# minBERT Multi-Extended: Fine Tuning minBERT for Downstream Tasks

CS224N Default Project

**Jayna Huang**
Department of Symbolic Systems
Stanford University
`jhuang23@stanford.edu`

**Isabella Lee**
Department of Computer Science
Stanford University
`leeij@stanford.edu`

**Sophie Zhang**
Department of Computer Science
Stanford University
`spzhang@stanford.edu`

## Abstract

In this paper, we explore multitask fine-tuning strategies to further enhance the capabilities of BERT across three distinct downstream tasks: sentiment analysis, binary paraphrase detection, and textual similarity scoring. Our approach leverages several key techniques: cross-encoding of sentence pairs, additional pretraining on domain-specific data, LLM-aided data augmentation, and SMART loss (Jiang et al. (2019)). Additionally, we explore the tradeoffs between training time, number of parameters updated, and performance through a deep-dive of Low-Rank Adapters as proposed by Hu et al. (2021). Through this comprehensive multi-task learning framework, we obtain a robust model that integrates knowledge from all three tasks, achieving an overall score of 79.1% on the dev set and 78.3% on the withheld test set for the three downstream tasks.

## 1 Key Information

Mentor: Arvind Mahankali | External collaborators: No | External mentor: No | Sharing project: No

Contributions: Jayna implemented default minBERT, as well as the Multi-task fine tuning, cross-encoding, additional pretraining, and data augmentation extensions. Sophie implemented SMART regularization and conducted its hyperparameter grid search. Isabella implemented the low-rank adaptation extension and conducted the deep-dive into its effectiveness. Isabella and Sophie worked on the milestone paper. All team members contributed to the writing of this report and the final poster.

## 2 Introduction

The rapid advancement in natural language processing (NLP) has been significantly propelled by the development of models such as BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al. (2018)). BERT's bidirectional training approach enables it to understand the context of words by considering both preceding and following words. Fine-tuning the model's robust pretrained embeddings by leveraging its powerful transformer architecture has led to substantial improvements across multiple performance benchmarks. In this paper, we investigate the performance of a minimally-implemented BERT model on three tasks: sentiment analysis, binary paraphrase detection, and semantic textual similarity.

Traditional fine-tuning is often done for a single task at a time and requires updating a large number of parameters, which can be computationally expensive and resource-intensive. To address these limitations, we explore multitask learning and methods of adapting, which aim to fine-tune the model simultaneously on multiple tasks while requiring a fewer number of parameters to be updated. This approach not only reduces the computational overhead but also enhances the model's ability to generalize across different tasks, leading to improved performance and more efficient resource utilization. Additionally, we explore additional approaches for improving fine-tuning including cross-encoding, additional pre-training, data augmentation, and SMART regularization.

## 3 Related Work

With the widespread adoption of pre-trained language models like BERT (Devlin et al. (2018)), there has been a spark in research on BERT's effectiveness as a multi-task learning model. In Bi et al. (2022), researchers explore the effectiveness of a multi-task learning framework for BERT for news recommendation tasks and demonstrate performance gains.

One limitation of BERT is that it is pretrained in a general domain that has a different distribution from the target domain for text classification tasks. In (Sun et al. (2020)), researchers perform further pre-training through within-task pre-training and in-domain pre-training and achieve improved performance on all datasets.

An additional challenge in large-scale pretraining and adaptation to downstream tasks is operational inefficiency during full fine-tuning. In Hu et al. (2021), researchers develop LoRA to freeze the pre-trained model weights and inject trainable rank decomposition matrices into each layer of the Transformer architecture. They found that in comparison to GPT-3, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times.

Another area of research addresses overfitting in the fine-tuning phase. Researchers have explored text data augmentation techniques in the context of adversarial attack and adversarial training. Wei and Zou (2019) propose EDA (easy data augmentation) to boost performance on text classification tasks. Given a sentence in the training set, they randomly choose and perform one of the following operations: synonym replacement, random insertion, random swap, and random deletion.

To reduce overfitting and aggressive updating issues, researchers also propose regularization techniques like SMART (Jiang et al. (2019)) which combines smoothness-inducing adversarial regularization with Bregman proximal point optimization to improve model generalization in multitask domain. They find that SMART consistently outperforms BERT on all GLUE tasks and improves the generalization of multi-task learning.

## 4 Approach

We first complete a minimally-implemented BERT (Devlin et al. (2018)) according to the default project handout. The BERT architecture consists of embedding layers and 12 encoder transformer layers; we implement the multi-head self-attention, additive and normalization layer, feed forward layer, and the Adam Optimizer step function. We utilize the [CLS] token embedding for our downstream tasks.

**Baseline.** To establish our first baseline, we fine-tune only the last linear layers. The SST task uses a dropout and linear layer with cross-entropy loss. The paraphrase task stacks embeddings vertically, before passing them through a dropout and linear layer; binary cross-entropy loss is used. The STS task passes embeddings through a linear layer and then uses cosine similarity, and mean squared error loss (Figure 1). Since only the gradients of these final layers are updating, this provides an idea of the performance capabilities of a minimally-trained BERT on the downstream tasks.

Our second baseline is three models individually fine-tuned to only a specific dataset according to the architecture described above. Because the model must only focus on one task instead of three, these are strong performance scores that we aim to achieve or surpass in just one model by implementing extensions.

**Multi-task Fine Tuning (MFT).** For full-model multitask learning, we employ the same general strategy as Bi et al. (2022): we sample a batch of data from each of the three tasks, calculate and sum their losses, then update the overall loss at once. Naturally, the uneven sizing of the datasets poses a problem, as the size of the QQP training dataset is more than thirty times the size of the training sets of each of the other two. We elect to undersample large datasets rather than oversample small ones in order to reduce the likelihood of overfitting on smaller datasets. We conduct a non-exhaustive grid search of different batch sizes (scaling losses accordingly), aiming to utilize as much of the given data as possible while maintaining accuracy. Later, because overfitting is observed for both the SST and STS tasks, we taper their contributions to the overall loss to decrease with each epoch ($\mathcal{E}$). The final weight updates are:

$$\mathcal{L} = \mathcal{L}_{sst} \cdot \frac{0.5}{\mathcal{E}+1} + \mathcal{L}_{para} + \mathcal{L}_{sts} \cdot \frac{1}{\log(\mathcal{E}+1)+1} \tag{1}$$

**Cross-Encoding.** Both QQP and STS are sentence-pair tasks, where one is asked to discern the nature of the relationship between two sentences. In the default implementation, each of these sentences are individually fed through BERT and then combined or classified in the last layer. With cross-encoding, as is done in Devlin et al. (2018), the two sentences are concatenated and the entire new vector is fed through BERT and classified at the end. This allows for attention to be performed on the entire sentence pair, instead of just the sentences individually. Because just one [CLS] token is produced, the head architecture for both QQP and STS is one dropout layer followed by a linear layer.

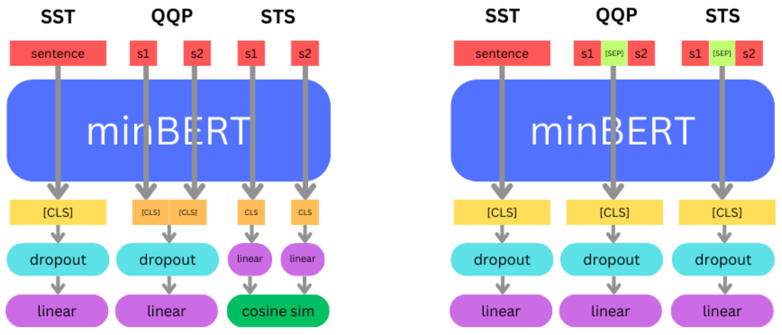

Figure 1: Architectures for baselines (left) and cross-encoding (right).

**Pretraining.** We implement additional pre-training with the Multi-Genre Natural Language Inference (MNLI) dataset by Williams et al. (2017), a collection of 433k sentence pairs from a range of spoken and written text and annotated with textual entailment information. Some examples include:

> You have access to the facts. The facts are accessible to you. *entailment*
>
> The economy could be still better. The economy has never been better. *contradiction*

By pretraining on a large annotated corpus of a related task, the BERT embeddings become more robust: the model becomes not only better at understanding language, but especially better at understanding relationships between sentence pairs. This is implemented using the same cross-encoding techniques as above, with lr=5e-6, dropout=0.1, for 5 epochs.

**Data Augmentation.** Overfitting—as shown by decreasing training loss, increasing train accuracy, but decreasing dev score—is noticed in the early epochs when evaluating SST. To enhance the generalizability of the model and mitigate overfitting, we implement data augmentation. To do so, we iterate through each of the 8,544 SST train examples. In batch sizes of 100, we choose our data augmentation technique (inspired by Wei and Zou (2019)) with probability $p$: replacing a random noun or verb with its synonym ($p = 0.3$), inserting a random word ($p = 0.2$), removing a random word ($p = 0.2$), choosing two words at random and swapping their positions ($p = 0.2$), and translating the sentence into Spanish and back again to English ($p = 0.1$) . We pass these through a publicly accessible large language model: Meta's Llama 3 8b [1]. Temperature is randomly sampled from the range of (0.55, 0.85). Sentiment remains the same and unique IDs are generated. We remove duplicates both within the augmented dataset and between the existing train dataset. This gives us 8,430 new examples, for a total of 16,974 total SST train examples.

**SMART Regularization.** To further address the challenge of overfitting in the SST dataset, we implement SMART as proposed by Jiang et al. (2019). We specifically add Smoothness-inducing

[1] `https://llama.meta.com/llama3/`

Adversarial Regularization to the model which effectively manages the complexity of the model through word embeddings with small amounts of noise. The regularization motivates the output of the model not to change much, enforcing the smoothness and controlling its capacity. It solves the following optimization problem for fine tuning as defined as:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta) \tag{2}$$

where $\mathcal{L}(\theta)$ is the loss function for our downstream tasks, $\lambda_s > 0$ is a tuning parameter, and $\mathcal{R}_s$ is defined as:

$$\mathcal{R}_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{||\tilde{x_i} - x_i||_{p \leq \epsilon}} l_s(f(\tilde{x_i}; \theta), f(x_i; \theta)). \tag{3}$$

Following Jiang et al. (2020), $l_s$ is chosen as the symmetric KL-divergence loss for classification tasks and mean-squared loss for regression tasks. SMART also incorporates Bregman Proximal Point Optimization, but we use the AdamW optimizer to solve the minimization instead. In our model, we only apply SMART to the SST classification tasks as that dataset had especially pronounced overfitting compared to the QQP or STS tasks.

**Low Rank Adaptation (LoRA).** We implement LoRA to address operational inefficiency during full fine-tuning as proposed by Hu et al. (2021), as it becomes less feasible to fully fine-tune models as they increase in size. Specifically, we freeze the pre-trained model weights and instead replace linear layers with trainable rank decomposition matrices with rank $r$ with inspiration from the implementation by Riggio (2023), decreasing the number of trainable parameters during fine-tuning. We decompose the linear layer by freezing the weight matrix $W$ in $\mathbb{R}^{d \times k}$ and representing the original update function $W + \Delta W$ as $W_f + BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$. We note that in order to effectively decrease the number of trainable parameters, $r < \min(d, k)$. In this implementation, $W_f$ is the frozen weight matrix and does not receive gradient updates while training. Instead, $B$ and $A$ contain trainable parameters. $B$ and $A$ are initialized with zeros and a random Gaussian distribution respectively, yielding $\Delta W = BA$ as 0 at the onset of training. $BA$ is also scaled by $\frac{\alpha}{r}$, where $\alpha$ is a scaling constant equal to 16 or the first $r$ tested. By setting $\alpha$ to a predetermined constant value, we reduce the need to return the model's hyperparameters as we change $r$. As a result, given the initial forward pass $h = Wx$, the modified forward pass is described as:

$$h = Wx = W_f x + \Delta W x = W_f x + \frac{\alpha}{r} BAx \tag{4}$$

We adapt attention weights and the last linear layers with the decomposed matrix, leaving the LayerNorm layers and embedding layers untouched similar to the method proposed by Hu et al. (2021).

## 5 Experiments

### 5.1 Data

We use three datasets for testing, each with an approximate breakdown of 70% train, 10% dev, and 20% test.

- Stanford Sentiment Treebank [2] (SST) dataset – 11,855 sentences, the output for this task will be one of five sentiment classification labels ranging from negative to positive.
- Quora question pairs [3] (QQP) dataset – 404,298 question pairs, the output for this task is a binary decision of whether or not question pairs are paraphrases of each other.
- SemEval STS Benchmark [4] (STS) dataset – 8,628 question pairs, the output for this task will be a number on a scale from 0 to 5 describing how related the meanings of two sentences are to each other.

[2]Socher et al. (2013)

[3]`https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs`

[4]Agirre et al. (2013)

## 5.2 Evaluation method

For the SST and QQP tasks, the model will be assessed on accuracy. For the STS test, its performance will be calculated via the Pearson correlation coefficient, a measure of linear correlation between two sets of data. We aim to produce the model with the best overall score, calculated by the following formula:

$$overall\_score = \frac{sst\_acc + \frac{1+sts\_corr}{2} + para\_acc}{3}$$

In addition to accuracy, we also make qualitative judgments on training time and number of parameters updated in order to explore the effects of tradeoffs between computational efficiency and performance. In particular, for evaluating LoRA performance, we track training time per epoch, GPU memory usage, and number of trainable parameters in addition to the individual task scores and Pearson correlation coefficient.

## 5.3 Experimental details

All experiments except for LoRA-related trials are conducted on a NVIDIA T-4 GPU. Unless otherwise noted, experiments are conducted with dropout probabilities of 0.3, a learning rate of $1e-5$, and run for 10 epochs. The optimizer is AdamW, with learning rate $1e-3$ and $\beta_1 = 0.9, \beta_2 = 0.999$.

**SMART Regularization.** In batch sizes of (SST=3, QQP=32, STS=2), we utilize the `smart_pytorch` module to incorporate SMART regularization into our model. We use the following parameter choices as recommended by Jiang et al. (2019): 1 sampling step, $\epsilon = 1e-6, \sigma = 1e-5$ and $\eta = 1e-3$. For the regularization weight $\lambda_s$, we experiment with different choices of $\lambda_s$ to find the optimal weight that gives us the best results.

**LoRA.** LoRA experiments are conducted on a NVIDIA L4 GPU. We conduct LoRA experiments with a dropout probability of $0.1$ in LoRA layers due to the low number of trainable parameters in the decomposed matrices. Experiments are conducted with learning rates of both $1e-5$ and $1e-3$ as described in the results below. The $\alpha$ is set to a constant value of 16, and the value of rank $r$ varies across tests.

## 5.4 Results

In Multi-task Fine Tuning (MFT), we vary the batch size for each task and observe its effect on overall performance A. We find that a batch size combination of SST=3, QQP=32, STS=2 both utilizes the most of the given data and is most performant (we could not scale up the batch sizing of QQP due to memory issues). Smaller batch sizes lead to noisier gradient updates which can increase generalizability: since SST and STS are prone to overfitting, it seems that smaller batch sizes perform better.

In testing SMART regularization, we conduct a hyperparameter grid search as seen in Table 2 to determine the optimal regularization weight.

| SMART weight | SST Dev | SST Train | QQP Dev | STS Dev | Overall Dev |
|---|---|---|---|---|---|
| Baseline: 0.0 | 0.501 | 0.949 | 0.752 | 0.555 | 0.677 |
| 0.5 | 0.509 | 0.957 | **0.770** | 0.547 | 0.684 |
| 1.0 | 0.510 | 0.960 | 0.766 | **0.567** | **0.687** |
| 3.0 | **0.514** | 0.929 | 0.7623 | 0.557 | 0.685 |

Table 1: Performance comparison of different SMART regularization weights

Without SMART regularization (weight = 0.0), the model achieves an overall dev score of 0.677 with the SST task showing signs of overfitting. We find that a weight of $\lambda_s = 1$ leads to the highest overall dev accuracy of 0.687, with notable improvements in all three tasks. For future experiments using SMART, we then adopt $\lambda_s = 1$.

For Low-Rank Adaptation, we vary the rank $r$ and note the percentage of original number of trainable parameters (% TP), the training time per epoch, and the dev score for each run. As shown in Table 2 and 8, we find the best learning rate to be 1e-3. As such, we choose a learning rate of 1e-3 and a LoRA dropout rate of $0.1$ for future experiments. Please see the Appendix for more detail.

| Hyperparameters | | Space and Time | | Dev Score | | | |
|---|---|---|---|---|---|---|---|
| **Rank** | **LR** | **% TP** | **Train Time** | **SST dev** | **QQP dev** | **STS dev** | **Overall** |
| 1 | 1e-3 | 21.73 | 2:00 | **0.350** | 0.731 | 0.307 | 0.578 |
| 32 | 1e-3 | 26.61 | 2:02 | 0.344 | 0.739 | **0.347** | **0.586** |
| 128 | 1e-3 | 41.74 | 2:24 | 0.342 | **0.745** | 0.307 | 0.580 |
| 128 | 1e-5 | 41.74 | 2:03 | 0.150 | 0.632 | 0.073 | 0.440 |
| 256 | 1e-3 | 61.90 | 3:08 | 0.323 | 0.740 | 0.308 | 0.572 |

Table 2: Effect of selected ranks and learning rates on Low-Rank Adaptation performance

The best performing LoRA trial results from setting $r = 32$ and the learning rate equal to 1e-3, despite this model containing only 26.61% of the original number of trainable parameters contained in minBERT. We do not see a decline in SST task performance as rank decreases, with the best performing rank being the lowest rank (1). Additionally, the performance across QQP and STS tasks does not demonstrate a significant decline as rank decreases, with the best performing trials being $r = 128$ and $r = 32$ respectively. Notably, despite LoRA with $r = 256$ containing the most number of trainable parameters, it performs worse than all other LoRA trials with the same learning rate and requires the most training time per epoch. Additionally, the best performing LoRA test has a training time of 2 minutes and 2 seconds per epoch, comparable to the LoRA test with the smallest possible rank. This is as expected. As found in Hu et al. (2021), LoRA with lower ranks still demonstrates robust performance. This indicates that while implementing LoRA, it is more effective to use lower rank matrices to save on training time, space utilized by the trainable parameters, and GPU utilization (Figure 3 and Figure 5) while not compromising on performance relative to higher ranks.

In Table 3, we also compare the results from the LoRA trial with the best results against two additional trials: utilizing LoRA with three models individually fine-tuned to specific datasets (similar to Baseline 2) and from testing minBERT with no extensions. For the individually-tuned trial, we set $r = 32$ and the learning rate equal to $1e - 3$ to match the best performing LoRA trial.

| Model Description | SST Dev | QQP Dev | STS Dev | Overall |
|---|---|---|---|---|
| **LoRA with $r = 32$, LR = $1e - 3$ (from above)** | 0.344 | 0.739 | 0.347 | 0.586 |
| **LoRA (Individually Tuned)** | 0.381 | 0.751 | 0.316 | 0.597 |
| **minBERT** | 0.262 | 0.632 | 0.044 | 0.472 |

Table 3: Dev score results from selected LoRA trials.

Next, we record the results from the baselines and other extensions as described above.

| Model Description | SST Dev | QQP Dev | STS Dev | Overall |
|---|---|---|---|---|
| **Baseline 1: last layers minBERT** | 0.364 | 0.676 | 0.270 | 0.559 |
| **Baseline 2: full model minBERT** | 0.486 | 0.808 | 0.394 | 0.664 |
| **Best LoRA (Baseline 2 Method)** | 0.381 | 0.751 | 0.316 | 0.597 |
| **Best MFT (from above)** | 0.503 | 0.764 | 0.520 | 0.676 |
| **Cross-Encoded MFT** | 0.499 | 0.878 | 0.872 | 0.771 |
| **Pretraining + Cross-Encoded MFT** | 0.512 | 0.875 | 0.887 | 0.777 |
| **Data Aug + Pretraining + Cross-Encoded MFT** | 0.520 | 0.869 | 0.885 | 0.777 |
| **SMART + Data Aug + Pretr + Cross-Enc MFT** | 0.502 | 0.882 | 0.886 | 0.776 |

Table 4: Dev score results from implementing extensions.

We observe that MFT yields significant gains over Baseline 1 because all BERT parameters are being updated instead of just the last linear layers. MFT also yields slight gains over Baseline 2, demonstrating that knowledge transfer between tasks through shared gradient updates can lead to more robust embeddings than models trained for just one task. As expected, cross-encoding also yields significant performance gains for both sentence-pair tasks, as attention is being performed on the pair of sentences as a whole instead of each individually.

We observe that LoRA (Baseline 2 Method) yields stronger performance than Baseline 1 across all tasks but weaker performance than full model minBERT. Moreover, LoRA trials described in

Table 2 demonstrate worse overall performance than LoRA (Baseline 2 Method). While we are not surprised to find that LoRA's performance is worse relative to Baseline 2 as the number of trainable parameters is significantly lower, we are surprised to find that the performance is not as comparable as we initially believed based on the findings of Hu et al. (2021). As a result, we separate LoRA trials from our best performing models with other extensions when testing only for performance.

Interestingly, using cross-encoded pretraining on MNLI does not improve performance on the QQP task, despite it being a sentence-pair task, but does so for SST and STS. The improvements for SST and STS are likely due to a more robust understanding of language; for the very slight decrease in performance on the QQP task, an examination of the graph 6 indicates that the initial boost performance boost from pretraining wanes after a number of epochs.

Further, we observe that data augmentation does increase STS dev accuracy as expected, though at the expense of QQP and STS dev accuracy. We also note that SMART loss does not seem to improve SST accuracy: this is confirmed by the dev accuracy graph, which oscillates.7 Insight from Mgbahurike et al. (2024) suggests that noise from SMART loss in addition to the dropout layers may create too much regularization to the point where the model does not fully learn necessary features.

To boost scores, we optimize by tapering the individual task contributions to the loss and increasing the number of epochs as follows. The best model uses SMART loss, tapered loss contribution, data augmentation, pretraining, cross-encoded MFT, and is trained for 25 epochs.

| Model Description | SST Dev | QQP Dev | STS Dev | Overall |
|---|---|---|---|---|
| **Tapered weight contribution, 20 epochs** | 0.514 | 0.891 | **0.893** | 0.783 |
| **Best Model, 25 epochs** | **0.529** | **0.902** | 0.887 | **0.791** |

Table 5: Dev score results from optimizing for overall score.

We then use our best model for the test set, and notice a slight decrease in performance, especially for the SST task.

| Model Description | SST Test | QQP Test | STS Test | Overall |
|---|---|---|---|---|
| **Best Model, 20 epochs** | 0.503 | 0.901 | 0.888 | 0.783 |

Table 6: Test results for our top-performing model on the dev set.

The strong scores on both the dev and test set well exceed both Baseline 1, where only the last layer is fine-tuned, and Baseline 2, where the model is individually-trained to focus on one specific task. This confirms the effectiveness of our extensions in fine-tuning minBERT on the three downstream tasks.

## 6 Analysis

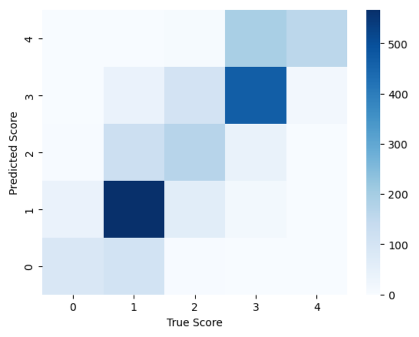

Figure 2: Confusion matrix displaying SST task performance for our best model

**Sentiment Classification.** We utilize a confusion matrix in Figure 2 to visualize our final model's performance on the SST-5 dataset. We see that there is a concentration of scores along the diagonal representing matching predicted and true sentiments, indicating that our model understands the sentiment analysis task. We notice that there are few scores in the far corners, demonstrating that there are very few times in which our model is far off from the true sentiment. However, the significant number of scores in prediction groups adjacent to the true diagonal also indicate that our model can be close to the true sentiment, but slightly off. This is likely a result of human sentiments being difficult to discern based on context. For example, the boundary between "negative" and "somewhat negative" is relative to the human interpreter, making it difficult to exactly classify each of these sentiments. Our model learns from the sentiment scores determined by a human, and these slight distinctions in classification scores may make it difficult for our model to accurately complete this task.

To further examine this, we calculate an "off-by-one" score, in which we determine the number of correctly classified sentiments and the number of sentiments whose predicted sentiment is off by 1. We then find that 96.83% of the predicted sentiments are either correct or off by one relative to the correct sentiment, demonstrating our model's ability to closely determine the range of the sentiment score accurately. This indicates that our model may be less able to discern the true boundaries between exact scores, where the distinction between "negative" and "somewhat negative" is vague. To address this, we may focus more on polarizing sentiments which are easier to classify and less dependant on human interpretation.

**Paraphrase Detection.** While examining our model's performance on the QQP dataset, we discovered that our model struggles with sentence pairs that have a high number of the same words but differ in specific details. For example, the model often inaccurately classifies a sentence pair as paraphrases of one another if they start and end with the same words. For example, the questions "How do warm and cold fronts form?" and "How does a cold front form?" are misclassified as paraphrases of one another when they are not. They both start with "How" and end with "cold front[s] form" but differ in the inclusion of "warm" which the model fails to pick up on. Conversely, the model often misclassifies a sentence pair as not paraphrases of one another if one sentence has an additional phrase. For example, the questions "Do eyebrows grow back after being shaved?" and "Do eyebrows grow back?" are incorrectly classified as not paraphrases of one another when they are. They differ in the phrase "after being shaved" which the model then interprets as giving the sentence another meaning.

**Semantic Textual Similarity.** To analyze our model's performance on the STS task, we created a scatter plot 8 visualizing the relationship between predicted and target similarity scores. Overall, there is a strong correlation between the scores, but we also highlighted a few notable outliers (calculated using 4 standard deviations). For instance, consider the sentences "Work into it slowly." and "It seems to work." Our model predicted a high similarity score of 2.995 while the target similarity score is 0.0. This suggests that our model relies on surface-level word matching. It struggles to understand deeper semantic meaning and contextual understanding, especially for shorter sentences.

## 7 Conclusion

We explore the adaptation and enhancement of BERT for multitask learning on downstream tasks. Our approach incorporates several key techniques including multi-task fine-tuning, cross-encoding, additional pre-training, data augmentation, SMART regularization, and LoRA. By carefully tuning the hyperparameters for each extension, we achieve significant performance improvements across all three tasks of sentiment analysis, paraphrase detection, and semantic textual similarity, with our model outperforming baseline results in tasks. We further conclude that while LoRA results in decreased performance relative to full fine-tuning, decreasing the rank of LoRA implementations does not significantly degrade performance while improving time and memory usage.

However, the primary limitations of our work include the time and resources necessary to test variations of minBERT. In the future, we propose expanding on these extensions by further optimizing hyperparameters, training for more epochs, or implementing modifications applicable to more types of layers in the case of LoRA. However, extensive additional testing was not feasible for this project due to the timeline of the class and compute credits necessarily to test a single minBERT variation.

## 8 Ethics Statement

The first broad category of risks include inherent bias or discrimination in the training data, which can stem from bias in human evaluation or source material. As a result of training on biased datasets, minBERT may also exhibit biased output and recommendations. For example, if a sentiment reads "Durian tastes disgusting.", the overall sentiment could be classified as negative despite a clear personal bias. However, when trained on this data, the BERT implementation may attribute "durian" to negative sentiments. Our model may later reveal this bias by classifying a new review as negative because of the mere presence of the word. While this example may seem innocuous, if the word was instead replaced by a name or group of people, our model faces a serious ethical challenge by potentially classifying that name or group in a biased way. To mitigate this ethical challenge, we could manually review the input data to remove inputs which demonstrate clear personal bias or discrimination. We could also redact the names of specific people or groups to ensure that the model is not attributing specific sentiments to individuals or people based on the biases of the training data, instead relying on the contextual evidence in the rest of the sentiment. Another similar ethical consideration and societal risk is that historical stereotypes or biases may be present in the training data, leading minBERT to learn offensive words, phrases, or sentiments. For example, if an offensive or stereotypical word was used to describe a group of people, our model may determine that those two phrases are paraphrases of each other. In doing so, our model may perpetuate stereotypes by outputting that two sentences are indeed paraphrases or have close semantic textual similarity, leading those reviewing the outputs to equate a highly offensive phrase with certain groups or people when those phrases should clearly not be paraphrases of each other. To mitigate this risk, we could once again review the inputs to manually remove offensive words and stereotypes, or we could add in additional categories of classification for the STS outputs to indicate the presence of problematic phrases.

A second possible ethical challenge of our minBERT implementation is the usage of a few datasets which are not representative of all people, dialects, or other populations. For example, British English can differ from American English in both word usage and intention, and the inclusion of only one group of speakers can result in the exclusion of a large fraction of language speakers whose sentiments are not being fully understood by the model. Thus, when our implementation of BERT is tested on the language of the excluded group, it is more likely to result in incorrect classifications or outputs. This particularly affects our model due to the limited datasets we are training on, specifically the three datasets described previously. To mitigate this exclusion risk, we could intentionally train our model on inputs from many dialects and groups of speakers to broaden the model's understanding of language.

## References

Eneko Agirre, Daniel Matthew Cer, Mona T. Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *sem 2013 shared task: Semantic textual similarity. In *International Workshop on Semantic Evaluation*.

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. MTRec: Multi-task learning over BERT for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669, Dublin, Ireland. Association for Computational Linguistics.

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

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. 2019. SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *CoRR*, abs/1911.03437.

Chijioke Mgbahurike, Iddah Mlauzi, and Kwame Ocran. 2024. Jack of all trades, master of some: Improving bert for multitask learning.

Alex Riggio. 2023. Lora: Low-rank adaptation from scratch — code and theory. `https://medium.com/@alexmriggio/lora-low-rank-adaptation-from-scratch-code-and-theory-f31509106650`.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to fine-tune bert for text classification?

Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. `https://arxiv.org/abs/1704.05426`. [Accessed 22-05-2024].

## A Appendix

### A1. Batch Size

| Batch Size | | | Data Utilization | | | Dev Score | | | |
|---|---|---|---|---|---|---|---|---|---|
| **SST** | **QQP** | **STS** | **SST** | **QQP** | **STS** | **SST dev** | **QQP dev** | **STS dev** | **Overall** |
| 16 | 16 | 16 | 0.71 | 0.02 | 1.0 | 0.501 | 0.738 | 0.463 | 0.657 |
| 16 | 32 | 16 | 0.71 | 0.04 | 1.0 | 0.500 | 0.743 | 0.491 | 0.662 |
| 8 | 32 | 8 | 0.71 | 0.09 | 1.0 | 0.505 | 0.721 | 0.469 | 0.654 |
| 4 | 32 | 3 | 0.94 | 0.23 | 1.0 | 0.492 | 0.762 | 0.543 | 0.675 |
| 3 | 32 | 2 | 1.0 | 0.32 | 0.94 | 0.503 | 0.764 | 0.520 | **0.676** |

Table 7: Effect of batch size on dev score for Multi-task Fine Tuning (MFT)

### A2. Additional LoRA Results

We examine two learning rates: $1e-3$ and $1e-5$. Decreasing the learning rate to $1e-5$ drastically decreases performance. In particular, the trial with $1e-5$ results in a nearly constant accuracy score, with no learning observed over time. The overall score was lower across all tasks relative to the the LoRA trial with the same rank ($r = 128$). Thus, we keep the learning rate at $1e-3$ for all other trials.

Additionally, we see that higher dropout rates result in significantly worse performance due to the smaller number of trainable parameters relative to full model fine tuning. In particular, increasing the LoRA layer dropout probability to 0.3 results in lower performance across all tasks for rank 256 relative to a dropout probability of 0.1. This is further detailed in Table 9.

| Hyperparameters | | Space and Time | | Dev Score | | | |
|---|---|---|---|---|---|---|---|
| **Rank** | **LR** | **% TP** | **Train Time** | **SST dev** | **QQP dev** | **STS dev** | **Overall** |
| 1 | 1e-3 | 21.73 | 2:00 | 0.350 | 0.731 | 0.307 | 0.578 |
| 4 | 1e-3 | 22.20 | 2:00 | 0.330 | 0.741 | 0.300 | 0.574 |
| 8 | 1e-3 | 22.83 | 2:01 | 0.339 | 0.729 | 0.314 | 0.575 |
| 16 | 1e-3 | 24.09 | 2:02 | 0.348 | 0.736 | 0.321 | 0.582 |
| 32 | 1e-3 | 26.61 | 2:02 | 0.344 | 0.739 | **0.347** | **0.586** |
| 64 | 1e-3 | 31.65 | 2:02 | **0.354** | 0.735 | 0.309 | 0.581 |
| 128 | 1e-3 | 41.74 | 2:24 | 0.342 | **0.745** | 0.307 | 0.580 |
| 128 | 1e-5 | 41.74 | 2:03 | 0.150 | 0.632 | 0.073 | 0.440 |
| 256 | 1e-3 | 61.90 | 3:08 | 0.323 | 0.740 | 0.308 | 0.572 |

Table 8: Effect of rank and learning rates on Low-Rank Adaptation performance

| Model Description | SST Dev | QQP Dev | STS Dev | Overall |
|---|---|---|---|---|
| **r=256, dropout=0.01** | 0.323 | 0.740 | 0.308 | 0.572 |
| **r=256, dropout=0.03** | 0.145 | 0.392 | 0.044 | 0.353 |

Table 9: Effect of dropout rates on Low-Rank Adaptation performance

In these figures, we observe the benefits of Low-Rank Adaptation on GPU Utilization and GPU Memory Allocated. In each of the three figures (5, 4, 3), we observe that the LoRA trial with rank 128 utilizes the most memory or percentage of GPU. By the end of the observation period, the trial with the next largest rank, $r = 64$, utilizes the second largest amount of memory in 5 and 4. The smallest LoRA decomposition matrix, with rank 8, consistently utilizes the lowest percentage of GPU and smallest amount of memory by the end of the observation period. Thus, our implementation of LoRA demonstrates the benefits of implementing rank decomposition matrices in decreasing the GPU usage and memory requirements as rank decreases. In conjunction with lower training times, we observe that lower ranks of LoRA matrices are effective in reducing the time and space requirements for training extremely large models without sacrificing performance relative to other higher ranks. The benefits of rank 32 or lower are apparent: we save on memory and GPU utilization while receiving similar or better performance on the three tasks assigned for minBERT.

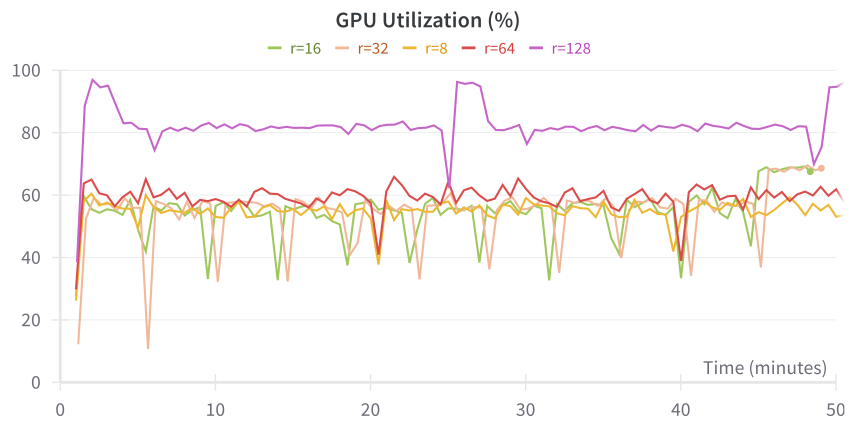

Figure 3: GPU Usage Percentage for Selected LoRA runs

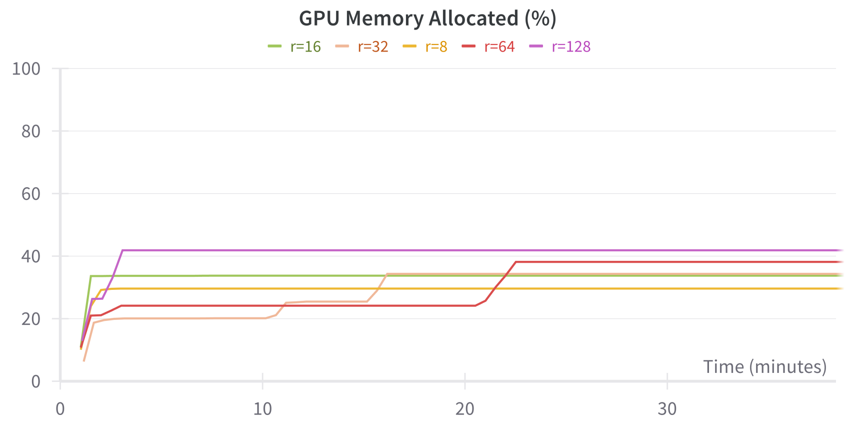

Figure 4: GPU Memory Usage Percentage for Selected LoRA runs

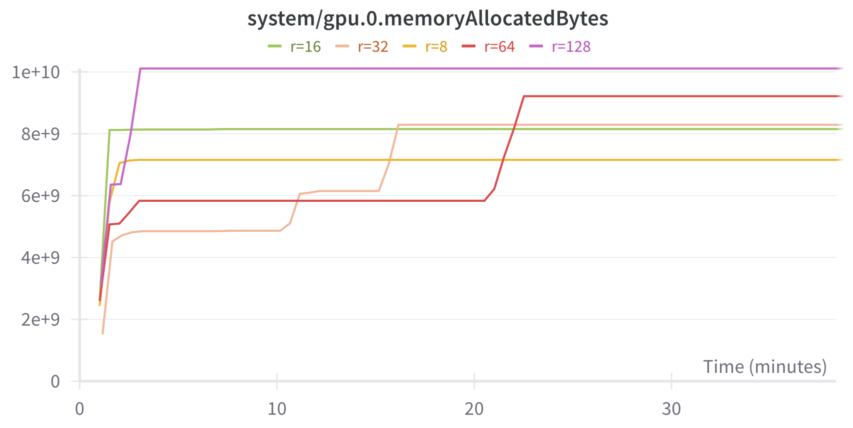

Figure 5: GPU Memory Allocated for Selected LoRA runs

## A3. MNLI Pretraining on QQP

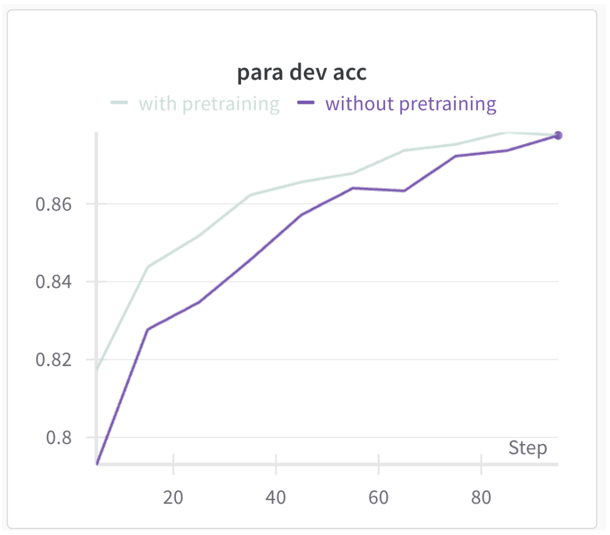

Figure 6: Performance boost of MNLI pretraining on QQP task wanes with more epochs

## A4. SMART loss on SST

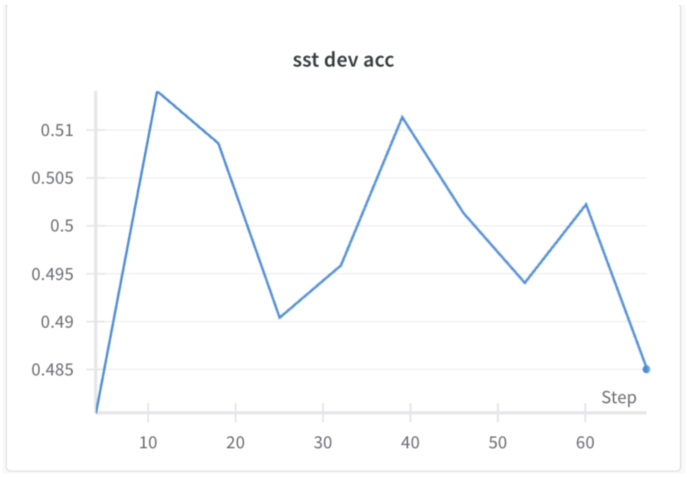

Figure 7: SST task with SMART loss does not indicate learning

## A5. Scatter plot for STS Analysis

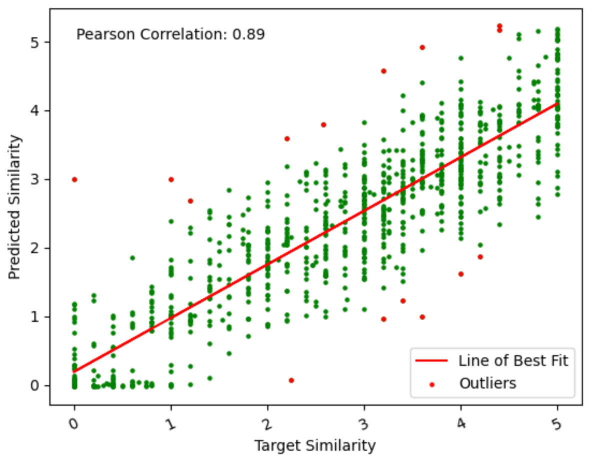

Figure 8: Scatter plot of predicted and target similarities for STS