Investigating BERT through Fine-Tuned Regularization and Layer-Level Adaptations for Multi-Task Performance.

Stanford CS224N Default Project

Arjun Pandey, Neel Narayan
Department of Computer Science
Stanford University
arpandey@stanford.edu, neelsn@stanford.edu

Abstract

This project evaluates the minBERT model under different optimizations to compare and contrast how it performs in multi-task scenarios. Our motivation is to understand how low-level basic model adaptations can help improve performance when working towards multi-task efficiency. Specifically, we aim to investigate how regularization techniques like Bregman Proximal Point Optimization and smoothness induced regularization compare to layer adaptation methods, where each layer between BERT is adapted to account for generalization across tasks. Here, we will highlight how BERT performs when its current parameters are fine-tuned without any new additions, and when layer adaptations are done to learn task specific parameters within a reasonable budget using SMART (**SM**oothness-inducing **A**dversarial **R**egularization and B**R**egman p**R**oximal poin**T** op**T**imization) [1] and adapter modules [2] as frameworks for executing fine-tuned regularization and layer-level adaptations, respectively.

1 Introduction

Multi-task learning is an inductive bias, meaning that a single large language model has the robustness to be generalized across several general language understanding tasks. Current literature largely relies on building custom layers on top of existing models to study multi-task scenarios, or fine-tuning hyper-parameters to adapt to the task at hand. BERT was originally trained on the Wikipedia dataset with small-scope language understanding objectives of next-sentence prediction and language modelling. However, the architecture can be generalized to realize strength across a variety of general language understanding tasks. Previous state-of-the-art results on the GLUE dataset have shown BERT's general language ability. This paper will explore the improvement of a base BERT model for downstream tasks using optimization and layer-level editing techniques. The primary goal is to investigate how BERT, specifically when fine-tuned using new data and objectives, can be utilized for multi-task learning.

Training BERT is computationally expensive since it has approximately 110 million parameters and was pre-trained on both the Wikipedia corpus of over 2.5 billion words and the Toronto BookCorpus of 800 million words. To re-train BERT from scratch requires expensive and potent computational resources. However, we hypothesize that BERT's trained parameters have sufficient strength to understand language-level meaning that can be leveraged for other tasks. Recently, researchers have been motivated to see how to build upon this base model to get BERT to generalize well across unknown tasks while taking advantage of optimization techniques.

Pre-trained models have exploded in popularity over the past several years. Starting from pre-trained vectors like GLoVe and word2vec which were essential in capturing singular word meaning, more robust large language models have now started to capture semantic technicalities of

language understanding. This paradigm shift started first with contextual models trained on large corpora, such as the OpenAI transformer, ELMo, and more. These models allow for pre-trained representations that represent a significant segment of language that encodes contextual information. Multi-task learning is based on using those pre-trained capabilities to capture the semantic and structural meaning behind language for generalization across several tasks only by using small dataset training and fine-tuning rather than training from scratch.

In this research project, we wanted to investigate how BERT's pre-trained version equipped with language understanding capabilities can be improvised to perform better in multi-task inference scenarios.

2 Related Work

2.1 BERT and PALs

BERT and PALs suggest a technique of editing layers within the model inspired by 'residual adapter modules' [3] so that instead of having task specific parameters as additions to the "top" of the model i.e. just before the output classification layer, these parameters are learned along with the rest of the model. One major drawback to adding layers on top of BERT is increasing the amount of parameters in the model - specifically, adding an extra layer for each task results in an approximately $1.67\times$ increase in the number of parameters. To avoid transformations that significantly increase the number of parameters, we will introduce task-specific functions of the form:

$$TS(\mathbf{h}) = V^D \times g(V^E\mathbf{h})$$

where V_e is the "encoder" matrix, V_d is the "decoder" matrix, and g is an arbitrary function which we will be experimenting with. Schematically (**Figure 1**) the BERT and PALs architecture looks as follows:

PALs work by adding an optional residual parallel connection and layer norm that are computed in parallel with each BERT layer. This is because we want to use our original BERT layer in the cases where PALs output a zero vector. Mathematically, we can define this to be calculated as such:

$$\mathbf{h}^{l+1} = LN(\mathbf{h}^l + SA(\mathbf{h}^l) + TS(\mathbf{h}^l))$$

The true motivation behind PALs is to move away from the bias of adding parameter transformations and instead use our "encoder" and "decoder" matrices, which operate on each sequence separately, as a means of making multi-task BERT more useful for sequences. BERT and PALs led us to explore how layer-level adaptations work when fine-tuning the model. Our initial goal was to use the proven architectures the authors came up with for our extension, but we ended up using the idea of having optional adapters as the basis for one of our extensions.

2.2 Parameter Efficient Learning for NLP

Parameter Efficient Learning for NLP is a technique used to mitigate the costs of fine-tuning models in the presence of multiple downstream tasks, as an entirely new model is required for each task. [4] Gelly et al. propose transfer learning with adapter modules as they add only a few trainable parameters per task, allowing for a compact and extendible model with a high degree of parameter sharing. Additionally, this adapter-based learning applies to both multi-task (which is what we are focused on) and continual learning, but since the tasks do not interact, all of the shared parameters are frozen, allowing the model to retain perfect 'memory' of previous tasks despite using a small amount of parameters specific to each individual task.

To achieve (1) good performance, (2) sequential training that does not require simultaneous dataset access, and (3) the addition of a small number of parameters per task, bottleneck adapter modules that add new layers to a pretrained network, while the original network's parameters are frozen and shared by multiple tasks, are used. In our project, we will adapt the idea of bottleneck adapter modules for task specific parameters in a smaller dimensional space.

2.3 Noise Stability Regularization for improving BERT fine-tuning

When training large language models like BERT, a problem that often arises is that when there are only a small number of training samples available, it is more difficult to adapt BERT to a specific task. As such, fine tuning to specific tasks becomes difficult. To mitigate this issue, **L**ayer-wise **N**oise **S**tability **R**egularization (LNSR) to induce stabilization and generalization can be used.

LNSR is a lightweight and effective regularization technique that improves the local Lipschitz continuity of each BERT layer and thus, the smoothness of the entire model.Empirical results show that the fine-tuned BERT models regularized with LNSR obtain significantly more accurate and stable results. As such, we will be tuning the perturbation of the embedding layers to induce smoothness when there is high variability between different tasks.

3 Approach

In this section, we will explore how we got our baseline models and outline ideas for training approaches. The baseline implementation was done entirely by us, whereas the code for the extensions was adapted from external research.

Our first extension was implementing the SMART technique, which is a regularization technique for multi-task learning that makes use of trust-based region methodology to prevent aggressive updating of loss, and subsequently, parameter values. In the original paper, the authors acknowledge how only one technique is not sufficient to realize the full effects of smoothness desired, so the following techniques outlined below are specifically used because they have been shown to complement each other when introduced in BERT.

3.1 SMART Framework

Here, we will describe how the Smoothness-Inducing Adversarial Regularization technique and Bregman Proximal Point Optimization work together to achieve results as defined in the SMART framework.

3.1.1 Smoothness-Inducing Adversarial Regularization

This regularization technique works by adding a regularization term to the loss function of the model that encourages the network to produce smooth and continuous output. The regularization term is typically based on an adversarial loss, which measures the difference between the model's output and a smoothed version of that same output. This smooth version can be obtained by applying a low-pass filter or a smoothing operation to the output. Specifically, let's consider a model $f(;,\theta)$ for a target task where x_i are input level embeddings and y_i are the associated outputs which, for the sake of explanation, can be considered labels. This method optimizes the following:

$$min_\theta F(\theta) = L(\theta) + \lambda_s \times R_s(\theta)$$

Where $L(\theta)$ is the loss function which is task-specific, $R_s(\theta)$ represents the regularization functions, and λ is the fine-tuning epsilon parameter which controls the strength of the regularization we desire. This can be adjusted as the variability in tasks go up.

3.1.2 Bregman Proximal Point Optimization

The SMART framework also considers the Bregman proximal optimization to prevent aggressive updating, which works by adding a strong penalty if the model takes an erratic or large step.

The basic idea behind Bregman proximal point optimization is to minimize a convex function $f(x)$ subject to some convex constraints by using an iterative approach to solving a sequence of proximal sub-problems. On each iteration, the proximal operator is applied to the current iterate x to obtain $x + 1$, which is closer to the optimal solution. Bregman proximal point optimization (PPO) is particularly useful when the objective function f has a separable structure, i.e. can be decomposed into a sum of simpler functions. In this case, each proximal sub-problem involves solving a simple proximal operator, which can be done efficiently using various algorithms. Specifically, if we con-

sider having the BERT pre-trained model $f(;,\theta)$, the Bregman PPO takes an approach of updating the t^{th} iteration as follows:

$$\theta_{t+1} = argmin_{\theta} F(\theta) + \nu \times D_{Breg}(\theta, \theta_t)$$

The Bregman PPO template provides flexibility since the choice of the ν parameter can help tune the Bregman PPO divergence, which is defined as:

$$D_{Breg}(\theta, \theta_t) = l_s(f(\bar{x}_i; \theta), f(x_i, \theta_i))$$

The primary rationale behind using Bregman PPO was to introduce a trust-region based regularization [5] to deter aggressive updating and ensure that the algorithm only takes steps within a small neighborhood of the previous iteration.

3.2 Adapter Modules

We were inspired by the methodology first introduced in the BERT and PALs framework, which introduced task-specific parameters and low-rank layers as alternative BERT blocks to train for multi-task learning. As shown above, projected attention layers introduced as adapters use encoder and decoder matrices that operate on each sequence separately to enhance the representational capacity of the model without adding a significant amount of parameters. However, as mentioned above, this quickly became hard to achieve since it required a complete re-training of BERT, which was not computationally feasible. Thus, we tried to replicate the ideology behind BERT and PALs by introducing adapter modules [4].

Houlsby et. al. introduce a methodology of adapters as an alternative to optimization based fine-tuning, showing that it requires orders of magnitude less parameters while maintaining model performance. We followed a similar bottleneck approach, but in two formats: **Figure 1** produces the adapter formations straight after the feed-forward layer and **Figure 2** executes the formations after the layer norm.

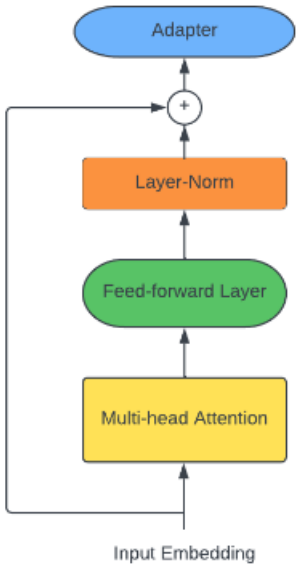


Fig 1: Adapter Transformations after feed-forward

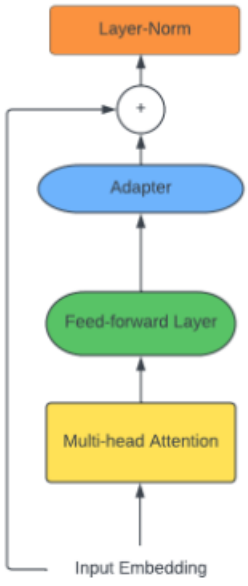


Fig 2: Adapter Transformations after layer-norm

This adapter block was added to all 12 transformer blocks we had in our models. The intention was to train it to retain task-specific information. The architecture follows a simple bottleneck method which is expressed as such:

$$\text{Adapter}(\mathrm{H}_i) = \mathrm{W}^D \left(g_{\text{non-linear}} \left(W^E h_i + b_E\right)\right) + b_D$$

Our goal was that the task-specific attention layers W^E will reshape and project the input to a smaller dimensional space which will be transformed under the non-linearity. Subsequently, W^D will project it back onto a higher dimensional subspace where b_E and b_D are included to account for biases. This is to ensure that only parameters enclosed in the smaller subspace are fine-tuned.

The adapter also inherently has a skip connection which we wanted to include from BERT and PALs. This is because if the parameters of the projection layers are initialized to near-zero, the adapter merely acts as an identity function.

Finally, we also added a learning hidden unit contribution (LHUC) transformation, which modifies a unit in a network by a reasonable scalar. Since there are way fewer units in a network than parameters, we included them to account for a reasonable parameter budget. We hypothesized that learnable units can only improve a network without significant downsides on our baseline.

3.3 Implementation

To achieve our desired implementation we adapted our initial code from the minBERT version [6] provided to us. For the SMART framework [7], we took inspiration from the implementation by Jiang et. al. We made the intentional choice of keeping our noise parameter low and steady at 1×10^{-5} since we were only dealing with three tasks, two of which had a similar scope of understanding (Paraphrase Detection and Semantic Textual Similarity). Lastly, to implement our adapter modules, we read through and took inspiration from Google Research [8], which was then used as the basis to code our adapter block. For the block, we chose our transformation size (i.e. the updated hidden size of the block) to be 64, and initialized our W_E and W_D tensors to be random, and then initialized the truncated normalization, with standard deviation of 1×10^{-3} to ensure it was within certain bounds.

To implement the SMART framework, we had to make certain design choices. We defined our fine-tuning epsilon parameter $\epsilon = 1 \times 10^{-6}$ and the noise perturbation was chosen to be 1×10^{-5}. These values were primarily inspired by the best results from the original paper.

The initial part of the project encompassed implementing a minimalistic version of BERT, which includes the multi-head self-attention, the transformer layers, and the embedding layers defined in the original BERT [9] model. Subsequently, we also implemented the Adam optimizer which was used to train across all different optimizations we implemented.

All the code was powered by PyTorch and trained on a singular GPU using Google Colab.

4 Experiments

For the purpose of this project we started by implementing a minimal version of BERT which was trained on the following two datasets: Stanford Sentiment Treebank (SST) and CFIMDB. This was done to have tested, trained, and executed a reasonable replication of the BERT architecture. Next, we wanted to get a baseline evaluation to compare to the results of our optimizations. For this purpose, we trained our BERT architecture using three datasets: Quora Question Pair, SST, and SemEval. This gave us a baseline working model to start with. We then implemented our subsequent extensions i.e. SMART and adapter modules. All of the evaluation and data information is outlined in the sections below.

4.1 Datasets

This section will outline the datasets we worked with across all the experiments we conducted.

Stanford Sentiment Treebank: The SST dataset contains 11,855 single sentences parsed using the Stanford parser, generating 215,154 unique phrases from the parse tree that were then labelled by human judges.

CFIMDB Dataset: This dataset consists of 2,434 movie reviews with human labelled polarity representations i.e. positive or negative.

Quora Dataset: Consists of 400,000 question instances with labels indicating what particular questions are paraphrases of each other. Since, this dataset was extremely comprehensive and require computational resources not available at the time of study, we stemmed the dataset to approximately 10,000 input samples.

SemEval STS Benchmark Dataset: 8,628 different sequence pairs of varying similarity on a scale from 0 to 5 i.e. from unrelated to equivalent meaning.

For initial baseline training of our sentiment classification model, we will be utilizing the pre-provided datasets from Stanford University which are the Stanford Sentiment Treebank and the CFIMDB dataset. In the cases of evaluation for extending to more general language understanding tasks, we used two separate datasets: the Quora dataset and the SemEval STS dataset.

4.2 Hyper-Parameters

The following table outlines the model configurations used. The choice to keep them uniform was done specifically to have uniform results free of hyper-parameter bias.

Configurations	
Parameter	Value
Learning Rate	1×10^{-3}
Number of Epochs	10
Batch Size	8
Dropout Probability	30%

4.3 Evaluation

It is important to note that for the tasks of sentiment classification and paraphrase detection, we used a basic accuracy score, and for semantic textual similarity, we used a Pearson correlation score, which finds the ratio between the covariance and standard deviations of two samples.

Our first step was to use our earlier BERT model trained only on the SST dataset (plain) as a test to see how it performs in multi-task scenarios i.e. instead of giving it relevant data for the two other downstream tasks, paraphrase detection and semantic similarity, we set out to evaluate its potency in a completely unsupervised scenario (these results were also published in our project milestone). Next, we started by training BERT using the Quora, SST, and SemEval datasets to obtain a reasonable baseline to see how a fine-tuned pre-trained version of BERT model performs in a multi-task scenario. Finally, we implemented our extensions using the SMART framework and adapter modules. The below table (**Table 1**) outlines our final results for all of the models:

Model	Dev Sentiment Acc	Dev Paraphrase Acc	Dev STS Corr.
BERT$_{plain}$	31.9%	55.2%	−0.067
BERT$_{baseline}$	29.3%	58.6%	−0.047
BERT$_{SMART}$	30.8%	58.2%	**0.202**
BERT$_{adaptbefore}$	31.2%	58.4%	**0.195**
BERT$_{adaptafter}$	30.1%	56.3%	−0.087

Table 1: Evaluation for Trained Models on Dev sets.

The sentiment accuracy for the BERT$_{plain}$ model being higher than the rest is indicative of how the model was only trained on that particular dataset and learned parameters that are specifically tuned for that task.

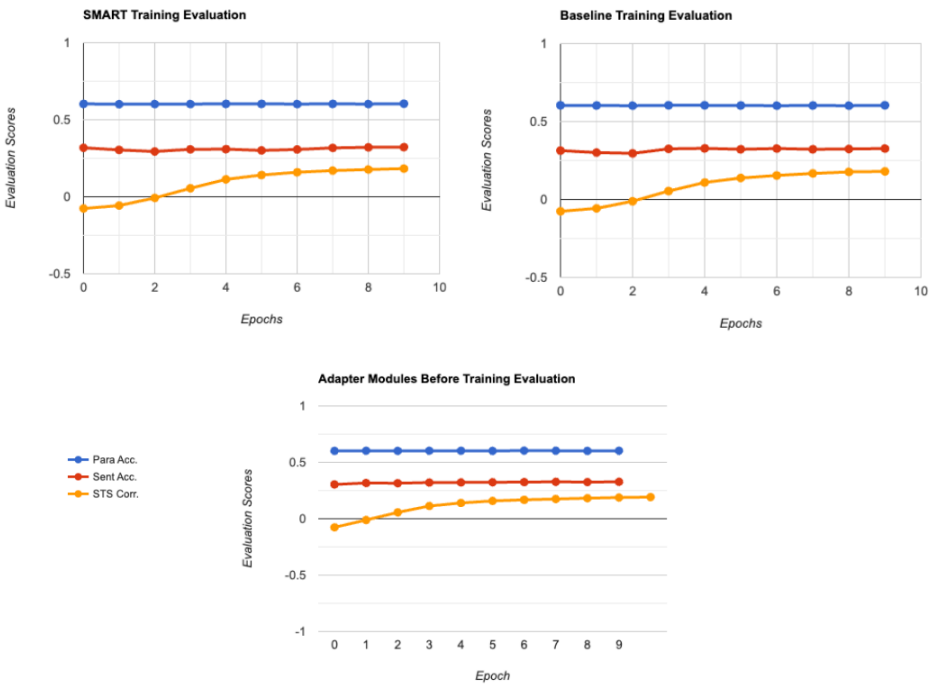
We also trained the model corresponding to Fig. 1. The results, however, were very standard and not comparable to the baseline. This can be explained by the fact that because the structure of our code enabled dropout alongside layer-norm, we were projecting to a smaller subspace with unaccounted or zeroed out units, and thus, we didn't include it in our analysis.

4.4 Analysis

These results indicate that the SMART model is comparable in accuracy to the sentiment task with the "plain" model. One particularly interesting output was a noticeably high correlation score for the "SMART" model as compared to the other scores, indicating that SMART helps smaller datasets (like the SemEval dataset) since it prevents overfitting by regularizing loss across all tasks. It makes sure that we generalize well by penalizing model complexity when it focuses on solely improving a singular task. Given the performance of SMART in the other two sectors, it is a very good indication of obtaining a robust model if trained on several more general language understanding tasks.

We also observed a good score for the *adaptbefore* model which suggests that a small injection of learned parameters for specific tasks are able to generate valuable results. This is indicative of adapter modules avoiding the problem of catastrophic forgetting i.e. the model still retains information that is initially given to it, since the pre-trained parameters are built top of the model by freezing previous parameters. Adapter modules tend to improve performance in higher layers, ensuring that we can take full advantage of BERT's semantic understanding capability and focus on parameters that are geared towards achieving the task at hand. While the results do not surpass those of state of the art models, we were able to generate a compact model that surpasses the baseline, which is the task we initially set out to achieve. One reason as to why the model had comparable scores for the remaining two tasks could be attributed to our choice of truncated initialization, which was done on a scale of 1×10^{-3} - this may have led the adapter module to act like an identity function and thus, update smaller changes.

To dive deeper into analyzing all three of our key models, we studied how they individually trained. The general patterns look very similar across all three models, but epoch level-data can give some idea into how our final dev scores vary:



We observe that the pattern trend across all three lines is very similar for both SMART and our baseline, which is expected since parameters are being learned in the same way. However, the trend varies for adapter modules since the direct injection of parameters kicks in from the beginning and gradually influences the inference being done. Another noticeable difference across all three graphs is the almost constant horizontal line for paraphrase accuracy, which suggests that the model could not perform very well on this specific task despite having extensions that optimize for task-specific parameters. However, there is still a big question of how the similar values in training

for SMART and baseline translate to radically different scores - this may be explained by overfitting on training data, but confirmation would require visualizing activation networks of BERT as heat maps to see how different parts of the network are working during inference.

5 Conclusion

In this study, we conducted experiments to see how two specific additions to the BERT model impact its performance in a multi-task scenario: the first was loss regularization inspired by the SMART framework, and the second was layer-level adaptations through adapter modules. We showed that both methodologies help generalize the model well and specifically, help boost semantic textual similarity correlation by preventing overfitting from the other models. The adapter module extension also creates a pathway to explore how more compact models can be utilized to achieve parameter-efficient extension of BERT.

Future work for this study will entail diving into activation networks to analyze the different model layers affected by our extensions and using these ideas to train across an entire suite of GLUE tasks. We would also like to experiment with different adapter modules by taking pre-trained task-specific ones available online.

References

[1] Haoming Jian, Penghcheng He, Weizhu Chen, Xiadong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Association for Computational Linguistics*, 2019.

[2] Asa Cooper Stickland and Iain Murray. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, 2019.

[3] Sylvestre-Alvise Rebuffi, Alkan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. 2017.

[4] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. 2019.

[5] Gould N. I. Conn, A. R. and P. L. Toint. Trust region methods. In *MOS-SIAM Series on Optimization: Trust Region Methods*, 2000.

[6] Gabriel Poesia. CS 224N Default Final Project - Multitask BERT. `https://github.com/gpoesia/minbert-default-final-project`, 2023.

[7] Xiaodong. Multi-Task Deep Neural Networks for Natural Language Understanding. `https://github.com/namisan/mt-dnn`, 2020.

[8] Google Research. Adapter-BERT. `https://github.com/google-research/adapter-bert`, 2020.

[9] Devlin Jacob, Min-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.