Regular(izing) BERT

Stanford CS224N Default Project

Parker Kasiewicz
Department of Computer Science
Stanford University
parkerka@stanford.edu

Eric Zhu
Department of Computer Science
Stanford University
ericz27@stanford.edu

Abstract

There are many potential ways to build an NLP model that generalizes to multiple different downstream tasks while using shared weights. For our default project, we aimed to extend the default min-BERT model to generalize better on sentiment analysis, paraphrase detection, and semantic similarity analysis. First, we utilized the Quora and SemEval STS benchmark dataset, in addition to the Stanford Sentiment Treebank dataset, in finetuning our model. We used a round-robin style training so that our model could learn weights relevant to all three downstream tasks without forgetting too much from the others ones. Through round-robin training, our model significantly outperformed the model trained just on the SST dataset. Then, we set out to attempt to regularize this new baseline, taking inspiration from Jiang et al. (2020), by implementing smoothness-inducing adversarial regularization and Bregman proximal point optimization. We show specific results in the paper, but we generally find that the SMART framework produces limited effect compared to more substantive shifts in model architecture, like round-robin training.

1 Key Information to include

Mentor: Arvind Venkat Mahankali

2 Introduction

Bidirectional Encoder Representations from Transformers, or "BERT", is a model built on the Transformer architecture which relies on attention mechanisms to generate contextual word representations. It's key innovation lies in its incorporation of bidirectional word representations, a significant alteration from previous models that only processed text sequentially.

BERT model development consists of two main steps: pretraining and finetuning. During pretraining, BERT captures general language representations that can be used across a wide range of downstream tasks through a process of unsupervised training on a corpus of unlabeled text data.

In the finetuning step, BERT adjusts the pretrained model to fit the specific needs of a given task through a process of supervised training on a set of labeled data. One of the major issues faced during the finetuning step is that the limited data for downstream tasks and the high complexity of the models can lead to over aggressive fine tuning resulting in overfitting and poor generalization on unseen test sets. Jiang et al. (2020) presents a pair of extension features which alter the loss function optimization framework in order to combat overfitting during the finetuning process. These two key features are:

1) Smoothing-Inducing Adversarial Regularization: This regularization aims to smoothen out predictions by imposing a penalty when the model makes different predictions on a datapoint and a perturbed datapoint that is very close by.

2) Bregman Proximal Point Optimization: This optimization method serves to prevent over-aggressive parameter updating by imposing a strong penalty at each optimization step which stabilizes the fine-tuning process and leads to much smoother parameter updates.

3 Related Work

The optimizer we are using for our baseline model is the AdamW optimizer, a modified version of the Adam optimizer that involves decoupling the weight decay from the gradient-based optimization steps taken with respect to the loss function. As detailed in Loshchilov and Hutter (2019), L2 regularization, as traditionally implemented, does not provide the same performance as true weight decay in terms of generalizing to unseen data. Thus, decoupling weight decay improves the generalization of the Adam optimization significantly and the authors present empirical evidence that with decoupled weight decay regularization, AdamW can compete with SGD with momentum on image classification datasets.

The primary inspiration for our project extensions is the paper by Jiang et al. "SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Prinicpled Regularized Optimization." In it, the authors explain the issue of overfitting of NLP models caused by the high complexity of pre-trained models and the limited data for downstream tasks. They propose a new framework aimed at preventing overfitting and improving generalization comprised of smoothness-inducing regularization and Bregman proximal point optimization. In their experiments, their proposed framework achieved state-of-the-art performance on a number of benchmarks including GLUE, SNLI, SciTail, and ANLI.

There are also several other papers that attempt to remedy the issue of over-aggressive fine-tuning with different methods which rely on hyper-parameter tuning heuristics. In particular, Howard and Ruder (2018) presents a strategy that adjusts the learning rate in a rule-of-thumb manner and progressively activate the layers of the language model to enhance the fine-tuning effectiveness. In their experiments, they find that their method outperforms state-of-the-art models on six text classification tasks. Peters et al. (2019), on the other hand, presents a method that adapts certain layers while freezing the others.

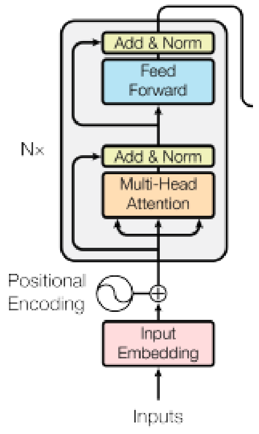
The Bregman proximal point optimization process outlined in the SMART paper is heavily informed by existing literature itself. In particular, the momentum parameter in our planned implementation of Bregman is based on the "Mean Teacher" method in Tarvainen and Valpola (2017) which maintains an exponential moving average of model weights instead of label predictions.

4 Approach

We used the vanilla minBERT model, finetuned on all three datasets in a round-robin style, as our baseline before implementing the two key features outlined in Jiang et al. (2020), Smoothing-Inducing Adversarial Regularization and Bregman Proximal Point Optimization. We will detail our baseline model as well as give a more in-depth explanation of our extensions.

4.1 minBERT Baseline Model

Our baseline for this project is the minimal BERT large language model implementation outlined in the default project handout with skeleton code. The architecture of our vanilla BERT model consists of a tokenization step, an embedding layer, transformer layers, and a pooling layer. The BERT model splits sentence inputs into word pieces with a WordPiece tokenizer. The BERT model has 12 transformer layers, each of which consist of Multi-head Self-attention and a Position-wise Feed-forward network with norm layers after each. Multi-head attention enables the model to take in information from different representation subspaces at different positions simultaneously. A visual representation of the transformer layer architecture is presented below.



The optimizer used for the baseline model is the AdamW optimizer, a stochastic gradient descent method that uses adaptive learning rates for different parameters based on the first and second moments of the gradients. It also incorporates a weight decay regularization parameter to prevent overfitting.

Instead of just training our baseline on the SST dataset, we decided to incorporate a round-robin training mechanism to include the other datasets. To do so, when fine-tuning, in each epoch, we process one batch from the SST dataset, then one batch from the paraphrase dataset, then one batch from the STS dataset. For each training method explored in this paper, including this baseline, we trained on 775 randomly-chosen, equally sized batches of size eight for each dataset. We chose this method of round-robin training to attempt to combat the forgetting problem, as generally discussed by Aljundi (2019), where they note "The main obstacle towards developing continually adapting systems is the "catastrophic forgetting" of old learned information once new knowledge is learned." As such, to balance the need to train on the other downstream tasks but not forget too much of what is learned about any particular task, we decided round-robin training might produce some of the most generalizable learnings from our model.

4.2 Smoothing-Inducing Adversarial Regularization (SIAR)

When defining the overall objective function for fine-tuning, SIAR adds an extra term $\lambda_s * R_s(\theta)$ to the already existing loss term, such that the optimization problem is finding

$$min_\theta F(\theta) = L(\theta) + \lambda_s * R_s(\theta). \tag{1}$$

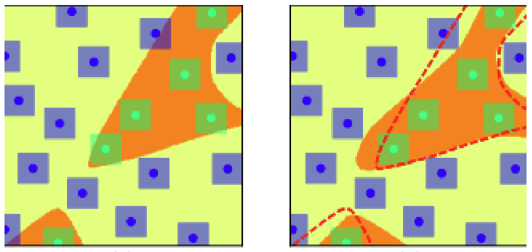
λ_s is a hyperparameter that controls the strength of the regularization, and $R_s(\theta)$ is a smoothness-inducing adversarial regularizer defined as:

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

Jiang et al. (2020) define l_s to be the symmetrized KL-divergence, which is:

$$l_s(P, Q) = D_{\text{KL}}(P \,||\, Q) + D_{\text{KL}}(Q \,||\, P) \tag{3}$$

In other words, for each datapoint, SIAR finds the perturbation that maximizes the difference between the original datapoint's predictions and the perturbed datapoint's predictions, subject to a norm constraint for the perturbation, and then adds that difference to the overall loss function. Intuitively, by penalizing the model when it predicts differently when only perturbed slightly, this sort of regularization should encourage our model to be smooth within the regions close to our training datapoints, reducing overfitting. An example from the original paper is below, where plot a is without SIAR and plot b is with SIAR and smoother decision boundaries are evident.


(a) (b)

As we can see from the equations, SIAR requires solving an optimization problem to find the perturbation that maximizes the KL-divergence, turning fine-tuning into a nested optimization problem that wasn't feasible for training on Colab. We implemented the optimization using various SciPy optimization methods, but it too compute intensive to get any results.

So, we modified the regularization equation in our model to be:

$$R_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} \text{random}\left\{\|\tilde{x} - x_i\|_p \leq \epsilon\right\} l_s(f(\tilde{x};\theta), f(x_i;\theta)) \tag{4}$$

This way, we continue to encourage smoothness in the epsilon neighborhoods around our training points, but this doesn't increase the computational complexity of our fine-tuning.

4.3 Bregman Proximal Point Optimization (BPPO)

Bregman Proximal Point Optimization is an optimization method designed to prevent over-aggressive parameter updates through imposing a strong penalty at each optimization step. The penalty term serves as a regularizer that prevents updates θ_{t+1} that deviate too far from θ_t and consequently smooths out parameter updates. Using the Bregman Proximal Point Optimization method has been shown to allow models to more effectively maintain knowledge learned from the out-of-domain data in the pre-trained model. Bregman optimization can be further improved by introducing a momentum term to the update process, which essentially captures all of the previous parameter states thorugh an iterative weighted average.

At the $(t+1)$-th iteration, the Bregman Proximal Point method, as defined by Jiang et al. (2020), takes:

$$\theta_{t+1} = arg\min_{\theta} F(\theta) + \mu D_{Breg}(\theta, \tilde{\theta}_t) \tag{5}$$

where $\mu > 0$ is a tuning parameter, $\tilde{\theta}_t = (1-\beta)\theta_t + \beta\tilde{\theta}_{t-1}$ where β is the momentum parameter, and $D_{Breg}(\cdot,\cdot)$ is the Bregman divergence:

$$D_{Breg}(\theta, \theta_t) = l_s(f(x_i;\theta), f(x_i;\tilde{\theta}_t)) \tag{6}$$

We have decided to apply the penalty term from Bregman, $\mu D_{Breg}(\theta, \tilde{\theta}_t)$, on top of the existing AdamW Optimizer method at each step in order to retain the adaptive learning rates and decoupled weight decay regularization that the AdamW Optimizer provides while further smoothing out the update process by discouraging over-aggressive updating with the penalty term.

We have opted to simplify the Bregman Divergance calculation in our optimizer in order to increase model efficiency, using Euclidean distance instead of symmetrized kl-divergence. When we tried to implement BPPO using symmetrized kl-divergence, finetune runtime increased significantly without improving model performance. We also introduced a second, smaller momentum parameter that is used to directly weight θ_{t-1} in the update step in order to further dampen and smooth out parameter changes and prevent over-aggressive updates. That is, $\theta_{t+1} = arg\min_{\theta} F(\theta) + D_{Breg}(\theta, \tilde{\theta}_t) + \beta_2\theta_{t-1}$.

5 Experiments

5.1 Data

For sentiment analysis we used the following datasets:

The Stanford Sentiment Treebank (SST), which consists of 11,855 single sentences from movie reviews, each of which was annotated by human judges with a sentiment label of one of five categories (negative, somewhat negative, neutral, somewhat positive, positive). The SST dataset is split into train (8,544 examples), dev (1,101 examples), and test (2,210 examples) sets.

The CFIMDB dataset, which consists of 2,434 polarized movie reviews. Each review has a binary classification of either negative or positive. The CFIMDB dataset is split into train (1,701 examples), dev (245 examples), test (488 examples) sets.

For paraphrase detection we used the following dataset:

The Quora dataset, consisting of 400,000 question pairs with labels for whether or not particular instances are paraphrases of each other. The Quora dataset is split into train (141,506 examples), dev (20,215 examples) examples, and test (40,431 examples) sets.

For semantic textual similarity we used the following dataset:

The SemEval STS Benchmark dataset, which consists of 8,628 sentence pairs of varying similarity on a scale from 0 to 5 where 0 is unrelated and 5 is equivalent meaning. The SemEval STS Benchmark dataset is splits into train (6,041 examples), dev (864 examples), and test (1,726 examples) sets.

5.2 Evaluation method

We used prediction accuracy to evaluate our model's performance on sentiment classification. We used prediction accuracy to evaluate our model's performance on paraphrase detection. We used the Pearson correlation coefficient of the true similarity values against our predicted similarity scores to evaluate our model's performance on semantic textual similarity.

5.3 Experimental details

For our baseline model, we used the default pretrained minBERT model and then finetuned on the SST, Quora, and STS datasets. We used a learning rate of $1e^{-5}$, trained for six epochs, and each epoch took around ten minutes. We chose to train for six epochs, since we observed that when we trained for much more than six the model would start to generalize worse to the dev set.

Then using the same learning rate and number of epochs, we finetuned our pretrained model on the SST, Quora, and STS datasets with three different strategies. First, we applied Smoothness-Inducing Adversarial Regularization when finetuning on the SST dataset. We chose to apply SIAR just to the SST finetuning steps because compared to paraphrase and semantic similarity analysis, sentiment analysis is a more subjective test, so we hypothesized that regularizing our model as to not overfit and produce unsmooth predictions on that particular task would be specifically important. Then, we applied Bregman Proximal Point Optimization when finetuning on all three datasets. Finally, we applied both at the same time.

For SIAR in particular, we tested different combinations of hyperparameters: $\lambda = .01, .1, 1, 10$ and $\epsilon = .01, .1, 1, 10, 100, 1000$. The values we settled on and used for our final experiments were $\lambda = .1$, $\epsilon = .1$. For BPPO in particular, the tuning hyperparameters we tested were $\mu = 0.1, 0.2$, $\beta_1 = 0.8, 0.9, 0.95$, and $\beta_2 = 0.05, 0.1$. The set of hyperparameters that achieved the highest overall model performance when finetuning with BPPO was $\mu = 0.2, \beta_1 = 0.8, \beta_2 = 0.05$.

5.4 Results

For overall performance, our results are about what we expected. Because we limited training to only a fraction of the paraphrase data and our extensions were mostly targeted toward the SST data, we didn't expect impressive scores for STS or paraphrase accuracies. We also expected model

Table 1: Experiment Test Scores

Model	Score			
	SST	**PARA**	**STS**	**Overall**
Baseline	.531	.714	**.370**	**.643**
BPPO ($\mu = 0.2, \beta_1 = 0.8, \beta_2 = 0.05$)	.530	**.717**	.343	.640
SIAR + BPPO ($\mu = 0.2, \beta_1 = 0.8, \beta_2 = 0.05, \epsilon = .1, \lambda = .1$)	**.535**	.714	.337	.639

Table 2: Experiment Dev Scores

Model	Accuracy		
	SST	**PARA**	**STS**
No Round Robin	.516	.483	-.012
Baseline	.522	.714	.378
SIAR ($\epsilon = .1, \lambda = .1$)	.520	.722	.370
BPPO ($\mu = 0.2, \beta_1 = 0.8, \beta_2 = 0.05, \epsilon = .1, \lambda = .1$)	.529	.717	.348
SIAR + BPPO ($\mu = 0.2, \beta_1 = 0.8, \beta_2 = 0.05, \epsilon = .1, \lambda = .1$)	.521	.710	.359

performance to significantly improve when implementing round-robin training, which we did in each model except the one titled "No Round Robin". Because the "No Round Robin" only trained on SST data, it didn't learn information useful to specifically answering paraphrase and semantic similarity questions, so finetuning on all three downstream tasks significantly improved performance, as we expected.

When comparing the models we trained, our results for SIAR are what we expected. Because we weren't able to adversarially choose the perturbations due to compute costs, the amount of substantive regularization was probably minimal at best. This is because when choosing a point in the epsilon neighborhood of each training point using random perturbation, the likelihood that the chosen point has substantively different predictions (and thus contribute substantively to the regularizing term) is extremely low. This is why the nested optimization is helpful, but because we relied on random perturbations, we expected SIAR to not have an effect.

As far as the results with BPPO, we see a slight increase in paraphrase detection accuracy over the baseline but no significant evidence of improvement on sentiment analysis and semantic textual similarity tasks. Bregman may offer a better regularization effect with its incorporation of prior knowledge and momentum term, which could be especially effective in a setting such as paraphrase detection where a model may be prone to overfitting and rely on lexical matching to determine paraphrases. However, the lack of improvement on the other downstream tasks suggests that the Bregman regularization term added on top of the AdamW Optimizer may not be effective or needed. This makes sense if we consider the fact that AdamW already has built in features specifically to discourage overfitting such as its weight decay term and adaptive learning rates. Thus, it makes sense that including another regularization term over an already robust optimization method would lead only to marginal returns.

6 Analysis

Overall, we can separate the different elements of our model and then look at their effects in tandem. First, we see that round-robin fine-tuning (and training on all three datasets, in general) significant improves results on downstream applications that hadn't already been seen. Round-robin training has the advantage of spreading out the learning on different datasets rather than sequential training, where the model might "forget" some of its learnings on datasets that were trained earlier in the sequential process.

Then, looking at smoothness-inducing adversarial regularization, we can see that empirically, the adversarial nature of the chosen perturbations is a crucial aspect of the regularization. Without being able to efficiently optimize for choosing the maximally disruptive perturbation, the effect on the overall loss function is very likely to be minimal, since predictions are likely to be similar when choosing a random point around a training point.

Looking at Bregman proximal point optimization, we see that implementing the momentum-based Bregman update penalty term on top of AdamW achieves only marginal accuracy improvements for one of the three downstream tasks. We interpreted this as evidence of the robustness of weight decay decoupling in AdamW as a regularization method. Thus, it seems that adding additional safeguards against overfitting may have been overkill and does not achieve a synergistic effect with AdamW as we had hoped.

Overall, it seems that while we were generally able to implement regularization techniques according to the SMART framework, they weren't nearly as impactful on the results of our BERT model as other, more substantive improvements in model architecture could be (like round-robin training). This is a somewhat expected result, especially considering the compute-intensiveness of the SMART framework (because of the nested optimization) compared to other tweaks in model architecture.

7 Conclusion

Overall, we implemented the SMART regularization framework (smoothness-inducing adversarial regularization and Bregman proximal point optimization) on a pre-trained BERT model and saw minimal impact on downstream prediction results. We learned both that the adversarial nature of some regularization techniques is crucial in producing substantive smoothing effects and that compute resources are a critical limiting factor in ambitious regularization methods (and that compute, generally, is really important). We also discovered how valuable substantive changes in model architecture can be compared to smaller tweaks in specific parts; for example, expanding our fine-tuning to include both the paraphrasing and semantic similarity produced significant improvements. We would be interested in future work investigating other adversarial regularization methods that don't significantly impact the computational complexity of training time or compute resources.

References

Rahaf Aljundi. 2019. Continual learning in neural networks.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. 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.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. ICLR.

Matthew Peters, Sebastian Ruder, and Noah Smith. 2019. To tune or not to tune? In *Proceedings of the 4th Workshop on Representation Learning for NLP*. Association for Computational Linguistics.

Antti Tarvainen and Harri Valpola. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*.