Exploring Improvements on BERT

Stanford CS224N Default Project

Sara Hong
Department of Computer Science
Stanford University
sara02@stanford.edu

Sophie Wu
Department of Computer Science
Stanford University
swulee@stanford.edu

Abstract

BERTram is our multitasking BERT model, which optimizes for three downstream tasks: sentiment classification, paraphrase detection, and semantic textual similarity. The process of exploring extensions on BERTram has not only significantly increased accuracies on the three said tasks compared to a baseline BERT model, but it has also revealed promising results about techniques to further improve performance, such as a combination of the right loss functions, hyperparameter adjusting, regularization, and gradient surgery.

1 Introduction

The default project tasks student researchers with the mission of exploring extensions to the basic BERT model to optimize for three seemingly similar yet subtly distinct tasks: sentiment classification, paraphrase detection, and semantic similarity. We responded to this challenge by building BERTram, our own BERT model that can obtain an overall accuracy nearly two and a half times greater than that of the basic BERT model, which was implemented with finetuning as well as the help of a number of different optimization techniques proposed by different scholars in this particular field of NLP research.

As effectively summarized in one paper that tackles multitask learning, Yu et al. (2020), there is a notable difficulty in the setting of multitask learning, which "presents a number of optimization challenges, making it difficult to realize large efficiency gains compared to learning tasks independently." In the first part of this project, we were able to witness this phenomenon firsthand with our minBERT classifier, which achieved an accuracy of 0.515 for the task of sentiment classification when finetuned and tested on the SST dataset alone, but dropped to an accuracy of 0.469 when finetuning and testing on all three datasets for the three downstream tasks. What made this project so interesting was the variety of explorations that could be made on so many aspects of the basic BERT model, and the uncertainty behind what exactly makes multitask learning so complex to optimize. Our task quickly morphed into that of finding the best combination of extensions to implement that would improve accuracies across the board for all of the downstream tasks. The most promising methods that we have discovered thus far are utilizing unique loss functions for each task, regularizing to prevent overfitting, and manipulating gradients to generalize learning across each of the downstream tasks. We will discuss each method with further detail in Section 3.

With the usage of large language models on the rise amongst the public, the objective of improving multitask learning is more important than ever, as more people try to utilize LLMs for more and more diverse arrays of tasks. We are more than grateful to be participating in the research of such a profound topic, advised by our project mentor, Olivia Lee (oliviayl@stanford.edu), who has graciously guided us on the proposal, milestone, and further explorations on this project.

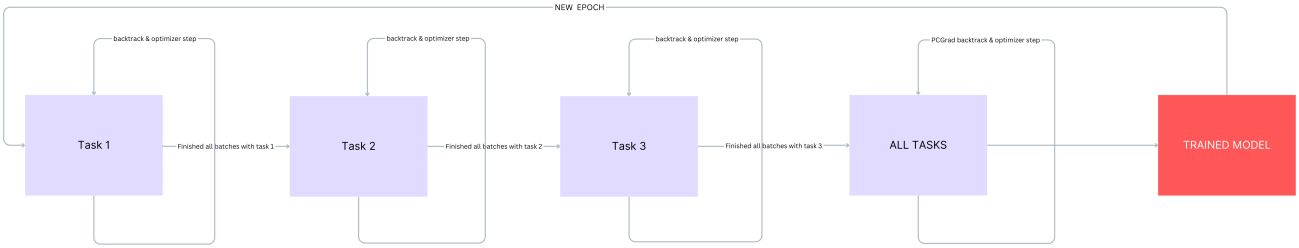
2 Related Work

There are a plethora of work to improve multitasking of BERT models. The works we have referred to for this project include SMART regularized optimization proposed in Jiang et al. (2020), as well as gradient surgery for multitask learning proposed in Yu et al. (2020). SMART regularized optimization was largely employed to resuscitate the accuracy for the sentiment classification task. We used the PyTorch implementation of the technique by the Archinet Open Source AI Research Lab published on GitHub (Archinet.ai, 2022). Gradient surgery, specifically PCGrad (projecting conflicting gradients) was employed across all three tasks using a PyTorch implementation of the technique by Wei-Cheng Tseng on GitHub (Tseng, 2020). We would also like to cite the procedures in Giulianelli et al. (2020), which is the research that began our explorations for this project, though we did not get the opportunity to explicitly utilize any specific techniques from this paper. We anticipate that knowledge of semantic change of words over time will benefit other downstream tasks that are not specific to the ones investigated in this project.

3 Approach

Our approach to this problem mainly involved the following three step process: (1) increase starting accuracy, (2) manage overfitting, (3) improve multitask finetuning. At each point in the process, we experimented with different hyperparameters. In the end, we concluded with the following framework for training our model.

Figure 1: Multitask training framework for BERTram



In the following subsections will break down the reasoning for the structure in the figure displayed above by detailing each of the three steps of our approach.

3.1 Increasing starting accuracy: ordering, similarity, and loss functions

Starting with the provided pretrained base BERT model yielded very low accuracy for every task. To resolve this issue, we finetuned our model on the datasets for each of the three tasks. We first began with one dataset (SST), then added another (Quora), and finally the last one (STS). Initially, we had always trained in the order of sentiment classification → paraphrase detection → semantic similarity, but we found that the model was always best on the last task it trained, so we switched up the orders in which the tasks were trained on in rounds. For example, first we would train sentiment → paraphrase → semantic, then for the next epoch we would train paraphrase → semantic → sentiment, and so on.

We also explored different metrics of similarity to explore ways to improve the accuracies for the comparative tasks, paraphrase detection and semantic similarity. We found that cosine similarity of two embeddings x, y: $cosine\_sim(x, y) = \frac{x \cdot y}{||x||||y||}$ yielded the most competitive results.

One of the last and most significant research we did was regarding different loss functions that would be suitable for each task. As classification tasks, for sentiment classification and paraphrase detection, the provided cross-entropy loss was an appropriate loss function for optimization. However, it was inappropriate for semantic similarity since it is closer to a regression task. Therefore, we first experimented with CosineEmbeddingLoss, which is calculated by the following equation:

$$\text{loss}(x, y) = \begin{cases} 1 - \cos(x_1, x_2), & \text{if } y = 1 \\ \max(0, \cos(x_1, x_2) - \text{margin}), & \text{if } y = -1 \end{cases}$$

However, the accuracy was relatively lower than what we had tried with just cross-entropy loss. Thus, we tried MSE loss, which is the mean squared error (squared L2 norm) between each element in the input x and target y. Using MSE loss yielded the best results.

3.2 Managing Overfitting: dropout and SMART

As briefly mentioned in Section 3.1, we initially addressed the problem of overfitting (before we even ran any trials) by adding a dropout layer during the computation of the logits for each task. We utilized the default dropout rate of 0.3 - we also experimented with rates of 0.4 and 0.5 but the best results came from the default rate.

The most significant approach to managing overfitting was applying SMART regularized optimization. However, this was only applied to the sentiment classification task, not on any of the other downstream tasks. SMART employs two specific techniques: (1) Smoothness-Inducing Adversarial Regularization and (2) Bregman Proximal Point Optimziation to both manage the complexity of the model and prevent aggressive updating in the finetuning process (Jiang et al., 2020).

Smoothness-Inducing Adversarial Regularization works by adding a small regularization term to the input, controlling the model's complexity such that it would not be extremely high. The paper Jiang et al. (2020) explains the process in the following manner:
Given the model $f(\cdot;\theta)$ and some datapoints denoted x_i, y_i, we solve for the optimization $min_\theta \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta)$ where $\mathcal{L}$ is our loss function, λ_s is a hyperparameter and $\mathcal{R}_s$ is the regularizer defined as

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

Here, ϵ is a hyperparameter and ℓ_s is defined the symmetrized KL-divergence $\ell_s(P,Q) = \mathcal{D}_{KL}(P||Q) + \mathcal{D}_{KL}(Q||P)$, because we are using this for the sentiment classification task.

The Bregman Proximal Point Optimization, which is used to solve the optimization task ($min_\theta \mathcal{F}(\theta)$) is explained in the same paper in the following manner.
The basic version of the Bregman proximal point method (which we applied) uses

$$\theta_{t+1} = argmin_\theta \mathcal{F}(\theta) + \mu \mathcal{D}_{\text{Breg}}(\theta, \theta_t) \tag{2}$$

where μ is a hyperparameter and $\mathcal{D}_{\text{Breg}}$ is defined as $\mathcal{D}_{\text{Breg}}(\theta, \theta_t) = \frac{1}{n}\sum_{i=1}^{n} \ell_s(f(x_i;\theta), f(x_i,\theta_t))$. Every iteration controls θ_{t+1} by keeping it close to the previous θ_t.

By combining the two methods, the proposed SMART technique is able to suppress the updates from deviating too much from its immediately previous iteration, and at the same time, contain the model's complexity within a reasonable amount. For further reading on this technique, please refer to the paper which initially proposed it, Jiang et al. (2020).

3.3 Improving multitask finetuning: combined loss, gradient surgery

As briefly mentioned in section 3.1, we initially addressed the problem of generalizing the model to multiple tasks by switching up the ordering of the datasets in which the model was trained on. We additionally approached the challenge of generalization by attempting additive loss, in which we added the loss of each task as the final loss function that we used to optimize upon, as suggested in the paper, Bi et al. (2022). Hence, the final loss function we tried was:

$$\mathcal{L}_{total} = \mathcal{L}_{\text{sentiment class}} + \mathcal{L}_{\text{paraphrase detect}} + \mathcal{L}_{\text{semantic sim}}. \tag{3}$$

Surprisingly, the additive loss was not very effective. Hence, we explored a different approach, gradient surgery. From the paper, Yu et al. (2020), we utilized projecting conflicting gradients, which, as explained by the paper, "alter the gradients by projecting each onto the normal plane of the other" if gradients are conflicting, so that we prevent "interfering components of the gradient from being applied to the [neural] network." The gradient of task i (denoted $\mathbf{g_i}$) would be projected onto the normal plane of the conflicting task j's gradient $\mathbf{g_j}$. For BERTram, we applied this gradient surgery by adding a second (Adam) optimizer that performs the surgery all at once for each epoch after every task has completed its batch training. Note that the back-propagation/optimizing step for each batch training of the tasks was separate from the gradient surgery done at the very end with the retained computation graphs of the very last batch training for each of the tasks.

4 Experiments

Finding the optimal model for multitasking on the three specified downstream tasks required many iterations of explorations with small and large changes to the training process. In this section, we outline the (1) data we used, (2) our evaluation methods, (3) details of experiments, and (4) results of said experiments.

4.1 Data

The following are the datasets we used for the project, which are all provided to us.

Stanford Sentiment Treebank (SST) dataset:

- Used for the sentiment classification task
- 11855 single sentences, total of 215154 unique phrases, annotated by 3 human judges
- input is the sentence being rated, and the output is a sentiment rating on a discrete scale of 0-4 where 0 is very negative is 4 is very positive

Quora dataset:

- Used for paraphrase detection task
- 400000 question pairs
- input are two sentences being compared, and the output is binary: 0 (two sentences are not paraphrases) or 1 (two sentences are paraphrases)

SemEval STS Benchmark dataset:

- Used for semantic similarity task
- 8628 different sentence pairs
- input are two sentences being compared, and the output is a rating on the continuous scale between 0.0-5.0 where 0.0 is “not related at all" and 5.0 is “same meaning"

4.2 Evaluation method

Accuracy was used as the evaluation metric for the tasks of sentiment classification and paraphrase detection as both are classification tasks. Pearson correlation was used for the task of semantic similarity.

To evaluate the progress and improvement of our transitioning model, we set a baseline BERT model, which achieves a 0.208 accuracy on sentiment classification, 0.375 on paraphrase detection, and -0.033 correlation on semantic similarity. For the sentiment classification task, our baseline model predicts the “neutral" class (2) for every datapoint. For the paraphrase detection task, the baseline model predicts 0.0 (not paraphrase of each other) for every datapoint. Lastly, for our sentiment textual similarity task, the baseline model outputs a random float between 0.0 and 5.0 for each datapoint. (The decision to randomize the output was made to make the Pearson correlation possible, since if every prediction was the same value, the standard deviation would be 0 so it would not be possible to calculate the Pearson correlation.)

4.3 Experimental details

To avoid redundancy, we have not included the individual experiment results for tweaking hyperparameters. However, we did experiment with different learning rates ($\eta = 10^{-3}, 10^{-4}, 10^{-5}$) and determined that the default value $\eta = 10^{-5}$ was the best. The dropout rate, as previously explained, was kept at the default value of 0.3 after experimenting with other values (0.4, 0.5). We also experimented with the number of epochs (10, 15, 20), and determined that 10 was the best for the overall score, though we did see some select improvements in some tasks.

There are also additional hyperparameters for SMART regularized optimization, which was employed only for the sentiment classification task. Our hyperparameters are reflective of the implementation described in Yu et al. (2020). We used $\epsilon = 10^{-5}$, $\mu = 1$, and $\lambda_s = 1$. There's also an extra hyperparameter in the random initialization of $\tilde{x_i}$s, σ^2, which is the variance of this random initialization. We

set $\sigma = 10^{-5}$ as suggested by the paper. The hidden size as well as the batch size was kept as the default (768 and 8, respectively).

Training time was a significant issue due to the limited financial sources and time on this project. In order to speed up training, we used a small (shuffled) sample (of about 8000 training samples) of the Quora dataset, which was the largest and took the most time (up to 50-60 minutes per epoch for training). The rest of the datasets took no more than 3 minutes per epoch, so their datasets were not reduced for the purposes of training. Therefore, unless otherwise stated, we will have always used 8000 datapoints for training paraphrase detection. (There are three occasions when we used a different dataset size; 9000 for experiment 8^{**}, exactly 6040 datapoints for each task for experiment 12, $13^{\dagger}$, and the full Quora dataset for the last experiment 14, which was also our final test submission.)

In addition, we used the ordering of training SST $\rightarrow$ Quora $\rightarrow$ STS for our initial trials. In experiment 7^{*}, we experimented with ordering last the dataset associated with the task with the lowest training accuracy, which proved to be unsuccessful and therefore abandoned. In experiments numbered 8-11^{***}, we utilized the alternating method for ordering described in Sections 3.1 and 3.3. In our last experiments 12,$13^{\dagger}$, we batch trained on all of our datasets at the same time.

4.4 Results

We report the result of our experiments in the table below, with each experiment numbered and labelled as to what kind of approach we took, as well as the accuracy for sentiment classification, paraphrase detection, Pearson correlation for semantic similarity, and finally the overall dev set score (or test set score for our test submission) if we decided to make a leaderboard submission.

Table 1: Results of Experiments

Experiment	Sentiment Classification	Paraphrase Detection	Semantic Similarity	Overall dev/test score
1. Baseline	0.208	0.375	-0.033	-
2. Pretrained HuggingFace model	0.316 (+0.108)	0.376 (+0.001)	0.019 (+0.052)	0.401
3. Finetune only on SST + Quora datasets	0.480 (+0.272)	0.525 (+0.15)	0.217 (+0.25)	-
4. Finetune on all 3 (SST + Quora + STS) datasets	0.469 (+ 0.261)	0.526 (+0.151)	0.534 (+0.567)	0.587
5. Finetune on all 3 datasets + CosineEmbedding Loss	0.440 (+0.232)	0.514 (+0.139)	0.429 (+0.462)	0.556
6. Finetune on all 3 datasets + SMART reg on sentiment	0.463 (+0.255)	0.534 (+0.159)	0.548 (+0.581)	0.590
7. Finetune on all 3 datasets + ordering* + SMART reg	0.500 (+0.292)	0.482 (+0.107)	0.558 (+0.591)	0.587
8. Finetune on all 3 datasets** + ordering*** + SMART reg + PCGrad	0.498 (+0.290)	0.524 (+0.149)	0.554 (+0.587)	0.599
9. Finetune on all 3 datasets + ordering*** + SMART reg + PCGrad + MSE Loss + Additive Loss	0.490 (+0.282)	0.486 (+0.111)	**0.622 (+0.655)**	0.595
10. Finetune on all 3 datasets + ordering*** + SMART reg + PCGrad + MSE Loss + 20 epochs	0.499 (+0.291)	0.533 (+0.158)	0.572 (+0.605)	-
11. Finetune on all 3 datasets + ordering*** + SMART reg + PCGrad + MSE Loss	0.516 (+0.308)	0.516 (+0.141)	0.587 (+0.620)	0.608
12. Finetune on all 3 datasets† + SMART reg + PCGrad + MSE Loss + 1 combined update per batch	0.494 (+0.286)	0.522 (+0.169)	**0.627 (+0.660)**	**0.610 (DEV)**
13. Finetune on all 3 datasets† + SMART reg + PCGrad + MSE Loss + 1 combined update per batch	0.500 (+0.292)	0.517 (+0.142)	0.555 (+0.588)	0.598 (TEST)
14. Finetune on all 3 datasets (FULL) + ordering*** + SMART reg + PCGrad + MSE Loss (**FINAL TEST submission**)	**0.521 (+0.313)**	**0.541 (+0.166)**	0.498 (+0.531)	0.604 (TEST)
Best Results	**0.521 (+0.313)**	**0.541 (+0.166)**	**0.627 (+0.660)**	**0.610**

In our best results, we were able to achieve an accuracy of 2.5 times the baseline for our sentiment classification task, about 1.44 times the baseline for the paraphrase detection task, and we had the greatest improvement on the semantic similarity task, where our Pearson correlation improved by a nearly 0.7 over a course of our 14 official experiments for our dev set. Ultimately, our overall accuracy for the test set was not very far off as well, sitting just a couple decimal points lower than our best overall dev accuracy.

We were surprised by the number of methods that didn't work as intended, including (1) CosineEmbeddingLoss, (2) Additive Loss, (3) increasing the number of epochs, and (4) training on the full Quora dataset. They all resulted in a lower accuracy/Pearson correlation score that resulted in us scrapping the method altogether.

We were not so suprised by the methods that did seem to work, such as (1) MSE loss, (2) SMART regularized optimization, and (3) PCGrad. However, they also didn't work as we expected them to. For example, for MSE loss (which was only applied to the semantic similarity task) and SMART regularized optimization (which was only applied to the sentiment classification task), we did not see immediate improvements in the specific tasks we applied the techniques on - instead, we saw improvements for the *other* tasks. However, down the road, we noticed that the specific technique did steadily help keep the score high for those respective tasks. In the case of MSE Loss, we saw an immediate boost in accuracy when accompanied with additive loss (a technique that did not ultimately improve our overall score and therefore abandoned).

5 Analysis

There are numerous qualitative evaluations that could be made on our model. We will address two specific points that we determined require the most attention: (1) Loss and Optimization, (2) Overfitting and SMART regularized optimization.

5.1 Loss and Optimization

After our experiments, it has come to our attention that our training framework may have been inefficient and ineffective in the way it is calculating the loss as well as the way it is optimizing in each batch iteration and epoch. This also relates to the way PCGrad is employed in our training process. We are specifically using two optimizers, a regular Adam Optimizer (one we implemented) that performs a step within each of the batch iterations for each task, as well as a second optimizer that is wrapped by the PCGrad implementation by Tseng (2020), which is used to project conflicting gradients at the very end of all individual processes for each task. The second optimizer performs one step per epoch, whereas the first optimizer performs a step per batch iteration for each of the three downstream tasks. We believe there may have been some conflicting actions between the two distinct optimizers, which could have caused some retrograde in the performance of BERTram.

What would have been more efficient and effective is to make use of additive loss in an accurate way - perhaps calculate the loss for each task exactly once altogether within each batch iteration, and then use *that* loss to back-propagate, then use PCGrad to perform an optimization step once for each batch iteration. This would have ensured that the model was, indeed, generalized for all tasks, whereas our current method technically still learns each task individually and then tries to even out the conflicting gradients far too late in the process. This could explain why the application of PCGradient was not as effective as we expected it to.

It is essential to note that we have already tried this technique during experiment numbers 12,13 of our trials, but it resulted in very little improvement on our overall dev accuracy (just +0.002 improvement from our best) and decreased our test accuracy. We believe that this particular model was not generalizing across all of the tasks enough, as it improved the last task (semantic similarity) heavily, but worsened the former two tasks (sentiment classification, paraphrase similarity). Hence, this is not the final model we decided on.

5.2 Overfitting and SMART regularized optimization

Right from the get go, we noticed that our model was overfitting extremely to our training dataset (some tasks reaching over 0.9 accuracy/Pearson correlation for training while the dev scores remained around 0.5). The fact that we had to reduce the number of training datasets used for paraphrase detection obviously did not help, as BERTram was exposed to lesser datapoints in the training process.

SMART regularized optimization was implemented on the sentiment classification task to assist with lessening the impact of overfitting. As seen in Table 1, when first implemented with a specfic ordering (datasets associated with least accurate tasks ordered last), SMART regularized optimization greatly improved the accuracy of sentiment classification. However, its effects seemed not last through the other experiments as the improvements plateaued around 0.5. There may be several explanations to this phenomenon.

First is directly related to the previous section 5.1. Because we were calculating the losses and back-propagating, performing an optimization step separately for each of the individual downstream tasks,

even if SMART regularized optimization did have a significant impact, if sentiment classification was not trained LAST, the effects of the technique would be dulled. Simply altering the orders in which the tasks were trained was most likely not enough to counteract the model acclimating to one specific task during the full batch iterations for that specific task. The best way to see the full effects of SMART, it would have been best to apply Smoothness-Inducing Adversarial Regularization on the additive loss of all three downstream tasks, and also continue to perform Bregman Proximal Point Optimization on this combined result.

Second is related to the hyperparameters of SMART. We utilized the hyperparameters used in the implementation detailed in the paper we referenced Jiang et al. (2020). Although the paper states that they "only observed slight differences in model performance when $\lambda_s \in [1, 10], \mu \in [1, 10], \epsilon \in [10^{-5}, 10^{-4}]$," it would not have hurt to try the different values for each hyperparameter. As long as they were within the bounds that the paper suggests. Because we noticed some extreme overfitting, perhaps λ_s could have been increased to $\lambda_s = 3, 5$ (which are also values that the paper mentions) to strengthen the effect of regularization. We anticipate that the increased size of perturbation would most likely decrease at least some of the overfitting we are currently observing.

6 Conclusion

The most significant finding from our explorations made us recall a very significant challenge that was posed years ago - the Netflix Prize, which was won by BellKor's Pragmatic Chaos that essentially gained huge improvements by combining numerous techniques in optimization, machine learning, and more - the discovery from our research was relevant to this even in that we found there was that no *one* extension or method could optimize the BERT model for multiple tasks. It was vital to use a number of different combination of extensions in order to achieve a high accuracy/correlation score across the board with all of the tasks in question. We also learned that some combinations of extensions to the model may have counteracting effects and therefore may not be as effective as other combinations of extensions. In our case specifically, we observed that the combination of finetuning with regularized optimization (SMART), gradient surgery (PCGrad), as well as certain settings of hyperparameters (mostly default values) led to promising increases in accuracy/correlation scores.

Our future work on this project may include expanding upon one of our last experiments (12, 13) which most efficiently performed the backpropagation/optimization step by doing all the datasets at once for each batch iteration. If we find a way to incorporate more datapoints and reduce the effect of overfitting even further, we expect to see greater improvements in our model with SMART regularized optimization and PCGrad. Additionally, while we were careful in applying each change in isolated and computable manners, with so many parameters and factors at play, it is difficult to determine which changes exactly contribute to which improvements. This is part of the challenge of multitask learning that many others (including future students of CS224N) will hopefully explore. As a part of our further investigations, we would have appreciated the opportunity to break down the effect of each of the changes we made, and how the positive changes can be amplified with some combination of techniques and how negative changes can be avoided by eluding specific combination of techniques.

Multitask learning is a field of study that is expected to have a huge impact on the way humans interact with LLMs. As NLP scholars continue their research, we hope to see further advancements in the interpretability and efficacy of multitasking BERT models like our very own BERTram.

References

Archinet.ai. 2022. archinetai/smart-pytorch.

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.

Mario Giulianelli, Marco Del Tredici, and Raquel Fernández. 2020. Analysing lexical semantic change with contextualised word representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3960 – 3973, Online. Association for Computational Linguistics.

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.

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

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