SMARTBert: Improving BERT Model Performance on Downstream Tasks Using Smoothness Inducing Adversarial Regularization

Stanford CS224N Default Final Project

Jennifer He
Department of Computer Science
Stanford University
jenhe@stanford.edu

Roy Yuan
Department of Computer Science
Stanford University
ryuan19@stanford.edu

Abstract

Our task is to implement and train a transformer model that can simultaneously perform on the following three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. The existing BERT model is pretrained and can be finetuned on downstream tasks. However, due to the complexity of the pretrained model as well as limitations in the datasets, the BERT model is often overfitted to the tasks. In order to allow the model to perform better on new, unseen examples, we adopt a Smoothness Inducing Adversarial Regularization (SMART) technique to improve the baseline BERT implementation. Using the BERT model finetuned on SMART, we achieved improved performance from the baseline on all tasks, scoring a sentiment accuracy of 0.372 on the SST dataset, a paraphrase accuracy of 0.656 on the Quora dataset, and a STS Correlation of 0.032 on the SemEval STS Benchmark dataset.

1 Key Information to include

- Mentor: Davey Huang, Tathagat Verma
- Special thanks: Rishi Desai, Elsa Wilbur, Constance Horng

2 Introduction

The field of natural language processing (NLP) has seen significant advancements in recent years, with the introduction of pre-trained transformer models like BERT, which have achieved state-of-the-art performance on a wide range of downstream NLP tasks, including sentiment classification, paraphrase detection, and semantic textual similarity.

However, despite their impressive performance, these models are not without their limitations. One key issue is that they are highly prone to overfitting to the training data due to the high complexity of pre-trained models and aggressive finetuning, which can lead to poor generalization to new, unseen examples. This is especially problematic in scenarios where the training data is limited or biased.

To address this challenge, researchers have proposed various regularization techniques that aim to encourage smoother and more robust model behavior. One such approach is smoothness inducing adversarial regularization, which manages the complexity of the model by imposing smoothness constraints on the model's decision boundary using adversarial training, which we discuss more in-depth later.

Another technique that has shown promise in this regard is Bregman proximal point regularization, which encourages the model to remain close to a reference point in its decision space to prevent the model from aggressive updating during the finetuning phase, thereby promoting smoother and more stable behavior.

In this paper, we propose a novel approach that combines both smoothness inducing adversarial regularization and Bregman proximal point regularization to improve the performance of the BERT model on sentiment classification, paraphrase detection, and semantic textual similarity tasks. Our approach is motivated by the intuition that these two regularization techniques complement each other by addressing different sources of instability in the model's decision boundary.

We demonstrate the efficacy of our proposed approach on several benchmark datasets and show that it outperforms existing state-of-the-art methods. Our results suggest that incorporating these two regularization techniques can significantly improve the robustness and generalization performance of the BERT model, and we believe that our approach has the potential to advance the state of the art in NLP.

3 Related Work

The field of NLP has seen a number of advancements in recent years, with many researchers proposing novel approaches to improve the performance of pre-trained transformer models on downstream tasks. In this section, we provide an overview of the related works in this area, with a particular focus on the contributions of the SMART paper by Jiang et al., as well as other NLP papers that have performed similar downstream tasks.

Jiang et al. proposed the use of smoothness inducing adversarial regularization and Bregman proximal point regularization to improve the performance of the BERT model on a range of downstream tasks, including sentiment classification, paraphrase detection, and semantic textual similarity. Their approach achieved state-of-the-art results on several benchmark datasets, including the GLUE benchmark, SNLI, SciTail, and ANLI, demonstrating the effectiveness of incorporating these two regularization techniques to improve the robustness and generalization performance of pre-trained transformer models.

Several other NLP papers have also proposed novel approaches to improve the performance of pre-trained transformer models on downstream tasks. For example, Wang et al. proposed a self-supervised pre-training method called ERNIE 2.0, which incorporates knowledge graph information to enhance the semantic representation capabilities of the model. Their approach achieved state-of-the-art results on several benchmark datasets, including the GLUE benchmark.

Similarly, Zhang et al. proposed a knowledge-enhanced pre-training method called K-BERT, which incorporates external knowledge from knowledge bases to improve the performance of the BERT model on a range of downstream tasks. Their approach achieved state-of-the-art results on several benchmark datasets, including the GLUE benchmark and the SuperGLUE benchmark.

In addition to these works, there have been many other papers that have proposed novel approaches to improve the performance of pre-trained transformer models on downstream tasks, including approaches based on multi-task learning, transfer learning, and model compression. While these approaches vary in their specific techniques and methodologies, they all aim to address the challenges of overfitting and limited data that are inherent to pre-trained transformer models, and have shown promising results on a range of downstream tasks.

The SMART framework proposed by Jiang et al. demonstrates the effectiveness of combining smoothness-inducing adversarial regularization with Bregman proximal point optimization on BERT for several limited downstream tasks, including GLUE, SNLI, SciTail, and ANLI. However, our paper aims to expand the applicability of the SMART framework to three additional downstream tasks, namely sentiment classification, paraphrase detection, and semantic textual similarity. We believe that this extension will further validate the effectiveness and versatility of the SMART framework on a wider range of NLP tasks.

4 Approach

We first use a vanilla BERT (Bidirectional Encoder Representations from Transformers) model to achieve baseline results before implementing SMART to finetune our BERT model.

4.1 Baseline Model

Our BERT implementation is based on the baseline model outlined by the default project code. Specifically, from a layer architecture standpoint, out BERT implementation consists of the following:

1. Embedding Layer: To get the word, position, and token type embeddings from the inputs, we use an embedding layer.

2. Embedding Post-Processing Layer: This layer normalizes the combined embeddings (word, position, and token type) across the hidden dimension and applies dropout.

3. Transformer Layers: Each transformer layer consists of the following components in the provided order:

- Multi-Head Self-Attention: This component uses linear layers for key, query, and value transformations and computes attention scores. The scores are used to weight the values, which are then summed and passed through another linear layer.
- Add Norm Layer: This layer normalizes the output of the multi-head self-attention component and adds it back to the input using a residual connection.
- Position-Wise Feed-Forward Network: This component consists of a linear layer followed by a GELU activation function.
- Another Add Norm Layer: This layer combines the output of the feed-forward layer with the input using a residual connection, enabling the model to learn more complex and non-linear relationships between the input tokens.

4. Pooling Layer: This layer passes the hidden state of the [CLS] token through a linear layer and a Tanh activation function to produce the final pooled output.

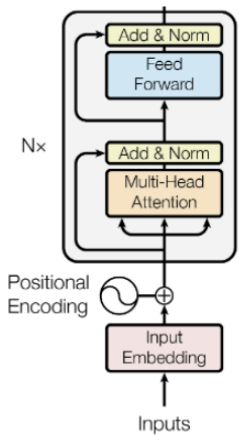


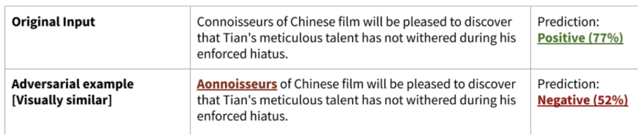
Figure 1: Encode layer in transformer of Bert

4.2 Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization (SMART)

We explore robust and efficient fine-tuning for pre-trained models to attain better generalization performance to improve our accuracies from the baseline results. SMART can be seen as two components:

1. Smoothness-inducing regularization, which effectively manages the complexity of the model
2. Bregman proximal point optimization, which is an instance of trust-region methods and can prevent aggressive updating.

4.2.1 Smoothness-inducing Adversarial Regularization We introduce our "adversaries" as our input (in this case, embeddings) with small perturbations (randomly generated noise) added to it. To help the model to be smooth with respect to the input data, we penalize the model when its output changes significantly with these added perturbations.



Original Input	Connoisseurs of Chinese film will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Positive (77%)**
Adversarial example [Visually similar]	**Aonnoisseurs** of Chinese film will be pleased to discover that Tian's meticulous talent has not withered during his enforced hiatus.	Prediction: **Negative (52%)**

Figure 2: Adversarial example

To change our perturbed embedding into a perturbed state, we feed in an eval function defined during our training in multitask_classifier.py. We want the to obtain the logits so it can be used in the SMART loss function and be compared to b_ labels, and to get the pooler output of the adversarial embedding, and we get the pooler output in the eval() function that we defined in multitask classifier, which does the following:

- we create an attention mask of all ones to use in encode, since noise is already added to the embedding before encoding so we don't need the attention mask to do this again
- run the encode layer on the perturbed embedding (input) to then get the pooler output
- use the pooler output and feed it into the specific downstream task function that we are fine tuning on, to get the logits

To incorporate this smoothing regularization to control the model complexity at the fine-tuning stage, SMART solves the following fine-tuning optimization:

$$\min_{\theta} F(\theta) = \mathcal{L}(\theta) + \lambda \mathcal{R}_s(\theta)$$

First, we look at $\mathcal{L}(\theta)$, the loss function, which is defined as:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; \theta), y_i)$$

We implement $\mathcal{L}(\theta)$ as the loss_last _fn method in our code, which defaults to the loss _fn method if not provided. The loss _last _fn we use is implemented as sym _kl _loss and the loss _fn is implemented as kl _loss, both defined in our mysmartloss.py file to reflect the equations provided by the original paper.

Here, x_i's denote the embedding of the input sentences obtained from the first embedding layer of the language model and y_i's are the associated labels (b_ label in our code). $\ell(\cdot, \cdot)$ is the loss function depending on the target task.

For predicting sentiment, our loss is initialized as the F.cross_entropy with the inputs as the logits outputted from our MultitaskBERT's predict_similarity function, and we use F.binary_cross_entropy_with_for predicting paraphrase and similarity with logits as output of predict_paraphrase and predict_similarity, respectively. This is because predict_similarity has output of 5 sentiment classes whereas predict_paraphrase and predict_similarity outputs a single logit representing if the sentences are paraphases/similar.

Moving on to the regularizer in the fine-tuning optimization defined above, we have $\mathcal{R}_s(\theta)$ is the smoothness-inducing adversarial regularizer with $\lambda_s > 0$ is a tuning parameter. This term penalizes the model when its output changes significantly with input perturbations. $\mathcal{R}_s(\theta)$ is defined as:

$$\mathcal{R}_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{||\tilde{x_i}-x_i||_p \leq \epsilon} \ell(f(\tilde{x_i};\theta), f(x_i;\theta))$$

the regularizer is adversarial because the regularization term, $\mathcal{R}_s(\theta)$, has an adversarial term in $\tilde{x_i}$ within the chosen loss function $\ell(\cdot)$. In our code, the adversarial term in here is the embedding that we are perturbing.

To "smooth out" the model's output, we give our model adversarial examples that we generated earlier with noise to help it achieve this. To calculate the $\mathcal{R}_s(\theta)$ term, we iteratively compute the loss_fn (defined as KL divergence) method. The process involves:

- first perturbing the input embed by adding noise
- computing the perturbed state state_perturbed from the perturbed embedding that we just created. This means getting the logits of the perturbed embedding, similar to how we get logits for a regular embedding (but task specific). The output is the perturbed state.
- then evaluating the perturbation loss using the loss_fn method

Once we have finished our iterations, we run last_loss_fn (defined as sym_kl_loss in mysmartloss.py) on the perturbed state to compute the symmetrized KL-divergence. Above in the ℓ function, in SMART, we use the KL divergence to compare the label of the perturbed input versus the actual label.

This term here prevents the model from relying too much on specific features from the training data and instead encourages the model to learn more generalizable features that are less sensitive to small input perturbations. In figure 2, we can see clearly that the decision boundary with smoothness-inducing adversarial regularization fits the sampled data points more accurately.

Altogether, we combine the main loss function $\mathcal{L}(\theta)$ and the smoothness-inducing adversarial regularizer $\mathcal{R}_s(\theta)$ to achieve our overall objective function to minimize $F(\theta) = \mathcal{L}(\theta) + \lambda\mathcal{R}_s(\theta)$.

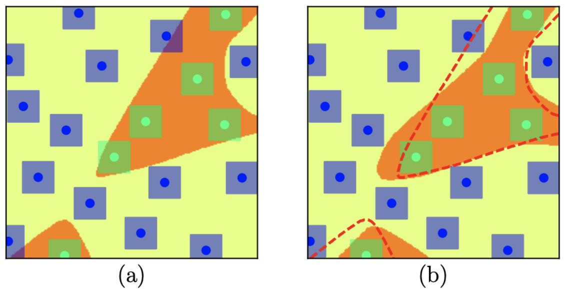


Figure 3: Decision boundaries, (a) without and (b) with (b) smoothness-inducing adversarial regularization, respectively. The dotted red lines in (b) represent the decision boundaries in (a).

4.2.2 Bregman Proximal Point Optimization (BPPO) The SMARTLoss class is designed to be used as part of the optimization process. The actual implementation of BPPO would involve integrating the SMARTLoss class into the training loop of the model and using an optimizer to update the model's parameters. The Bregman divergence is defined as:

$$D_{Breg}(\theta, \theta^t) = \frac{1}{n}\sum_{i=1}^{n} l_s(f(\mathbf{x}_i;\theta), f(\mathbf{x}_i;\theta_t))$$

and this measure would be used to guide the optimization process, ensuring that the model remains robust while fine-tuning.

5 Experiments

5.1 Data

We use the sst dataset (11,855 single sentences from movie reviews extracted from movie reviews, Each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive) and cfimdb dataset (2,434 highly polar movie reviews categorized as negative or positive) to do tasks such as sentiment analysis with our classifier. We use the Quora Dataset (400,000 question pairs with binary labels indicating whether particular instances are paraphrases of one another) and SemEval STS Benchmark Dataset (8,628 different sentence pairs of varying similarity on a scale from 0=unrelated to 5=equivalent meaning) to do tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity with out multitask classifier. Each dataset has train, dev, and test splits.

5.2 Evaluation method

We evaluate the model's performances on specific downstream tasks and compare between a fine-tuned model with and without SMART implemented.

For the sentiment analysis and paraphrase detection tasks, we evaluate the model based on accuracy. For semantic textual analysis, we will calculate the Pearson correlation of the true similarity values against the predicted similarity values across the test dataset.

5.3 Experimental details

We ran a number of experiments to optimize performance on multitask classification. In order to obtain some baseline metrics for the three tasks, we finetune the pretrained model only on sentiment classification. In order to analyze how SMART improves our model in the three tasks, we then apply our SMART model and train the tasks individually. Finally, we aggregate the model's learning on all three tasks and run an experiment with all three losses. In this experiment, we update the loss by training all batches for each task and running the optimizer after each task. Seeing that finetuning on sentiment and paraphrase with SMART improves our model performance the most, we ultimately do one final aggregated experiment with tuned hyperparameters to try to improve our model with a round robin approach in which we alternate taking batches from the sentiment and paraphrase datasets only.

For all training, we use 10 epochs, batch size of 8, and learning rate of 1e-3. Trained on Colab GPU, each task takes about 2-3 hours.

5.4 Results

Model	Sentiment Acc	Paraphrase Acc	STS Corr
Baseline without SMART	0.318	0.625	0.015
Finetune sentiment only with SMART	0.344	0.625	0.015
Finetune paraphrase only with SMART	0.135	0.660	0.030
Finetune similarity only with SMART	0.135	0.376	0.025
Finetune all three tasks with SMART	0.332	0.371	0.015
Finetune interleaved with SMART	0.372	0.656	0.032

Table 1: Dev Set Performance on BERT Multitask Datasets

- The dev set results are reported above. We report test set results on our final experiment which involved BERT finetuned on interleaved sentiment and paraphrase batches. This model obtained the following dev set results: 0.372 on sentiment accuracy, 0.656 on paraphrase accuracy, and 0.032 on STS correlation. We submitted finetune interleaved with SMART to leaderboard and obtained the following: SST test Accuracy: 0.373, Paraphrase test Accuracy: 0.657, STS test Correlation: -0.032, Overall test score: 0.333.
- Overall, implementing SMART improved the model on the tasks and our approach for interleaving the learning on sentiment and paraphrase allowed the model to simultaneously achieve better performance on all three tasks, as expected.

6 Analysis

The results we obtained are as expected. We finetuned on each individual task with SMART marginally improves performance on that task. We see that finetuning on sentiment only improves sentiment accuracy while not changing paraphrase or STS. This makes sense because finetuning to a certain dataset should fit the model to that data. And since sentiment is a multiclass classifier, the last layer has number of features corresponding to the number of classes while paraphrase and sentiment only linear layer has only one dimension in the output (indicating either if a phrase is a paraphrase in the case of paraphrase or indicating how similar two phrases are in the case of similarity), so finetuning on sentiment (5 features in linear layer output) doesn't affect paraphrase and similarity (1 feature in linear layer output) because they are a different flavor of tasks. Meanwhile, finetuning paraphrase improves paraphrase accuracy and STS correlation while slashing performance on sentiment. This makes sense given the previous explanation that paraphrase and sentiment are a more similar flavor of tasks. The slashed performance from baseline also makes sense since baseline was trained on sentiment. Finetuning STS improves STS while slashing both sentiment and paraphrase. This may be a reflection of the STS dataset or the task of similarity correlation. This could be explained by the fact that similarity itself is a difficult, unique task, so fine tuning on such a non-generalizable dataset would slash performance in other tasks. This could also be explained by the fact that the dataset for STS is a lot smaller and has 8,628 data points compared to the 11,855 for sentiment SST dataset and 400,000 for paraphrase Quora dataset.

Finetuning on all three one after the other slightly improves sentiment while slashing paraphrase and STS. We hypothesize that this is the case because we finetune on sentiment, paraphrase, and similarity in that order, and since finetuning with the worst learning happens at the end, the model forgets some of the good learning that it might've done before and generalizes more to similarity, which we have seen does not result in good performance for the other two tasks. Given this, we wanted our model to take into account the strengths of finetuning sentiment and paraphrase (and not similarity), while also not forgetting what it's learned from previous iterations of finetuning, so we implement a round-robin in which we interleave batches from sentiment and paraphrase. We find that this improves the performance of our model on all three tasks. This makes sense because sentiment was probably finetuned on its own dataset, while simultaneously paraphrase was finetuned on its dataset, and STS improved by the model's learning on the paraphrase dataset since paraphrase and similarity are similar tasks.

We also observe differences in the loss curves of the round robin finetune with SMART experiment and the baseline experiment. As you can see from the figure, our finetuned model learns a lot faster than the baseline model. While baseline plateaus, our better performing model continues to learn over the epochs, highlighting the strengths of multitask learning.

While training our model with SMART, we see that dev accuracy is almost the same, if not higher than the train accuracy, whereas training on baseline BERT without SMART does not have as good dev accuracies compared to train accuracies. This, in addition to the fact that the model performs better on the test set than dev set, indicates that SMART is helping the model with not overfitting.

Seeing as the sentiment accuracy we got was 0.372 and paraphrase accuracy was 0.656, while we didn't worsen baseline, that is only slightly better than random guessing which has an accuracy of 0.2 and 0.5 respectively. Given this, we would've liked to achieve better overall results, and we observe some of the shortcomings of SMART.

One potential issue with SMART is that it may not be effective for all types of datasets or models. For example, SMART may not be as effective for datasets with highly imbalanced classes.

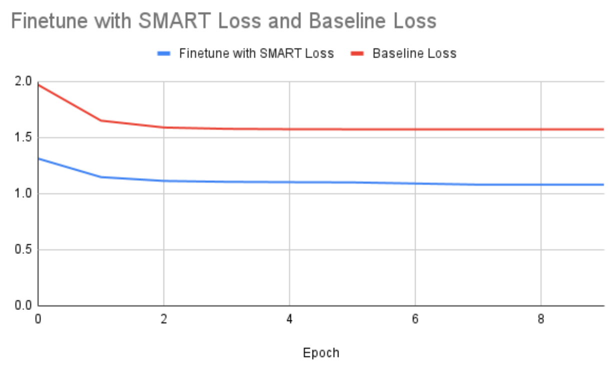


Figure 4: Loss Curves

Another potential issue is that SMART may increase the training time and computational resources required to train the model. This is because the additional regularization term requires additional computations during each training iteration. As a result, it may not be practical to use SMART for very large datasets or in situations where training time is a limiting factor.

Additionally, SMART may introduce a trade-off between model performance and interpretability. This is because the regularization term can make the model weights more diffuse and less interpretable. While this may improve generalization, it can make it more difficult to understand how the model is making its predictions.

Finally, SMART may not be effective for all types of tasks. For example, it may not be as effective for tasks that require a high degree of precision or that are highly dependent on specific features of the input data. In such cases, other regularization techniques may be more appropriate.

In summary, while SMART can be an effective technique for improving the generalization and robustness of BERT multitask classification models, it is important to carefully consider its potential shortcomings and limitations.

7 Conclusion

Overall, BERT model implemented with SMART produces significantly better results. However, although we implemented smoothness-inducing regularization, only the complexity of the model is managed. With a less complex model, we can finetune on each of the respective downstream tasks, and our results showed that the accuracy performance with this model. Even with our less complex model, it was difficult to improve the accuracy on all three downstream tasks simultaneously, as there were often tradeoffs between task performances. Highlights of our work include exploring the effects of smoothness-inducing regularization on a model's performance, and reducing a model's complexity can bolster performance as well. Despite having several downstream tasks to train on, we learned that training on all three simultaneously may not necessarily yield the best result, and future avenues include exploring training on these same tasks but with weights. Additionally, other future avenues include exploring other methods to prevent aggressive finetuning that can be used in tandem with our less complex model.

8 Future Work

Given more time and compute resources, we would've done more experiments and explored different combinations of training BERT with and without SMART and using various combinations of finetuning on particular tasks. We also would've liked to explore how our model does using different loss functions like contrastive loss using and applying additional finetuning such as cosine similarity.

References

[1] Zaid Alyafeai, Maged Saeed AlShaibani, and Irfan Ahmad. A survey on transfer learning in natural language processing, 2020.

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

[3] 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. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[4] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning, 2018.

[5] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. Ernie 2.0: A continual pre-training framework for language understanding, 2019.