

3.2 Improved Performance

3.2.1 Data

Our multi-classifier, capable of performing multiple different tasks, is fine-tuned and tested on the Quora dataset, and the SemEVAL STS Benchmark dataset. The train, dev and test holdout has the following splits:

Table 3: Number of Samples: Train, Dev, and Test

	Quora	STS
Train	283010	6040
Dev	40429	863
Test	80859	1725

3.2.2 Evaluation Method

After augmenting model parameters, we evaluate performance using accuracy values that are calculated using the test and dev splits. We also consider runtime as an additional metric.

3.2.3 Experimental Details

We perform three different experiments to evaluate our three potential model improvements, including hyper-parameter tuning, regularized optimization and cosine similarity fine tuning. After comparing each of these individual improvements with the baseline, we implement a final model that includes all of them.

For the hyper-parameter tuning, we change the learning rate from 1e-5 to 2e-5, the batch size from 8 to 4, the number of epochs from 10 to 15, and finally, we switch from last-linear-layer fine tuning to full-model. We hypothesize that these changes collectively result in higher accuracy at the expense of training time as they are all individually associated with richer embeddings.

For regularized optimization we update our loss equation to include the symmetric K-L divergence. We hypothesized that this would reduce over-fitting and, therefore, also result in an increase in accuracy for all tasks.

Finally, the cosine-similarity fine-tuning involves the utilization of cosine embedding loss. While we expected to observe improvements, we hypothesized that they would be more limited as this experiment only affects the fine-tuning on the SemEval STS dataset.

3.2.4 Results

Figure 2 confirms our hypotheses with the dev and test accuracies improving in all cases compared to the baseline model. More specifically, the overall dev score improves by 30% with the updated hyper-parameters, 22% with the regularized optimization, and 4% with the Cosine-similarity fine-tuning.

Our final combined model has an overall accuracy of .594 with the highest improvement over the baseline. While the individual experiments did not collectively increase the model accuracy as much as we had hoped, we do note a much higher STS dev correlation in the final model (Figure 3).

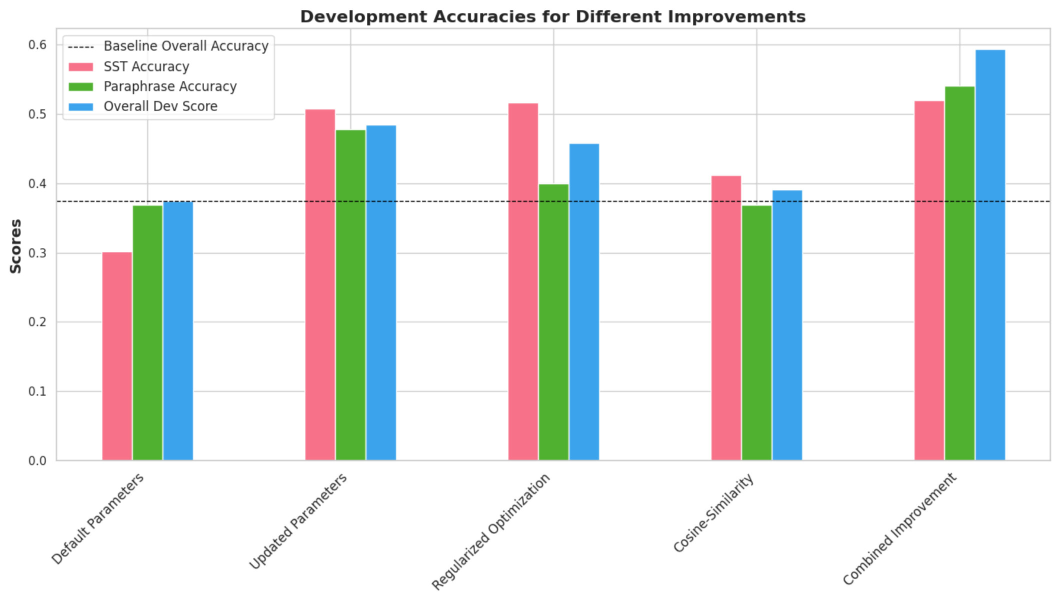

Figure 2: The accuracy results of our base line, individual improvements and final model.

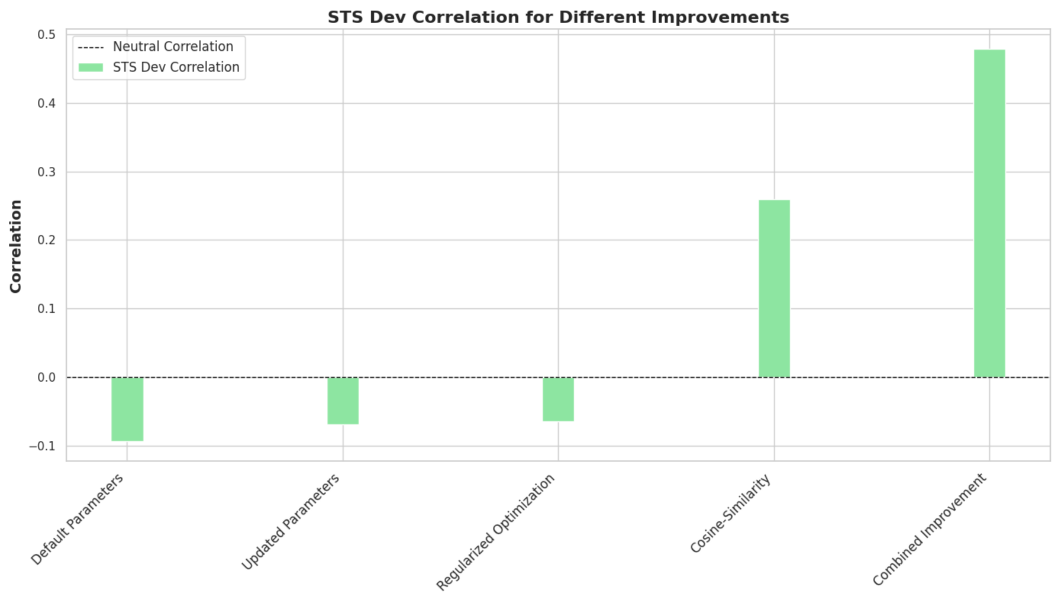

Figure 3: The the STS correlations for the different improvements.

4 Analysis

Figure 4 demonstrates our visualization of the final model's attention weights. The dark purple color indicates low attention weights among most tokens, demonstrating that most tokens are not attending strongly to each other or themselves. We do observe higher weights with the separator token and function, which might suggest that the model is giving importance to understanding the sentence structures. The relatively low weights across the board, coupled with our low model accuracy demonstrate there is still many improvements to be made that we did not have time to implement!

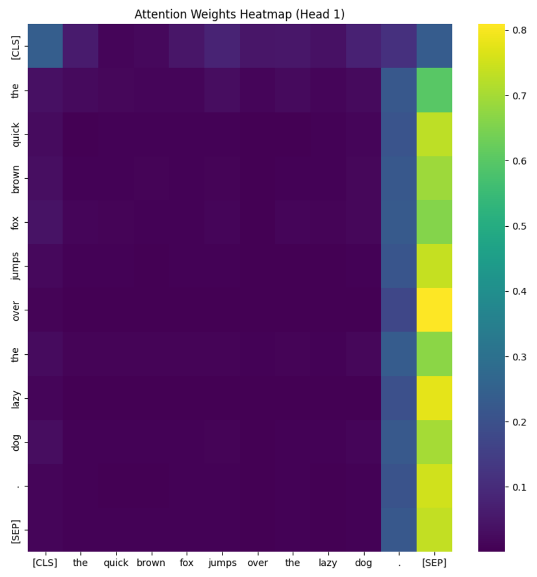

Figure 4: Visualization of the final model's attention weights.

5 Conclusion

This project demonstrates the successful implementation of a version of the original BERT model and the development of its application for several different classification tasks including sentiment classification, evaluation of textual similarity and paraphrase detection. We exceed the provided baseline accuracies for sentiment classification. We also implement several improvements to our model including a fine-tuning of hyper-parameters, smoothing-inducing adversarial regularization, and cosine-similarity fine-tuning. All of these additional modifications resulted in the advancement of our model's performance, resulting in a final dev accuracy of .593. This being said, we failed to implement fine-tuning on a new novel dataset as we originally planned, and there are several other model improvements, such as additional pre-training and multi-task fine tuning, that we hope to explore in the future. Furthermore, we note larger improvements in our SST accuracy in comparison to our paraphrase accuracies. Ideally, our model would perform well across all tasks implemented, but we acknowledge that this is ambitious. Overall, we demonstrate the potential of transformer-based large language models in performing several tasks that would be laborious for a human to complete. We also demonstrate the large improvements in model accuracy that can be achieved with fine-tuning and small tweaks in model architecture.

6 Ethics Statement

We recognize two possible ethical risks associated with our project including bias in the data used for fine-tuning and misinterpretation of model accuracy. Our model is fine tuned on data such as the Quora movie review dataset and STS dataset. We do not have information on the demographic profile of participants. It is possible that a non-representative sample of movie reviewers could lead to the encoding of certain biases into our model. One way to address this ethical concern in future is to try and broaden the range of datasets used for fine-tuning. More data will inherently expose the model to text created by a more diverse audience.

Several steps can also be taken to address potential misinterpretation of our model's accuracy. First and foremost, we will carefully manage access to our model. Our model is not performing at level that is ready for any application beyond educational purposes. Therefore access to it should be limited to our team, and teaching staff. In future, we would also like to provide confidence scores on our predictions to further emphasize the uncertainty that is baked in.

7 Team Contributions

Responsibilities were divided equally amongst team members. Antonio helped to develop code, write the various parts of the written report, and design the poster. Kyra also developed parts of the code, worked on the report, and helped create the poster. Similarly, Anqi helped with coding, writing the report, and making the poster. Sometimes, we each worked on separate tasks and then came together and combined our work. Other times, we all worked simultaneously on the same part of a document.

References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2023). Attention is all you need. arXiv. https://arxiv.org/abs/1706.03762v7

Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171–4186). Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv:1711.05101, 2017. arXiv preprint

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. arXiv preprint arXiv:1911.03437, 2019.

Stanford NLP Group. Sentiment Analysis. Retrieved from https://nlp.stanford.edu/sentiment/treebank.html