SMARTCS: Additional Pretraining and Robust Finetuning on BERT

Stanford CS224N Default Project

Yasmine Mabene
Department of Computer Science
Stanford University
ymabene@stanford.edu

Ayesha Khawaja
Department of Computer Science
Stanford University
akhawaja@stanford.edu

Rachel Clinton
Department of Computer Science
Stanford University
rclinton@stanford.edu

Abstract

We aim to enhance the capabilities of the BERT architecture for natural language processing tasks. Motivated by the need for models that better capture domain-specific nuances, we propose three strategic modifications to the original BERT framework. These are (1) additional pretraining on domain-relevant data, (2) the incorporation of cosine similarity metrics during finetuning, and (3) the application of the SMART regularization to mitigate overfitting. We assess the efficacy of these enhancements in improving model performance across three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We found that incorporating cosine similarity reduces the total execution time for finetuning our model but does not improve prediction accuracy. Additionally, we found that pretraining with SMART loss and Bregman Optimization produces the highest semantic analysis accuracies across all of our models. Finally, our best overall test score (0.770) is produced by finetuning with SMART loss and Bregman Optimization, *without* performing additional pretraining or utilizing cosine similarity.

1 Key Information

CS224N Default Project Mentor. Timothy Dai (timdai@stanford.edu)

Team Contributions. Yasmine constructed the baseline BERT model and the SMART implementations. Rachel designed the model classification heads and implemented the cosine-similarity finetuning technique. Ayesha implemented an adapted masked language modeling objective and conducted additional pretraining on domain-specific data. We all assisted with model experimentation, analysis, and manuscript preparation.

2 Introduction

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based language model that uses bidirectional context to better understand language patterns. Its approach to pretraining allows it to employ transfer learning to achieve state-of-the-art results on several NLP benchmarks (Devlin et al., 2018). Risks of overfitting to training data, lengthy training periods, and optimizing performance on multiple tasks with different requirements make improving the model difficult, but multitask learning helps the model learn a generalizable shared representation for all tasks.

We implement and improve upon a baseline BERT model to enhance model performance on three tasks: sentiment classification with the Stanford Sentiment Treebank (SST), paraphrase detection with Quora's Question Pairs (QQP), and semantic similarity assessment with the Semantic Textual Similarity Benchmark (STS). We extend the model in three ways to respond to three different problems associated with the baseline. First, we conduct additional pretraining with the target datasets (SST, QQP, and STS-B) using an original implementation of the masked language modeling objective. Second, we implement adversarial (SMART) regularization and Bergman optimization to prevent aggressive updating and overfitting, a problem especially salient with our additional pretraining using the training data (Jiang et al., 2019). Third, we implement cosine-similarity finetuning to ensure that the model preserves what it learns during each round of finetuning and calculates loss more efficiently (Reimers and Gurevych, 2019). We call our framework **SMARTCS**: **SM**oothness-inducing **A**dversarial **R**egularization and B**R**egman p**R**oximal poin**T** op**T**imization, pre**T**raining, and **C**osine **S**imilarity finetuning.

Our findings suggest that the SMART loss and Bregman methods improve model performance, while additional pretraining and cosine similarity fine-tuning lead to individual improvements on the SST accuracy and total execution time, but decrease model performance overall.

3 Related Work

A common way to improve artificial intelligence models is pretraining on unlabeled data. Letting a model explore unlabeled data and detect patterns allows the model to learn useful language representations that it can apply to downstream tasks (Han et al., 2021). A variety of pretraining objectives have been explored to optimize natural language models, such as masked language modeling (MLM), next sentence prediction, generalized autoregressive methods, and sentence-order prediction (Devlin et al. (2018), Arocha-Ouellette and Rudzicz (2020), Yang et al. (2019)). When developing the original BERT model, Devlin et al. (2018) found that pretraining on the MLM objective achieved high performance on a large set of tasks for evaluating natural language models that includes the SST, QQP, and STS-B tasks. Consequently, we seek to improve our minBERT model by pretraining on this objective.

Pretraining is an aspect of transfer learning, a common technique used in natural language processing in which models are trained on a large amount of data from particular domains and then are used for other tasks in a target domain (Pan and Yang, 2009). While transfer learning has many advantages, this method often results in models that generalize poorly to new data due to over-fitting during fine tuning. In (Jiang et al., 2019), the authors propose a SMART framework to prevent model over-fitting. SMART relies on the introduction of a smoothness-inducing adversarial regularizer, which has been explored in applications such as semi-supervised learning, unsupervised domain adaptation, and image classification (Miyato et al. (2018), Zhang et al. (2019), and Shu et al. (2018)). Additionally, SMART draws on the existing work on Bregman proximal point optimization to propose a new optimization method that prevents aggressive parameter updating. By leveraging innovative regularization techniques as described in these works, we seek to improve the adaptability and generalization of our baseline BERT model.

A groundbreaking work in the cosine similarity realm is (Reimers and Gurevych, 2019), which introduces Sentence-BERT (SBERT), a modification of the pre-trained BERT network, to derive semantically meaningful sentence embeddings for the STS task. SBERT uses siamese and triplet network structures to enable the computation of semantically meaningful sentence embeddings that can be effectively compared using cosine similarity. This approach significantly reduces the computational overhead associated with the STS task, reducing the runtime from 65 hours with BERT to approximately 5 seconds with SBERT, while maintaining BERT's accuracy levels.

4 Approach

Our research extends the BERT Large Language model (Devlin et al., 2018). We begin by constructing MINBERT, a minimal BERT implementation that consists of 12 encoder transformer layers, Multi-Head Self attention, a feed-forward layer, and layer normalizations.

For our multitask model baseline, we design three classification heads for our transfer learning tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. The sentiment analysis head

consists of a linear layer followed by a RELU activation layer. For the semantic analysis task, we pass our embeddings through the BERT model directly to receive pooled outputs for the linear layer. For the paraphrase and semantic textual similarity tasks, we pass the concatenated embeddings of the two sentence inputs to the BERT model. We then input the pooled outputs from BERT through a shared layer between the two tasks and then through individual RELU and linear layers. The combination of individual and shared layers allows for the model to utilize patterns to generalize across similar tasks while maintaining specialization. Figure 1 depicts how these layers are implemented in our model.

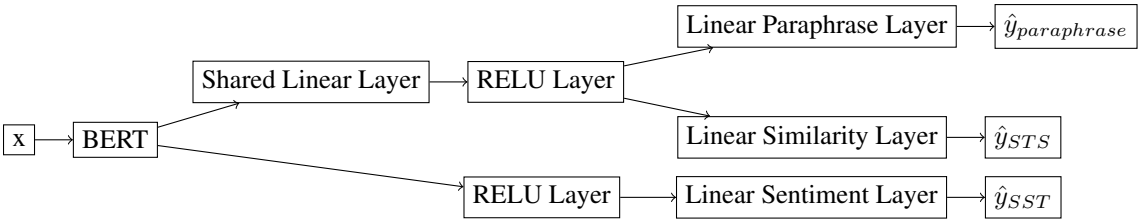

Figure 1: BERT multitask classifier architecture.

We utilize a "Round-Robin" multitask learning framework in our finetuning approach. In each epoch, we cycle through batches from all three datasets from each task. We use Cross Entropy Loss for semantic analysis, Mean Squared Error loss for semantic textual similarity, and Binary Cross Entropy Loss for paraphrase detection. Because the QQP dataset is substantially larger than the STS and SST datasets, we use a larger batch size for the paraphrase detection task. This prevents us from exhausting the smaller datasets, which could lead to poor generalization or overfitting on those tasks.

Principled Regularized Optimization. To address potential overfitting within our model, we adopt the SMART method detailed in Jiang et al. (2019). This method has been found to improve model generalization and performance across a variety of tasks including semantic textual similarity and semantic analysis. This approach consists of two main components: Smoothness Inducing Adversarial Regularization and Bregman Proximal Point Optimization.

In the first component of the model, we mitigate overfitting by applying regularization to control the model complexity. For n data points, $(x_i)_{i=1}^n$ represents the first-layer embeddings of input sentence i in the model while $(y_i)_{i=1}^n$ is the label for input sentence i. $F(\theta)$ is defined as a function taking in the first-layer embeddings and outputting either a probability or scalar for classification and regression respectively. The following equation is solved during fine tuning:

$$min_\theta F(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta) + \lambda_s(R_s(\theta)), \tag{1}$$

where $l(\theta)$ is the loss function, λ_s is a positive smoothing parameter, and $R_s(\theta)$ is the smoothness-inducing adversarial regularizer. (We use KL-Divergence as the loss function for our classification tasks and Mean Squared Error as the loss function for our regression task.)

$R_s(\theta)$ is defined as the following:

$$R_s(\theta) = \frac{1}{n}\sum_{i=1|||\tilde{x_i}-x_i||_p\leq\epsilon}^{n} max(l_s(f(\tilde{x_i};\theta), f(x_i);\theta)). \tag{2}$$

Here, $\epsilon > 0$ constrains the perturbation size.

By adding the regularizer to the loss function, we can ensure that the outputs of f do not change substantially as a result of small perturbations in the input. (These perturbations are represented by $||\tilde{x_i} - x_i||_p$ in Equation 2.) This property of f maintains its smoothness around neighbors of x_i which can prevent overfitting and lead to better generalization.

The second component of the SMART method is the Bregman Proximal Point Optimization. This optimization is used to solve Equation 1 during fine tuning and is characterized by the following equation:

$$\theta_{t+1} = argmin_\theta F(\theta) + \mu D_{breg}(\theta, \theta_t), \tag{3}$$

where D_{breg} is the Bregman Divergence and μ is a tuning parameter. Bregman Divergence is defined as the following:

$$D_{breg}(\theta, \theta_t) = \frac{1}{n} \sum_{i=1}^{n} l_s(f(x_i; \theta), f(x_i; \theta_t)), \tag{4}$$

where l_s is the symmetrized KL-Divergence.

The Bregman Divergence term inhibits large changes in parameters between iterations which helps prevent overfitting.

We construct our own classes **from scratch** to implement the SMART method. While we follow the general structure outlined in the original paper, we modify the approach to allow for layer specific regularization. To do so, we only perturb the embeddings of a selected task and incorporate the regularization loss for this task only. Details on our algorithm can be found in Appendix Figure 5.

Pretraining. We implement an adapted version of masked language modeling (MLM) pretraining objective described in (Devlin et al., 2018) **from scratch** on portions of the SST, QQP, and STS datasets. To make use of our existing model architecture, we mask one random token in the input sequence and construct an MLM head consisting of a linear layer to project the model output onto the vocabulary. In line with the original BERT model's implementation, to reduce the mismatch between pretraining and finetuning we mask the token 80% of the time, replace it with a random token from the vocabulary 10% of the time, and leave it unchanged 10% of the time. We pass the model outputs through the log softmax function and measure model performance using cross entropy loss against one-hot vectors that encode the values of the replaced tokens.

Cosine Similarity. We implement cosine similarity finetuning to improve the performance and efficiency of the STS task. The process of using cosine similarity to improve STS classification is described in the study conducted by (Reimers and Gurevych, 2019). In this work, Reimers et. al. discuss the development of SBERT, a modification of the traditional BERT model that uses cosine similarity, as well as siamese and triple network structures, to improve the efficiency of the existing BERT model, all while preserving accuracy. To integrate cosine similarity into our BERT model, we generate embeddings for each sentence pair and then compute the cosine similarity between each pair. Separately, the concatenated embeddings of the sentence pairs are passed through the shared linear layer shown in Figure 1. The scaled cosine similarity scores are then concatenated with the outputs from this layer. This concatenated output merges the information provided by cosine similarity with the information already derived from BERT embeddings. This combined output is then passed through an additional linear layer to make the final STS predictions.

5 Experiments

5.1 Data

For the sentiment analysis task, we use the Stanford Sentiment Treebank, consisting of 9,645 sentences extracted from movie reviews. For the paraphrase detection task, we use the Quora Dataset, which was designed to reduce the amount of duplicate questions on Quora and currently has over 400,000 lines of potential duplicate pairs. Finally, for the semantic textual similarity task, we use the SemEval 2016 Shared Task Semantic Textual Similarity (STS) dataset.

5.2 Experimental Details

Layers. We update the mulitask classifier to contain separate layers for each task. This allows for task-specific optimizations in the model architecture, enabling each task to have its dedicated processing layers tailored to its unique requirements and complexity.

Hyperparameter Tuning. We experiment with five learning rate values (see Appendix Figure 6) drawn from (Devlin et al., 2018). We perform this initial hyperparameter-tuning step using just the SST and STS datasets in order to expedite experimentation while still yielding insightful findings on effective hyperparameter settings. After selecting the optimal learning value, we ran our baseline model on all three datasets with a variety of batch size configurations. All of these experiments were conducting using 10 epochs. The optimal learning rate ($3e-5$) and batch sizes (6 for SST, 40 for QQP, and 4 for STS) were then used for all subsequent model approaches.

SMART. We conduct our initial experiments with just the adversarial regularizer and afterwards incorporate Bregman optimization. We begin by assigning the hyperparameters as the following: $\lambda = 1$, $\sigma = 1e-5$, $\epsilon = 1e-5$, $\alpha = 1e-3$, and $T = 1$. We then experiment with three values for λ: $1, 2$ and 5. We select $\lambda = 2$ for our experiments. Finally, to improve generalization of the SST task, we assess the performance of the regularization when applied solely on layers for semantic analysis. To evaluate the performance of our SMART implementation, we construct regularization baselines in addition to our baseline model. The first regularization baseline introduces a dropout layer with dropout rate of 0.15 in each classification head. Our second regularization baseline modifies our multitask finetuning approach and is inspired by the annealed sampling framework detailed in (Stickland and Murray, 2019). Rather than equally cycling through each dataset as done in our baseline, we deterministically cycle through the largest dataset (Quora) for a given number of training steps before incorporating the other two datasets. We utilize this method as a regularization baseline to account for overfitting that may occur as a result of our Round-Robin finetuning strategy on unbalanced datasets.

Pretraining. To determine the best approach for pretraining, we experiment with two masking strategies: masking the token 100% of the time, or masking the token 80% of the time and replacing it with a random token in the vocabulary 10% of the time and leaving it unchanged 10% of the time. We experiment with pretraining on solely the SST dataset versus all three datasets, since model performance on the SST task was much lower than performance on the other two tasks. We also experiment with including a dropout layer during pretraining to reduce overfitting. Finally, we experiment with different hyperparameters: we pretrain for 3, 5 and 10 epochs, and experiment with learning rates of $1e-5, 2e-5$, and $3e-5$. Ultimately, the best model performance arises from masking the token 80% of the time, pretraining on all datasets, forgoing the dropout layer, and using a learning rate of 1e-05 for five epochs. We use batch sizes of 6, 4, and 40 for the SST, STS, and QQP data sets, respectively, to keep the pretraining as similar to our finetuning procedure as possible. Pretraining for five epochs takes 3924 seconds total, or a little over an hour.

Cosine Similarity. Initial experiments for this technique involved calculating the cosine similarity between the embeddings of two sentences and using these values directly as STS predictions. We scaled the embeddings and labels to be within the same range and computed Mean Square Error between the cosine similarity scores and the scaled target labels. We found that while this approach had moderate performance on the SST task, it performed significantly worse on the STS task, even dropping to a correlation score as low as -0.190. To address this, we instead integrate the cosine similarity scores with the concatenated embeddings as described in Section 4. We maintain the same hyperparameter configurations as our baseline model.

5.3 Evaluation method

To evaluate model performance on the SST and QQP tasks, we utilize prediction accuracy. For the STS task we use the Pearson Correlation Coefficient.

5.4 Results

Hyperparameter Tuning. After initial hyperparameter tuning, we selected $3e-5$ as our optimal learning rate for finetuning (see Appendix Figure 6). We then ran our baseline model on all three training and dev sets on several different batch sizes, producing the results in Appendix Figure 7.

In our finetuning approach, we cycle through each dataset for a fixed number of iterations. Thus, the batch size determines how much of the training data is used per epoch. With the QQP dataset being 16 times that size of the smallest dataset, the batch sizes are crucial in ensuring model generalization across all tasks. Initially, we hypothesized that batch sizes that are proportional to the size of the datasets would produce the most optimal results. However, we found that the best dev score came from a combination of larger batch sizes for larger datasets while also disproportionately enlarging the batch size for SST. We believe this is because the model struggled to generalize well to the SST task, so it benefitted from increased SST data. These insights confirm the necessity of additional modifications to our model to address overfitting.

SMART Results. The results of our initial SMART experimentation can be found in Appendix Figure 8. While our baseline model performed well on the SST and STS tasks, it struggled to attain similar performance on the sentiment analysis task. Even with the addition of dropout layers, our

model performance did not increase for sentiment analysis and even decreased for the other tasks. On the other hand, our default implementation of SMART led to a 7% increase in the QQP dev accuracy. After applying the SMART loss only on the sentiment analysis task, we saw a modest increase in the overall dev score compared to both the baseline (0.769 vs 0.764) and the default SMART implementation (0.769 vs 0.759). Interestingly, we obtained the same dev score from our modified SMART approach as that of our modified multitask learning (Stickland) baseline (0.769), while using only half the run-time. This highlights the ability of our SMART implementation to efficiently mitigate overfitting that arises as a result of multitask learning on unbalanced data.

5.5 Combined Results.

Model Type	SST	QQP	STS	Dev Score	Total Execution Time (hrs)
Baseline	0.480	**0.887**	**0.882**	0.764	4.500
Pretrain	0.485	0.859	0.870	0.760	3.934
Cosine Similarity	0.478	0.866	0.879	0.761	4.010
SMART Loss	0.509	0.864	0.866	0.769	5.000
SMART Loss, Bregman Only	0.494	0.879	0.876	**0.770**	5.389
Pretrain, SMART Loss, Bregman	**0.512**	0.859	0.858	0.767	4.572
Pretrain, Cosine Similarity	0.472	0.856	0.864	0.753	4.384
SMART Loss, Bregman, Cosine Similarity	0.487	0.863	0.872	0.762	4.932
Pretrain, SMART Loss, Bregman, Cosine Similarity	0.478	0.862	0.867	0.758	5.034

Figure 2: Development set performance for all model approaches. *Note: "SMART Loss" refers to the SMART implementation on the SST task only. "Pretrain" contains the results after finetuning using our additionally pretrained model.*

Our best model was the combination of the layer-specific SMART approach and Bregman optimization which obtained **test set** performances of 0.493 (SST), 0.878 (QQP), 0.878 (STS), and 0.770 (Test Score). Although this approach caused modest decreases in the QQP and STS scores compared to the baseline, it increased the SST score by nearly 3%, improving the overall test score (Figure 3). Generally, across modifications to our baseline model, we found that increases in the SST scores were associated with decreases in the QQP and STS scores (Figure 2). Both of our SMART models along with our pretrained model improved performance on the SST task. However, our pretrained model struggled to balance performance across all three tasks. When we combined SMART loss and Bregman optimization with our pretrained model, we obtained a 6% increase in the SST accuracy, the highest SST accuracy of all our combined approaches. This indicates the overall benefit of both approaches for improved generalization of our model for semantic analysis. We found that our cosine similarity modification when implemented alone, produced a dev score commensurate with that our baseline. However, its combination with other approaches hindered the performance of these models. Further, parameter tuning may be needed to combine cosine similarity with our other modifications.

Model Type	SST	QQP	STS	Test Score
SMART Loss	0.501	0.862	0.863	0.765
SMART Loss, Bregman	0.493	**0.878**	**0.878**	**0.770**
Pretrain, SMART Loss, Bregman	**0.511**	0.855	0.858	0.765

Figure 3: Test set performance for our best three model approaches. *Note: "SMART Loss" refers to the SMART implementation on the SST task only. "Pretrain" contains the results after finetuning using our additionally pretrained model.*

6 Analysis

When training our models we generally found that the performance of the semantic analysis task was inversely related to the performance of the semantic textual similarity and paraphrase detection tasks. We originally hypothesized that the latter two tasks would be similar and constructed our baseline architecture to include a shared layer between the two tasks to exploit this. However, this relationship presented a challenge in terms of optimizing performance for all three tasks. The introduction of

SMART made our model more robust to changes within the inputs as well as less prone to aggressive updates that contribute to overfitting. Enriched embeddings following additional pretraining was also found to be useful for improving semantic analysis performance. However, this occurred at the expense of the performance of the STS and QQP tasks.

Additional Pretraining Effects. Pretraining on all three datasets yielded slightly lower results than the baseline—while the pretraining improved the SST dev score, it decreased performance on the QQP task and STS tasks. When paired with SMART regularization and Bregman optimization, we achieved our highest SST accuracy. However, when additional pretraining was paired with cosine similarity finetuning, and when the model included additional pretraining, SMART and Bregman, and cosine similarity, model performance dropped (Figure 3). Altogether, additional pretraining was most successful with SMART loss and Bregman optimization.

The mixed success of the added pretraining contradicts the findings for Devlin et al.'s implementation. We suspect this is due to two reasons. First, the adapted MLM objective that we implemented was a minimal implementation of Devlin et al.'s—namely, Devlin et al. masked 15% of tokens instead of one token, so it's possible that the model did not learn as much from the additional pretraining as it would have had more tokens been masked per sentence. Second, the pretraining pushed the model to overfit significantly on the training data, and our regularization and Bregman optimization implementations helped a little, but may not have been strong enough to combat that overfitting.

SMART and Bregman Effects. The incorporation of SMART into our model was effective for reducing overfitting. When comparing SMART implementations, we found a minor increase in the dev score when incorporating Bregman optimization to the SST SMART Loss approach. Thus, we believe that the perturbations we introduce when incorporating the adversarial loss are nearly enough to mitigate overfitting that is addressed by Bregman optimization in our model.

Out of all our SMART implementations, using the default parameters produced the highest SST accuracy (7% increase from the baseline). After modifying SMART to apply regularization on only the SST task, we improved the overall dev score (Figure 2). This improvement was primarily driven by a 4% increase in the QQP accuracy. As a result, we believe that the noise introduced into the QQP embeddings in our default SMART implementation may have been too large and negatively affected this task. In all, we have shown that layer-specific SMART can improve overall model performance. Additionally, our results open the door to the potential of task-based variation of other SMART parameters (eg. noise) for improving model performance.

Cosine Similarity Effects. Contrary to the findings of (Reimers and Gurevych, 2019), the integration of cosine similarity into our model did not produce significant improvements on the STS task. Specifically, our baseline model produced an STS score of 0.882, whereas the cosine similarity incorporated with our baseline produced a lower STS score of 0.879. Furthermore, pretraining alone produced an STS score of 0.870, whereas pretraining with cosine similarity decreased the STS score to 0.864. This may be because the addition of cosine similarity requires alternative hyperparameter configurations than our baseline model. For example, the noise introduced into the embeddings in SMART may not be at an appropriate scale for computing cosine similarity. Further hyperparameter tuning may be needed to fully harness the benefits of cosine similarity finetuning.

Cosine similarity did, however, result in time improvements, with the cosine similarity and baseline performing 10.9% faster than the baseline alone, and the pretrain and cosine similarity performing 4.1% faster than the pretrain with SMART loss and Bregman. This improvement in time is consistent with the findings of (Reimers and Gurevych, 2019).

Overall Observations.

For the QQP task, the recall and precision are very similar (see Appendix Figure 9). This means that the proportion of paraphrase (positive) and non-paraphrase (negative) inputs that our model correctly classifies are nearly equal. However, the positive predictions for our model are nearly 10% more accurate than the negative predictions. This may be a result of imbalanced classes during training as only 37% of our training data belonged to the true paraphrase class. To improve QQP accuracy, our next step would be to reduce the number of false negatives, potentially through the introduction of more data from the positive class.

After computing the recall and precision for all five classes in our semantic analysis task, we find the our model obtains the most accurate predictions when classifying sentences as positive (see Appendix

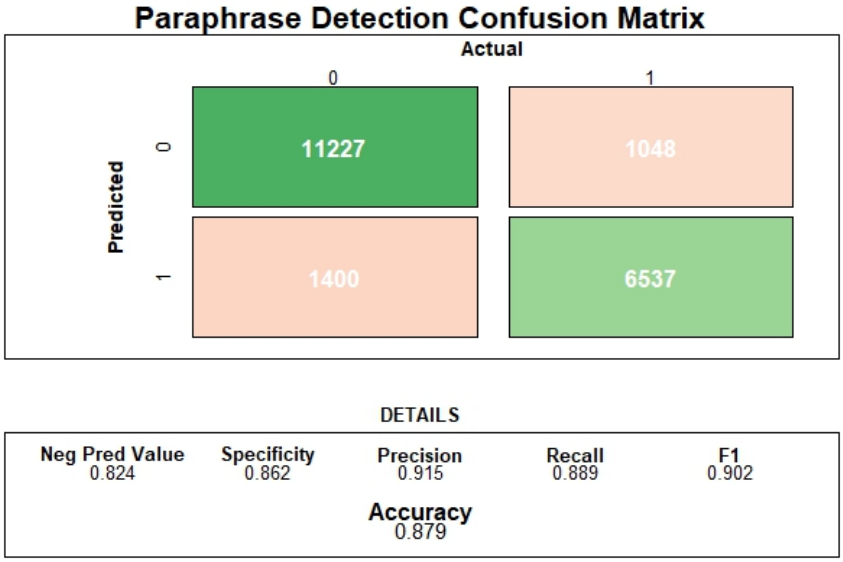

Figure 4: Confusion matrix for paraphrase detection using SST SMART + Bregman model on development set.

Figure 9). Interestingly, our model is better at correctly classifying inputs as somewhat negative than those that are negative. This is non-intuitive as "somewhat negative" would appear to be a more ambiguous description than "negative". Upon further analysis, we found that across all classes, 70% to 93% of falsely classified inputs were predicted as being in an "adjacent" class. For example, 87% of the wrongly classified inputs that were actually from the "somewhat negative" class were predicted to be either "negative" or "neutral". This means our model can identify general sentiments of inputs but may struggle in distinguishing between similar labels.

The distribution of the predicted STS labels and the true labels are fairly similar (Appendix Figure 10). However, the true label distribution is clearly multimodal, with evident peaks at the integer values,while the distribution for our predicted values is more smooth. In the future, we can experiment with an additional layer to round outputs that are within a certain range of the integer values. This would mimic the natural approach humans have for labeling data.

7 Conclusion

We explore the effects of additional pretraining on the MLM objective, SMART loss and Bregman optimization, and cosine similarity finetuning on BERT's performance on three natural language benchmarks: the Stanford Sentiment Treebank, Quora Question Pairs, and Semantic Textual Similarity Benchmark. Among these extensions, we found that our model performed best when finetuned with SMART loss and Bregman optimization. Additional pretraining improves performance on the SST task but decreases performance on the other two benchmarks. Finally, cosine similarity finetuning reduces the total execution time but does not improve model accuracy.

The primary limitations of our work include the data used for pretraining and finetuning, limited hyperparameter tuning, and the compute power available. We only worked with the SST, QQP, and STS datasets, so our findings may not be generalizable to other datasets and limited data made our model prone to overfitting, particulary during pretraining. Due to GPU memory constraints, we could only explore some modifications. In the future, we may try to pretrain on other objectives, incorporate additional datasets, or explore ensembling our different models together to improve performance.

References

Stéphane Arocha-Ouellette and Frank Rudzicz. 2020. On losses for modern language models. *In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayn Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *AI Open*.

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

Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. 2018. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41:1979–1993.

Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE transactions on knowledge and data engineering 22 (10)*, 1345.

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, Hong Kong, China. Association for Computational Linguistics.

Rui Shu, Hung H Bui, Hirokazu Narui, and Stefano Ermon. 2018. A dirt-t approach to unsupervised domain adaptation. *arXiv preprint arXiv:1802.08735*.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. "PMLR".

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *33rd Conference on Neural Information Processing Systems*.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. 2019. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*.

A Appendix

Notation: j: *index representing the current training step. Note,* $j \in [1, J]$ *and* θ_0 *is the parameter from the pretrained model.* $g_i(\overline{x_i}, \theta_j)$ *the gradient of the loss function wrt. the noise* v_i. $AdamUpdate_\beta$ *is the Adam Rule for optimizing Equation* 3.

$\theta_j = \theta_{j-1}$
For batch β *from* χ:
  $v_i \sim N(0, \sigma^2)$
  For $t = 1, \ldots t = T$:
    For x_i *in* β:
      $\overline{x}_i = x_i + v_i$
      $\overline{g}_i = g_i(\overline{x}_i, \theta_j)$
      $v_i = v_i + \overline{g}_i(\alpha)$
      $v_i = \frac{v_i}{\left\| g_i(\overline{x}_i, \theta_j) \right\|_p}$
$\overline{x}_i = x_i + v_i$
$\theta_{j+1} \leftarrow AdamUpdate_\beta(\theta_j)$

Figure 5: SMART method algorithm based on method from original paper Jiang et al. (2019).

Learning Rate	SST Dev Accuracy	STS Dev Correlation
1e-5	0.474	0.868
2e-5	0.459	**0.879**
3e-5	**0.498**	0.877
4e-5	0.489	0.871
5e-5	0.486	0.874

Figure 6: Model performance on SST and STS tasks after ten epochs with varying learning rates. The learning rate of 3e-5 performed best across both tasks; although the model performed slightly better on the STS dev set with a rate of 2e-5, the difference in correlation was small enough (0.879 vs. 0.877) to motivate using 3e-5 instead. Additionally, we determined the loss converged with this learning rate.

Batch Sizes	QQP Dev Acc	SST Dev Acc	STS Dev Corr	Dev Score
32, 4, 3	0.625	0.262	0.314	0.514
40, 4, 3	0.833	0.443	0.854	0.734
45, 4, 3	0.842	0.478	0.853	0.724
40, 6, 4	0.848	**0.487**	0.866	**0.756**
40, 8, 6	**0.873**	0.438	**0.871**	0.749

Figure 7: Model performance on SST, QQP, and STS tasks after ten epochs. Batch sizes are reported for QQP, SST, and STS data respectively.

Model Type	SST Dev Acc	QQP Dev Acc	STS Dev Corr	Dev Score	Run Time (sec)
Baseline	0.480	**0.887**	**0.882**	0.764	1620
Dropout Baseline	0.480	0.861	0.867	0.758	1620
Stickland Baseline	0.494	0.883	0.861	0.769	3200
SMART Loss Default	**0.515**	0.828	0.867	0.759	1800
SMART-SST Loss	0.509	0.864	0.866	0.769	1800
SMART-SST Loss + Bregman	0.494	0.879	0.876	**0.770**	1940

Figure 8: SMART method performance and epoch run time. Note, the Stickland Baseline refers to our modified multitask learning approach described in section 5.2.

Class	Recall	Precision
negative	0.324	0.474
somewhat negative	0.512	0.548
neutral	0.489	0.397
somewhat positive	0.487	0.548
positive	0.624	0.500

Figure 9: Recall and Precision values for semantic analysis using Dev set.

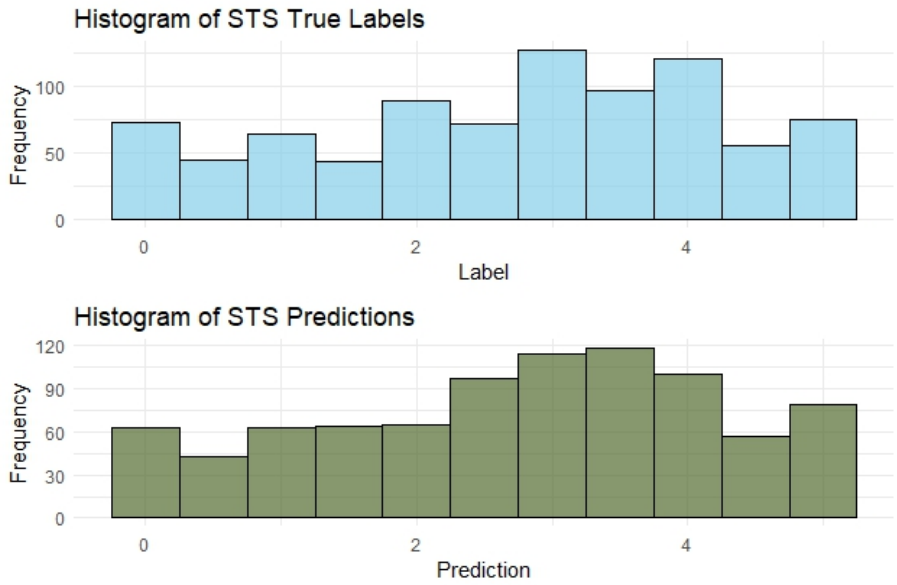

Figure 10: Histogram comparison of true and predicted STS labels for SST SMART + Bregman Model on development set.