BEST-FiT: Balancing Effective Strategies for Tri-task Fine Tuning

Stanford CS224N Default Project

Elijah Kim
Department of Electrical Engineering
Stanford University
elijahkim@stanford.edu

Rohan Sanda
Department of Electrical Engineering
Stanford University
name@stanford.edu

Abstract

In this paper, we explore multitask learning (MTL) with the minBERT model to address three distinct tasks from the GLUE Benchmark: semantic sentiment analysis, paraphrase detection, and semantic textual similarity. We implement and evaluate four task-scheduling schemes alongside various regularization and data augmentation techniques to mitigate overfitting. Our work also incorporates SMART regularization to stabilize training and PCGrad to manage conflicting task gradients. We find that our custom scheduling and regularization strategies significantly improve performance and stability among tasks with high dataset size variability. The insights from our experiments provide useful conclusions about the effectiveness of various MTL techniques.

1 Key Information to include

Mentor: Zhoujie Ding, External Collaborators (if you have any): N/A, Sharing project: No

2 Introduction

As natural language processing (NLP) models continue to grow in complexity and size, efficient fine-tuning for downstream tasks has become increasingly important. Multitask learning (MTL) offers a promising approach by training a model on multiple related tasks simultaneously, leading to shared learning and improved generalization. In our project, we extend the minBERT model – a simpler version of the famous Bidirectional Encoder Representations from Transformers (BERT) model that was state-of-the-art in the early 2020s – to handle three distinct tasks from the General Language Understanding Benchmark (GLUE): semantic sentiment analysis (SST), paraphrase detection (QQP), and semantic textual similarity (STS), Wang et al. (2019), Devlin et al. (2019). Our modifications include exploring and optimizing task-scheduling strategies, loss functions, and data augmentation techniques to enhance model performance and stability.

While multitask learning offers numerous benefits for downstream task performance, current methods for fine-tuning large language models often face significant hurdles such as overfitting to smaller datasets and unstable parameter updates. These issues are particularly pronounced in MTL scenarios where the model must balance learning from tasks with vastly different data volumes and complexities. In this paper, we examine several strategies to correct these shortcomings of MTL. Our findings highlight the potential of custom scheduling schemes and augmentation methods in MTL as well as provide insights into the challenges of multitask fine-tuning in large-scale NLP models.

3 Related Work

In the original release of BERT by Devlin et al. (2019), the transformer architecture, bidirectional contextualization, and pre-training and fine-tuning are used to achieve state-of-the art results on NLP tasks. However, because of the frequent lack of data for downstream tasks as well as the enormous complexity of modern transformer models, fine-tuning often overfits to the training set, hurting the model's performance on unseen data. Furthermore, fine-tuning complex models on smaller datasets can lead to aggressive and unstable parameter updates, again hurting its performance.

This problem has primarily been approached via hyperparameter tuning. For instance, Howard and Ruder (2018) propose a scheme that consists of discriminative fine-tuning (separate learning rates for each layer), slanted

triangular learning rates (learning rates are first linearly increased, then decreased), and a two-layer target task classifier. While this method did improve fine-tuning performance, it comes at the cost of high complexity.

Jiang et al. (2020) proposes a method to reduce overfitting through SMART loss. In this process, small amounts of noise added to the embedding input and the model is penalized for the corresponding changes in the output. This efficiently stabilizes parameter updates, smoothing model fitting and reducing overfitting, improving performance by over 1% on several tasks on the GLUE benchmark. Another approach to reducing overfitting is improving the dataset itself. Wei and Zou (2019) introduce techniques they call Easy Data Augmentation. In this, they randomly swap, replace, insert, and delete words in the dataset in order to generate additional rows. The method is most effective for small datasets, and the authors report significant improvements.

Other work has sought to tackle additional challenges in MTL, besides overfitting. Yu et al. (2020a) tackled the problem of gradient interference between tasks. They propose projecting a task's gradient onto the normal plane of another task's conflicting gradient to prevent destructive interference during optimization. This technique, known as PCGrad, was found to improve learning efficiency and performance by resolving conflicting gradient updates across tasks.

4 Approach

We implement the minBERT model, inspired by the original BERT paper by Devlin et al. (2019). Our BERT backbone consists of an embedding layer followed by 12 encoder transformer layers. These layers utilize sinusoidal positional encodings along with multi-headed attention layers, based on the architecture proposed by Vaswani et al. (2023). Each layer is followed by a residual connection and layer normalization. Finally, we pass the CLS token embedding into a single-layer multilayer perceptron (MLP) for downstream tasks. As an optimizer, we implement the weight-decayed Adam Optimizer.

4.1 Baseline

In addition to our single-task baseline, we also sought to establish baselines for our model's multitask learning performance. To this end, we implement a rudimentary multitask minBERT model with a two-layer MLP classifier head that uses round-robin scheduling to simultaneously train over all three tasks. We fine-tune the full multitask model, which we will henceforth call *baseBERT*.

4.2 Sentence concatenation

We tried concatenating the two sentences prior to passing them into the BERT model, separated by a separator (`SEP`) token. We hypothesized that this would provide significantly better results in the QQP and STS tasks compared to passing the two sentences in separately and concatenating the resulting embeddings since the BERT model could learn from seeing both sentences being compared at the same time. This architecture difference is diagrammed in Figure 1.

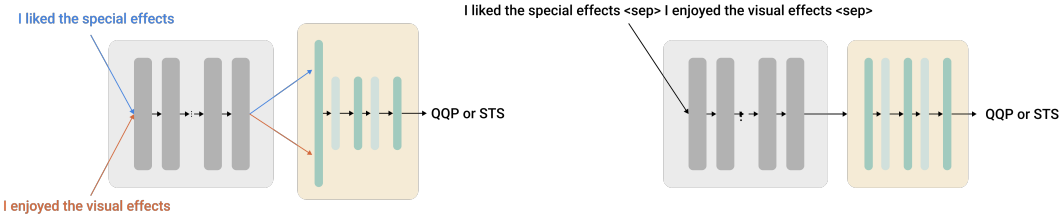


Figure 1: Concatenating embeddings vs tokens. Note that in the approach diagramed on the left, the two sentences are being passed into the BERT model separately. The resulting embeddings are then concatenated.

4.3 Loss Functions

Since SST involves categorical classification, cross entropy loss is used as the task's loss function. Binary cross entropy loss is used for QQP since there are only two discrete categories ("yes" and "no"). STS, on the other hand, is a regression task that predicts similarity scores rather than discrete classification, so (mean-squared error) MSE loss is used to minimize the distance between predicted and actual values.

4.3.1 Contrastive Loss

We also experimented with contrastive loss, inspired by Hadsell et al. (2006) by adding a contrastive loss factor to the task loss function, encouraging similar sentences to have similar embeddings and dissimilar sentences to have different embeddings.

Let y represent the label (1 for similar meanings, 0 for different meanings), e_1 and e_2 be the embeddings from the two sentences, and c be the cosine similarity between e_1 and e_2. Define $d = 1 - c$. The contrastive loss L is given by:

$$L_{\text{categorical}} = \frac{1}{2}\left[y \cdot d^2 + (1-y) \cdot \max(m-d, 0)^2\right] \qquad L_{\text{continuous}} = \left|\frac{y}{20} - d^2\right|$$

For the QQP task, when $y = 1$, the $L_{\text{categorical}}$ function encourages embeddings to be close together, and for $y = 0$, it encourages the cosine similarity between embeddings to be less than m. We used $m = 0.5$. For the STS task, where the label ranges continuously from 0 to 5, the loss function $L_{\text{continuous}}$ aims for d to be 0 when $y = 0$ and 0.5 when $y = 5$, maintaining a target cosine similarity of 0.5 for unrelated sentences in both QQP and STS tasks.

4.3.2 Ordered Cross Entropy Loss

To improve performance on the SST task, we tried creating a custom loss function by modifying the cross entropy loss to penalize predictions that are far from the true class in terms of their average value. For example, the loss should be higher for a sentence that is classified as a 0 but is actually a 4, compared to if the sentence is classified as 3 but is actually 4. To do this, we add a distance penalty D which is given by: $D = \left(\sum_{j=0}^{N-1} p_j \cdot j - y\right)^2$, where p_j is the predicted probability for class j and y is the true class label. This distance penalty is then scaled down by a factor of 10 and added to cross entropy loss to give the final custom loss function L.

$$L = \text{CrossEntropyLoss}(\mathbf{p}, y) + \frac{1}{10}D$$

4.4 Scheduling Policies

As shown in section 5.1, each of the three downstream tasks has vastly different dataset sizes. For instance, the QQP dataset contains over 20 times the number of examples as the STS dataset. This naturally leads to a tradeoff between training on more batches per epoch and overfitting to smaller datasets, and training on fewer batches but underfitting on the data-abundant tasks. We explore this tradeoff by experimenting with three different scheduling policies: 1) round-robin (default), 2) random scheduling, 3) annealed sampling, and 4) a modified round-robin schedule.

In the random schedule, we sample a batch from a task with uniform probability. This serves as an interesting comparison to the annealed sampling technique, where batch $\mathcal{B}_i$ from Task i is sampled with weighted probability $p_i \propto N_i^{\alpha}$, where N_i represents the size of Task i's dataset. Following the definition proposed by Stickland and Murray (2019), we set $\alpha = 1 - 0.8\frac{e-1}{E-1}$, where e is the current epoch and E is the total number of epochs. This scheme effectively trains on larger datasets towards the beginning of training, but gradually samples tasks with uniform probability in later epochs. In our custom schedule, we first train only the QQP task (corresponding to our largest dataset) on 80% of its training data for two epochs, before switching to a round-robin policy. This is motivated by the fact that gradient surgery techniques, like PC Grad, require an even distribution of tasks to effectively correct gradient disparities across tasks.

4.5 SMART Regularization and Gradient Surgery

To address overfitting, we implement the SMART framework developed by Jiang et al. (2020). This framework consists of two contributions: 1) a smoothness-inducing adversarial regularizer, and 2) a proximal point method to solve the new optimization problem. In this project, we focus on the first point. The regularization term is added to the standard loss, formulated here as an optimization problem:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta) \quad \text{where} \quad \mathcal{R}_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_s(f(\tilde{x}_i;\theta), f(x_i;\theta))$$

The smoothness-inducing adversarial regularizer stabilizes the output of f for small perturbations $\tilde{x}_i$ of the input x_i within a distance ϵ, measured by the p-norm. This encourages f to be smooth in a neighborhood around x_i, thereby preventing gradient blow-ups and consequent overfitting. Note that for classification tasks, $\ell_s(\cdot)$ is chosen to be the KL-Divergence. For regression tasks, MSE is used. We implement SMART regularization using the implementation provided by Archinet [1] and solve the optimization problem using the AdamW optimizer. Following the advice of (Hayes (2023)) that SMART regularization performance decreases in the presence of unlearnable noise caused by dropout layers, we disable dropout layers while computing the SMART regularization term.

To address conflicting gradient updates, we adopt the PCGrad technique proposed by Yu et al. (2020b). If two tasks' gradients, $\nabla\mathcal{L}_a$ and $\nabla\mathcal{L}_b$, conflict (so they negative cosine similarity), we will project the second gradient

[1](Archinet) Smart PyTorch Github: https://github.com/archinetai/smart-pytorch

onto the normal plane of the first gradient:

$$\nabla_a^{PC} = \nabla \mathcal{L}_a - \frac{\nabla \mathcal{L}_a \cdot \nabla \mathcal{L}_b}{\|\nabla \mathcal{L}_b\|_2^2}$$

This is repeated for every pair of gradients. The modified gradients are then summed to create the combined PC gradient. Because the PC Grad algorithm requires having separate gradients from each task, a round-robin scheduling policy is preferable. We were inspired by the implementation by Tseng [2]. However, we made several improvements to the code to reduce its memory footprint, which previously severely limited our batch size (to be around 4) on GPU. Our version can tolerate a batch size of 32. We validated the implementation by running our version and the Tseng version for a batch size of 2, and demonstrated comparable results.

4.6 Data Augmentation Methods

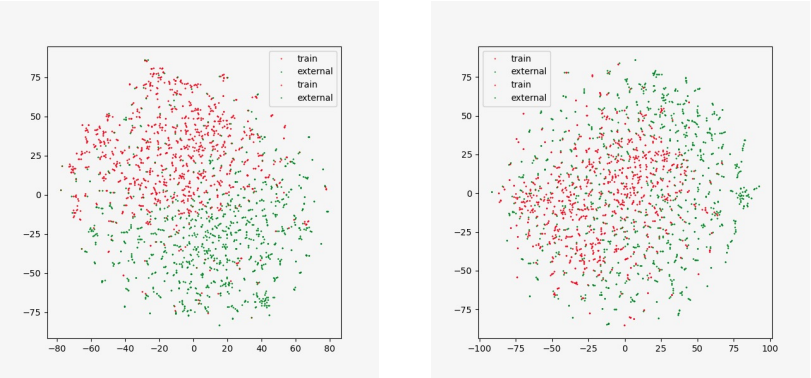
4.6.1 Easy Data Augmentation (EDA)

We tried using Easy Data Augmentation (EDA) methods proposed by Wei and Zou (2019) to augment our training datasets. Each sentence was augmented with 8 other sentences using synonym replacement, random insertion, random swap, and random deletion methods. We modified 10% of the words in the newly generated rows.

4.6.2 Embedding Base Data Augmentation

We also tried using an external dataset to augment the existing SST dataset. To do this, we used Amazon reviews for movies and TV shows from Jianmo Ni's Amazon review dataset with 8.8 million reviews Jianmo Ni (2019). Reviews were filtered so that the distribution of sentence lengths and the number of reviews per category were the same as the original SST train dataset (these distributions are shown in the Appendix). Originally, most of the sentences in the reviews were not very similar to the sentences in the SST dataset, as many reviews discuss topics not relating to the quality of the movie (such as packages getting lost in the mail).

To remedy this, embeddings for each review in the external dataset and each sentence in the SST dataset were generated with the `all-MiniLM-L6-v2` sentence BERT model by Nils Reimers (2021). For each label in the SST dataset, the average embedding was computed, and the 3000 sentences with the greatest cosine similarity from reviews with the corresponding number of stars were selected. This significantly improved performance over the training on the external dataset without filtering based on the embedding.



(a) Embeddings sampled from original Amazon dataset

(b) Closest embeddings in Amazon dataset

Figure 2: External dataset embeddings (green), before and after filtering for the closest embeddings

We also tried to leverage more powerful language models to augment our dataset. To this end, we tried using Llama3-8b from Meta (2024) to label sentences in our external dataset. Llama was given a movie review and instructed as follows.

```
Label the following sentence as negative_sentiment, somewhat_negative_sentiment,
neutral_sentiment, somewhat_positive_sentiment, or positive_sentiment.
```

Llama was also given 11 examples of sentences and their labels from the SST train set. Llama was then used to label 5000 sentences from the Amazon dataset which was pre-filtered for reviews close in the embedding space. These labeled sentences were then combined with the original SST train set.

[2](Tseng) PCGrad PyTorch Github: https://github.com/WeiChengTseng/Pytorch-PCGrad

5 Experiments

We present our baselines and some experimental results below. We achieved the required results on the base minBERT model (without multitask training): **Last Linear Layer Fine-Tuning: {SST dev = 0.416, CFIMDB dev = 0.784}**, **Full Model Fine-Tuning: {SST dev = 0.520, CFIMDB dev = 0.963}**. These results demonstrate the efficacy of our core implementation of minBERT.

Our baseline multitask model achieved an accuracy of: baseBERT: SST dev = 0.490, QQP dev = 0.778, STS dev = 0.483}. The hyperparameters used throughout this section are (unless stated otherwise): learning rate of $2e-5$, $(\beta_1, \beta_2) = (0.9, 0.999)$ weight decay of 0.1, 2-layer MLP classifier head (ReLU activiation), dropout rate of 0.3, batch size of 8, 2000 batches per epoch.

5.1 Datasets and Evaluation Metrics

We utilized three benchmark datasets for our experiments, each addressing different natural language processing tasks. These datasets include tasks for paraphrase detection, sentence similarity evaluation, and sentiment classification.

- **Quora Dataset (QQP)**: Contains 404,298 question pairs labeled to indicate if they are paraphrases. The dataset is split into 283,010 training examples, 40,429 validation examples, and 80,859 test examples. We use accuracy to evaluate model performance on this binary classification task.
- **SemEval STS Benchmark**: Comprises 8,628 sentence pairs with similarity scores ranging from 0 (unrelated) to 5 (equivalent). The splits include 6,040 training examples, 863 validation examples, and 1,725 test examples. Performance is measured by the Pearson correlation (R^2) between the predicted and true similarity scores.
- **Stanford Sentiment Treebank (SST)**: Consists of 11,855 sentences from movie reviews with labels for five levels of sentiment: negative, somewhat negative, neutral, somewhat positive, and positive. The dataset includes 8,544 training examples, 1,101 validation examples, and 2,210 test examples. BERT embeddings are used to predict these sentiment labels, and evaluation is based on classification accuracy.

5.2 Sentence Concatenation and Loss Functions

Our baseline where we are concatenating embeddings performs very poorly, but can be improved significantly through contrastive learning. Switching to concatenating sentences before passing into BERT *marked with (SC)*, dramatically improved performance. Sentence concatenation with Ordered Cross Entropy Loss helps the model train more quickly, but fails to increase performance above the sentence concatenation model with standard cross entropy loss. Figure 3 illustrates these findings. The numerical results from these runs can be found in Table 4. Thus, we continued development using the standard cross entropy loss with sentence concatenation model.

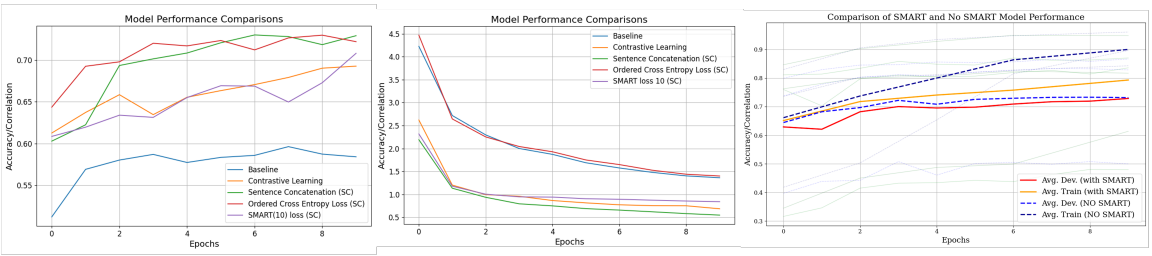


Figure 3: (Left) Loss function accuracy vs. epochs, (Middle) Loss function average loss vs. epoch, (Right) SMART regularization ($\lambda_s = 8$) significantly reduces overfitting compared to baseline.

5.3 Scheduling Policies

Having improved our baseline with sentence concatenation, we now turn our attention to improving scheduling schemes so take advantage of larger datasets while not overfitting to smaller ones. We show our results for the four scheduling policies tested below. We define our baseline as: *baseBERT + Sent. Concat. + Round Robin*.

Training Approach Dev Set	**SST-5** Accuracy	**QQP** Accuracy	**STS** R^2	**Overall** Score
Round Robin Baseline	0.508	0.833	0.858	0.733
Random	0.501	0.835	0.851	0.729
Annealed Sampling	0.508	0.865	0.859	0.744
Custom Schedule	0.501	**0.889**	**0.863**	**0.751**

Table 1: Comparison of Different Training Approaches

We notice that annealed sampling provides a small benefit to the QQP accuracy, which makes sense given that we are more likely to sample data from the QQP dataset under this policy, where sampling probability is proportional to dataset size. Nonetheless, the gain is not significant as our batch size is still 2000, meaning that we are not leveraging the Quora train dataset. Under the custom policy, this changes. Now the first two epochs each train on 80% of the Quora dataset. This increases training time but dramatically improves QQP results. STS also benefits. Because of the large benefits to QQP and STS that outweighed slightly worse SST performance, we used the custom schedule for future models.

5.4 SMART Regularization and Gradient Surgery

We present our results for SMART regularization and gradient surgery below. For SMART regularization, we use the default parameters suggested by Jiang et al. (2020): number of steps $T = 1$, step size $\eta = 1e-3$, $\epsilon = 1e-6$, noise variance $\sigma^2 = 1e-5$. The regularization weight λ_s is tuned with varying behavior for large and small values. We also run PC Grad and compare against the *baseBERT + Sent. Concat.* baseline. We define our baseline as: *baseBERT + Sent. Concat. + Round Robin*. All experiments use the round-robin schedule.

Added Technique Dev Set	**SST-5** Accuracy	**QQP** Accuracy	**STS** R^2	**Overall** Score
Baseline	0.508	0.833	0.858	0.733
PC Grad	0.495	0.821	0.857	0.724
SMART Regularizer ($\lambda_s = 0.50$)	**0.512**	0.817	0.864	0.731
SMART Regularizer ($\lambda_s = 8$)	0.480	**0.846**	**0.870**	0.732

Table 2: Comparison of Different Training Approaches

First, we observe that PC Grad does little on its own to improve the baseline performance. While surprising at first, this is investigated further in Section 6. SMART regularization is shown to have wide-ranging effects depending on λ_s. Smaller values of λ_s lead to better performance for SST while harming QQP (which does not overfit as much due to its larger dataset), while larger values can boost both STS and QQP. However, the primary goal of SMART regularization reducing overfitting is achieved, as demonstrated by the reduced training-validation accuracy gap for SMART regularization compared to the baseline, shown in Figure 3.

5.5 Dataset Augmentation

Training SST on an dataset augmented to 9 times its original size with EDA methods gave slightly improved results above our baseline. Training on random Amazon reviews from our external dataset produced very poor results. After filtering them for the reviews closest (in the embedding space) to our original train examples, performance improved slightly over the baseline, though not to the level of of the EDA-trained model. However after using Llama to label our reviews, performance improved to significantly above baseline.

Data Augmentation Dev Set	**SST-5** Accuracy	**QQP** Accuracy	**STS** R^2	**Overall** Score
EDA	0.501	0.832	0.855	0.729
Random Amazon reviews	0.380	0.837	0.858	0.694
Amazon reviews close in embedding space	0.492	0.833	0.861	0.729
Llama-labeled reviews	**0.519**	0.837	0.844	0.733

Table 3: Comparison of Different Data Augmentation Approaches

5.6 Additional Experiments

In addition to the experiments described above, we also tuned the batch size and learning rate hyperparameters and experimented with different pooling schemes (max, mean, and attention-based pooling). For brevity, we move discussion of our results on these experiments to the Appendix. As a summary, we found that a larger batch size generally led to better performance. None of the pooling mechanisms we tried out-performed using the CLS token. We also tried training the single task SST classifier with various batch sizes, and found a batch size of 16 yielded the best SST results.

5.7 Results

Overall, we observe new behavior upon stacking our best results from the preceding sections. First, for large SMART regularization weights under the custom schedule, the STS accuracy collapses to about 0.265. We believe that this is because SMART regularization penalizes abrupt parameter updates, and that the SST accuracy experiences abrupt changes in its training accuracy under the custom schedule. Overall, we see that our best

Table 4: Summary of Dev Experiment Results

Model	SST-5 Accuracy	QQP Accuracy	STS Correlation	Overall Score
Baseline (baseBERT)	0.490	0.778	0.483	0.584
Contrastive Learning	0.408	0.812	0.859	0.693
Sent. Concat. (SC)	0.501	0.832	0.855	0.729
SC + Ordered Cross Entropy Loss	0.495	0.828	0.844	0.722
SC + Custom Schedule (CS)	0.501	0.889	0.863	0.751
SC + SMART(0.50)	0.512	0.817	0.864	0.731
SC + CS + SMART(0.50)	0.491	0.888	0.870	0.750
SC + CS + SMART(7)	0.286	0.891	0.872	0.683
SC + CS + SMART(0.50) + PCGrad	0.500	0.845	0.870	0.738
SC + Llama SST data augmentation	0.519	0.837	0.844	0.733
SC + CS + SMART(8) + (α, $\|\mathcal{B}\|$)=(1e-5, 16)	0.262	0.899	0.888	0.683
SC + CS* + SMART(10) + (α, $\|\mathcal{B}\|$)=(2e-5, 32)	0.262	**0.899**	**0.893**	0.686
minBERT, Batch Size 16	**0.528**	N/A	N/A	N/A

Table 5: Best performance (9th Place on Test Leaderboard)

Model	SST-5 Accuracy	QQP Accuracy	STS Correlation	Overall Score
DEV	0.528	0.899	0.893	0.773
TEST	0.538	0.898	0.879	0.792

results for QQP and STS come from using SMART regularization with optimal hyperparameter configurations found in the Appendix. Note that CS* refers to a schedule where we train 4 epochs just on QQP followed by another 8 epochs round-robin. The best results for SST come from training it in isolation of other tasks on the minBERT model with optimal hyperparameters. Curiously, we find that data augmentation techniques are not particularly effective in boosting SST performance in the single-task setting – which is investigated further in our Analysis section.

We ensemble our best results from the best QQP/STS model and the best STS model. **In the end, we are 9th on the leaderboard (TEAM NAME = BEST-FiT) at the time of submission (9 PM on Sunday, June 9th)**, demonstrating the efficacy of our approach.

6 Analysis

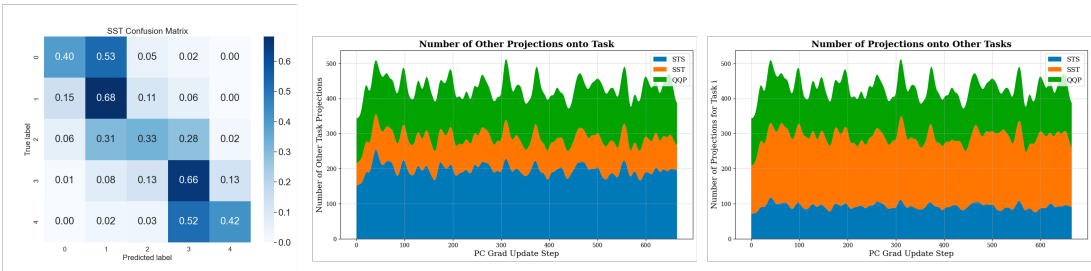


Figure 4: (Left) SST Confusion Matrix (Middle) Number of projections of other task-gradients onto Task i's gradient, (Right) Number of projections of Task i onto other task-gradients.

6.1 PC Grad and Task Malleability

To understand PC Grad's limited effectiveness, we examined two metrics over the first 2000-step epoch for six task-gradient pairings: 1) the number of Task i's gradient projections onto Task j, and 2) the percent change in Task i's gradient magnitude after projection onto Task j. Each update step involves 1350 gradient pairings, with an average of 32% resulting in a projection. This rate declines in later epochs. The smoothed stacked area plots reveal that QQP and SST tasks frequently project onto STS, indicating STS's distinct gradient path influences other tasks. SST, in particular, often aligns its gradients with others, showing high adaptability. Data further

supports this, with SST's gradients undergoing on average a 25% magnitude change post-projection, more than twice that of any other task.

These findings indicate SST's gradients are highly responsive to the learning paths of other tasks. This adaptability facilitates SST's integration in multitask learning but also means its gradient updates are heavily influenced by other tasks, causing instability in SST's learning trajectory. This may explain why PC Grad negatively impacts SST performance. Training SST independently could reduce these issues by isolating its learning process from conflicting task interactions, stabilizing its dynamics for more consistent and improved performance. Indeed, our best results are achieved when SST is trained separately.

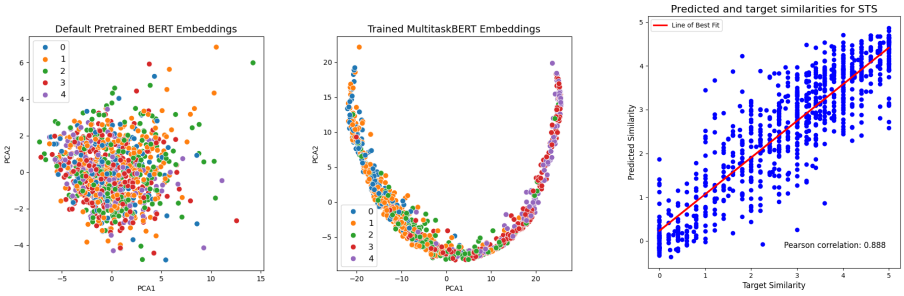


Figure 5: SST dev set BERT embeddings, STS prediction scatterplot

6.2 Data Augmentation and SST Challenges

As seen in the confusion matrix (Figure 4), our final SST model has a hard time predicting neutral sentiment accurately, and over-predicts somewhat positive and somewhat negative sentiment. This is also observable in the fact that the dev embeddings for labels 0 and 1, and the embeddings for labels 3 and 4 fall on similar parts of the embedding plot (Figure 5). This may be due in part to the fact that the SST train set has more `somewhat positive` and `somewhat negative` examples, (See Figure 7 in the appendix) biasing the model to select those categories. Additionally, when examining the original SST train data manually, there are a number of cases where we personally disagree with the original labeling. For example:

- I couldn't recommend this film more. (somewhat positive)
- A gem, captured in the unhurried, low-key style favored by many directors of the Iranian new wave. (neutral)
- Cold, pretentious, thoroughly dislikable study in sociopathy. (somwhat negative)

These potential internal inconsistencies in the training set likely contributes to why the model struggles to achieve a high accuracy, even with a large, well-labeled augmented dataset.

6.3 QQP and STS Accuracy

Looking at the sentences our MultitaskBERT model gets incorrect for QQP and STS, the models tend to think two sentences have the same meaning when they share many words. For example, our best multitask model thought the sentences "what is the importance of communication skills?" and "what is the importance of communication skills in school?" had the same meaning in a QQP task. The STS model output also has some outliers, visible in the STS scatterplot in Figure 5. For example, the model thought "indian stocks open lower" and "indian stocks close lower" had a similarity of 4.2, even though those sentences actually have opposite meanings. This indicates that the model is likely largely looking at how many words are similar and fails to pick up on finer nuances of language.

7 Conclusion

In our investigation, we found that custom scheduling methods that are tuned for multitask training set sizes can significantly improve model performance. We found that existing embedding models and foundation models can be used to effectively augment training datasets. Our experiments validate the efficacy of SMART regularization in improving model performance in multitask learning. Additionally, our custom ordered cross entropy loss function lets a model reach the same accuracy in less time. A key limitation is that much of our work, such as custom schedulers and hyperparameter searches, are specific to this set of tasks and training datasets and won't generalize to broader NLP tasks. Moving forward, we want to see how techniques like SMART regularization and PC Grad affect overfitting on new downstream tasks. We also seek to investigate multitask learning in state space models like MAMBA.

Team Contributions: Rohan implemented minBERT, and was responsible for SMART regularization, PC Grad, custom schedulers, pooling techniques, and doing a grid search of hyperparameters. Elijah implemented MulitaskBERT and was responsible for easy data augmentation, augmentation with external data, ordered cross entropy loss, contrastive loss, confusion matrix and final embedding plots. We did experiments, analysis, and writing together.

8 Ethics Statement

All language models come with the risk of reflecting biases in their training data, as explored in class. However, our data augmentation technique has the potential to magnify existing biases in the training data when expanding the dataset. Choosing sentences that are close in the embedding space magnify existing biases in the data, for example if in the original dataset reviews for movies with stereotypical white savior tropes tend to be higher, our data augmentation technique would reinforce this bias. Additionally, by using Llama to label movie reviews, we additionally inherit the biases of the Llama foundation model. If this model was used to decide what movies to display as most popular, the biases could reduce public exposure to stories about underrepresented minorities. This can be mitigated by performing a bias audit of the original SST dataset by analyzing the embedding space for distances between sentences representing different groups, and having a sample of the reviews themselves manually inspected by a multicultural human review board. Then, with the data from the bias analysis, adversarial debiasing can be used to reduce these biases in the model.

Additionally, this project carries with it the societal risk of people misusing the model for nefarious means, such as using the SST classifier to target people for harassment by scraping social media feeds to find fans of a politicized movie. This could be mitigated by making the model only available for use through an API, and monitoring usage for suspicious activity such as requesting sentiment analysis for many reviews of a trending politicized movie. Users could be required to sign an agreement prohibiting use for harassment, and accounts engaging in suspicious activity can be suspended.

References

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

R. Hadsell, S. Chopra, and Y. LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742.

Matthew Hayes. 2023. Serbertus: A smart three-headed bert ensemble.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

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.

Julian McAuley Jianmo Ni, Jiacheng Li. 2019. Justifying recommendations using distantly-labeled reviews and fined-grained aspects. *Empirical Methods in Natural Language Processing (EMNLP)*.

Meta. 2024. Meta-llama-3-8b.

Nils Reimers. 2021. all-minilm-l6-v2.

Asa Cooper Stickland and Iain Murray. 2019. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Jason W. Wei and Kai Zou. 2019. EDA: easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, abs/1901.11196.

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

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020b. Gradient surgery for multi-task learning. *CoRR*, abs/2001.06782.

A Appendix (optional)

If you wish, you can include an appendix, which should be part of the main PDF, and does not count towards the 6-8 page limit. Appendices can be useful to supply extra details, examples, figures, results, visualizations, etc. that you couldn't fit into the main paper. However, your grader *does not* have to read your appendix, and you should assume that you will be graded based on the content of the main part of your paper only.

A.1 Pooling Methods

As stated earlier, we experiment with three different pooling techniques in addition to using the CLS token for downstream tasks. In mean pooling, we take the average of all token embeddings across the sequence length dimension – motivated by wanting to capture trends from across the embedding sequence. In max pooling, we select the maximum value from each feature dimension across all token embeddings – hoping to select the most prominent features from the sequence. Finally, we introduce a custom pooling scheme we call "attention pooling", that uses a single attention layer to compute attention scores for each token embedding, then (after passing scores through a softmax), uses the associated probabilities to weight the sum of the token embeddings. In this way, we seek to focus on the most "important" embeddings defined by the attention layer. We define our baseline as: *baseBert + Sent. Concat. + Custom Schedule*.

Pooling Approach Dev Set	**SST-5** Accuracy	**QQP** Accuracy	**STS** R^2	**Overall** Score
Baseline (CLS)	0.501	0.889	0.863	0.751
Mean	0.489	0.887	0.869	0.748
Max	0.480	0.881	0.861	0.741
Attention	0.491	0.884	0.852	0.742

Table 6: Comparison of Different Training Approaches

We observe that the CLS baseline performs best under the custom schedule. As a result, we did not explore or use any of the variant pooling schemes. Mean pooling did seem the most promising of these approaches. Clearly, additional model complexity in the form of an attention layer did not help performance.

A.2 Hyperparameter search

We ran a simple grid search hyperparameter search over learning rate and batch size to optimize performance. The results are shown in the table below. Each model is of the following: *baseBERT + Sent. Concat. + Custom Schedule* run for 6 epochs. From this analysis, we use a batch size of 16 and learning rate of 1e-5 for our best

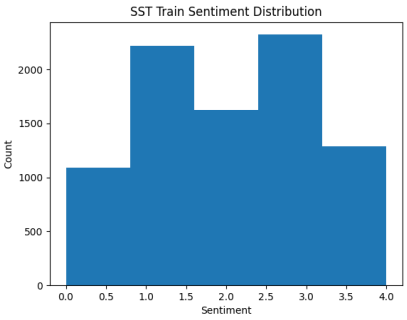
Batch Size	**Learning Rate**	**SST-5 Accuracy**	**QQP Accuracy**	**STS Correlation**
8	5×10^{-5}	0.391	0.776	0.829
8	1×10^{-5}	0.501	0.821	0.854
8	2×10^{-5}	0.500	0.821	0.856
8	1×10^{-4}	0.262	0.632	0.077
12	5×10^{-5}	0.253	0.632	0.063
12	1×10^{-5}	0.479	0.831	0.863
12	2×10^{-5}	0.472	0.850	0.879
12	1×10^{-4}	0.253	0.632	0.129
16	5×10^{-5}	0.490	0.840	0.859
16	1×10^{-5}	0.493	0.850	**0.884**
16	2×10^{-5}	0.493	0.843	0.868
16	1×10^{-4}	0.262	0.632	0.039
32	5×10^{-5}	0.497	0.844	0.866
32	1×10^{-5}	**0.506**	0.855	0.869
32	2×10^{-5}	0.498	**0.858**	0.877
32	1×10^{-4}	0.479	0.825	0.850

Table 7: Hyperparameter Tuning Results: Batch Size and Learning Rate Sweep

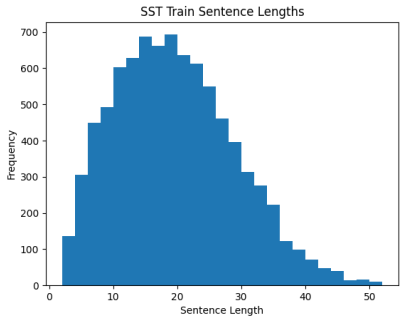
model runs - as we were particularly interested in boosting STS performance at the time of this run. QQP accuracy for this hyperparameter configuration is also quite good.

A.3 SST Train Label Distribution

We show the distribution of the training labels for SST below.

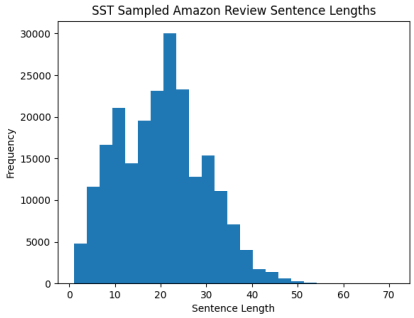


(a) SST train label distribution



(b) SST train sentence length distribution

Figure 6: SST train dataset distributions

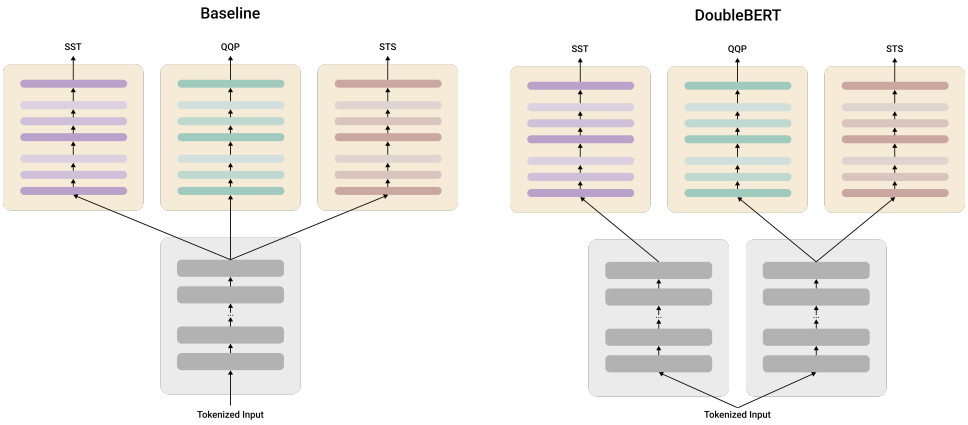


(a) SST train sentence length distribution

Figure 7: External Amazon dataset distribution after sampling

A.4 Double BERT

Since the QQP and STS tasks are very similar, we tried a model where the QQP and STS task heads share a single BERT model, and SST gets its own model.



(a) Baseline model architecture (b) Double BERT architecture

Figure 8: SST train dataset distributions