BERT: Battling Overfitting with Multitask Learning and Ensembling

Stanford CS224N Default Project

Mentor: Timothy Dai

Javier Nieto
Department of Computer Science
Stanford University
jgnieto@stanford.edu

Annabelle Jayadinata
Department of Computer Science
Stanford University
abellej@stanford.edu

Abstract

Transformers have revolutionized the field of neural NLP thanks to the advantages of the self-attention mechanism. Fine-tuning a Transformer which has previously been pre-trained on a large corpus of data to carry out a downstream task has been found to be a highly successful approach used by many SOTA models. In this paper, we fine-tune a BERT Transformer for three tasks: sentiment classification, paraphrase detection, and semantic textual similarity. To combat overfitting, we use multitask training across all tasks, SMART regularization, LoRA and ensembling. We also explore hyperparameter tuning, PCGrad, L1L2 regularization and DoRA, and examine their impact on scores. We observe much better scores than the baseline and find conclusive evidence that our techniques to fight overfitting are effective and our downstream model performs well with unseen data.

Member Contributions: Javier wrote around two-thirds of the code and one-third of the report, while Annabelle wrote around one-third of the code and two-thirds of the report. Both contributed equally to planning the project and researching extensions.

1 Introduction

The realm of machine analysis and processing of human natural language data, known as natural language processing (NLP), has seen significant advancements and widespread applications from translation to speech-to-text transcription among others. At present, we have been able to achieve fantastic results for such tasks due to massive data collection efforts, compute power, and costs undertaken by large organization to train large models on billions of words. However, for smaller players interested in solving more specific tasks, data deficit and poor model generalization remain a hindrance, and this is where transfer learning comes into play by allowing us to use fine-tune large pre-trained models and re-use them in new settings.

In this paper, we will improve on fine-tuning approaches on Bidirectional Encoder Representations from Transformers (BERT) Devlin et al. (2019), which is originally designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. We will investigate methods that can can help such representations generalize better for different downstream tasks, namely sentiment analysis, paraphrase detection, and semantic textual similarity. By incorporating several advanced fine-tuning techniques including DoRA, LoRA, an original multitask training process, PCGrad to minimize weaknesses associated with multitask learning, and other optimizations such as Elastic Net and SMART regularization, we seek to promote more efficient knowledge transfer and improve effectiveness across the different tasks. Our results demonstrate notable improvements over baseline models. In the following sections, we will review related works, outline our approach, investigate experimental results, and discuss the implications of our findings for future advancements in multitask BERT NLP.

2 Related Work

In order to adapt BERT to perform particular tasks, the process of fine-tuning still presents a challenge regarding parameter efficiency. Hu et al. introduced a novel parameter-efficient fine-tuning (PEFT) method known as Low-Rank Adaptation (LoRA) which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks Hu et al. (2021). Statisically, LoRA is able to reduce the number of trainable parameters by 10,000 and the GPU memory requirement by 3 times as compared to GPT-3 175B fine-tuned with Adam.

Building on top of LoRA is Weight-Decomposed Low-Rank Adaptation (DoRA)Liu et al. (2024), another novel parameter-efficient fine-tuning (PEFT) method that incorporates weight decomposition, achieving a learning capacity closely resembling full fine-turning (FT) while mitigating overfitting, without any additional inference latency over LoRA Hu et al. (2021). The main upgrade that sets DoRA apart is that their method limits LoRA to concentrate mainly on directional adaptation that is optimized and made more stable through weight decomposition, although in the process, still allowing the magnitude component to be tunable.

Furthermore, although BERT was able to achieve pioneering performance across eleven NLP tasks, the original paper did not explore it as a multitask learning model. We believe that investigating this aspect is essential for improving BERT's adaptability and potential in diverse real-world NLP applications. However, a significant challenge with multitask learning is the difficulty of optimizing one task without negatively impacting the performance of another. Yu et al recommended a technique known as PCGrad which which solves this problem by projecting the gradient of each of the two tasks in conflict onto the normal plane of the gradient of the other task, thereby reducing the amount of destructive gradient interference between tasks Yu et al. (2020). Additionally, we will investigate regularization techniques like SMART Jiang et al. (2020) to mitigate model overfitting.

3 Approach

Baseline. We started with minBERT [1], a minimal version of BERT as described in the DFP handout, along with the utilization of pre-trained BERT-base weights[2] from Huggingface. For our baseline, we extend the model with a linear layer for each task, which takes pooled encodings from minBERT as input and produces logits or a raw value, depending on the task, as output. This baseline model establishes the initial performance benchmark before fine-tuning the minBERT weights.

Task-specific approach.

- Sentiment analysis: use BERT to encode the input sentence ("[CLS] Sentence [SEP]") and a single linear layer with dropout to output five logits, one for each sentiment classification option. Those values are fed through a sigmoid and optimized with cross-entropy loss.
- Paraphrase detection: use BERT to encode each of the two input sentences separately as above, as well as encoding both sentences at the same time as follows "[CLS] Sentence A [SEP] Sentence B [SEP]". That way, we can take advantage of the self-attention mechanism, as described by Devlin et al. We then take those three embeddings and pass them as input to a linear layer with dropout with a single output logit. That value is fed through the sigmoid function and optimized with binary cross-entropy loss. We observed empirically that not feeding both sentences to BERT together and only doing that both worsen scores, which is why we used this combined approach.
- Semantic textual similarity: use the same approach as paraphrase detection, but the output is instead fed through ReLU and optimized with smooth L1 loss. We used ReLU since STS scores are bounded between 0 and 5, and we observed that while scores did not surpass 5, they frequently were slightly negative. We tried both MSE loss and smooth L1 loss and found the latter to be slightly better.

Apart from the baseline sentiment analysis approach described in the DFP handout, all code for this section is fully original. The intuition to encode both sentences together is original, and came as

[1] `https://github.com/barneyhill/minBERT`

[2] `https://huggingface.co/google-bert/bert-base-uncased`

a result of an in-depth reading of the BERT paper. It required us to hack the dataset preprocessing mechanism. We thank our mentor, Tim, for suggesting we also encode each sentence separately.

Multitask Batch Sampling. Upon reviewing the data, we identified a significant imbalance: the sentiment analysis dataset (SST) contains 8,500 examples, the textual similarity dataset (SemEval) 6,000 examples, and the paraphrase detection dataset (Quora), 210,000 examples. Initially, we sampled 6,000 examples each epoch from each dataset to compensate for the imbalance, in effect pretending that SST and Quora were "as small as" SemEval for a given epoch. While the difference between SST and SemEval in this case was negligible, we were severely underutilizing the Quora dataset. Over 5 epochs, the model would see around 14% of the dataset in the best case. Consequently, we proposed a down-weighting strategy for the Quora data. Specifically, at each iteration we sample 10 examples from Quora and average the losses.

We implemented this strategy ourselves.

Multitask Learning. We primarily focus on a single multitask model consisting of a single BERT model and three task-specific heads as described above, where the weights of the BERT model are shared across all tasks. We then performed multitask fine-tuning to find parameters θ of the BERT model f_θ that achieves high average performance across all three training tasks. We adapted the provided code to, each iteration, run a training batch on each of the three tasks and do backpropagation with the sum of the losses:

$$\mathcal{L} = \mathcal{L}_{\text{sentiment}} + \mathcal{L}_{\text{paraphrase}} + \mathcal{L}_{\text{STS}}$$

We hypothesized that the reason we were not getting the scores we expected was a clash between gradient vectors of different tasks. A comparatively larger gradient of one task will dominate the average gradient, and a high positive curvature may significantly overestimate improvement in performance from the dominating task while underestimating its degradation. Hence, we integrated a technique recommended bu Yu et al, PCGrad, which solves this problem by projecting the gradient of each of the two tasks in conflict (negative cosine similarity) onto the normal plane of the gradient of the other task, thereby reducing the amount of destructive gradient interference between tasks Yu et al. (2020). The algorithm is as follows:

Algorithm 1 PCGrad Update Rule

```
Require: Model parameters $\theta$, task minibatch $\mathcal{B} = \{\mathcal{T}_k\}$
$\mathbf{g}_k \leftarrow \nabla_\theta \mathcal{L}_k(\theta)\ \forall k$
$\mathbf{g}_k^{\text{PC}} \leftarrow \mathbf{g}_k\ \forall k$
for $\mathcal{T}_i \in \mathcal{B}$ do
    for $\mathcal{T}_j \overset{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order do
        if $\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j < 0$ then
            // Subtract the projection of $\mathbf{g}_i^{PC}$ onto $\mathbf{g}_j$
            Set $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i^{\text{PC}} - \frac{\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$
return update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$
```

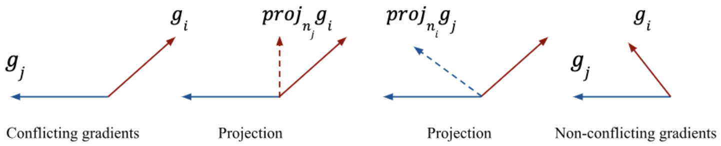


Figure 1: Task i's gradient is projected onto the normal vector of j's gradient using the formula: $\mathbf{g}_j = \mathbf{g}_i - \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$

We developed the multitask model from scratch and implemented the PCGrad technique on top of it by adapting the code provided in the paper.

Fine-Tuning Methods for BERT: LoRA & DoRA. We explored parameter efficient finetuning, specifically, LoRA, to improve our model. LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers' change during adaptation instead, while keeping the pre-trained weights frozen, as shown in 2. This reduces the number of trainable parameters whilst still achieving a higher training throughput.

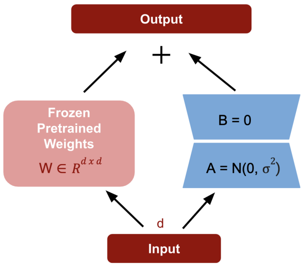
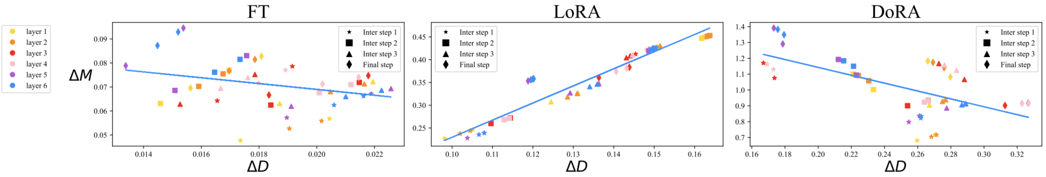


Figure 2: Reparametrization where only the injected smaller low-rank matrices, A and B, are trainable and optimized.

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA constrains its update by representing it with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, the modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx \tag{1}$$

A is initialized using a random Gaussian while B is initialized to zero, so that $\Delta W = BA$ is zero at the beginning of training. $\Delta W x$ is then scaled by $\alpha/\sqrt{r}$, where α is a constant. When optimizing with Adam, tuning α is roughly the same as tuning the learning rate if we scale the initialization appropriately. As a result, we simply set α to the first value we try and do not tune it in order to reduce the need to retune hyperparameters when we vary r Hu et al. (2021).



(a)

(b)

(c)

Figure 3: (a) Magnitude and direction updates of FT (b) LoRA, (c) DoRA, taken from Liu et al. (2024)

However, Liu et al. found that LoRA exhibits proportional relationship between changes in direction and magnitude when updating the weight matrix, which is opposite of learning patterns found in FT, as shown in 3, where a more varied learning pattern with a relatively negative slope is observed, indicating a different learning capability. It shows that LoRA falls short in executing slight directional changes alongside more significant magnitude alterations which is a feature characteristic of FT that is desirable Liu et al. (2024). Hence, they proposed DoRa, an adaptation method that decomposes the pre-trained weight into its magnitude and directional components and finetunes both of them. To avoid initialization concerns, DoRA is initialized with pre-trained weight W_0, such that magnitude vector $\mathbf{m} = \|W_0\|c$ and directional matrix $\mathbf{V} = W_0$. The directional component is then updated through LoRA, resulting in the following equation: $W' = \frac{m(V+\Delta V)}{\|V+\Delta V\|_c} = \frac{m(W_0+\underline{BA})}{\|W_0+\underline{BA}\|_c}$, where ΔV is the incremental directional update learned by multiplying two low-rank matrices B and A, and the underlined parameters denote the trainable parameters. We can derive the gradient of loss $\mathcal{L}$ as follows: $\nabla V'\mathcal{L} = \frac{m}{\|V'\|_c}\left(I - \frac{V'V'^{\top}}{\|V'\|_2^c}\right)\nabla W'\mathcal{L}$. This equation reveals that the weight gradient $\nabla W'\mathcal{L}$ is scaled by $\frac{m}{\|V'\|_c}$ and is projected away from the current weight matrix, which aligns the gradient's covariance matrix more closely with the identity matrix, thereby enhancing the learning stability of LoRA.

The study showed that similar to FT, DoRA is characterized by a distinct negative stope and was able to demonstrate the ability to make only substantial directional adjustments with minimal changes in magnitude, or vice versa, thereby showing learning patterns closer to FT. The reason behind this might be due to the fact that pre-trained weights might in fact already possess sufficient knowledge suitable for downstream tasks. Hence, having a larger magniture or direction alteration alone is sufficient for adaptation.

We adapted code from the LoRA and DoRA papers respectively. LoRA required significant rework to adapt.

Elastic Net Regularization. Initial iterations hinted that we were experiencing significant overfitting challenges. Although training loss continued to decrease, accuracy on dev set peaked around epoch 3 and worsened thereafter. Hence, we implemented Elastic Net Regression, which merges both L1 and L2 penalty on the loss function across all three tasks using the following equation:

$$\mathcal{L} = \mathcal{L}(y, y_{\text{pred}}) + \alpha_1 \sum_{i=1}^{m} |\theta_i| + \alpha_2 \sum_{i=1}^{m} |\theta_i|^2 = \mathcal{L}(y, y_{\text{pred}}) + \alpha_1 \|\theta\|_1 + \alpha_2 \|\theta\|_2^2$$

We implemented this regularization ourselves.

SMART Framework. Aggressive fine-tuning often causes the models to overfit and fail to generalize to unseen data. To address this issue, we explored the SMoothness-inducing Adversarial Regularization and BRegman pRoximal poinT opTimization (SMART) framework proposed by Jiang et al Jiang et al. (2020). It effectively manages the complexity of the model and the utilization of Bregman proximal point optimization, which is an instance of trustregion methods, can prevent aggressive updating.

Specifically, given the model $f(\cdot;\theta)$ and n data points of the target task denoted by $\{(x_i, y_i)\}_{i=1}^n$, where x_i denote the embedding of the input sentences obtained from the first embedding layer of the language model and y_i are the associated labels, their method solves the following optimization for fine-tuning:

$$\min_{\theta} F(\theta) = \mathcal{L}(\theta) + \lambda_s R_s(\theta),$$

where $\mathcal{L}(\theta)$ is the loss function defined as

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

and $\ell(\cdot,\cdot)$ is the loss function depending on the target task, $\lambda_s > 0$ is a tuning parameter, and $R_s(\theta)$ is the smoothness-inducing adversarial regularizer. $R_s(\theta)$ is defined as

$$R_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|x_i^e - x_i\|_p \leq \epsilon} \ell_s(f(x_i^e;\theta), f(x_i;\theta)),$$

where $\epsilon > 0$ is a tuning parameter annd ℓ_s is the symmetrized KL-divergence for the classification task and the squared loss for the regression task.

Significant adaptations had to be made in order to properly integrate SMART into our framework, and we mostly wrote original code based on the original algorithm.

Ensembling. Ensemble models are a machine learning approach to combine multiple models in the prediction process. Each model will make some mistakes due to randomness introduced during training. By aggregating the outputs of diverse models, the ensemble can achieve better generalization and accuracy than just a single model alone. As our final model, we ensembled three single-task models trained separately on only one of the downstream tasks, as well as our multitask model, and used a voting approach to get the final prediction of the ensemble model. In particular, we averaged values for the regression task (semantic similarity) and we took the mode in paraphrase and sentiment analysis, classification tasks.

We implemented the ensemble model ourselves from scratch.

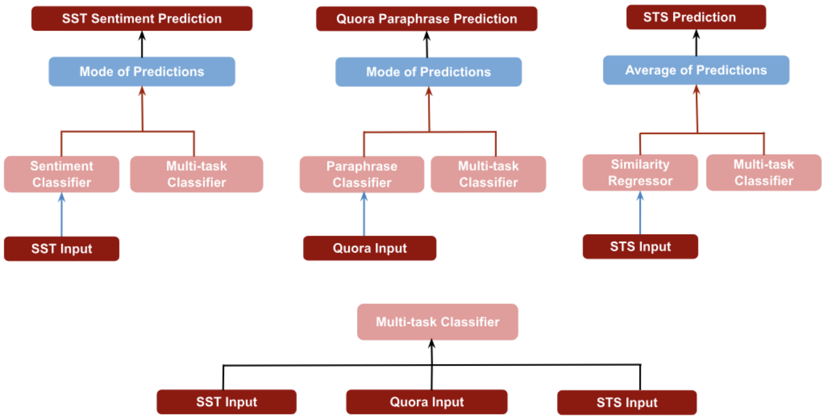


Figure 4: Our final model was composed of an ensemble of four models. One for each task, plus another multitask model.

4 Experiments

Data. To train and evaluate or model, we used the three datasets specified in the DFP handout. Namely, Stanford Sentiment Treebank (SST) for sentiment classification, Quora dataset for paraphrase detection, and SemEval STS Benchmark dataset for semantic textual similarity analysis.

Evaluation method. Following DFP, we evaluate the model using accuracy for SST and Quora, and Pearson correlation for STS. The overall score is computed as an average among the three, with correlation linearly mapped from $[-1, 1]$ to $[0, 1]$. In addition to that, we also track the number of total and trainable parameters and epoch time to evaluate the trade-off between performance and computational cost.

Experimental details. We ran our model on Modal labs with NVIDIA T4 or A10G GPU's, as available. We trained BERT using AdamW Optimizer, learning rates of $\{1 \times 10^{-5}, 2 \times 10^{-5}, 3 \times 10^{-5}\}$, and performed training for about 1.5 hours. We used a dropout rate of 0.3 for BERT weights and 0.5 for the last linear layer, which we determined empirically.

Results.

Model	**Score**	**Sent. Acc.**	**Para. Acc.**	**STS Corr.**	**Params (Train/Total)**
Baseline (last linear layer)	0.588	0.381	0.661	0.445	8K / 109M
Single task (full model)	0.735	0.490	0.791	0.848	324M / 324M
Multitask Equitable Sampling	0.743	0.517	0.785	0.851	109M / 109M
Multitask Down-weighting	0.756	**0.527**	0.813	0.755	109M / 109M
Multitask Down-weighting PCGrad	0.500	0.802	0.842	0.741	109M / 109M
Multitask Down-weighting SMART	0.759	0.526	0.823	0.856	109M / 109M
Multitask Down-weighting SMART PCGrad	**0.762**	0.525	0.836	0.850	109M / 109M
Multitask Down-weighting SMART PCGrad LoRA	0.760	0.510	**0.841**	**0.859**	25M / 110M
Multitask Down-weighting SMART PCGrad DoRA	0.714	0.442	0.793	0.813	25M / 110M
Ensemble	0.768	0.532	0.840	0.863	1.2B/1.2B

Table 1: Dev set scores for selected models.

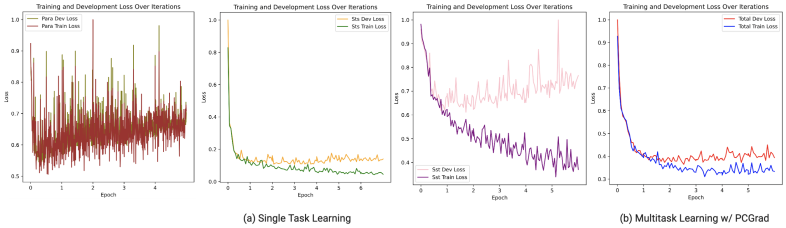
Overall Score	Sentiment Acc.	Paraphrase Acc.	STS Corr.
0.770	0.538	0.840	0.862

Table 2: Leaderboard test results from final model ensemble.

Models limited to pre-trained weights shared across the three downsteam tasks have the lowest parameter overhead since only the final classification layers are trained. However, they also demonstrate the lowest accuracy at 0.588. Fully fine-tuned models perform significantly better, reaching an accuracy of approximately 0.7. However, they are the most expensive to train, requiring around 200M additional parameters.

5 Analysis

Multitask Learning.



(a) Single Task Learning

(b) Multitask Learning w/ PCGrad

Figure 5: Comparison of Single Task and Multitask Learning. Rightmost is Multitask.

Single-task models tend to perform with low accuracy. However, our multitask model, which employs PCGrad, effectively leverages the inherent correlations among these tasks, improving its overall training and prediction capabilities.

Parameter efficient Fine-tuning (PEFT). We can achieve comparable overall performance with our optimized models while using far fewer parameters with PEFT methods. By utilizing LoRA and DoRA techniques, the number of trainable parameters is significantly reduced from 109 million to 25 million. We found that LoRA enhanced our model's performance and reduced overfitting by decreasing the number of trainable parameters, thus avoiding excessive complexity. In contrast, DoRA did not perform as well. This is likely because, as the authors suggest, DoRA's similarity to full fine-tuning compared to LoRA mitigates its efforts to reduce overfitting.

Gradient Surgery. PCGrad took a much longer runtime of approximately 80% more time overall, as pairwise comparison of loss gradients is performed at every timestep during training. This is inefficient and also did not generate substantial improvement to our overall accuracy, but instead, dropped the accuracy of our multitask model. We hypothesize that PCGrad undermines to some extent the benefits of multitask learning. In a sense, conflicting gradients are a feature, not a bug, which prevent the model from overfitting to training data too much, and projecting them might allow the model to better memorize examples.

Furthermore, in fact, implementing PCGrad on top of both LoRA and DoRA reduced accuracy significantly. This might be due to the fact that PCGrad changes the gradient directions to resolve conflicts, and such orthogonal projections can interfere with the fine-tuning dynamics and low-rank structures that LoRA relies on, leading to suboptimal updates. Furthermore, LoRA already limits the parameter space by focusing on low-rank adaptations, and with PCGrad further restricting it, will cause insufficient capacity for proper learning and adaptation.

PCGrad	PEFT	LR	Hidden, Last Dropout	Dev Sent.	Dev Para.	Dev STS	Dev overall
✗	✗	1×10^{-5}	0.3, 0.5	0.516	0.821	0.857	0.755
✓	✗	1×10^{-5}	0.3, 0.5	0.500	0.802	0.842	0.741
✗	DoRA	2×10^{-5}	0.3, 0.5	0.302	0.769	0.810	0.659
✓	DoRA	2×10^{-5}	0.3, 0.5	0.253	0.632	-0.030	0.457
✗	LoRA	1×10^{-5}	0.3, 0.5	0.495	0.801	0.837	0.738
✓	LoRA	1×10^{-5}	0.3, 0.5	0.257	0.734	0.808	0.632

Hence, we decided not to combine these two techniques in a single model, even if we ensembled some of them together.

Hyperparameter tuning.

Overfitting was a significant issue in our initial trials, so we introduced dropout to regularize the model. To mitigate overfitting, we experimented with different dropout rates in the hidden layers and the final layer of our model. The overall development score was higher (0.723) for the higher dropout rates, compared to 0.706 for the lower dropout rates, reinforcing the benefit of higher dropout rates in improving the model's robustness and performance. This configuration provided a balanced improvement across various tasks, making it a better choice for our experiments.

Hidden, Last Dropout	PEFT	PCGrad	LR	Dev Sent.	Dev Para.	Dev STS	Dev overall
$0.5, 0.6$	✗	✓	2e-5	$\mathbf{0.464}$	$\mathbf{0.783}$	0.844	$\mathbf{0.723}$
$0.1, 0.5$	✗	✓	2e-5	0.425	0.768	$\mathbf{0.852}$	0.706

We also experimented with different learning rates on our model. Specifically, we found that as the learning rate increased, the overall score generally decreased as we are overshooting optimas and had poor generalization, indicating that lower learning rates performed better in our experiments. A learning rate of 1e-5 allowed us to achieve the best overall score of 0.750, with significant improvements in sentiment classification and paraphrase detection accuracy.

Learning Rate	Overall Score	Sentiment Classification Acc.	Paraphrase Detection Acc.	Semantic Textual Similarity Corr.
1×10^{-5}	$\mathbf{0.750}$	$\mathbf{0.510}$	$\mathbf{0.814}$	0.854
2×10^{-5}	0.736	0.476	0.805	$\mathbf{0.856}$
3×10^{-5}	0.720	0.445	0.789	0.853
4×10^{-5}	0.696	0.406	0.764	0.836
5×10^{-5}	0.699	0.427	0.765	0.809

Table 3: Learning rate fine-tuning results

In addition, we also fine tuned the λ in the SMART framework. Using their recommended 1 sampling step, $\epsilon = 1e^{-6}$, $\eta = 1e^{-3}$, $p = \infty$, we experiment with different choices for λ. As shown in 6, we find that λ = 2 performed well and hence, we decided to adopt this value for future experiments. Lower values had little effect and higher values had too great an impact.

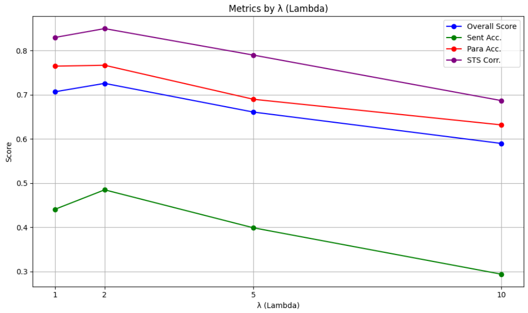


Figure 6: Effect of λ in SMART regularizer on dev model accuracy

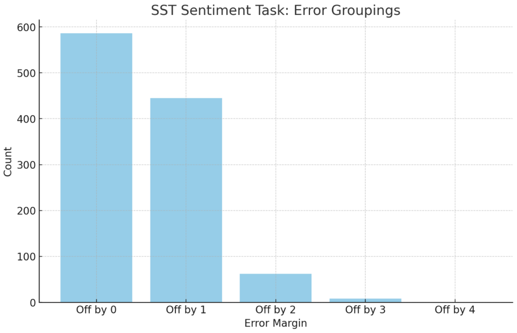


Figure 7: SST Result Breakdown

We also examined our lowest-performing task, sentiment classification. Instances where the prediction is slightly off, such as predicting *very negative* instead of *slightly negative*, are often subjective. Considering only exact matches (Off by 0) as correct, makes our model appear to be less accurate than it really is.

6 Conclusion

Fully fine-tuned models excelled across all tasks due to their extensive trainable parameters and the capacity for task-specific fine-tuning. Supplementary enhancements like LoRA can further elevate efficiency by significantly reducing parameter overhead while maintaining competitive performance across tasks through reduced over-fitting. Conversely, the performance of PCGrad with LoRA or DoRA resulted in inferior performance as it resulted in over-correction and simplification of the model. Conflicting gradient adjustments underscores the importance of carefully balancing regularization techniques with model architecture adaptations.

Moreover, our hyperparameter tuning efforts revealed that higher dropout rates and lower learning rates were essential to improved generalization and overall performance. While our study achieved promising results in parameter efficiency and task performance, it is essential to acknowledge the limitations, including the complexity introduced by multitask learning dynamics and the sensitivity of results to hyperparameter settings. Future research could explore more sophisticated strategies for integrating regularization techniques with parameter-efficient fine-tuning methods, as well as investigate adaptive approaches to gradient correction that align more closely with low-rank model adaptations.

References

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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*, pages 2177–2190, Online. Association for Computational Linguistics.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning.

7 Ethics Statement

One potential ethical concern of using our BERT system to analyze sentiment is the introduction of unfair bias. Suppose that our system were to be employed as a review aggregator by taking as input social media posts which mention a given movie to produce “audience score” on a movie review website. Not only must we ensure that overall accuracy is reasonably good, but there is a danger of learning that, say, the use of African American Vernacular English (AAVE) in a post implies a more negative review, or that “I love how *It* takes horror to the next level” is a negative review because it says the word “horror.” In those examples, movies more likely to have reviews in AAVE or mentioning the word “horror” would get statistically worse audience scores, which could disproportionately hurt niche yet important genres. A potential mitigation strategy is to train a model for each movie category in isolation, and then force all genres to have the same average.

A concern about paraphrasing detection is that, if the system were to be employed to identify plagiarism in academic works, there would be a risk of false positives. In this case, the great risk is user’s overdependence and trust of the system. For example, the Texas A&M professor who failed students after ChatGPT told him that it had written the students’ papers, even though Transformers only predict the next token and ChatGPT has no recollection of specific usage by other users in the past. That case exemplifies that our model too can, as a result of human negligence, be the cause of great anxiety in innocent students who did not know of its existence, let alone authorize its use. Perhaps the only way to mitigate this risk is to clearly warn users that paraphrasing detectors are fallible and must always be corroborated by a human reviewer.