Strategies for Optimization of minBERT for Multi-Task Learning

Stanford CS224N Default Project

Ankur Jai Sood
`jaisood@stanford.edu`

Cameron Heskett
`cheskett@stanford.edu`

Shinwoo Lee
`shinwoo@stanford.edu`

Abstract

In this project we aim to implement a multi-task minBERT model that achieves superior performance across three tasks: sentiment analysis (SST), paraphrase detection (PARA), and semantic textual similarity (STS) when compared to task specific fine-tuned minBERT models. We experiment with a variety of strategies, including various task scheduling techniques, low-rank adaption layers (LoRA), parallel adaption layers and projected attention layers (PALs), and SMART regularization to improve our model performance. We investigate the effect of these techniques individually on task specific performance and combine them to improve results compared to our baselines. The result of our efforts is a significantly improved model that achieves an overall accuracy of **78.8%** on our test set and outperforms our baseline multi-task and task specific fine-tuned models.

1 Key Information

- **TA Mentor**: Olivia Y Lee **External Collaborators:** No **Sharing project:** No
- **Team Contributions:**
  - Ankur Jai Sood: minBERT, task scheduling & PALs implementation, technical writing
  - Cameron Heskett: minBERT, LoRAs implementation, technical writing
  - Shinwoo Lee: minBERT, SMART implementation, technical writing

2 Introduction

One of the foundational challenges in natural language processing (NLP) is developing models that generalize effectively across multiple tasks, rather than only excelling at a single task. Even with the rise in popularity of LLMs, which can excel at performing a wide variety of NLP tasks, multi-task learning can be leveraged to optimize overall memory footprint, power consumption, and compute overhead for constrained use cases like mobile devices, autonomous vehicles, and robotics applications where models run on edge devices (Rebuffi et al. (2018)).

The complexity of multi-task learning arises from the need to balance performance across diverse tasks without sacrificing efficiency and effectiveness. Traditional approaches often involve increasing the number of parameters to boost performance, but this method does not scale efficiently, leading to increased memory requirements and prolonged training times. To address these challenges within the context of fine-tuning a pre-trained minBERT model, we have experimented with several recent techniques and evaluated their effectiveness: Low-Rank Adaptation (LoRA), Projected Attention Layers (PALs), and Smoothness-inducing Adversarial Regularization (SMART).

We show how applying these techniques can significantly improve multi-task minBERT performance across our three tasks of interest: sentiment analysis (SST), paraphrase detection (PARA), and semantic textual similarity (STS). These techniques are designed to enhance model performance and training scalability and our experiments demonstrate that these strategies not only help improve multi-task performance but can also help reduce training time and memory requirements during training. We find that by combining these techniques our multi-task model significantly outperforms our baseline multi-task model while matching the performance of task specific fine-tuned models. Our best model achieves an overall accuracy of **78.8%** with task specific scores of **52.4%** (SST Acc.), **90.0%** (PARA Acc.), and **87.9%** (STS Corr.) on our test set.

3 Related Work

Transformers, introduced by Vaswani et al. (2017), are now widely used in language modeling. BERT further (Devlin et al. (2018)) highlighted the importance of bidirectional pre-training in transformer models, demonstrating that the masked language model objective enables a model to excel at various language tasks without extensive task-specific architecture modifications. In this project, we explore techniques to further improve pre-trained minBERT through fine-tuning, enhancing multi-task performance across our three tasks of interest.

LoRA Hu et al. (2021) is a parameter-efficient fine-tuning technique that represents full neural network layers with low-rank matrix approximations, significantly reducing the number of trainable parameters. This approach addresses the challenge of fine-tuning extremely large models as parameter counts grow faster than available hardware memory. In our project, we use LoRA layers to optimize training speed and memory efficiency during multi-task fine-tuning.

Stickland and Murray (2019) introduce a modified BERT architecture which adds task specific adaption layers in parallel to the original BERT layers. Although this increases the number of trainable parameters overall, the goal is to perform as well as models fine-tuned for a specific task while only minimally increasing the total number of trainable parameters. The authors experiment with different forms of parallel adaption layers but find that low rank adaption layers and *Projected Attention Layers (PALs)* perform optimally especially on the GLUE Recognizing Textual Entailment (RTE) task. We believe that RTE is similar to our PARA task and so we also experiment with various forms of parallel adaption layers.

SMART (Jiang et al. (2020)) explores how a regularization method while fine-tuning can affect the performance of a model. The existing research is highly applicable due to its similarity in the base architecture and target tasks compared to ours. The paper aims to enhance performance for multiple NLP tasks including the SST benchmark which is part of our dataset, using BERT and RoBERTa architectures as baselines. Enhancing the performance of fine-tuning e.g. by reducing overfitting will be useful beyond this project as it effectively addresses the issue of a general shortage of fine-tuning (labeled) data, and improves the scalability of models moving forward.

4 Approach

As our baseline, we implemented a multi-task minBERT model consisting of 12 BERT transformer layers to perform SST, PARA and STS tasks. We implemented multi-headed self-attention and an AdamW optimizer (Kingma and Ba (2017)) with weight decay regularization (Loshchilov and Hutter (2019)). We also implemented three task specific output heads, one for each task. For SST, we added a classification head which takes the pooled representation of each input and classifies it by applying dropout and projecting it using a linear layer. For PARA, we implemented a binary classification head consisting of a dropout layer followed by a linear layer with one output. For STS, we implemented a head with one dropout layer and one linear layer with one output. For PARA and STS, we followed the original BERT approach in Devlin et al. (2018) and concatenated the two input sequences, using the `SEP` token as a delimiter, and concatenated the two attention masks with `1`'s as a delimiter. We used Cross Entropy loss for SST, Binary Cross Entropy loss for PARA, and Mean Squared Error loss for STS.

We trained our multi-task minBERT model across the three tasks, randomly selecting a task to train on every epoch. After each epoch, we evaluated on the tasks' validation datasets, using the average of SST accuracy, PARA accuracy, and STS Pearson correlation to decide if model parameters should be updated. The following sections describe our various extensions to our multi-task baseline model in detail.

4.1 Task Scheduling

We first extended our baseline multi-task minBERT model by implementing improved task scheduling methods. We implemented a scheduler which picks a task based on a provided policy every batch and trains for a specified number of batches for every epoch. We experimented with implementing three different task scheduling policies: random scheduling, round-robin scheduling, and annealed sampling (AS) scheduling.

Our random scheduling policy selects a task to train on every batch with a uniform random distribution. Since our random seed is fixed for the remainder of our tests, this is deterministic and can be compared across tests. Our round robin scheduling cycles through each task in a fixed order (SST, PARA, STS).

Annealed sampling, first introduced by Stickland and Murray (2019), addresses a fundamental issue posed by our training data, the stark imbalance in dataset size between the Quora dataset (used for PARA) and the SST/STS datasets. By sampling each dataset proportionally based on the number of entries in each dataset, AS helps balance the amount of training time spent on each task based on the following formula:

$$p_i \propto N_i^{\alpha}, \alpha = 1 - 0.8\frac{e-1}{E-1} \tag{1}$$

Here i refers to the task, N_i is the number of training examples for the task, and p is the probability of selecting a batch of examples from the task. α changes on each epoch e where E is the total number of epochs. α converges as e approaches E resulting in more even training on all the datasets towards the end of the run.

4.2 LoRAs

We hypothesized that introducing Low-Rank Adaption (LoRA) layers into the minBERT model could help improve training speed without significantly altering performance, potentially allowing us to train more epochs per unit time and increase the final model's multi-task performance. As described by Hu et al. (2021), LoRA works by freezing the pre-trained model weights and injecting trainable low rank decomposition matrices into each layer of the transformer architecture. To calculate layer output with LoRA layers, we introduce two new matrices, A and B, as shown in appendix E which are used to update the original weights W_0 at each layer based on equation 2.

$$h = W_0 + \alpha BA, \text{ where } B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} \tag{2}$$

Here, r is a tunable hyperparameter, with a minimum value of 1 yielding 1-D matrices $A \in \mathbb{R}^{d\times 1}$ and $B \in \mathbb{R}^{1\times k}$. This configuration requires updating only $k + d$ parameters, compared to $k \times d$ in the original layer. Increasing r raises the number of trainable parameters, potentially improving model performance at the cost of more memory and gradient calculations. Another tunable parameter, α, magnifies the result of the B and A multiplication before adding it to the original weights. While a higher α can speed up convergence, it may also cause instability in fine-tuning.

We implemented LoRAs from scratch and in our implementation we introduced a new layer type, LinearWithLoRALayer, with tunable r and α values to replace standard linear layers. This layer freezes the original layer's parameters from gradient updates during training and adds a new LoRA layer with corresponding A and B matrices. We replaced the key, query, and value components of the BERT self-attention mechanism, as well as the feed-forward layers, with LinearWithLoRALayers. In most tests, we used the *last-linear-layer* fine-tune mode, freezing almost all original parameters and updating only the low-rank A and B matrices during training, **reducing updatable parameters by over 500x** compared to the baseline. We also tested the *full-model* fine-tune mode, which freezes only the LinearWithLoRALayer-injected layers, **reducing updatable parameters by about 3.4x**.

4.3 Projected Attention Layers (PALs) & Parallel Adaption Layers

We implemented various forms of parallel adaption layers, inspired by Stickland and Murray (2019). In BERT layers (BL), the hidden vector h_j is processed by multi-headed attention (MH) and single self-attention (SA) layers, followed by a feed-forward network (FFN) and layer normalization (LN) as shown in equations 3, 4, and 5 below:

$$\text{MH}(\mathbf{h}_j) = W^o[\text{Attention}_1(\mathbf{h}_j) \dots \text{Attention}_n(\mathbf{h}_j)] \tag{3}$$

$$\text{SA}(\mathbf{h}_j) = \text{FFN}(\text{LN}(\mathbf{h}_j + \text{MH}(\mathbf{h}_j))) \tag{4}$$

$$\text{BL}(\mathbf{h}_j) = \text{LN}(\mathbf{h}_j + \text{SA}(\mathbf{h}_j)) \tag{5}$$

The original authors found that adding adaptation layers in parallel to the original BERT layers resulted in optimal performance. These layers use encoder (V^E) and decoder (V^D) matrices to project the input for task-specific functions (TS), where V^E is a $d_s \times d_m$ matrix, V^D is a $d_m \times d_s$ matrix ($d_s < d_m$) as described in Equation 6 ($d_m = 768$ in BERT). The original BERT layers are replaced with new layers (Equation 7) where the task specific function is inserted in parallel (See Appendix F, figure 5). The original paper uses the following syntax where l is the layer index:

$$\text{TS}(\mathbf{h}_j) = V^D g(V^E \mathbf{h}_j) \tag{6}$$

$$\mathbf{h}^{(l+1)} = \text{LN}(\mathbf{h}^l + \text{SA}(\mathbf{h}^l) + TS(\mathbf{h}^l)) \tag{7}$$

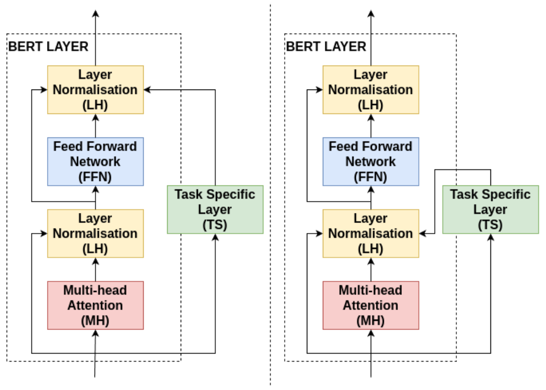

Figure 1: **Left**: Adaption layers original architecture (Stickland and Murray (2019)). **Right**: Our adaption layers alternative architecture (AA).

The authors introduce a novel task-specific layer called *Projected Attention Layer* (PAL), a low-dimensional multi-headed attention layer that adds task-specific parameters to each BERT layer. Importantly in PALs, V^E and V^D are shared across tasks but not across layers. We experimented with two architectures for adding adaptation layers: the original PALs as proposed by the authors, and a **new alternative architecture (AA)** we established for this project which we view as a form of task specific attention. The architectural differences are shown Figure 1.

We experiment with the following forms of TS for both architectures. The options marked with a *, we believe, are **novel or partially novel experiments** with adaption layers. For all tests we fine-tuned the full BERT model and adaptation layer parameters together, following the Stickland and Murray (2019) approach, and quantified the impact of different adaptation methods and our alternative architecture (AA) on task performance.

- TS as a low-rank transformation with $d_s = 100$ as described in the original PALs paper
- TS as PALs with $d_s = 204$ but without sharing V^E and V^D *
- TS as PALs with $d_s = 204$ and with shared V^E and V^D across layers (not tasks)
- Mixed adaptation: TS as PALs (shared V^E and V^D, $d_s = 204$) for the first six BERT layers, and TS as low-rank transformation ($d_s = 100$) for the last six BERT layers *
- TS as PALs ($d_s = 204$) with shared V^E and V^D and shared multi-headed attention across layers *

4.4 SMART

We implemented a variation of SMART (Smoothness-inducing Adversarial Regularization and Bregman Proximal Point Optimization, Jiang et al. (2020)) with the goal of reducing overfitting and improving prediction on unseen data. We implemented SMART partially by applying smoothness-inducing adversarial regularization. We set the fine-tuning objective to optimize for:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda \mathcal{R}(\theta), \tag{8}$$

$$\mathcal{R}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \alpha} \ell(f(\tilde{x}_i;\theta), f(x_i;\theta)), \tag{9}$$

$\mathcal{L}(\theta)$ is the target task's loss function, $\mathcal{R}(\theta)$ is the regularizer, λ is a hyperparameter (weight or magnitude of $\mathcal{R}(\theta)$ to be applied) while computing the total loss. In equation 9, n is the batch size and α is a hyperparameter (noise scaling variant). ℓ is set as the symmetrized KL-divergence loss function for classification tasks such as our project's multi-task problem.

$$\ell(P, Q) = \mathcal{D}_{\mathrm{KL}}(P\|Q) + \mathcal{D}_{\mathrm{KL}}(Q\|P) \tag{10}$$

SMART measures local Lipschitz continuity of the prediction functions under symmetrized KL-divergence loss. We add a small random perturbation as noise to the input samples during fine-tuning without significantly compromising the prediction outputs. By minimizing the objective from equation 8, the fine-tuning process motivates the prediction functions to be smooth within the proximity of all samples. This smoothness-encouraging characteristic is especially helpful when fine-tuning data is not abundant, which is common in real world scenarios.

The perturbation is applied on the BERT embedding output before it is fed to the transformer. We compute the SMART loss via symmetric KL-divergence between the perturbed prediction and unperturbed prediction, which is then scaled with the hyperparameter λ and added to the unperturbed loss, which we optimize for with backpropagation. We referenced an implementation of SMART from a public repository (Schneider (2022)), before heavily modifying it to fit to our needs.

5 Experiments

5.1 Data

We used the datasets below to train and evaluate our model performance (See appendix A, dataset sizes).

- (**Sentiment Analysis**) SST and CFIMDB datasets. We combined the SST and CFIMDB training datasets and re-scaled the CFIMDB scores from 0 - 1 to 0 - 4 to match the range of the SST scores. (`data/ids-sentiment-combined-train.csv`)
- (**Paraphrase Detection**) Quora dataset: training (`data/quora-train.csv`)
- (**Semantic Textual Similarity**) SemEval STS benchmark dataset: training (`data/sts-train.csv`)

5.2 Evaluation Method

We used mean accuracy for SST and PARA, and Pearson correlation for STS. Accuracy for SST and PARA was computed as the count of correctly predicted outputs divided by evaluation sample count. The Pearson correlation coefficient for STS was computed following equation 11 as shown in appendix B.

We evaluated our model loss after every epoch during training using the SST, Quora, and STS dev sets. We re-scaled the STS Pearson correlation to 0 - 1 and then averaged the SST/PARA accuracies and Pearson correlation to determine if our model parameters should be updated.

5.3 Experimental Details

Unless otherwise specified, all tests were executed for 30,000 steps using annealed sampling (AS) as the task scheduling policy, with 100 epochs and 300 steps per epoch, a batch size of 16, a learning rate of `1e-5`, dropout probability `0.3`, and AdamW optimizer with weight decay of `0.0` . The tests in each table in our results section were profiled on the same GPU for consistency in timing, which would otherwise vary depending on the GPU used.

For some experiments we applied a sparse evaluation scheme where we evaluated predictions against the dataset every $n > 1$ epochs. Using a number larger than 1 for n slightly reduced the best scores but also reduced the time required to complete the fine-tuning job. This was particularly useful when debugging and for determining optimal hyperparameter values within a single extension in less time.

5.4 Results

5.4.1 Baseline Models

As baselines, we developed four models (Appendix C, table 5): three task specific fine-tuned models and a baseline multi-task model. All models were trained for 30 epochs, with a batch size of 16 and a learning rate of `1e-5`. The task specific models were trained on the entire task dataset for each epoch and for the multi-task baseline model we randomly selected a task to train on for each epoch. Since we select a task randomly each epoch, and our seed is fixed, we trained on the SST training set 8 times, the STS training set 9 times, and the Quora training set 13 times. These baselines will be used to evaluate the effectiveness of our improvements.

5.4.2 Task Scheduling

We extended our baselines by implementing improved task scheduling methods. Appendix D, table 6 summarizes our findings when training over 100 epochs (300 batches/epoch), with a batch size of 16 and a learning rate of 1e-5.

We found that due to dataset size imbalances (Appendix A), annealed sampling (AS) significantly outperformed other task scheduling policies while also taking significantly less time to train. With AS we trained over each dataset much more evenly, looping over the SST dataset $\approx$ 5x, the Quora dataset $\approx$ 1.5x, and the STS dataset $\approx$ 7x, compared to random or round-robin scheduling ($\approx$ 15x, $\approx$ 0.5x, $\approx$ 26x). We noticed that although the distribution of batches across tasks was much better it could be further improved by treating 0.8 in equation 1 as a **new hyperparameter**. We propose that others in the future tune this parameter per their projects' dataset sizes.

5.4.3 LoRAs

Model Configuration	Trainable Parameters	SST Accuracy	Paraphrase Accuracy	STS Pearson Correlation	Average Accuracy	Training Time Without Eval
FM (Baseline)	109,487,623	0.519	0.892	0.873	0.782	0:47:20
LLL + LoRA r=1, α=1	149,767	0.454	0.828	0.833	0.733	0:41:30
LLL + LoRA r=4, α=1	599,068	0.508	0.835	0.850	0.756	0:42:59
LLL + LoRA r=8, α=1	1,198,136	0.496	0.851	0.850	0.757	0:45:05
LLL + LoRA r=16, α=1	2,396,272	0.496	0.855	0.849	0.759	0:45:50
LLL + LoRA r=1, α=8	149,767	0.481	0.842	0.842	0.748	0:42:56
LLL + LoRA r=4, α=8	**599,068**	**0.510**	**0.867**	**0.870**	**0.771**	**0:44:33**
LLL + LoRA r=8, α=8	1,198,136	0.501	0.871	0.843	0.765	0:45:13
LLL + LoRA r=16, α=8	2,396,272	0.502	0.870	0.850	0.766	0:44:46
LLL + LoRA r=16, α=16	2,396,272	0.468	0.845	0.853	0.747	0:45:38
FM + LoRA r=4, α=8	32,150,812	0.529	0.875	0.859	0.778	0:54:47

Table 1: Dev set results for different LoRA configurations. LLL is last-linear-layer, FM is full-model

Table 1 summarizes the results for testing the performance of integrating LoRA layers into the model compared to our baselines. "LLL" denotes testing in the last-linear-layer fine-tuning mode where nearly all model parameters are frozen except for the LoRA parameters. "FM" refers to full-model tuning mode where only the self-attention and feed-forward layers are frozen, and the rest of the parameters are updatable.

Overall, the performance with LoRA layers slightly trails the baseline, particularly in the PARA task. While the performance is modestly lower, the training time is **reduced by up to 14%**. LLL+LoRA models utilize up to **731 times fewer** trainable parameters than the baseline model, significantly reducing memory usage during training and model size on disk. For the LLL+LoRA models, the final model sizes ranged from 419-445 MB depending on the rank (lower ranks yield smaller models), compared to the full-model baseline output of 1.22 GB - **a reduction of up to 2.91 times**. Generally, as the rank increases, both training time and performance improve, though the model with rank=4 and alpha=8 outperformed models with higher ranks. A downside to the LoRA models is that evaluation time increased, by up to about 23%, likely due to the additional computations required by the LoRA layers during the forward pass.

5.4.4 PALs & Parallel Adaption Layers

Table 2 summarizes the results of testing our parallel adaption layer configurations. Similar to Stickland and Murray (2019), PALs with shared V^E and V^D across layers performed optimally in terms of average accuracy. A 5% increase in total parameters from our baseline multi-task model resulted in a **3% increase in SST performance** compared to our baselines (Appendix C). However, the overall average accuracy improvement with shared PALs was minimal due to a drop in PARA accuracy and STS correlation which we discuss further in the analysis section. We also tested applying PALs to SST only as opposed to all tasks, but it performed less optimally than using PALs with shared V^E and V^D for all tasks.

Interestingly, our alternative architecture (AA) **outperformed the original** when NO shared layers were used but performed worse when ANY shared layers were used. While this is notable, any conclusive claims on our AA performance would require further investigation.

We attempted a mixed adaption approach as suggested in the future work section of Stickland and Murray (2019) where we used PALs with shared V^E and V^D in parallel for the first six BERT layers and low-rank adaption layers in parallel for the final six BERT layers. We found that this mixed adaption approach did not perform as well as either using low-rank layers or PALs layers alone.

We experimented with adding sharing attention (SA) along with shared V^E and V^D with PALs. We expected that shared attention would perform worse since sharing attention across layers would likely result in confusion and interference between tasks and hence in the resultant output and it indeed did perform worse.

Model Configuration	Trainable Parameters	SST Accuracy	Paraphrase Accuracy	STS Pearson Correlation	Average Accuracy	Training and Eval Time
Low-rank (AA)	115048471	0.524	0.888	0.870	0.782	2:30:41
Low-rank	115048471	0.522	0.887	0.865	0.781	2:29:37
PALs (AA)	125319559	0.522	0.888	0.875	0.783	2:52:08
PALs	125319559	0.514	0.888	0.868	0.779	2:52:21
PALs Shared $V^{E/D}$ (AA)	114947131	0.532	0.888	0.866	0.784	2:52:32
PALs Shared $V^{E/D}$	**114947131**	**0.540**	**0.887**	**0.867**	**0.787**	**2:58:22**
Mixed Adaption (AA)	124748995	0.511	0.888	0.870	0.778	2:45:12
Mixed Adaption	124748995	0.518	0.884	0.875	0.780	2:50:02
PALs Shared $V_{E/D}$ (AA + SA)	110806951	0.506	0.889	0.876	0.778	2:42:58
PALs Shared $V_{E/D}$ (SA)	110806951	0.522	0.881	0.876	0.780	2:42:48
PALs Shared $V_{E/D}$ (SST only)	111307459	0.529	0.887	0.869	0.784	2:30:34

Table 2: Dev set results for parallel adaption layer experiments with annealed sampling. AA indicates our alternative architecture, SA indicates shared attention

5.4.5 SMART

Table 3 summarizes the testing of our partial SMART implementation in comparison with our baselines. We tested by varying the two SMART hyperparameters, α and λ. We set $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and $\alpha \in \{10^{-5}, 10^{-2}\}$.

α dictates the noise variation i.e. the magnitude of perturbation we apply to the embeddings which should be small enough to not significantly affect the prediction outputs. λ dictates the weight to apply to the SMART loss $\mathcal{R}(\theta)$ before being added to the target task loss. Increasing λ results in a larger total loss ($\mathcal{F}(\theta)$ from Equation 8), which may affect backpropagation, although we find that larger total loss does not necessarily translate to lower scores.

Due to doubling the computations (unperturbed and perturbed) of embeddings, predictions, and losses, SMART regularization takes longer and uses much more memory than our baselines. In order to reduce training and evaluation time, we evaluated the results every four epochs instead of one. This slightly reduces the best score during the test due to smoothing, but allows us to more quickly compare results across tests on different hyperparameters.

Variance in SMART hyperparameters resulted only in slight variance of average accuracy (roughly 0.780 ± 0.002). In contrary to the original paper, best performance in our implementation was with $\alpha = 10^{-2}$, $\lambda = 10^{-3}$, although the configuration scored only marginally higher.

Model Configuration	**SST Accuracy**	**Paraphrase Accuracy**	**STS Pearson Correlation**	**Average Accuracy**	**Training and Eval Time**
SMART+AS $\alpha = 10^{-5}$, $\lambda = 10^{-4}$	0.518	0.885	0.869	0.779	1:41:34
SMART+AS $\alpha = 10^{-5}$, $\lambda = 10^{-3}$	0.515	0.887	0.884	0.781	1:44:40
SMART+AS $\alpha = 10^{-5}$, $\lambda = 10^{-2}$	0.506	0.892	0.881	0.780	1:45:39
SMART+AS $\alpha = 10^{-2}$**,** $\lambda = 10^{-3}$	**0.519**	**0.892**	**0.877**	**0.783**	**1:41:02**
SMART+AS $\alpha = 10^{-2}$, $\lambda = 10^{-2}$	0.502	0.893	0.880	0.778	1:39:59
SMART+AS $\alpha = 10^{-2}$, $\lambda = 10^{-1}$	0.522	0.885	0.864	0.780	1:39:22

Table 3: Dev set results for SMART regularization and annealed sampling (evaluation per 4 epochs)

5.4.6 Combined Model

Finally, we tested combined configurations of adding annealed sampling, LoRAs, shared PALs, and SMART (Table 4). Compared to the baseline, the combined model with SMART and PALs showed better performance across all tasks. For the continuous STS task, predictions were more accurate, with fewer instances of being off by 3.0 or more. In the PARA and SST tasks, the model was more often correct, particularly in the challenging middle classification range of 2-4. The confusion matrices for SST and PARA, as well as the residuals for STS, are shown in Figure 2. Baseline performance can be found in Appendix G.

1. Configuration 1: PALs Shared + SMART (100 epochs, 1000 batch/epoch, batch size 16, λ 1e-5, α 1e-2)
2. Configuration 2: PALs Shared + SMART + LoRA (100 epochs, 1000 batch/epoch, batch size 16, λ 1e-5, α 1e-2, LoRA rank 4, LoRA α 8)
3. Configuration 3: PALs Shared + XL batch size (100 epochs, 1000 batch/epoch, batch size 64)

Model	**Dev Accuracy / Correlation**				**Test Accuracy / Correlation**				**Training and**
Configuration	**SST**	**PARA**	**STS**	**Avg**	**SST**	**PARA**	**STS**	**Avg**	**Eval Time**
Configuration 1	**0.530**	**0.900**	**0.847**	**0.789**	**0.524**	**0.900**	**0.879**	**0.788**	**7:02:04**
Configuration 2	0.508	0.878	0.864	0.772	0.517	0.880	0.860	0.776	8:21:00
Configuration 3	0.524	0.903	0.879	0.788	0.518	0.905	0.875	0.787	8:21:24

Table 4: Dev and test set results for our combined models

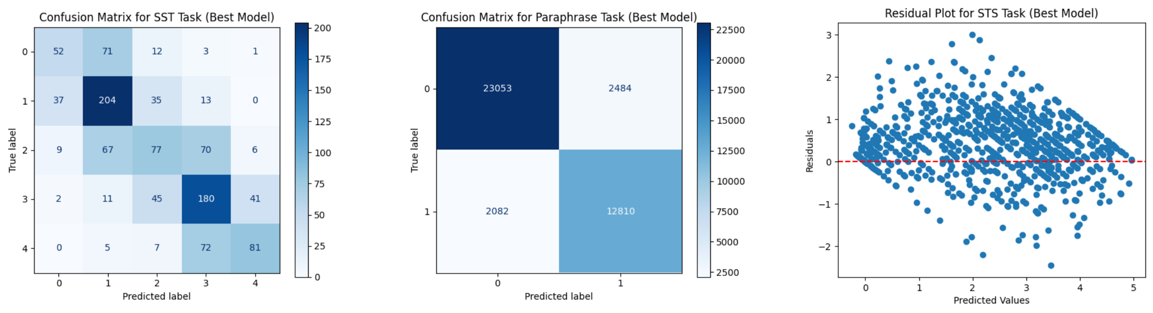

Figure 2: Comparison of confusion matrices for SST and PARA and residual plot for STS

6 Analysis

In some tests, training loss over time did not converge, particularly with large λ values in SMART regularization. When λ exceeded 10^{-2}, the loss became unstable rather than converging. This was likely due to the regularization adding more noise than the model could successfully adapt to during gradient updates. Additionally, a low LoRA rank appeared to prevent optimization of training loss beyond a certain point due to the significant reduction in updatable parameters, limiting model adaptation during fine-tuning. Interestingly, the final training loss over 100 epochs did not necessarily correlate with better results; for instance, LoRA with $r = 4$ outperformed $r = 16$ despite having higher training loss.

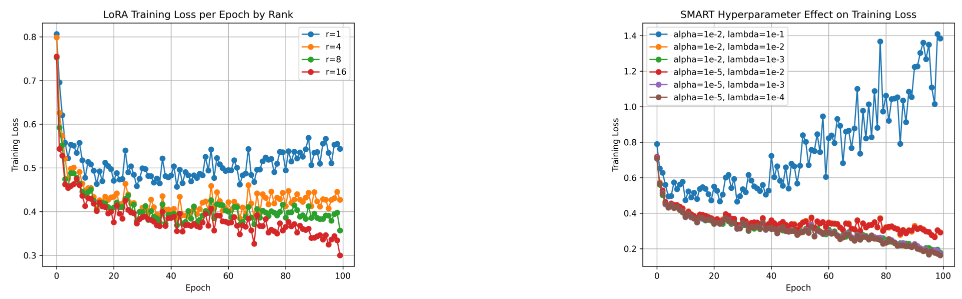

Figure 3: Hyperparameter effects on training loss

LoRA layers demonstrated significant value in reducing model size and memory requirements by nearly **3x** and accelerating training speed, with only a minor performance trade-off of around 2%. Performance was most notably affected in the PARA task, while the other tasks were less impacted. We don't fully understand the cause for this yet and it might be worth exploring ways to maintain PARA performance with LoRA in the future. We also observed that evaluation time per epoch increased when using LoRAs. Therefore, future users of LoRAs may want to consider running evaluations less frequently to maintain the time performance improvements over the baseline.

Tuning SMART regularization hyperparameters together affected overall accuracy, but did not always yield optimal results. This variability can be attributed to the need for varying optimal levels of random noise and weight of the SMART loss across datasets, tasks and layers. Additionally, an initial bug in the SMART implementation (which was fixed later), where a perturbation was incorrectly added to the unperturbed embeddings, produced surprisingly strong results. We were not able to explore this deeper, but it seems a promising avenue for potential future work.

When examining the results of our PALs tests, we saw that the average accuracy improvement with shared PALs was minimal due to a drop in PARA accuracy and STS correlation. This was contrary to our hypothesis that PARA accuracy would improve due to its similarity with the RTE task, which drove performance improvement in Stickland and Murray (2019). The significant SST improvement compared to the lack of improvement in PARA and STS likely results from PARA and STS already benefiting from transfer learning due to their similarity and hence not being able to leverage the additional task specific parameters as well.

This is supported by comparing our best model's SST confusion matrix in figure 2 to our baseline SST confusion matrix in Appendix G. Binary classification on PARA is already accurate in the baseline model whereas we see much more mis-classification of adjacent categories in SST. In our best model, which includes shared PALs with additional task specific parameters, the frequency of SST mis-classifications significantly decreases. Similarity, for STS our MSE metrics (Appendix H) are already good in the baseline but the final model has better predictive accuracy. Additionally, the R^2 score (Appendix H), which measures variance in the prediction, is higher for the final model, indicating that the model explains more of the variability in the data compared to the baseline.

7 Conclusion

Through model architecture enhancements, improved task sampling, and parameter optimization, we achieved notable improvements in multi-task minBERT fine-tuning compared to our baseline models. Our best model achieved an overall accuracy of **78.8%** on our test set (SST: **52.4%**, PARA: **90.0%**, and STS: **87.9%**). By integrating SMART regularization and PALs with shared parameters, alongside annealed sampling, we observed a **2.4%** accuracy improvement across all tasks, with a notable **5.6%** increase in the SST task as compared to our baseline multi-task model. We were also able to approach or exceed the baseline task-specific model performance across all tasks using our improved multi-task training approaches. LoRA layers also demonstrated their potential by reducing model size by nearly **3x** and accelerating training speed, with only a minor 2% performance trade-off. Future research could further explore more selective use of LoRA layers to approach baseline model performance while minimizing the total number of tunable parameters.

8 Ethics Statement

As we push the boundaries of natural language processing (NLP), it is critical to consider the potential consequences and secondary effects of our technology. Any model can have uses which the original developers did not intend and can be misused by malicious actors with harmful intent.

Extremely effective NLP systems can be misused by tyrannical governments or corporations with abundant resources to survey and manipulate individuals. For instance, a dictator with extensive resources could implement an active surveillance system using sentiment analysis on the internet, phone calls, or even in-person conversations (in conjunction with audio recognition technology) to counteract dissidents, either directly (e.g., wrongfully prosecuting individuals) or indirectly (e.g., swaying public opinion by rebutting opposing statements via automated bots). Corporations could use similar technologies to target users with intrusive ads via sentiment analysis, exploiting emotions to expose people to advertisements. For example, an advertiser might attempt to sell a gun or other potentially lethal weapon to a user predicted to be extremely unhappy, potentially nudging the individual toward violent action. While such actions may be legal, they raise significant ethical and philosophical questions about the manipulation of emotions for profit.

Some of our extensions, particularly SMART, required more computational power for training and evaluation. This highlights the potential for a resource race where entities with more resources (e.g., computational power, energy) dominate, potentially depleting these resources and exacerbating social inequalities. Increased resource consumption can lead to environmental damage due to the depletion of natural resources such as water, coal, and materials used in the production of computers and data centers. We are mindful of the environmental impact of our work and strive to optimize our methods to minimize resource usage.

We used publicly available datasets for training and evaluation purposes, ensuring that all data does not contain personally identifiable information. No sensitive or private data were collected or used in this research. We have documented our methodology, experiments, and findings thoroughly to promote transparency and enable reproducibility. The results and improvements achieved through our work are clearly reported, including any limitations and potential areas for further research.

To mitigate the risks associated with our research, we propose several strategies for future consumers of this work. First, we advocate for the incorporation of bias and personally identifiable information detection techniques throughout the development process for additional datasets used for fine-tuning. By continuously evaluating and addressing the inputs to our models, both during training and during evaluation, we can help ensure the resulting models behave equitably and avoid violating individuals' privacy. Second, we suggest the adoption of environmentally sustainable practices, such as optimizing computational resources and exploring energy-efficient algorithms, to minimize the environmental impact of our research. Additionally, we encourage open discussions and collaborations within the research community to develop guidelines and standards that promote the responsible use of NLP technologies.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805. https://dblp.org/rec/journals/corr/abs-1810-04805.bib.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/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*. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. https://arxiv.org/abs/1711.05101.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2018. Efficient parametrization of multi-domain deep neural networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8119–8127. https://api.semanticscholar.org/CorpusID:215822736.

Flavio Schneider. 2022. Pytorch – smart: Robust and efficient fine-tuning for pre-trained natural language models. https://github.com/archinetai/smart-pytorch.

Asa Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of the 36th International Conference on Machine Learning*. https://proceedings.mlr.press/v97/stickland19a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

A Dataset Sizes

Dataset Size			
Dataset Name	Train examples	Dev examples	Test examples
SST	8544	1101	2210
CFIMDB	1701	245	488
Quora	282841	40430	80789

B Pearson Correlation

The Pearson correlation coefficient between two samples x and y is given by:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}, \tag{11}$$

where n is sample size (dimensions), x_i (and y_i) are individual sample points (indexed by i), $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$, and $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ (the sample means).

C Baselines: Task specific fine-tuned models and multi-task baseline

Model Configuration	**SST Accuracy**	**Paraphrase Accuracy**	**STS Pearson Correlation**	**Average Accuracy**	**Training and Eval Time**
Baseline Finetune SST	**0.512**	0.592	0.-0.026	0.530	1:06:50
Baseline Finetune Para	0.268	**0.907**	0.689	0.673	11:42:24
Baseline Finetune STS	0.217	0.725	**0.873**	0.626	0:44:13
Baseline Finetune Multitask	0.496	0.886	0.851	**0.769**	5:59:27

Table 5: Dev set results for baseline finetuned models and baseline multi-task minBERT

D Results: Task Scheduling Methods

Model Configuration	**SST Accuracy**	**Paraphrase Accuracy**	**STS Pearson Correlation**	**Average Accuracy**	**Training and Eval Time**
Multitask Random Batch	0.506	0.867	0.877	0.771	2:23:45
Multitask Round-Robin Batch	0.520	0.855	0.864	0.769	2:20:55
Multitask AS Batch	**0.527**	**0.893**	**0.876**	**0.786**	**2:25:57**
Multitask AS Batch (Eval Epochs = 5)	0.520	0.893	0.870	0.783	1:09:21

Table 6: Dev set results for multi-task minBERT with different task scheduling methods

E Low-Rank Adaption Layers

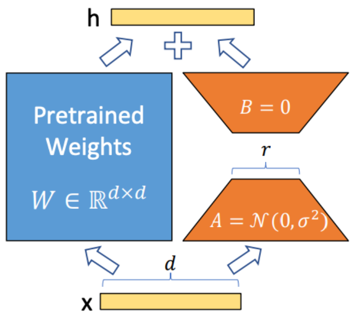
h
Pretrained Weights
$W \in \mathbb{R}^{d\times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Figure 4: LoRA Hu et al. (2021) Architecture

F Projected Attention Layers and Parallel Adaption Layers

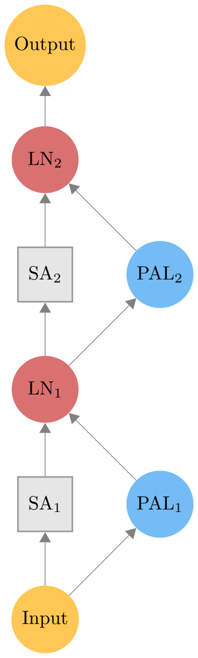

Figure 5: BERT with PALs in parallel. From Stickland and Murray (2019).

G Confusion Matrices (SST and PARA) and Residual Plots (STS) for Best and Baseline Model

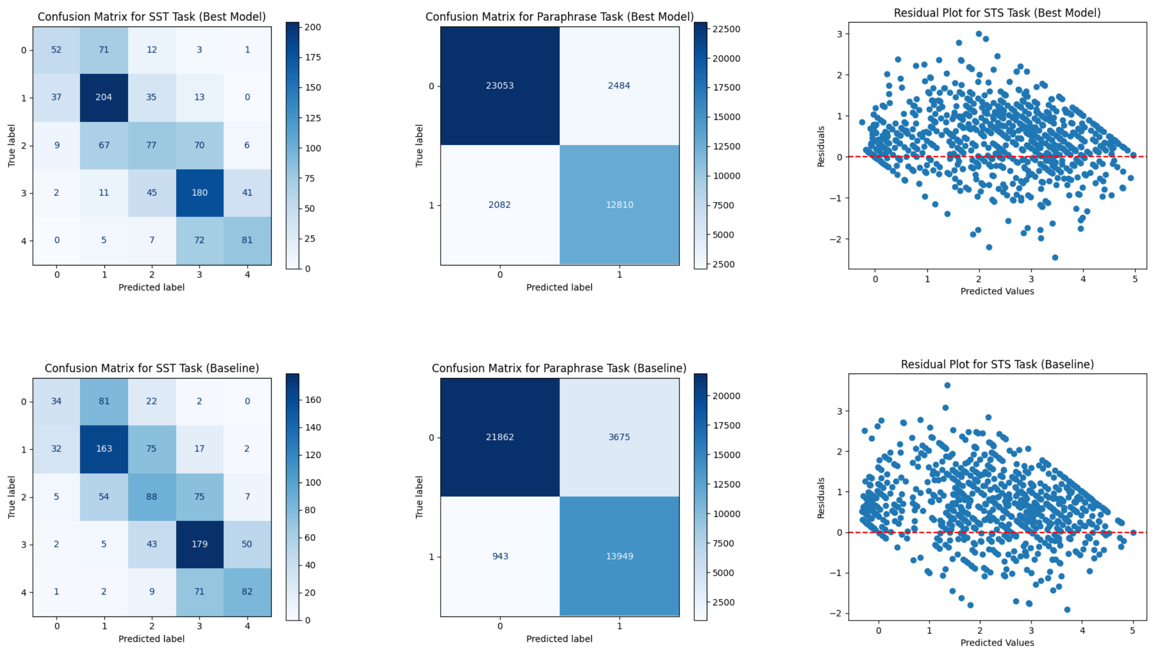

Figure 6: Comparison of confusion matrices for SST and PARA and residual plot for STS

H STS MSE: Baseline compared to Final Combined Model

	Baseline		Final	
	Mean	St. Dev.	Mean	St. Dev.
MSE	1.023	0.012	0.718	0.007
RMSE	1.011	0.008	0.847	0.005
MAE	0.808	0.006	0.660	0.004
R² Score	0.530	0.003	0.670	0.002

Table 7: Summary of Evaluation Metrics for Baseline and Final Models