Fine-Tuning BERT for Sentiment Analysis, Paraphrase Detection and Semantic Text Similarity NLP Tasks

Stanford CS224N Default Final Project

Andrew Cheng
Stanford University
ac1792@stanford.edu

Swathi Gangaraju
Stanford University
swathig1@stanford.edu

External collaborators: No, External mentor: No, Sharing project: No.

Abstract

In this default final project, we train BERT model to perform well on Sentiment Analysis, Paraphrase Detection, and Semantic Text Similarity tasks. Furthermore, we implement several core components of BERT and fine-tune a pre-trained BERT to perform well on the three aforementioned NLP tasks. In our experiments, we demonstrate advantages of first further pre-training on target data, followed by targeted fine-tuning on individual tasks while keeping the BERT layers frozen, and lastly end to end multi-task fine-tuning of the whole network. We find that this strategy, coupled with a scheduled freezing of BERT layers, gradient surgery, and other extensions, lead to the best evaluation scores. We provide a general multi-task learning framework and provide several ablation and comparative experiments to examine the effectiveness of our approaches. We conclude that our general fine-tuning strategy is effective, but that gradient surgery does not help in most cases if a sufficiently low learning rate is used.

1 Introduction

Our goal is to implement and perform multi-task fine-tuning using a pre-trained BERT model to perform well on 3 downstream tasks: sentiment analysis, paraphrase detection, and semantic text similarity. Due to the relatively distinctiveness of these three tasks, the challenge is to use creative extensions to both enrich the underlying BERT embeddings and train robust task-specific head layers. While [1] and [2] explore multi-task learning as a way to simultaneously do well on multiple tasks, the varying degree of difficulty of each task in our problem and the differences in dataset sizes means that the model needs to work harder learning the difficult task compared to the others. As a result, naive multi-task learning may not be appropriate for this specific context. Motivated by the "How to Fine-Tune BERT for Text Classification?" paper [1], we specifically aim to explore both further pre-training using the masked language model objective [3] and the multi-task fine-tuning method mentioned in the paper, with our own modifications. We introduce a robust combined pre-training and multi-task learning strategy utilizing a weighted sum of individual task-specific losses. For our main approach we initially generated embeddings of input data using original BERT architecture and Base-BERT pre-trained weights. We then further pre-trained the model using our data and implementing MLM from scratch. Then the model was fine-tuned on each task separately keeping rest of the model layers frozen. These weights are used to further fine-tune multitask model to train on all tasks together. In the end, we unfroze all BERT layers and fine-tuned the model using multitask loss. Our results show there is 20% improvement in performance when using further pre-training. Model is performing best on paraphrase task compared to others. We addressed over-fitting for Sentiment Analysis and Semantic Text Similarity tasks which we countered by L2 regularization, gradient surgery and dropout. We also observed more even performance between tasks with Weighted Random Sampling than without. But overall accuracy was slightly less when we applied Weighted Random Sampling than

without all other conditions remaining same. Overall this project's contribution is implementation and analysis of multiple strategies to further pre-train and fine-tune BERT to perform well on 3 given NLP tasks.

2 Related Work

As the task is to create a model that does well on multiple NLP tasks, we researched prior work on multi-task BERT implementations including the suggested readings in the project handout. In the original Bidirectional Encoder Representations for Transformers (BERT) model [3], Devlin et. al. propose a new pre-training approach and model that is designed for natural language processing (NLP) tasks. The key innovation of the BERT model is the use of a bidirectional Transformer architecture, which allows the model to capture dependencies between the left and right context of a given word in a sentence using the masked language model (MLM) and next sentence prediction (NSP) objectives for pre-training. We utilize the same MLM objective to perform further pre-training on target data.

To understand approaches to implement multi-task BERT model, we researched the How to Fine-Tune BERT for Text Classification? paper by Chi Sun et. al. [1]. In relation to multi-task learning, the authors cite [4]'s success in simultaneously training a language model and a task-specific model with a multi-task strategy. Inspired by [2]'s MTL model, a multi-task deep learning network which uses BERT as text encoding layers, Sun et. al. mainly focus on tackling three text classification tasks: sentiment analysis, question classification, and topic classification. In their paper, they use three main strategies to analyze BERT: further pre-training BERT using in-domain or within-tasks datasets, utilizing multi-task fine-tuning, and directly fine-tuning BERT for target specific tasks. While their work achieves state-of-art-performance for these tasks on seven English and one Chinese datasets, they do not explore any extensions to vanilla multi-task fine-tuning such as task weighting, gradient weighting, etc.

Sun et. al. also implement layer-specific learning rates with the assumption that model learns generic relationships in lower layers and more granular language understanding in higher layers. However, setting layer-specific learning rates for each transformer layer in BERT is a tedious process that adds more hyperparameters and increases the combinatorial hyperparameter search space. As a result, we explore an alternative approach to leveraging the layer-specific differences: freezing the earlier layers that do not contribute to higher language understanding during multi-task fine-tuning. Thus, we leverage the differences in representation in the BERT layers to maximize downstream task performance while only controlling the number of BERT layers to freeze while multi-task fine-tuning.

We studied "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations" paper by Zhenzhong Lan et al. [5]. This paper proposes a more efficient variant of BERT called ALBERT. ALBERT uses a parameter-sharing technique to reduce the number of model parameters, making it easier to fine-tune on small datasets. The main drawback of this method is that it sacrifices some performance compared to the original BERT model in order to reduce its size and computational cost. One clever extension to improve performance of multi-task learning is to apply gradient surgery, as proposed by Tianhe Yu et al. [6]. In this work, the authors identify a set of three conditions of the multi-task optimization landscape that cause detrimental gradient interference and propose a form of gradient surgery that projects a task's gradient onto the normal plane of the gradient of any other task that has a conflicting gradient. We implement gradient surgery and perform some experiments to analyze the effect of this on end to end multi-task fine-tuning.

We also reviewed Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks paper by Nils Reimers et al. [7]. In this paper, they use Sentence-BERT to fine-tune the pre-trained BERT model on a sentence-pair classification task. The fine-tuning process is carried out using a contrastive loss function that encourages similar sentences to have similar embeddings and dissimilar sentences to have dissimilar embeddings. They use cosine similarity to calculate degree of similarity and use Triplet Loss function during training. None of the research we reviewed specifically worked on this combination of NLP tasks for multi-task training using BERT architecture. Nor have we seen impressive performance when BERT is trained on small dataset and does well on multiple tasks. Given the prior work and research findings, our implementation is to use further pre-training and fine-tuning for model to do well on small datasets like one indicated in the Default project.

3 Approach

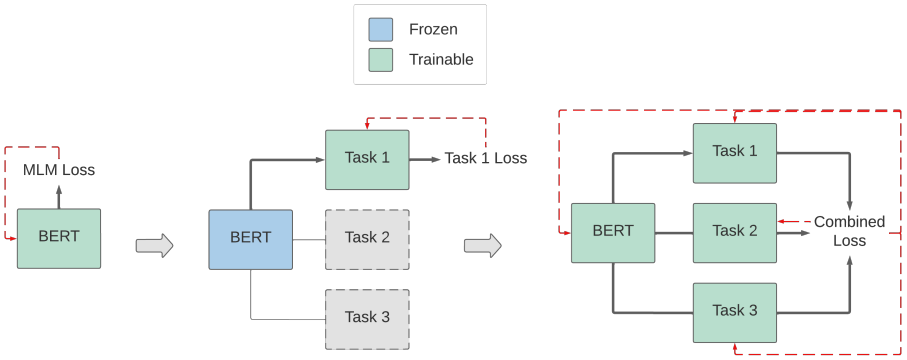

Figure 1: (Left to Right) Further pre-training, targeted fine-tuning for each task, followed by full end to end multi-task fine-tuning of the entire network. Modules with trainable parameters are shaded in green, while frozen modules are shaded in blue. Dashed red lines represent the flow of gradients during the backward pass.

Our overall strategy can be summarized by Figure 1. We initially pre-train sequentially on each task-specific dataset using the masked language modeling (MLM) objective. We then freeze the BERT layers that were further pre-trained on the target data to separately fine-tune each task head. The task head weights are then saved and all weights are loaded and integrated into an end to end model that is trained with multi-task learning. Stage 1 adapts the transformer layers to the target data distribution. Stage 2 allows for greedily finding the best task-specific weights that are compatible with the frozen BERT layers from Stage 1. Stage 3 is used to tie everything together so that all layers can be re-updated simultaneously to satisfy all tasks.

3.1 Further pre-training

Sun et. al. demonstrate further pre-training on target data can be advantageous [1], especially when the pre-training is done on in-domain data. Since the encodings produced by BERT are designed to give rich bi-directional contextual information to downstream task-specific layers [3], it is critical that this contextual information accurately represents the information contained in each task-related dataset. Otherwise, the encodings will be less useful and task performance will suffer.

Following the steps of [3], we randomly mask 15% of the tokens with the "[MASK]" token, feed the masked tokens into BERT, then use an additional linear layer followed by a softmax function to produce a distribution over the vocabulary for each masked token. The model then predicts the masked word token id. Furthermore, we follow the original steps of only masking 80% of the time, replacing the token to mask with a random token from the vocabulary 10% of the time, and doing nothing the other 10% of the time [3]. We implement this MLM objective code from **scratch** by defining a custom collate function and further pre-training the given BERT model on the datasets associated with each task.

3.2 Targeted fine-tuning

One of the most challenging aspects of multi-task learning in this context is that the BERT layer parameters will typically be updated by multiple tasks since the gradient of each task-specific loss with respect to the BERT layer parameters will exist if the BERT layers are trainable.
One alternative would be to separately fine-tune a different model composed of BERT + task-specific head for each task. However, doing so will defeat the purpose of obtaining shared generalized representations across tasks through multi-task learning and will also be computationally demanding.

To establish a compromise on these two extremes, we propose to freeze the BERT layers containing the further pre-trained weights and train the parameters of each task head separately for this stage. Since the BERT layers are frozen, each task head can be trained with a "greedy" approach, greatly

simplifying and eliminating the risk of over-updating the BERT layers. With this approach, each task head is tied to a single loss, and the weights of each task head can be separately tuned and saved to be loaded in the third stage of Figure 1. In the third stage, we can load the BERT weights that were further pre-trained in stage 1, load the task-specific weights that were greedily updated in stage 2, and unfreeze the whole network to then perform multi-task fine-tuning. The advantage of this is now the weights of each task head are already starting off at good values, and can be integrated with the now unfrozen BERT layers to do traditional multi-task learning, where the now unfrozen BERT layer parameters can be tweaked to satisfy all tasks simultaneously. Stage 2 allows for minimal multi-task fine-tuning steps, since task-head weights are already starting off at good values. To the best of our knowledge, this specific approach is original in the scope of this course.

3.3 Weighted Multi-Task learning

We simultaneously fine-tune a BERT model with pretrained weights on all three downstream tasks using different weights on the sub-task losses as follows (**Original**):

$$\mathcal{L}_{multi} = \lambda_1 \mathcal{L}_{SST} + \lambda_2 \mathcal{L}_{Para} + \lambda_3 \mathcal{L}_{STS} \tag{1}$$

where $\lambda_1, \lambda_2, \lambda_3$ are hyperparameters representing the weight on the subloss for sentiment analysis (SST), paraphrase detection (PARA), and semantic text similarity tasks (STS), respectively. The benefit of this modified objective is that if a separate optimizer is used for the unfrozen BERT layers in stage 3, then we can increase the learning rate of this optimizer while weighting the losses so that we can finely control which task contributes the most to the update of the BERT layers. For example, if we observe that the STS task is heavily dependent on the BERT layers, we can increase λ_3 and decrease λ_1 and λ_2 so that the BERT layers are updated more in favor of the STS task.

3.4 Freezing of BERT Layers

While [1] vary the learning rate per layer, we hypothesize that lowering the number of trainable BERT layers when fine-tuning with smaller datasets will reduce over-fitting, since less parameters are considered during optimization. Since many epochs are needed for multi-task learning to generate embeddings generalized for all tasks, the risk of over-fine-tuning increases. Thus, we choose to dynamically change the number of frozen BERT layers when performing the multi-task fine-tuning step (**Original**).

3.5 Task Specific Classifier Layer Networks

For each task we have task specific networks. We used two blocks of fully connected layers, followed by layer norm, RELU and dropout layer. In first Fully Connected Layer we take output of BERT and map to 512 hidden size. Second fully connected layer is to take output of first block to generate output of 64 hidden size. to reduce the effect of the internal covariate shift, which is the phenomenon of the distribution of the inputs to a layer changing as the model parameters change. RELU layers are to introduce non-linearity for model to learn complex representation. Dropout layers is to prevent over-fitting. Finally an output linear layer to convert output from second block to and map it to the number of output classes required for the specific task.

3.6 Strategy to manage paraphrase dataset imbalance

Due to imbalance in paraphrase detection task, model can predict label of 0 for all examples and get 63% accuracy as the Quora dataset used in the project has 63% non-duplicate inputs. To correct model behavior to predict non-trivial label, we implemented `class_weight` following[8] and passed it to the loss function to ensure the model is penalized more for predicting the minority class incorrectly:

$$|class_weight| = \frac{\#of inputs with 0 label}{\#of inputs with 1 label} \tag{2}$$

3.7 Cosine Similarity for Semantic Text Similarity task

We used cosine similarity to calculate similarity between two sentences in the semantic text similarity task head. This acts as effective feature extraction step before we input the data to the final layers,

since cosine similarity is effective in discriminating similar sentences from dissimilar sentence embeddings.

4 Experiments

4.1 Data and Tasks

We used the 4 datasets provided by the CS224N default project handout, as well as its pre-processing steps. The datasets follow those listed in the handout (see Table 1), and task-specific information is omitted for redundancy (please refer to default project handout).

Task	Name	# Train	# Dev	# Test
Sentiment Analysis	Stanford Sentiment Treebank [9] (SST)	8,544	1,101	2,210
Sentiment Analysis	CFIMDB	1,701	245	488
Paraphrase Detection	Quora (PARA)	141,506	20,215	40,431
Semantic Text Similarity	SemEval (STS) [10]	6,041	864	1,726

Table 1: Summary of default datasets and splits.

4.2 Evaluation method

Following the default project handout, the evaluation metrics we will be using for each of the tasks are:

1. Sentiment Analysis: Accuracy (Number of examples with correctly predicted sentiment/total number of examples)
2. Paraphrase Detection: Accuracy (Number of examples with correctly predicted paraphrases /total number of examples)
3. Semantic Text Similarity: Pearson coefficient = cov(X, Y) / ((standard_deviation(X) * standard_deviation(Y)) where X is true similarity score and Y is predicted similarity score

4.3 Experimental details

For our baseline, we performed multi-task learning for 10 epochs using a learning rate of 1e-05, a batch size of 32, and a dropout probability of 0.0. We used a loss weight ratio of 1:1:1 for SST:PARA:STS. We used single linear layer as the final output layer for each task head. We also experimented with and without weighted random sampling for above config. We ran multi-task end to end learning for 15 and 40 epochs to observe where the learning is slowing down. We trained the model on Google Colab. Each epoch took 5-10 mins and consumed 16-17 GB of GPU RAM space for these hyperparameter settings running with a single Tesla A100 40GB GPU.

4.4 Results

Test Leaderboard scores our model achieved on 3 tasks are:
SST Accuracy: 0.485, Paraphrase Accuracy: 0.739, STS Correlation: 0.781 and Overall score: 0.668
Dev Leaderboard scores our model achieved on 3 tasks are:
SST Accuracy: 0.468, Paraphrase Accuracy: 0.743, STS Correlation: 0.787 and Overall score: 0.666
Our assumption was further pre-training should result in better embeddings and aid in overall model performance. Using our approach of further pre-training and fine-tuning individual tasks separately and combined multi-task fine-tuning we achieved much higher score than using pretrained BERT weights and using multi-task fine-tuning as we expected. While we addressed some of the over-fitting in SST and STS tasks using dropout for both tasks and L2 for STS tasks, over-fitting to SST is still observed.

Additionally, our BERT+Adam implementation, we get SST pretrain dev accuracy: 0.388, SST finetune dev accuracy: 0.513, CFIMDB pretrain dev accuracy: 0.718, and CFIMDB finetune dev accuracy: 0.967.

Experiment	Epochs	SST Acc	Para Acc	STS Corr	Split
BERT-Base	5	0.152	0.630	0.216	Train
BERT-FPT	5	0.241	0.607	0.548	Train
BERT-FPT-FT	5	0.405	0.720	0.535	Train
BERT-FPT-FT-Multi	5	0.569	0.726	0.771	Train
BERT-FPT-FT-Multi	15	0.642	0.749	0.930	Train
BERT-FPT-FT-Multi	40	0.899	0.776	0.927	Train
BERT-Base	5	0.155	0.624	0.245	Dev
BERT-FPT	5	0.247	0.599	0.529	Dev
BERT-FPT-FT	5	0.395	0.716	0.563	Dev
BERT-FPT-FT-Multi	5	0.476	0.724	0.715	Dev
BERT-FPT-FT-Multi	15	0.468	0.743	0.787	Dev
BERT-FPT-FT-Multi	40	0.476	0.758	0.791	Dev

Table 2: Comparisons against baselines for each stage. Base: BERT with only given weights loaded (initial baseline). FPT: further pre-trained on target data with MLM (Stage 1 baseline). FPT-FT: further pre-trained on target data and fine-tuned with BERT layers frozen (Stage 2 baseline). FPT-FT-Multi: further pre-trained on target data, fine-tuned on each task with BERT layers frozen, then end-to-end fine-tuned using multi-task learning (Stage 3). Scores are for the best model saved based on the highest average dev score across epochs.

4.5 Further pre-training

We run separate fine-tuning experiments using base BERT layer weights as well as BERT layers further pre-trained on target data. We can see from the results in Figure 2 that in all cases, further pre-training on target data improves training stability and scores.

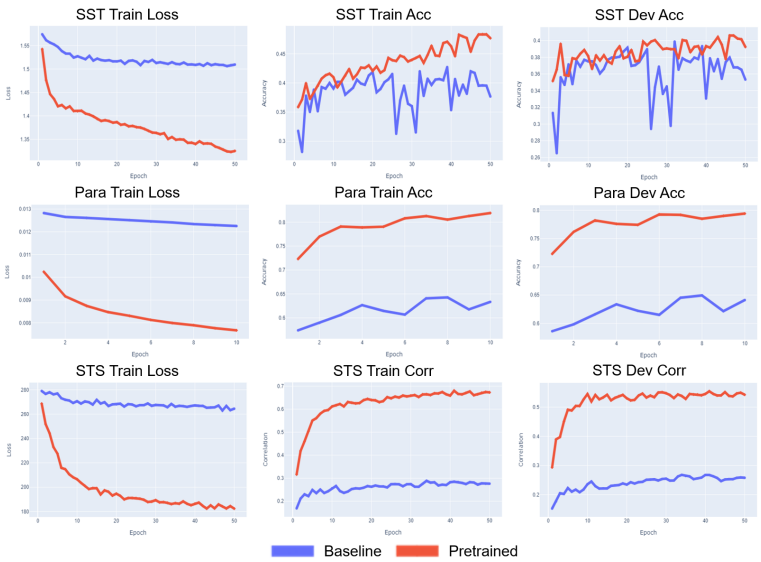

Figure 2: Stage 2 fine-tuning with and without Stage 1 further pre-trained BERT weights.

4.6 Gradient surgery

We run an ablation experiment to determine whether gradient surgery is effective. The experiments use a batch size of 32, the same learning rate of 1e-6, and are run for 3 epochs across 3 separate random seeds. The resulting scores are averaged across random seeds and epochs. From our results in Table 3, we see that there is no significant difference between adding and not adding gradient surgery when using a very low learning rate of 1e-6.

Furthermore, we run an experiment using gradient surgery where we vary the learning rate, using 1e-3, 1e-4, and 1e-5 and then count the number of collisions (where the cosine similarity between the gradient of one task and a gradient of another task is negative). We observe 1187, 1087, and 1130 average collisions per batch step for learning rate of 1e-3, 1e-4, and 1e-5, respectively.

Experiment	SST Acc	Para Acc	STS Corr	Split
With Gradient Surgery	0.4997	0.8029	0.5770	Train
Without Gradient Surgery	0.4999	0.8029	0.5772	Train
With Gradient Surgery	0.3982	0.7863	0.5256	Dev
Without Gradient Surgery	0.3982	0.7866	0.5257	Dev

Table 3: Multi-task fine-tuning (Stage 3) scores for with and without using gradient surgery. Each score is averaged across 3 epochs and 3 different random seeds.

4.7 Weighted Random Sampling

We applied Weighted Random Sampling to Sentiment Analysis(SST), Paraphrase and Semantic Text Similarity (STS) datasets during training. The intuition is since paraphrase and STS datasets are imbalanced datasets, weighted random sampling should help counter the imbalance for generating representative samples from non-uniform populations. However on experimenting with and without Weighted Random Sampling(WRS) noticed following accuracies for sentiment analysis and paraphrase detection tasks and correlation score for STS task on scale of (0-1):

Experiment	SST Acc	Para Acc	STS Corr	Split
With WRS	**0.537**	**0.814**	0.922	Train
Without WRS	0.522	0.813	**0.936**	Train
With WRS	0.393	0.797	0.468	Dev
Without WRS	**0.434**	0.797	**0.793**	Dev
With WRS	0.395	0.792	0.581	Test
Without WRS	**0.429**	**0.793**	**0.623**	Test

Table 4: Scores with and without weighted random sampling.

We used batch size of 32 and 10 epochs in both cases. This shows that either WRS had no impact or minor decrease in performance compared to training model without Weighted Random Sampling.

5 Analysis

The model has better downstream performance on the paraphrase detection (Para) and semantic text similarity tasks (STS) compared to sentiment analysis (SST) (see Table 2). The good performance of the model in all cases on paraphrase detection can be attributed to the large training dataset available for this task. Based on review of output for three tasks, we observe model is generally capturing semantic sense of sentence well. This validates model's learning based on one dataset is benefiting other tasks when their objective is aligned and when tasks have shared parameters.

Model performance on STS task was better when neural network was used in conjunction with cosine similarity. This could be because cosine similarity is both agnostic to the norm of the last hidden cls states for each sentence pair element and because cosine similarity forces the BERT layers to learn to place similar sentence encoding vectors closer together. Based on model output on this task, we observed that model is generally generating lower scores than ground truth. As expected it is doing well when same words or phrases are used in sentence pairs than when there is some semantic difference. For example: Model correlation coefficient score is 4.4 for "Kenyan forces caused mall collapse add "Official says Kenyan forces caused mall collapse". Ground truth is 5. Model scored 1.7 for "Someone is boiling okra in a pot" and."Someone is cooking okra in a pan." Ground truth is 4.4. Looks like model is giving importance to difference between pot and pan.

In the case of Sentiment Analysis, SST dataset labels seem somewhat open to interpretation and inconsistent at times. For example: 'It 's a lovely film with lovely performances by Buy and Accorsi' has somewhat positive score label in the dev dataset however our model predicted positive review which seems more appropriate for this example. Another example is 'you 've got ice water in your veins' has somewhat positive label in dev dataset but has negative predicted label from our model which seems more accurate. This means that the SST dataset may have noisy labels.

Initially model was over=fitting to Sentiment Analysis and Semantic Text Similarity. But based on final results it is evident our strategies of applying L2 norm and dropout helped combat over=fitting for Semantic Text Similarity to great extent. While it helped some, it didn't make as much impact in addressing over-fitting in the case Sentiment Analysis. Maybe based on datasets while model is picking semantic sense it is not able to generalize emotion of sentence in alignment with SST dataset ground truth.

The model is able to pick up nuances in usage of pronouns like "you", "one" or proper nouns in detecting question pairs as paraphrases. For example: the model accurately predicted "Do you prefer to ask or answer questions on Quora?" and "Do Quora Users prefer to ask questions or to answer them?" as the same question. However the model sometimes misses the significance of qualifier clauses. For example: the model predicts "Why do I fall asleep when sitting?" and "Why do I keep falling asleep?" as paraphrases when they are not. Despite the relatively high performance of the model on paraphrase detection, the examples above indicate that there is still room for designing a better classifier for this task.

Lastly, we performed hyperparameter tuning using task weights for loss, epoch, batch size and learning rate. As predicted, fine-tuning was working better with a lower learning rate. This is consistent with finding of [1] that lower learning rate can help combat catastrophic forgetting. Our experiments with gradient surgery and gradient smoothing didn't help much with improving model performance (see Table 3). This could be because the learning rate is so low for our multi-task fine-tuning experiments that in expectation, the model still converges despite small non-optimal steps caused by collisions in task-specific gradients.

6 Conclusion and Future work

Although we do not achieve the top test leaderboard scores, our work demonstrates that our overall strategy for pre-training into multi-task learning is effective. We also demonstrated that further pre-training the model using MLM on the target datasets significantly improves performance, and applying task specific fine tuning and transferring weights before a combined multitask-fine tuning can significantly improve model performance compared to only further pre-training. For future research, we would pursue pre-training on other forms of self-supervised objectives, and perhaps use contrastive learning to improve pre-training.

We also found that a multiple layer neural network for task specific heads yields better results than a single naive linear layer for each task-head module. In the future, weight sharing between some of these task-specific modules may improve parameter efficiency and lead to more generalized learning across tasks.

BERT is known to do better on larger datasets. We noticed significant over-fitting for the sentiment analysis and semantic text similarity tasks. In our future efforts, implementing data augmentation would be good way to lower over-fitting and help model generalize better. Based on the results, it is clear that model is doing better in capturing semantic similarity between sentences than in qualifying emotion of the sentence. It might also help to experiment with other neural network architectures for the sentiment analysis head to improve performance for classifying sentiment. Another future extension would be to look into using label smoothing as a way to dull the negative affects of noisy labels on classification for the SST dataset.

Lastly, there is also more scope for hyperparameter tuning, as we only tested a few combinations due to time and computing resource constraints.

References

[1] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? *Chinese Computational Linguistics*, 2019.

[2] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[4] Liyuan Liu, Jingbo Shang, Frank F. Xu, Xiang Ren, Huan Gui, Jian Peng, and Jiawei Han. Empower sequence labeling with task-aware neural language model. In *AAAI Conference on Artificial Intelligence*, 2017.

[5] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

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

[7] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019.

[8] Ravi Prakash and Karmanya Kumar. Class weight technique for handling class imbalance, 07 2022.

[9] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, A. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing*, 2013.

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