A SMARTer minBERT

Stanford CS224N Default Project

Arisa Chue
Department of Computer Science
Stanford University
achue@stanford.edu

Daphne Liu
Department of Computer Science
Stanford University
daphnehl@stanford.edu

Poonam Sahoo
Department of Computer Science
Stanford University
pnsahoo@stanford.edu

Abstract

Our goal is to increase the performance of our minBERT model on multiple downstream tasks, specifically on sentiment analysis, paraphrase detection, and semantic textual similarity. We compare smoothness-induced regularization and Bregman proximal point optimization (SMART) from Jiang et al. (2020) with the gradient surgery PCGrad algorithm from Yu et al. (2020) along with cross-encoding sentence pairs to determine the best additional extensions on top of our pre-existing multi-task minBERT model. We also combine these extensions together to further evaluate performance. We find that the extension that had the most impact on our paraphrase detection and semantic textual similarity results was cross-encoding sentence pairs. However, our strongest model overall was a combination of SMART, cross-encoding, and PCGrad which achieved an overall test score of 0.752.

Key Information to include

Mentor: Josh Singh External Collaborators (if you have any): N/A Sharing project: N/A

1 Introduction

Advancements in data-driven neural models have drastically improved the field of Natural Language Processing. However, deep learning models on a single task are often bottlenecked by data scarcity when training and are limited by the possibility of overfitting. This can cause extensive computing costs and can be inflexible when given a variety of tasks, thus failing to replicate human language ability. Recent research on multi-task learning, which leverages multiple related tasks to improve performance on various tasks, shed light on the positives of multi-task learning models (Chen et al., 2021).

In addition to multi-task learning improving the performance of models, this multi-task learning approach resembles the human brain the most. Humans learn the natural language using a single brain on a multitude of tasks, from translation to question answering. To truly replicate our model to mimic human behavior, a model that can accurately learn on multiple tasks would be the most relevant (Zhang et al., 2023).

However, our original minBERT model is designed for sentiment analysis. Our project takes inspiration from multi-task learning and extends the original minBERT model to perform well on a variety of downstream tasks. We acknowledge that training a minBERT model on sentiment analysis alone is not enough to mimic human behavior. Through this project and exploration, we improve the model's performance on all three different tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.

2 Related Work

There have been many efforts to improve BERT performance on several NLP tasks. Because these tasks often involve finetuning a preexisting BERT model, multi-task learning often has pitfalls with overfitting. As a result, the efforts to improve BERT multi-task learning performance have often involved some form of regularization. Jiang et al. (2020) introduce an "adversarial" regularization method that promotes models' smoothness within the neighborhoods of all model inputs. A popular way to benchmark performance on these tasks is GLUE (General Language Understanding Evaluation), a benchmark suite that spans NLP tasks as diverse as sentiment analysis, sentence entailment, and semantic similarity (Wang et al., 2018). Jiang et al. demonstrate SMART's success with multi-task learning by showing that the model with SMART outperforms BERT on all five GLUE benchmark tasks. Because of how well SMART performed on benchmarks, and how the authors suggested default hyperparameters, we were particularly interested in SMART.

Another technique for improving the performance of multi-task learning more generally was proposed by Yu et al. (2020). The authors describe their PCGrad gradient surgery algorithm which resolves conflicting gradients between different tasks by projecting a task's gradient onto the normal plane of the gradient of the conflicting task (Yu et al., 2020). We were especially interested in PCGrad because we noticed during our initial implementation of multi-task minBERT that an improvement in one task would lead to a decrease in improvement for a different task.

Finally, improvements on BERT such as RoBERTa and ALBERT in the multi-task learning realm give us a better sense of the approach to multi-task learning with regards to NLP and the sorts of architecture changes that can be made to improve performance (Liu et al., 2020; Lan et al., 2020). For example, just small changes in embeddings in RoBERTa led to an improvement in performance when compared to BERT (Liu et al., 2020). This inspired us to later pursue cross-encoders to improve our model performance.

3 Approach

Our starting point was the $BERT_{BASE}$ model released by the original authors as our baseline (Devlin et al., 2019). We have been given a skeleton for this code. This model has 12 layers, 768 hidden size, 12 attention heads, and 110M total parameters. The $BERT_{BASE}$ has been widely used as a baseline in this field, which can help us compare our results to other studies. We adapted this code for a multi-task learning approach and then proceeded with the following extensions.

3.1 SMART

A common problem when finetuning multi-task models is overfitting, so we were interested in using regularization techniques to help combat this. As discussed earlier, SMART is a regularization method that promotes model smoothness within the neighborhoods of all model inputs. More specifically, in the case when an input is perturbed by a small amount of noise, the desired output would still be the same; SMART helps during training in the context of this case by optimizing the loss $\mathcal{F}(\theta)$ by choosing the θ that minimizes $\mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_f(\theta)$.

We define

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

where ℓ is the task-specific loss function, λ_s as the tuning parameter, and $\mathcal{R}_{\mathrm{s}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_{\mathrm{s}} \left(f \left(\tilde{x}_i ; \theta \right), f \left(x_i ; \theta \right) \right)$ as the smoothness-inducing adversarial regularizer. This regularization term measures the local Lipschitz continuity under the KL-divergence metric, which is precisely what promotes local smoothness within neighborhoods of all model inputs. In the original paper, Jiang et al. (2020) use symmetrized KL-divergence $\ell_{\mathrm{s}}(P, Q) = \mathcal{D}_{\mathrm{KL}}(P \| Q) + \mathcal{D}_{\mathrm{KL}}(Q \| P)$, for classification tasks, and a squared loss of $\ell_{\mathrm{s}}(p, q) = (p - q)^2$ for regression tasks. The discrete definition of KL-divergence is $\mathcal{D}_{\mathrm{KL}}(Q \| P)=\sum_i Q_i \log \left(Q_i / P_i\right)$. The most notable distinction between using some form of KL-divergence vs. squared loss is that squared loss amplifies the effect of outliers, and that KL-divergence measures "information loss" between distributions. We noticed while reading the paper on SMART that the authors use KL-divergence as a metric of success for their models (Jiang et al., 2020). Because of this, we also run experiments where KL-divergence is used as a regularizer function for all tasks and where KL-divergence is used as a regularizer function for all steps except for the final one, where symmetrized KL-divergence is used instead.

Finally, to solve the regularization term, Jiang et al. introduce Bregman proximal point optimization methods that find the desired θ that minimizes $\mathcal{F}(\theta)$ without perturbing the given model parameters too much. The specific update method for the model parameters θ for the $(t+1)$th iteration is $\theta_{t+1}=\operatorname{argmin}_\theta \mathcal{F}(\theta)+\mu \mathcal{D}_{\text{Breg}}\left(\theta, \theta_t\right)$ where μ is a tuning parameter and $\mathcal{D}_{\text{Breg}}$ is the Bregman divergence defined as $\mathcal{D}_{\text{Breg}}\left(\theta, \theta_t\right)=\frac{1}{n} \sum_{i=1}^n \ell_{\mathrm{s}}\left(f\left(x_i ; \theta\right), f\left(x_i ; \theta_t\right)\right)$. When μ is large, the Bregman divergence is a strong regularizer that ensures that θ_{t+1} does not deviate too much from the previous iteration.

We adapted our minBERT code to match the structure in the smart-pytorch python package (Archinet, 2022). In addition, when we used minBERT with two embeddings for the paraphrase detection and sentence similarity tasks, we modified the smart-pytorch code to take in two embeddings. We use the default hyperparameter values that are suggested in the paper for our tasks, as the authors had already verified their performance for BERT and RoBERTa on similar tasks.

3.2 Random Sampling

The initial framework of our model trained the paraphrase task with a Quora dataset with significantly more data samples than the datasets used for sentiment classification and similarity analysis ($141,506$ vs $8,544$ and $6,401$ respectively). We felt that this could overtrain the paraphrase task. In fact, during our baseline experiments, we did see that our paraphrase accuracy was significantly higher than sentiment and similarity. Moreover, the paraphrase task took approximately ten times longer during training, which lengthened our training process significantly.

We decided to move forward by randomly sampling only 10% of the Quora dataset. We randomly sampled indices without replacement from the dataset and only used data with those indices. The size of this data is more comparable to the SST and STS datasets while still representative of our original dataset. Our sample sizes are $14,150$, $8,544$, $6,401$ now for paraphrase, sentiment, and similarity respectively.

This same procedure was used on other datasets when we added extra data during training.

3.3 Gradient Surgery (PCGrad) and Round-Robin Training Loop

The gradient surgery algorithm is as follows: if a gradient of a task conflicts with a gradient of another task, we will remove the conflicting component by projecting the gradient of one task onto the other. Say the gradient of a task is $\mathbf{g}_i^{\mathrm{PC}}$ and the gradient of the other task is $\mathbf{g}_j$. We first check if they are conflicting using cosine similarity by checking if $\mathbf{g}_i^{\mathrm{PC}} \cdot \mathbf{g}_j < 0$. Then, if conflicting, we perform a projection:

$$\mathbf{g}_i^{\mathrm{PC}}=\mathbf{g}_i^{\mathrm{PC}}-\frac{\mathbf{g}_i^{\mathrm{PC}} \cdot \mathbf{g}_j}{\left\|\mathbf{g}_j\right\|^2} \mathbf{g}_j$$

By modifying gradients through projections, we can decrease the amount of gradient interference within different tasks.

To incorporate gradient surgery into our minBERT implementation, we took inspiration from Tseng's gradient surgery implementation in PyTorch (Tseng, 2020). To do so, we downloaded their Python file that contained the gradient computation functions and wrapped our original Adam Optimizer in this class.

The nature of gradient surgery requires the utilization of all three tasks' losses, and therefore their gradients. Instead of training our model on each task's dataset independently, we had to modify the structure of our code to adopt a round-robin training loop strategy. To compute each respective batch per task together, we front-loaded each task's training data in the beginning and transitioned to a while loop that would retrieve the respective batches in the same iteration. The while loop will continue to the longest dataset and compute each loss only if we have remaining data in that task since we had varying data lengths. Thus, we only incorporated the gradient surgery functions when there were at least two losses computed, and reverted to the Adam optimizer when there was one loss left.

3.4 Cross-Encoding of Sentence Pairs

For the paraphrase and sentence similarity tasks, the data consists of pairs of sentences. Originally, our minBERT model took in each sentence as a separate embedding. However, we decided to implement a cross-encoding of the sentence pairs to see if it would improve performance for the paraphrase and sentence similarity tasks. A cross-encoding of a sentence pairs involves combining the embedding of sentence pairs in some way to better capture their relationship to each other (Documentation, 2014). The benefits of cross-encodings are that they tend to improve accuracy and can capture the relationship between two texts in context with each other (Ntongana, 2024). We implemented the cross-encoding technique ourselves by modifying the Datasets class so that sentence pairs are automatically collated together before being passed into the embedding function.

3.5 Training on More Data

Initially, we felt that our models could be overfitting too closely to the movie dataset (SST dataset). Since the Quora dataset was significantly larger than the other datasets, we also wanted to find more data for the sentiment and similarity tasks.

For the sentiment task, we obtained Yelp review data, where each review corresponds to its "star evaluation" that ranked sentiment on a 5-point scale (1-5). We manipulated the data a bit to have ratings 0-4 instead and removed extra columns to match the SST dataset exactly.

For the similarity task, we use the Stanford Natural Language Inference Corpus. The SNLI corpus used the majority vote for similarity from five people; the three labels were contradiction, neutral, or entailment. Meanwhile, the existing SST dataset is simply a value of sentiment. We did computation on the votes, mapping contradictions to 0s, neutral to 3s, and entailment to 5s, and took the average to come up with a similarity score from 0-5, similar to the STS dataset.

4 Experiments

4.1 Data

We are using the Stanford Sentiment Treebank (SST) for sentiment analysis (Socher et al., 2013). It has 11,855 single sentences from movie reviews. Each of the 215,514 unique phrases in the dataset has a negative, somewhat negative, neutral, somewhat positive, or positive label. We have 8544 samples for training and 1,101 samples for dev.

We used the Quora Dataset for paraphrase detection (Quora Data, 2016). This dataset consists of 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another. We are using a subset of this dataset, with 141,506 training examples, and 20,215 dev examples.

We are using SemEval STS Benchmark dataset for semantic textual analysis (Agirre et al., 2013). This consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). We have 6401 training examples, and 864 dev examples.

We also experimented with Bowman et al's (Bowman et al., 2015) Stanford Natural Language Inference (SNLI) Corpus for the sentence similarity task and Yelp's dataset which contains ratings and reviews (Yacharki, 2022) for the sentiment analysis task. The SNLI Corpus contains 570,000 human-written English sentence pairs manually labeled for balanced classification with entailment, contradiction, and neutral labels. The training dataset contains 550,152 pairs of samples. The Yelp dataset has 130,000 data points, each including a review alongside a rating in stars from 1 to 5. More details about how these datasets are used will be described later on in this section.

4.2 Evaluation method

We evaluate each model on sentiment classification on the dev SST dataset, paraphrase detection on the dev Quora dataset, and Semantic Textual Similarity correlation on the dev STS dataset. We then look at the training curve for each task.

4.3 Experimental details

We have run different combinations to see the impact of each addition on the baseline minBERT model. For each model, we used a pretraining and fine-tuning learning rate of 0.001 and 0.00001 respectively, When incorporating gradient surgery, we decreased the batch size from 8 to 5 due to our CUDA GPU running out of memory from the additional computational costs. We later lowered the batch size from 5 to 3 when we included additional datasets. For SMART, we use the suggested hyperparameters mentioned in Jiang et al. (2020). This includes a step size of $1e-3$, $\epsilon = 1e-6, \mu = 1$, and $\sigma = 1e-5$. We discussed most of these hyperparameters earlier in the implementation of SMART, besides σ which is the standard deviation of the noise which perturbs the inputs.

Below are our described experiments; we previously described our justification for the different variations on SMART in Section 3.1.

- Baseline: *Our baseline implementation with the SST dataset, Quora Dataset, and STS dataset respectively for the three tasks.*
- Randomly Sampled Baseline: *For the rest of the experiments, we proceeded with random sampling on the paraphrase dataset, sampling* $1/10$ *of the dataset (n=14,150).*
- SMART: *We added SMART to each of the three tasks. Initially, we updated the smart-pytorch package to take in two embeddings, since the original code only took one embedding.*
- PCGrad: *We added PCGrad to resolve the issue of conflicting gradients between tasks. The idea is that we can use gradient surgery to find the optimal search direction without confusing our model due to conflicting gradients.*
- PCGrad + SMART: *We combined PCGrad and SMART into the same model.*
- PCGrad + SMART (5 epochs): *We noticed from our literature review that most models only trained on 3-5 epochs. We wanted to explore if that was a reason our model was overfitting, so we retrained with 5 epochs.*
- PCGrad + SMART (additional data): *We added data from SNLI and Yelp.*
- Cross-Encoding: *We cross-encoded sentence pairs so that we could pass singular embeddings into our models, rather than a pair of sentence embeddings.*
- Cross-Encoding + PCGrad + SMART 0: *We implemented a model combining all three extensions. For SMART, we used symmetrized KL-divergence for the classification tasks and squared loss for the regularization tasks.*
- Cross-Encoding + PCGrad + SMART 1: *This model is the same as the one above, except the regularizer used KL divergence for all tasks.*
- Cross-Encoding + PCGrad + SMART 2: *This model is the same as the one above, except the regularizer used symmetrized KL divergence for the last iteration and KL divergence for all previous iterations.*

4.4 Results

We decided to drop two of our experiments: PCGrad + SMART (5 epochs) and PCGrad + SMART (additional data). With 5 epochs, the model was still quickly reaching above 90% training accuracy, and the model was performing a bit worse. We didn't feel a need to continue experimenting with 5 epochs, so all of our experiments were run on 10 epochs.

When adding additional data, our virtual machine instance could not handle the amount of data and the longer sentence lengths that we were training on, so we had to reduce our batch size. We also tried to randomly sample data as described in Section 3.2, but this did not help. We ultimately improved our performance via other extensions such as cross-encoders, so we moved forward without adding extra data. Although we did not add extra data, our implementation has been rewritten to take in a list of data filenames for scalability for the addition of datasets. Lastly, random sampling cut our training time from about 8 hours to about 3 hours, but adding PCGrad (and reducing the batch size to 5) increased the training time to about 5 hours again. Our best results on the dev leaderboard were from

	Sentiment Acc	**Paraphrase Acc**	**Similarity Corr**
Baseline	0.450	0.739	0.356
Random Sampling Baseline	0.460	0.650	0.374
SMART	0.482	0.659	0.363
PCGrad	0.469	0.721	0.368
PCGrad + SMART	0.486	0.719	0.351
Cross-Encoding	0.480	**0.830**	0.838
Cross-Encoding + PCGrad + SMART 0	0.460	0.818	0.845
Cross-Encoding + PCGrad + SMART 1	0.459	0.821	**0.847**
Cross-Encoding + PCGrad + SMART 2	**0.500**	0.825	0.843

Table 1: Dev accuracies for each model and each corresponding task

the Cross-Encoding + PCGrad + SMART 2 model, which we then submitted to the test leaderboard to get an overall score of 0.752. We had an SST test accuracy of 0.512, a paraphrase test accuracy of 0.821, and an STS test correlation of 0.845.

The sentiment analysis was the task that our model performed the worst on, and we were interested in delving into why. Overall, the highest reported accuracy on the SST-5 dataset was 0.598, which shows that this task is difficult in general (Heinsen, 2022). Further, the dataset itself is skewed, which can be shown by a graph of the distribution of labels vs. predicted results in Figure 1. The predictions holistically seemed to be made more based on the distribution of the dataset rather than any specific attributes, as the precision for most classes was below 0.50 besides for the strong positive label 4, where the precision was 0.70. Below we have provided the precision and F1-scores for our best model performance on the SST dev set.

The very high precision for class 4, and lower F1 score indicates there are few false positives, but also not many true positives. This is expected since there are 5 classes in the sentiment task, so there is higher probability for the model to make a misclassification, and 4 is more of a extreme label. Classes 1 and 3, however, have high F1 score and low precision, suggesting that our model is over-classifying 3's, and getting false positives. This also makes sense since the middle classes 1 and 3 are "slightly" positive and negative, which are probably safer bets for the model to choose as its predictions and also occur more frequently in the dataset.

Class	**Precision**	**F1-score**
0	0.48	0.44
1	0.51	0.58
2	0.42	0.33
3	0.49	0.57
4	0.70	0.42

Table 2: Precision and F1-score for our best model on the SST dev set

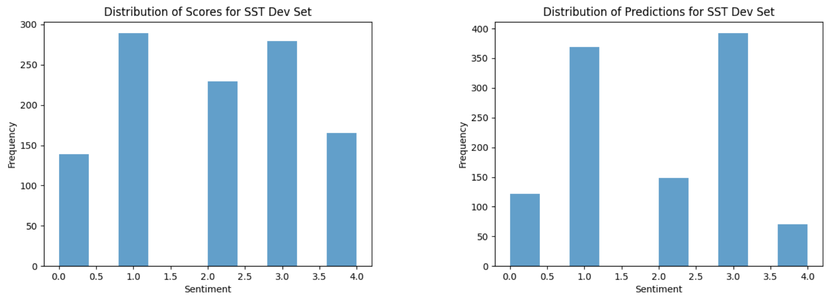

Figure 1: Distribution of Scores and Predictions for the SST Dev Set.

5 Analysis

5.1 Qualitative Evaluation of Sentiment Analysis

As mentioned above, the accuracy of sentiment analysis was always lower than the other tasks. We wanted to explore some qualitative reasons for this.

Our sentiment classification is multiclass classification with 5 classes, 0-4, which correspond with negative, somewhat negative, neutral, somewhat positive, and positive respectively. This means that something negative and somewhat negative, or positive and somewhat positive will have been classified with the correct sentiment, but just with an incorrect magnitude.

When we take a close look at our dev outcomes, we noticed that of the 1101 STS dev samples, 110 were cases where the actual sentiment was 1 and our prediction was 2 (or vice versa), and 126 were cases where the actual sentiment was 3 and our model predicted 4 (or vice versa). If we had some sort of binary sentiment task, for example, the binary version of the SST dataset (SST-2), our model's accuracy would have immediately increased by $\approx 21\%$.

Since paraphrase detection is a binary classification problem, there is already a random guess baseline of 50% accuracy, explaining why its accuracy is so much higher than sentiment (which would have a random guess baseline of 20%). Additionally, the similarity task is a regression problem, so the metric of evaluation is not comparable with that for classification.

5.2 Qualitative Evaluation of Paraphrase Detection

Overall, the model performance on the Quora dataset was much better than it was for the sentiment analysis task. For our best model, the accuracy was 82% on the dev set. The model performed better on the negative class, with its precision and F1-scores for the $0-$label being 0.89 and 0.85 respectively. For the $1-$label, the precision and F1-scores were 0.73 and 0.78 respectively. This made sense as the training and dev sets consisted of $62-63\%$ of $0-$label sentence pairs, so the model was able to train on more such data.

5.3 Qualitative Evaluation of Semantic Textual Similarity

One thing that is interesting about the STS dataset is that it has a fairly bimodal distribution where there are clear spikes at every integer value such as $0, 1, 2$, etc. This makes sense especially if there are human scorers who are more likely to default to a "round" number. Our distribution is slightly skewed to the right, which makes sense given that the highest frequency of numbers (especially the integers $3, 4$, and 5) was at the higher end of the labels.

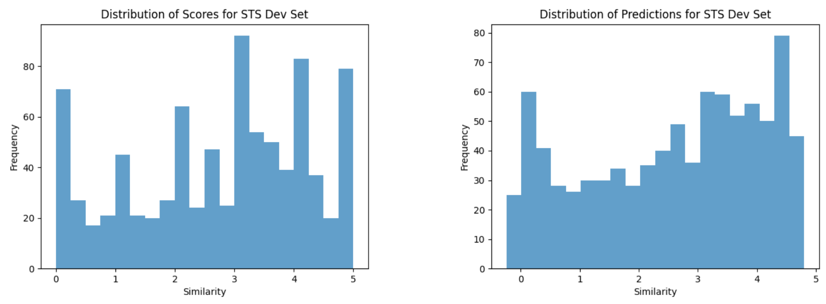

Figure 2: Distribution of Scores and Predictions for the STS dev set.

5.4 Experiment Findings and Failings

As observed in Table 1, the random sampling baseline model performed worse than our previous baseline. This makes sense since we were training our model on less data and thus limited its learning capacity. To combat this, however, we included our extensions of SMART, PCGrad, and cross-encodings to improve our model while lowering the computational cost and time needed to train our model. A notable improvement to our model was the cross-encoding extension, which made the biggest improvements in the paraphrase accuracies and similarity correlations. This is reasonable since cross-encodings are done on sentence pairs, and only the paraphrase and sentence similarity tasks examined two sentences at a time.

Lastly, it is interesting to observe that incorporating SMART 0 produced lower accuracies than SMART 1 and SMART 2. This conflicts with the loss functions that were recommended in Jiang et al. (2020). SMART 1 and SMART 2 both used KL divergence instead of symmetrized KL divergence or squared loss. Since KL divergence essentially functions as a generalization of squared distance, we were wondering if that was why using KL divergence instead of squared loss didn't really change the overall correlations for the STS dataset. The most notable impact was with the sentiment analysis accuracy, which jumped from 0.460 to 0.500 when using KL divergence for every iteration besides the last one, where we still used symmetrized KL divergence for the regularizer. Our suspicion is that because KL divergence is not symmetric, it penalizes the amount of information lost due to the perturbation more harshly, which may have led to better performance on the classification tasks.

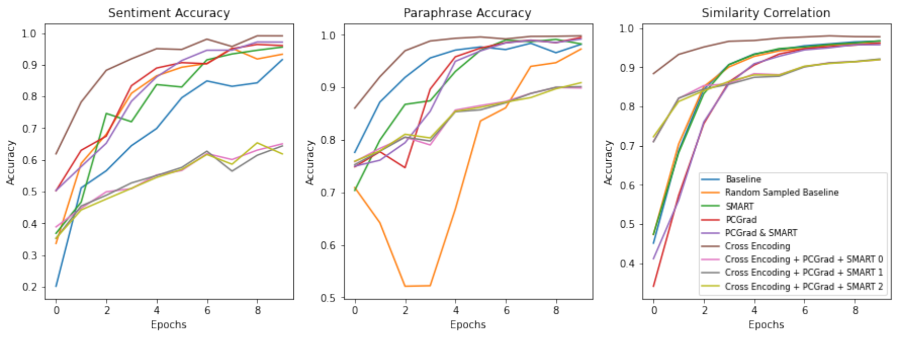

Figure 3: Training Trends Throughout Epochs for All Models

The training plots in Figure 3 provide an insight into our training accuracies throughout the epochs. Understandably, our task accuracies exhibit an increasing arc, but many of our models produced an extremely high accuracy by the last epoch. This could indicate our model overfitting to the given datasets. Our prediction of overfitting proved to be true, given that the models with the lowest training accuracies (pink, gray, light green) performed the best of the dev sets rather than the other models.

It is also interesting to note the steep decline in accuracy for paraphrase detection around epochs 2-4 for the random sampled baseline model. However, we can also observe this trend with the slight dips in the other models (red, pink, green, gray) as well. Additionally, the similarity correlation training curves are more stable than the other tasks. This can be due to correlation being a less rigid metric than accuracy, causing less erratic scores throughout the epochs.

6 Conclusion

To improve our original minBERT performance on multiple downstream tasks, we incorporated: 1) SMART regularization, 2) gradient surgery on conflicting task gradients, 3) cross-encoding embeddings for sentence pairs, and 4) additional training datasets to prevent overfitting. We achieved the best training accuracy results when combining cross-encoding, gradient surgery, and SMART 2 regularization (see Table 1) and obtained an overall dev score of 0.748 and a test score of 0.752. Out of all of the extensions we implemented, we found the biggest improvements by adjusting embeddings to be cross-encoding.

The main limitation of our project is the GPUs we trained our model on. With our current T4 GPUs on GCP, training took around 6-8 hours to complete even after decreasing our initial dataset. With a more powerful GPU, we see the potential of improving our model even further if we could train our model on the entire Quora dataset rather than our random sampling approach. Another limitation would be the scope of our project. Currently, we only measure our minBERT model on three tasks and only train on datasets catered to these tasks. We recognize that to replicate human language ability, a multi-task learning model must be more flexible.

For future work, we hope to examine the performance of a variety of downstream tasks on the model. When improving the accuracies of these tasks, we can take inspiration from the ROBERTa model to further extend our project (Liu et al., 2020).

Contributions: Arisa incorporated the PCGrad package for the gradient surgery extension and converted the training loops to a round-robin approach. Daphne performed all of the model training on GCP, added random data sampling, and changed the infrastructure for adding datasets. Poonam

changed the smart-pytorch package to accommodate two embeddings, modified the structure of the training code to fit the package implementation, and implemented the cross-encoding from scratch.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. Sem 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics (*SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, pages 32–43.

Archinet. 2022. smart-pytorch. `https://github.com/archinetai/smart-pytorch/`.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference.

Shijie Chen, Yu Zhang, and Qiang Yang. 2021. Multi-task learning in natural language processing: An overview.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Sentence Transformers Documentation. 2014. Cross-Encoders; Sentence-Transformers documentation — sbert.net. `https://www.sbert.net/examples/applications/cross-encoder/README.html`. [Accessed 13-03-2024].

Franz A. Heinsen. 2022. An algorithm for routing vectors in sequences.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2177–2190, Online. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Ro{bert}a: A robustly optimized {bert} pretraining approach.

Jones Ntongana. 2024. Decoding Sentence Representations: A Comprehensive Guide to Cross-Encoders and Bi-Encoders — plainenglish.io. `https://plainenglish.io/community/decoding-sentence-representations-a-comprehensive-guide-to-cross-encoders-and-bi-encoders-3a67` [Accessed 13-03-2024].

Quora Data. 2016. First quora dataset release: Question pairs. `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`. Accessed on Date Accessed.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y 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.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Yacharki. 2022. Yelp reviews for sa fine-grained (5 classes). `https://www.kaggle.com/datasets/yacharki/yelp-reviews-for-sa-finegrained-5-classes-csv`.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*.

Zhihan Zhang, Wenhao Yu, Mengxia Yu, Zhichun Guo, and Meng Jiang. 2023. A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods.