An Exploration of Fine-Tuning Techniques on minBERT Optimizations

Stanford CS224N Default Project

Gabriela Cortes
Department of Computer Science
Stanford University
gcrt0701@stanford.edu

Victoria Hsieh
Department of Computer Science
Stanford University
vhsieh02@stanford.edu

Iris Fu
Department of Computer Science
Stanford University
irisfu@stanford.edu

Abstract

The main objective of this paper is to present a study of the minBERT model to optimize it for sentiment classification, paraphrase detection, and semantic textual similarity. We introduce extensions such as multitask fine-tuning, cosine similarity, and SMART to optimize the model's performance. We also look into the impact of parameters such as number of iterations and the effect of different pooling strategies on each of our NLP tasks. To evaluate this, we look for the model iterations that produce the highest accuracies across all three tasks. We find that increased iterations upon our extensions of gradient surgery, cosine similarity, and SMART produce the highest accuracies for each task. We hope our paper contributes to a more efficient and accurate understanding of applications that rely on NLP tasks because our model enhances the performance of sentence embeddings in the three NLP tasks mentioned.

1 Key Information to include

- Mentor: Arvind Venkat Mahankali
- No External Collaborators
- No Shared Project

2 Introduction

The advancement of natural language processing (NLP) has been significantly shaped by the advent of advanced pre-trained language models like BERT, which have shown impressive outcomes across various NLP tasks. Before this, linguistics and/or statistics-based methods served as NLP baselines but performed poorly on tasks such as identifying synonyms. However, these models often face challenges in adapting to different tasks due to their design for specific tasks, limiting their broad applicability. The original BERT model consists of around 110 million parameters and training data comprising over 2.5 billion words from Wikipedia and 800 million words from the Toronto BookCorpus. Training a model that large is resource-intensive and demands substantial computational power. Although it focuses on simple language understanding goals like next-sentence prediction and language modeling, its architecture has the potential to excel in a broad range of language understanding tasks, as evidenced by its performance on the GLUE dataset. Recent efforts have focused on how to adapt this foundational model to perform well on new tasks by employing optimization methods.

To address this issue, multitask learning has become an effective strategy, enabling the training of models on various tasks at once. This approach enhances knowledge sharing among tasks, leading to better efficiency, generalization, and performance overall. This is especially applicable in our BERT as it sets out to accomplish three tasks: sentiment classification, paraphrase detection, and semantic textual similarity.

Fine-tuning the BERT model enables a more efficient resolution of NLP tasks despite little pre-training. Since the introduction of BERT, there has been a wave of innovative methods and improvements aimed at this objective. Key developments include integrating domain-specific text collections into pre-existing language models for further refinement, and advancements in multitask learning, which entails refining a language model across several related tasks at once. The techniques for fine-tuning we explore in this paper are 1) multitask learning 2) cosine similarity and 3) SMART along with variations such as gradient surgery, feature count, and number of iterations per epoch.

3 Related Work

The foundation of our work is based on the original BERT implementation by Devlin et al. (2019) and the Transformer architecture developed by Ashish Vaswani and Polosukhin. (2017). Devlin et al. (2019) introduces BERT as a pre-trained language model consisting of three main parts: sentence conversion, the embedding layer, and the transformer layer. In particular, the use of self-attention in BERT differentiates itself from previous models because each word's importance is learned from the complete context of the sentence. This model can improve on various NLP tasks like the three addressed in this paper: sentiment classification, paraphrase detection, and semantic textual similarity. This paper employs minBERT, a smaller version of this model, and experiments with multiple methods of fine-tuning to improve upon it.

For multitask learning, we reference Qiwei Bi and Yang. (2022)and how they employ a general method of multitask fine-tuning which involves adding each task's loss into a cumulative loss to be optimized with gradient descent. However, this fine-tuning may not be beneficial if gradients conflict and could potentially remove learning progress from pre-training. Thus, we also look into Yu et al. (2020)'s method of gradient surgery which resolves conflicting gradients using projections.

Reimers and Gurevych (2019) introduces an adaptation of the BERT framework by incorporating a Siamese network structure to efficiently derive sentence embeddings titled SBERT. SBERT can be used for sentence pair tasks like paraphrasing and semantic textual similarity. The generated embeddings can be directly compared using cosine similarity which reduces the computational costs of the original BERT model significantly for large datasets. The paper also experiments with different pooling strategies: the output of the CLS-token, computing the mean of all output vectors, and computing a max of all output vectors over time.

When adopting models for additional downstream tasks, there is a danger of aggressive fine-tuning which can result in overfitting on these models. Jiang et al. (2019) introduces a regularization technique titled Smoothness-Inducing Adversarial Regularization, or SMART, to manage the complexity of the model and their work saw large improvements in benchmarks such as GLUE, SNLI, SciTail, and ANLI. This regularization technique introduces small modifications to the input while maintaining small changes in the output. We aim to extend this work by applying SMART to the three tasks discussed in this paper.

4 Approach

4.1 Baseline Model

The main architecture of our model is MinBERT where we use 12 Transformer Encoder layers like the original BERT model (Devlin et al., 2019). Our model utilizes a tokenizer to break down sentences into individual words and then into individual word parts which is finally converted into word tokens to be used by our model. Each of our twelve layers consists of a multi-head attention, followed by an additive and normalization layer with a residual connection, a feed-forward layer, and a final additive and normalization layer with a residual connection. Unlike BERT, our minBERT does not use segmentation embeddings as only single sentence tasks are performed.

The optimizer function we utilize is the AdamW Optimizer by Kingma and Ba (2014). The optimizer updates the moment estimates, applies bias correction to these steps and finishes off parameter updates by using weight decay. Finally, we implement a multitask classifier to train the model on all three tasks in parallel.

4.2 Multitask Fine-tuning

To fine-tune our model on all three tasks simultaneously, we sum our losses from each task and proceed to train on the combined loss in accordance with the equation from Qiwei Bi and Yang. (2022):

$$L_{total} = L_{sentiment} + L_{paraphrase} + L_{similarity}$$

Our experiments differ from the original paper by using three different datasets for each of our tasks rather than the same dataset across classification and named entity recognition.

Using multitask learning comes with the risk of losing information during updates due to conflicting gradients between tasks. Thus, we use the gradient surgery method, or PCGrad, to account for this (Yu et al., 2020). This approach projects the gradient of the i-th task onto the normal plane of another conflicting task's gradient. In our implementation, we identify whether gradients are conflicting by checking for a negative cosine similarity and update it accordingly:

$$g_i = g_i - \frac{g_i \cdot g_j}{||g_j||^2} g_j$$

4.3 Cosine Similarity

With a focus on improving the Pearson correlation of our semantic textual similarity task, we explore many variations of cosine similarity. Our first approach utilizes Cosine Embeddings loss on the BERT embeddings of the two sentences and the 2.5-threshold-adjusted label to determine whether sentences are similar or not. Separately, we also conduct experiments with our own processing of the BERT embeddings using cosine similarity and Mean Square Error loss. We run three variations with different numbers of features and pooling strategies.

4.4 SMART Loss

We also introduce the regularization technique SMART (Jiang et al., 2019) to manage the potential overfitting of our model. The smoothness-inducing regularization adds an extra regularization term after the loss function. This formulation utilizes the KL divergence as the regularization term in the optimization equation to control the complexity by using a smoothness-inducing adversarial regularizer:

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

where the regularization loss function is defined as

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

We use the first embedding layer of the input sentence as x_i's and the associated labels as y_i's.

To implement SMART, we utilize the framework of a preexisting library (Jiang et al., 2020), where we set the parameters as $\lambda = 5$, $\eta = 10^{-3}$, $\varepsilon = 10^{-5}$, and $\sigma = 10^{-5}$.

Finally, following what we use in our baseline model, we employ cross-entropy loss for sentiment classification, binary cross-entropy loss for paraphrase detection, and mean squared error loss for semantic textual similarity, and these losses are appended to by the regularization loss. These updated losses for each task are then combined to calculate the total training loss which we seek to minimize.

4.5 Increased Iterations

As a final improvement on our best-performing experiments, we increase the number of iterations per epoch so that we are no longer capped by the smallest dataset. To optimize for time constraints,

we choose to triple the number of iterations. With this approach, the SST dataset is repeated approximately twice and STS is repeated exactly thrice per epoch. Since QQP is over 20x larger than our smallest dataset, it sees exclusively new data points on each iteration within an epoch.

5 Experiments

5.1 Data

All of our experiments are trained and evaluated on the same three datasets. For the sentiment analysis task, we use the Stanford Sentiment Treebank (SST) which contains 215,154 unique phrases extracted from movie reviews annotated by three human judges into negative, somewhat negative, neutral, somewhat positive, or positive classifications. For paraphrase detection, we utilize the Quora Question Pairs (QQP) dataset of 400,000 question pairs with binary labels indicating whether particular instances are paraphrases of one another. For semantic textual analysis, we use the SemEval STS Benchmark Dataset (STS) dataset which consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning).

5.2 Evaluation method

We evaluate accuracy for the tasks of sentiment analysis and paraphrase detection because they are both classification problems. For the STS task, we use Pearson correlation because it is a regression problem based on the original SemEval paper. We then find an overall evaluation of the model on all three tasks by normalizing the Pearson correlation and finding the average of these three normalized metrics.

5.3 Experimental details

We run a series of experiments guided by the approach detailed in Section 3. All experiments are run with a learning rate of 1e-5 over ten epochs using 768-dimensional BERT sentence embeddings. The models each include a linear layer for each task, of which the semantic and paraphrase classification layers remain the same with a hidden size of 768 and 1536, respectively. Their loss functions also remain fixed, with Cross Entropy Loss for semantic classification and Binary Cross Entropy Loss for paraphrase classification. For semantic textual similarity, we vary the loss function depending on the experiment. The rest of the details specific to each experiment are summarized below.

5.3.1 Multitask Baseline (Score: 0.641)

Our multitask baseline implementation runs 755 iterations per epoch, during which each task is trained individually, but in parallel. For the semantic textual similarity task, we use a linear layer with a hidden size of 1536 and Mean Square Error Loss.

5.3.2 Extension 1: Multitask Learning

For all experiments under this extension, the number of iterations per epoch and loss functions for each respective task remain the same as the implementation in Multitask Baseline.

1a. Multitask Learning (Score: 0.636)

First, we implement multitask learning where the model's parameters are updated based on the combined loss of the tasks rather than their individual losses.

1b. Gradient Surgery (Score: 0.639)

After seeing that multitask learning performed worse than our baseline, we hypothesize that we may be losing information due to conflicting gradients. We fully implement PCGrad on our own (without external libraries) and update the model using the adjusted gradients.

1c. Gradient Surgery Experiment (Score: 0.644)

As an experiment, we create our unique implementation of PCGrad that performs two updates per epoch based on both the adjusted combined loss and the original combined loss. We first update our

parameters using the mean of the adjusted gradients for each loss, and immediately after, re-update them using the combined loss.

5.3.3 Extension 2: Cosine Similarity (on Gradient Surgery Experiment)

Seeing that our gradient surgery experiment (GSE) performs best in Extension 1, we build all experiments in Extension 2 off of GSE. However, we no longer use a consistent linear layer for the semantic textual similarity task.

2a. Cosine Embeddings Loss (Score: 0.589)

Our first approach leverages Cosine Embeddings Loss on the BERT embeddings of the two sentences and the 2.5-threshold-adjusted label to determine whether sentences are similar or not. We do not use a linear layer for this experiment since we would need to apply it on the per-sentence BERT embeddings and do not expect it to learn any particularly unique information.

2bi. Cosine Similarity with d features (Score: 0.610)

In response to the large decrease in our score in comparison to the baseline, we decide to use cosine similarity directly. In addition, we reintroduce the linear layer for semantic textual similarity. For this experiment, we fetch the per-sentence BERT embeddings, apply a d-hidden-size linear layer on each embedding (where d = 768), and then calculate the cosine similarity between the new embeddings. We multiply this score by 20 (a large scalar), then run sigmoid over this score, and scale it by 5, thereby matching the 0-5 scale followed by the semantic textual similarity data and allowing us to use Mean Square Error Loss.

2bii. Cosine Similarity with 2d + 1 features (Score: 0.643)

Another approach we consider is to introduce dropout and extend our linear layer to compute a logit per sentence pair with which to calculate the Mean Square Error loss. To do this, we compute the cosine similarity using the per-sentence BERT embeddings, concatenate the embeddings with the similarity score, perform dropout, and feed it into a (2d+1)-hidden-size layer (where 2d+1 = 1537).

2c. Cosine Similarity with 2d + 1 features with mean pooling (Score: 0.641)

To build off our best cosine similarity experiment, we explore different pooling strategies beyond the default [CLS] token one. We choose to implement mean pooling to provide a smoother representation of each sentence. To do this, we multiply the expanded attention mask to zero-out padding token embeddings before summing, and then we divide by the number of non-padding tokens to calculate the mean.

5.3.4 Extension 3: SMART Loss

After plotting training accuracies for each of our previous experiments, we notice that SST and STS are overfitting. Therefore, we perform SMART loss as a regularization technique.

3a. Smart Loss (Score: 0.642)

To each individual task's loss, we add a regularization loss that is computed based on the SMART framework.

3b. SMART Loss w GSE (Score: N/A)

GSE remains our top-performing extension, but it shows signs of overfitting for SST and STS. Therefore, we implement SMART on top of GSE. In running this experiment, we find that the PyTorch CUDA runs out of memory, implying the memory-intensive nature of this experiment.

5.3.5 Extension 4: More Iterations per Epoch

For our last extension, we take the three best-performing experiments so far and run three times more iterations per epoch on each: Gradient Surgery Experiment (Score: 0.644), Cosine Similarity 2d+1 (Score: 0.650), and Smart Loss (Score: 0.651), noticing score improvements.

5.4 Results

Table 1: Dev Leaderboard Results

Model	SST	QQP	STS	Overall
Baseline	0.512	0.731	0.361	0.641
Multitask Learning	0.511	0.727	0.341	0.636
Multitask Learning w/ GS	0.510	0.718	0.378	0.639
Multitask Learning w/ GSE	0.507	0.748	0.353	0.644
Cosine Embeddings Loss	0.510	0.706	0.103	0.589
Cosine Similarity (d features)	0.508	0.730	0.185	0.610
Cosine Similarity (2d+1 features)	0.510	0.737	0.365	0.643
Cosine Similarity (2d+1 features) + Mean Pooling	0.499	0.747	0.357	0.641
SMART Loss	0.487	0.750	0.376	0.642
SMART Loss w/ GSE	N/A	N/A	N/A	N/A
GSE w/ 3x Iterations	0.489	0.762	0.362	0.644
Cosine Similarity (2d+1 features) w/ 3x Iterations	0.499	0.761	0.379	0.650
SMART Loss w/ 3x Iterations	0.507	0.758	0.376	0.651

We submit our three best-performing models on the dev datasets to run on the test datasets and obtain the following results:

Table 2: Test Leaderboard Results

Model	SST	QQP	STS	Overall
GSE w/ 3x Iterations	0.524	0.768	0.329	0.652
Cosine Similarity (2d+1 features) w/ 3x Iterations	0.516	0.761	0.327	0.647
SMART Loss w/ 3x Iterations	0.514	0.760	0.345	0.649

The tables above reports our results for the set of experiments run on the dev and test dataset. GSE represents our gradient surgery experiment and GS represents gradient surgery.

For implementing cosine similarity, the usage of MSE greatly improved performance compared to the usage of CosineEmbeddingLoss. An increase in features also generated better accuracy scores, especially for STS. The default use of d features on Cosine Similarity with GSE resulted in an accuracy of 0.185 while the use of $2d + 1$ features resulted in an accuracy of 0.365. Because of this, we chose to run our increased iterations experiment on the Cosine Similarity with $2d + 1$ features.

We also recognized that our use of data in earlier experiments only explored a limited number of data points on the STS dataset. In order to expand this, we chose to run three times more iterations and our results demonstrate that increased iterations on our extensions improves the performance of all three natural language processing tasks compared less iterations. The increased iterations produced our highest accuracy scores yet.

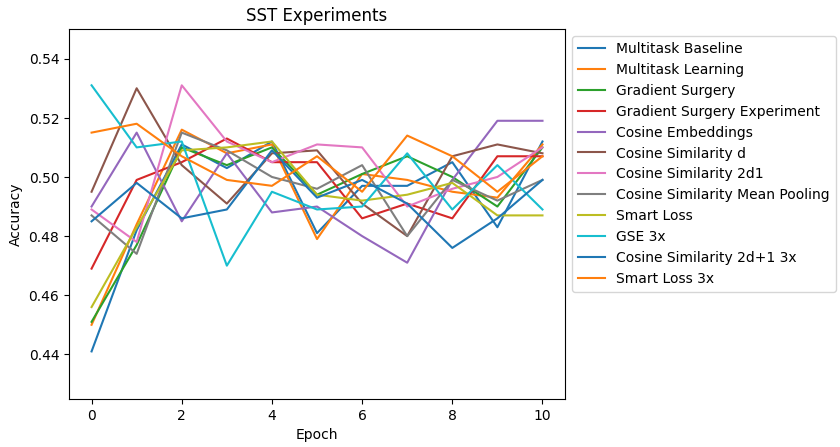

Figure 1: SST Experiments

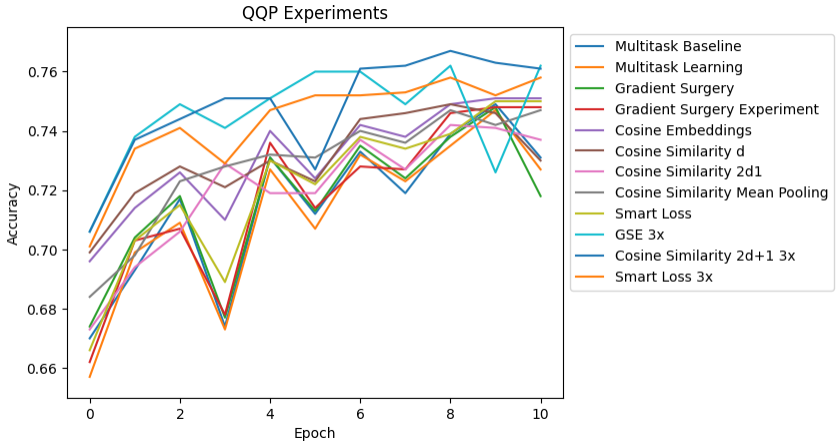

Figure 2: QQP Experiments

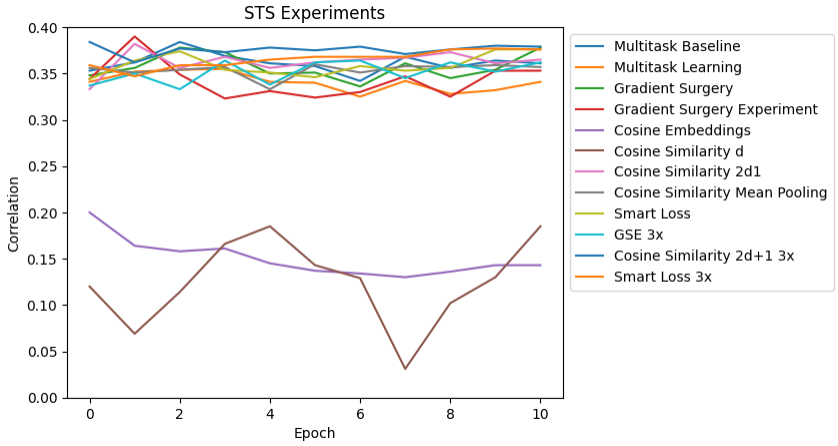

Figure 3: STS Experiments

6 Analysis

We notice that multitask learning does not necessarily improve upon the baseline but rather worsens the performance for each task, likely due to conflicting gradients. We find that our experimental gradient surgery technique performs better than raw gradient surgery. One possible theory as to why this is so is because it reaches a happy medium where it tempers the effects of conflicting gradients through the gradient surgery update, but also retains the full information of the combined gradients in the second update of the same epoch.

For our cosine similarity extension, we see that our cosine similarity with $2d + 1$ features model vastly outperforms Cosine Embeddings Loss and our d-features implementation, scoring the best on overall score and our focus task (semantic textual similarity) out of all experiments in this section. Cosine Embeddings Loss likely performs the worst due to its lack of a linear layer as well as the loss of information from the need to generalize labels to similar or dissimilar classifications based on a 2.5 threshold in order to use the provided function. In our d-features experiment, the linear layer is likely not learning any useful information since it was applied directly on top of the BERT embeddings. This could also explain why the loss over epochs fluctuated—wavering around 10—rather than steadily declining into a plateau near 0, like our other experiments. In our $2d + 1$ implementation, on the other hand, our model's semantic textual similarity linear layer manages to learn more information about the relationship between the sentence pairs with the help of their cosine similarity score. As for mean pooling, we hypothesize that averaging each output's vectors, which treats each token vector

equally, can lead to a loss of information as it could dilute the significance of more critical tokens. This is likely why adding mean pooling hurt, rather than helped, the performance of cosine similarity.

SMART improves scores for paraphrase detection and semantic textual similarity from baseline but harms the score for sentiment classification. While we intend for SMART to reduce overfitting, our results show that improvements still need to be made. As we can see in the following graph, train accuracy on SST and STS approach 1.0 after epoch 8, meaning that the model is still overfitting to the training data.

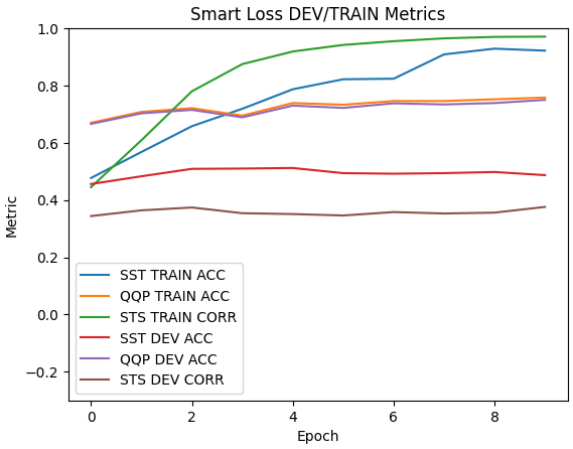

Figure 4: SMART Loss Dev/Train Metrics.

Our overall best-performing model is 3x iterations on our gradient surgery experiment. This model performs best on paraphrase detection, followed by sentiment classification, and then semantic textual similarity. We hypothesize that the two tasks associated with detecting meaning (sentiment and semantics) are the most difficult to encode with BERT, as a word can often mean different things in different contexts, and therefore requires contextual data and knowledge often beyond the scope of BERT's exposure. The paraphrase detection task sees the greatest improvement likely because it is training on 1500 new data points, while sentiment analysis and semantic textual similarity both repeat data points 2-3 times per epoch. Both of these tasks may be at risk of overfitting due to cumulatively training on the same data points 20 to 30 times. However, seeing that paraphrase detection and semantic textual similarity are conceptually similar tasks in that they are trying to identify similar sentences, it is possible that semantic textual similarity also benefits from the new information being learned by paraphrase detection, while semantic analysis is left with nothing to counteract its overfitting.

7 Conclusion

In this project, we propose multiple approaches to refining and optimizing the original BERT model and improve upon the baseline for all three tasks: sentiment classification, paraphrase detection, and semantic text similarity. We achieve a final best accuracy score of 0.652 on the test dataset with 3x iterations on the GSE model.

In the future, we would be interested to try a higher number of iterations on our extensions to check if accuracies further improve. Furthermore, while SMART assists with the complexity of the model, adding on Bregman proximal point representation could help with aggressive updating prevention and reduce overfitting. We would also like to obtain results made by layering GSE with SMART, so further interest lies in exploring ways to reduce the amount of memory used by this layering.

Our accuracy scores remain relatively similar across extensions which are all primarily focused on fine tuning. We want to explore pretraining in future avenues of work and see if it could further improve our current performance.

References

Niki Parmar Jakob Uszkoreit Llion Jones Aidan N Gomez Łukasz Kaiser Ashish Vaswani, Noam Shazeer and Illia Polosukhin. 2017. Attention is all you need.

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

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization.

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 *ACL*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Lifeng Shang Xin Jiang Qun Liu Qiwei Bi, Jian Li and Hanfang Yang. 2022. Mtrec: Multi-task learning over bert for news recommendation.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 5824–5836.