Leave It To BERT: Exploring Methods for Robust Multi-Task Performance

Stanford CS224N Default Project

Finn Dayton
Department of Computer Science
Stanford University
`finniusd@stanford.edu`

Abhi Kumar
Department of Computer Science
Stanford University
`abhi1@stanford.edu`

Chris Moffitt
Department of Computer Science
Stanford University
`cmoffitt@stanford.edu`

Abstract

The ability to accurately perform multiple sentence-level tasks with a single model has significant applications. We investigate the efficacy of fine-tuning a BERT model to achieve optimal performance on a range of sentence-level tasks. As a baseline, we expand the minBERT implementation by utilizing pretrained weights to perform sentiment classification, paraphrase detection, and semantic textual similarity. Then, we perform additional pretraining on adjacent datasets, multi-task fine-tuning with gradient surgery, and single-task fine-tuning using a variety of model architectures to train the model weights for deeper embeddings that generalize well to each of the three tasks. Our experiments demonstrate that the additional pretraining and task-specific head fine-tuning contributes mild boosts in performance, but the greatest increases in performance come from the combination of performing gradient surgery and using a model that concatenates input sentences so that BERT can generate one embedding for the entire input. Essentially, performing gradient surgery while allowing BERT to do most of the work on the input sentences together results in the highest performance across all objective tasks. We provide a comprehensive analysis of the individual performances, advantages, and disadvantages of each proposed training method and model architecture.

1 Key Information

Our mentor is Shai Limonchik. We have no external collaborators nor are we sharing projects.

2 Introduction

Deep NLP models have achieved great success on single tasks, but the ability to perform well on multiple tasks is becoming more crucial for many practical applications of NLP. Developing multi-task models is challenging, however, because different tasks have different requirements in terms of input representation, output format, and training data. Moreover, tasks may have conflicting objectives, making it difficult to optimize a single model for all tasks simultaneously. Transformer-based models, such as BERT [2], have led to significant improvements in the performance of NLP models across a range of tasks. For our project, we train and evaluate a series of models that use shared BERT embeddings to perform three tasks simultaneously: sentiment analysis, paraphrase detection, and similarity detection.

In this report, we present a novel approach that combines additional pretraining, gradient surgery, and final layer fine-tuning to optimize multi-task deep learning for NLP. Additional pretraining is a technique that involves training a model on additional data before fine-tuning on specific tasks, improving model performance by enabling it to learn general features that can be transferred to a range of downstream tasks. Gradient surgery, proposed by Yu et al [10], is a technique used for multitask learning that manipulates gradients to selectively dampen or amplify the contributions of each objective task to the overall loss function. Gradient surgery not only helps the model focus on the more challenging or informative parts of each task but also is beneficial for preventing over-fitting, under-fitting, and negative transfer. Our final step is final layer fine-tuning, whereby the shared BERT weights are frozen and the final layers (task-specific head) for each objective task is fine-tuned on labeled data. Final layer fine-tuning can help the model adjust its final layer parameters more precisely to each specific task without conflicts between the tasks. While each of these techniques has been shown to improve the performance of NLP models in isolation, their combined effect has not been thoroughly explored. Our project aims to fill this gap by investigating the potential benefits of many combinations of these techniques and evaluating their effectiveness on a range of NLP tasks. Our results show promising improvements in the performance of multi-task NLP models, especially in cases where the tasks have different complexities or variances.

3 Related Work

Recent research, such as (Yang et al [9]) and (Liu et al [5]), highlights the advantages of additional pretraining in natural language processing models on large, diverse datasets to enhance performance on downstream tasks. Nevertheless, this is a very active area of research and there exist many techniques for pretraining and finetuning for good multi-task performance.

Liu et al 2019 [5] introduces Roberta, an improved bi-directional encoder based on the BERT architecture. The paper validates the fundamental approach of BERT but emphasizes that longer training and better hyperparameter optimization lead to much better results. The model, though improved, remains just an encoder, so it is fine-tuned for the downstream tasks tackled by our model.

Chen et al 2021 [1] provides an overview of the use of Multi-Task Learning (MTL) in NLP tasks. They review MTL architectures used in NLP tasks and categorize them into parallel, hierarchical, modular and generative architecture. A parallel architecture—what we elected to use—shares most of its weights for each of its tasks while each task has its own task-specific output layer. For our experiments, since each input produces a specified output (which is specific to each task), the generative architecture is not applicable. Future research into multi-task training of BERT, however, could investigate the efficacy of using a hierarchical or modular architecture.

Yu et al 2020 [10] introduced gradient surgery, a technique for reconciling gradient conflicts for multi-task learning introduced above. Our approach builds on this technique by applying it to the three tasks of sentiment analysis, paraphrase detection, and similarity detection.

Lastly, Weller et al 2022 [7] examine and compare three methods of multi-task finetuning. They first train on an intermediate task before training on the objective task. Second, they using multi-task learning to train jointly on the main task and a supplementary task. And third, they using multi-task learning to jointly train on all available datasets. They found that for primary tasks of STS and SST (two of our objective tasks) and a pretraining dataset of MultiNLI (Multi-Genre Natural Language Inference) [8] (which we use for additional pretraining), there was no significant difference between their first and second methods. Given these findings, we will build on their first method: we first pretrain on MultiNLI dataset and subsequently perform multi-task learning to jointly train on our three end-tasks.

4 Approach

4.1 Baseline

For our baseline, we train a naive Multitask Classification model that uses the pre-trained BERT weights and is only finetuned for the Sentiment Classification task but then evaluated for each of the three objective tasks: sentiment classification, paraphrase detection, and semantic textual similarity. Our aim was to beat these baseline scores for each of the three tasks by utilizing the model architectures and neural training methods described below.

4.2 Model Architectures

In this report, we investigate three model architectures. Our first model architecture, Concat After BERT (ConcatA), feeds each sentence from a sentence pair separately into BERT and concatenates the two resulting embeddings. It includes one linear layer and one dropout layer in each task-specific head (See Figure 1).

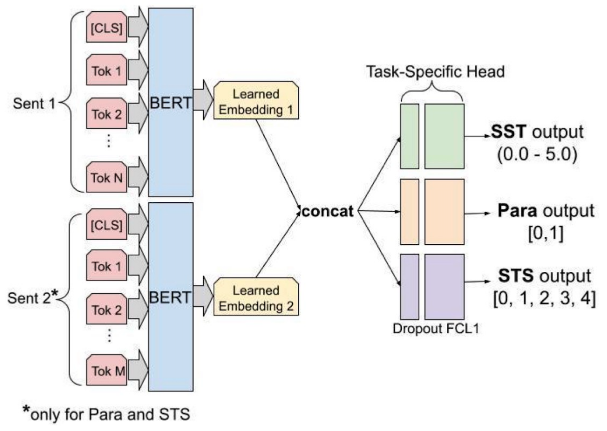
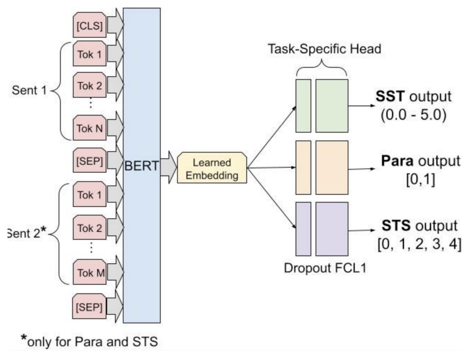
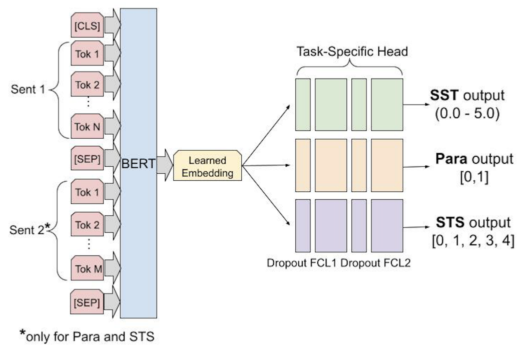


Figure 1: Concat After BERT (ConcatA)

Our second model architecture, Concat Before BERT (ConcatB), concatenates tokenized sentences from a pair and then feeds the result into BERT to generate one embedding for both sentences. Similarly to ConcatA, it also includes one linear layer and one dropout layer in each task-specific head (See Figure 2(a)). Our last model architecture, Concat Before BERT + Added Layer (ConcatB + AL) follows the same concatenation pattern of inputs as ConcatB, but there are two linear layers and two dropout layers in each task-specific head. In this model architecture, we introduce a Leaky ReLU activation function in between the first set and second set of dropout and linear layers (See Figure 2(b)).



(a) Concat Before BERT (ConcatB)



(b) Concat Before BERT + Added Layer (ConcatB + AL)

Figure 2: Concat Before BERT (ConcatB) Architecture

Note, the difference between the ConcatA and ConcatB model architectures only affects how learned embeddings are generated in the PARA and STS datasets since they are both pair datasets (having two sentences as input). Since the input for the SST dataset is a single sentence, we feed the single sentence token ids for SST directly into BERT to generate a learned embedding in all model architectures. However, the task-specific heads in each model architecture are still accordingly modified for all three tasks (i.e. ConcatA and ConcatB have 1 set of dropout and linear layers in each task-specific head; ConcatB + AL has 2 sets of dropout and linear layers in each task-specific head).

4.3 Training Methods

4.3.1 Additional Pretraining

Our first training method is to conduct additional pretraining on target-domain data using the Multi-Genre MultiNLI Corpus, which has 433k sentence pairs from multiple different contexts, all labeled with textual entailment information. Like both the paraphrase detection and similarity detection task, since the MultiNLI dataset contains sentence pairs that are related in some way, we hypothesized pretraining on MultiNLI will allow us to improve BERT embeddings for sentence pairs.

4.3.2 Gradient Surgery

Our second training method is gradient surgery for multitask fine-tuning on all of our three objective tasks. Multitask fine-tuning with gradient surgery, as described in [10], is a technique that selectively scales gradients from each objective task in a model to improve its generalization performance on the different but related tasks. To perform gradient surgery, we first add together the loss on each of our three tasks.

$$L_{total} = L_{sst} + L_{para} + L_{sts} \tag{1}$$

where L_{sst}, L_{para}, and L_{sts} represent the Loss values of the sentiment classification, paraphrase, and STS tasks, respectively.

To reconcile gradient conflicts, we use gradient surgery, proposed by Yu et al. [10], where the gradient of one task g_i is projected onto the normal plane of another conflicting task's gradient g_j, as shown in this equation: $g_i = g_i - (\frac{(g_i g_j)}{||g_j||^2})g_j$. This removes the component of the gradient that conflicts with the other task's objective, while preserving the component that contributes to the current task's objective. Since we have three tasks and thus three gradients to calculate, we project each gradient onto the normal plane of the other two conflicting gradients:

$$g'_{sst} = g_{sst} - (\frac{(g_{sst} g_{para})}{||g_{para}||^2})g_{para} - (\frac{(g_{sst} g_{sts})}{||g_{sts}||^2})g_{sts} \tag{2}$$

$$g'_{para} = g_{para} - (\frac{(g_{para} g_{sst})}{||g_{sst}||^2})g_{sst} - (\frac{(g_{para} g_{sts})}{||g_{sts}||^2})g_{sts} \tag{3}$$

$$g'_{sts} = g_{sts} - (\frac{(g_{sts} g_{sst})}{||g_{sst}||^2})g_{sst} - (\frac{(g_{sts} g_{para})}{||g_{para}||^2})g_{para} \tag{4}$$

where g_{sst}, g_{para}, and g_{sts} represent the gradients for the sentiment classification, paraphrase, and STS tasks, respectively. Then, we simply take the sum of each of the three gradients in order to calculate the final gradient, g', which is used to perform gradient descent:

$$g' = g'_{sst} + g'_{para} + g'_{sts} \tag{5}$$

In order to perform gradient surgery and back propagation for each batch simultaneously during training, we attempt two different Round Robin training techniques to reconcile the different sizes of the SST, Quora, and STS datasets:

(1) Round Robin: Different Batch Sizes (GS$_{bd}$)

In this approach, we define different batch sizes b_{sst}, b_{para}, and b_{sts} based on the lengths of each dataset, l_{sst}, l_{para}, and l_{sts} such that for n number of batches,

$$n \cdot b_{sst} \approx l_{sst}, \quad n \cdot b_{para} \approx l_{para}, \quad n \cdot b_{sts} \approx l_{sts} \tag{6}$$

By doing this, for each training iteration in an epoch, we can take one batch from SST, Quora, and STS, calculate the loss for each respectively, perform gradient surgery, and take a gradient step. After going through all n batches, we will have exhausted most of the training examples in all three datasets (See figure 3(a)).

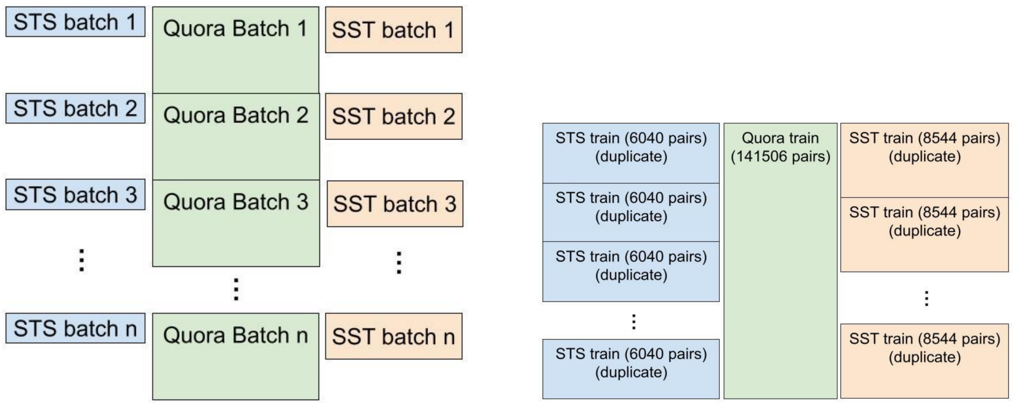
When combining the losses for gradient surgery, this approach also requires us to rescale the losses according to the minimum batch size among the three batch sizes, b_{min}, in the following manner:

$$Loss'_{sst} = Loss_{sst}\frac{b_{min}}{b_{sst}}, \quad Loss'_{para} = Loss_{para}\frac{b_{min}}{b_{para}}, \quad Loss'_{sts} = Loss_{sts}\frac{b_{min}}{b_{sts}} \tag{7}$$

Rescaling in this manner allows us to weight each loss equally rather than over-weighting losses that were calculated from larger batch sizes.

(2) Round Robin: Data Wrapping (GS$_w$)

In the Data Wrapping approach, we define equal batch sizes for each dataset. This means that each training iteraction sees an equal number of examples from each dataset. Since this will cause, shorter datasets (SST and STS) to run through all of their training examples before longer datasets (Quora), we wrap the shorter datasets such that we see repeat exemples until exhausting all training examples of the largest dataset. This means each epoch sees all of Quora (the largest dataset) once and sees many duplicates of STS and SST (the smaller datasets), which could potentially lead to over-fitting (See figure 3(b)).



(a) One epoch of GS$_{bd}$ (b) One epoch of GS$_w$

Figure 3: Two Round Robin Strategies for Multi-Task Training with Gradient Surgery

4.3.3 Final Layer Fine-tuning (FL)

Our last training method is Final Layer Fine-tuning (FL). Our model has one head for each of our objective tasks (SST, PARA, SST), and we vary the depths of these heads (as described in section 4.2.2). In this method, we finetune the parameters for each head by training the model on each task once again, but freezing the shared BERT weights so that we only perform back propagation on the head specific to that task. The reasoning behind this final method is that, unlike for the shared BERT weights, each task-specific head does not need to consider the losses from the other tasks when performing back propogation. Thus, in this final training method, we are able to finetune the heads to learn task-specific features and specialized parameters for each of our objective tasks.

5 Experiments

5.1 Data

We use 4 datasets in our experiments. From the default final project, we use the Stanford Sentiment Treebank (SST) [6] for sentiment classification, Quora [3] for paraphrase detection between two sentences, and SemEval[4] for semantic textual similarity between two sentences. We also use the Multi-Genre MultiNLI Corpus [8], a dataset used for logical inference detection between two sentences, for additional pretraining. This dataset has 402,517 examples of sentence pairs from 10 different "genres" (ex: fiction, government, travel, etc). Each sentence pair is labeled as either contradictory, entailment, or neutral; therefore, it has been used for the textual entailment task. We will use 392,702 examples for our train set, and 9,815 examples for our dev set when pretraining the model.

5.2 Evaluation method

We use accuracy as the primary metric for both sentiment classification on the SST dataset and paraphrase detection on the Quora dataset. We use the Pearson correlation coefficient as the primary metric for sematic textual similarity detection on the SemEval dataset.

5.3 Experimental details

To establish a baseline performance for our multitask model, we ran a baseline experiment in which we tested our model on all three tasks using the pretrained weights imported from BERT and fine-tuned on a single linear final layer head for the predict sentiment task using the SST dataset, as described in section 4.2. This baseline experiment serves as a reference for the following experiments.

We assessed the performance of our proposed neural techniques by training and evaluating every permutation of the proposed training methods on each of the three tasks for each of the model architectures we implemented:

First, we conducted a series of experiments in which we trained separate models using the GS$_{bd}$, GS$_{w}$, and FL training methods individually. These experiments were conducted to understand the effectiveness of each training method in isolation and whether it improves the model performance for each task.

Then, we pre-trained our BERT weights on the Natural Language Inference (MultiNLI) task and subsequently trained on GS$_{bd}$, GS$_{w}$, and FL, yielding the following experiments: MultiNLI + GS$_{bd}$, MultiNLI + GS$_{w}$, and MultiNLI + FL. In these experiments, we pretrained MultiNLI and loaded these weights in before training on GS$_{bd}$, GS$_{w}$, and FL. This set of experiments provided a direct comparison to the first set of experiments, illustrating the effect of pretraining MultiNLI prior to gradient surgery and fine-tuning. We also conducted experiments to test whether gradient surgery can be improved by fine-tuning, given by the following experiments: GS$_{bd}$ + FL and GS$_{w}$ + FL. To conduct these experiments, we fine-tuned on the model weights from the previous experiments GS$_{bd}$ and GS$_{w}$.

Finally, we also assessed all three of our training methods in series: MultiNLI + GS$_{bd}$ + FL and MultiNLI + GS$_{w}$ + FL. For these experiments, we loaded in the model weights trained by the experiments MultiNLI + GS$_{bd}$ and MultiNLI + GS$_{w}$ and fine-tuned using FL.

We ran each of the 10 experiments described above on each of our model architectures (ConcatA, ConcatB, and ConcatB + AL), giving us a total of 30 experiments. In all 30 experiments, we used a learning rate of 1e-5 and 10 epochs. Additionally, for every experiment with MultiNLI, we used a batch size of 16 for the MultiNLI stage. For every experiment with GS$_{bd}$, we used batch sizes of 1, 24, and 1 for the SST, PARA, and STS tasks, respectively, for the GS$_{bd}$ stage. For every experiment with GS$_{w}$, we used a batch size of 16 for all tasks for the GS$_{w}$ stage. For every experiment with FL, we used a batch size of 8 for the FL stage. To reduce over-fitting, we employed dropout rates of 0.3 for the first dropout layer (applies for ConcatA, ConcatB, and ConcatB + AL) and 0.45 for the second dropout layer (only applies to ConcatB + AL). While we would have liked to conduct more experiments with alternative model hyperparameters to continue exploring the most optimal multitask model configuration, we were limited by compute resources and time.

5.4 Results

As shown in Table 1, all of our experiments outperform the baseline model, and this is expected since our baseline model was only fine-tuned on the SST task and did not utilize any sophisticated model architectures or training methods. Our best model was trained with MultiNLI + GS$_{w}$ and used the ConcatB + AL architecture (highlighted in Table 1) as it had the highest average scores across all three tasks. The best SST score comes from MultiNLI + GS$_{bd}$ + FL with the ConcatA architecture; the best PARA score comes from multiple experiments using GS$_{w}$ with ConcatB + AL architecture; and the best STS score comes from MultiNLI + GS$_{bd}$ with the ConcatB + AL architecture (See Table 1). When we evaluated our best model, MultiNLI + GS$_{w}$ with ConcatB + AL, on the Test Set, we observe outstanding scores across all three tasks (See Table 2).

Baseline:	**SST**	**PARA**	**STS**
Finetune on SST	.310	.380	-.008

	Model Architectures								
	ConcatA			**ConcatB**			**ConcatB + AL**		
Methods	**SST**	**PARA**	**STS**	**SST**	**PARA**	**STS**	**SST**	**PARA**	**STS**
GS_{bd}	.500	.720	.362	.520	.838	.866	.511	.832	.865
GS_{w}	.528	.732	.352	.511	.843	.850	.520	**.873**	.844
FL	.305	.636	.206	.303	.665	.154	.337	.696	.330
MultiNLI + GS_{bd}	.529	.708	.390	.495	.836	.881	.500	.839	**.892**
MultiNLI + GS_{w}	.514	.740	.326	.503	.850	.868	.516	**.873**	.877
MultiNLI + FL	.342	.661	.104	.343	.753	.635	.330	.751	.711
GS_{bd} + FL	.503	.719	.373	.510	.838	.867	.511	.830	.866
GS_{w} + FL	.527	.742	.356	.508	.847	.851	.518	**.873**	.844
MultiNLI + GS_{bd} + FL	**.531**	.713	.389	.499	.840	.882	.508	.843	.891
MultiNLI + GS_{w} + FL	.508	.748	.332	.499	.853	.868	.514	**.873**	.878

Table 1: Experiment Results on Dev Set

Model	**SST**	**PARA**	**STS**
MultiNLI + GS_{bd} with ConcatB + AL	0.539	0.873	0.876

Table 2: Best Model Performance on Test Set

5.5 Model Architectures Findings

Our findings indicate that the performance of the Paraphrase and STS Similarity detection tasks is significantly improved when we employ a model architecture that concatenates sentence tokens prior to inputting them into BERT, ConcatB, as opposed to ConcatA. This observation is particularly intriguing as it suggests that ConcatB enables BERT to generate embeddings that can more effectively capture the inter-sentence relationship between two input sentences. However, it is also worth noting that ConcatB appears to negatively impact SST Sentiment Classification performance, which is the only task that employs single sentence inputs. This suggests that there may be a trade-off between the ability of the model to generate high-quality sentence pair embeddings and its ability to generate single sentence embeddings. Our analysis of this observation leads us to speculate that the over-specialization of BERT in generating sentence pair embeddings for tasks that require them may come at the cost of its ability to generate effective single sentence embeddings. Additionally, we found that for most of the methods that we tested, the ConcatB + AL model architecture outperformed ConcatB across all three tasks on average. This observation suggests that the addition of extra layers on each task head makes the model more expressive, enabling it to learn more nuanced representations of the underlying data and task-specific features.

5.6 Training Methods Findings

We observed that MultiNLI pretraining typically results in small improvements in performance across all tasks, except for SST performance in the ConcatB models. This may be attributed to the fact that MultiNLI pretraining may cause the model to further specialize for sentence pair embeddings rather than single sentence embeddings, leading to a decrease in performance for single sentence tasks. This observation highlights the importance of carefully selecting the appropriate pretraining techniques based on the nature of the objective tasks. Moreover, our experimental results suggest that gradient surgery is a highly effective technique for multi-task learning of shared BERT weights since our gradient surgery experiments achieve superior performance across all three tasks. Our experiments revealed that GS_{w} consistently under-performs GS_{bd} in STS similarity detection performance. This is likely because wrapping causes the model to repeatedly see the same STS examples multiple times which leads to over-fitting. Similarly, we see that GS_{w} consistently performs better than GS_{bd} on the PARA task since GS_{w} trains on the entirety of the PARA dataset, while GS_{bd} is trained on a portion of the PARA dataset; this allows GS_{w} to learn more nuanced features for the PARA Paraphrase task.

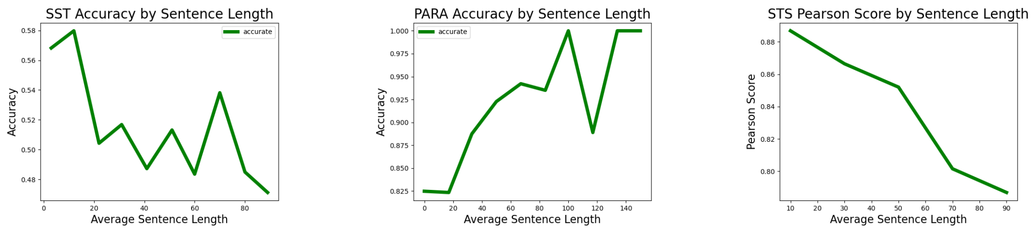
Finally, our results indicate that final layer fine-tuning can result in minor performance improvements across all tasks. However, it is worth noting that final layer fine-tuning, when performed individually, under-performs all other experiments, except for baseline. This suggests that final layer fine-tuning should be used in conjunction with gradient surgery and additional pretraining, if used for multitask learning, in order to achieve optimal performance.

6 Analysis

6.1 Analysis of Model Performance and Data Variation

We performed additional data analysis on the results of our best model (MultiNLI + GS$_w$ with ConcatB + AL architecture) in order to get a better sense of where our model is succeeding and failing. In doing so, we made two interesting discoveries with regards to what type of input our model performs the best upon.

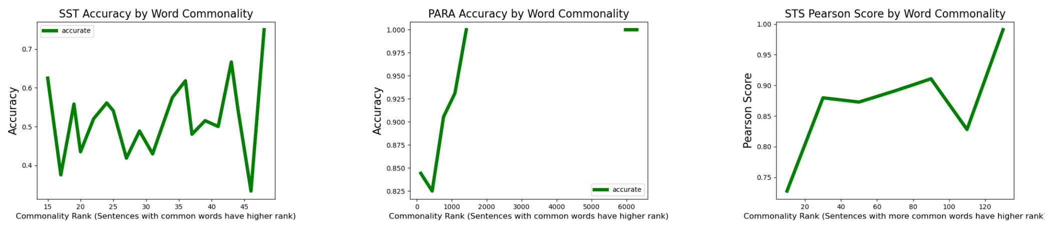
1. First, we found that our model performs worse at SST sentiment classification and STS similarity detection when given longer sentences (See figures 4(a) and 4(c)). This is likely because longer sentences increase ambiguity for SST and STS, making it harder to make accurate predictions. However, PARA paraphrase detection interestingly performs better when given longer sentences (See figure 4(b)). This is likely because the paraphrase detection task requires simpler predictions (simple binary classification) and the longer sentences provide more context on which the model can base its predictions.



(a) (b) (c)

Figure 4: Performance Analysis by Sentence Length

2. Second, we found that our model performs better across all tasks when the sentences contain more words that appeared commonly in the training dataset (See figures 5(a), 5(b), and 5(c)). As might be expected, the model is less sure about sentences with words that rarely appear in the training dataset or don't appear at all. However, as is evident in the chart and worth noting, this correlation is much more noisy for the SST task.

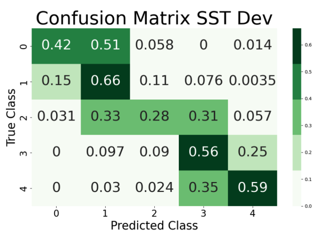


(a) (b) (c)

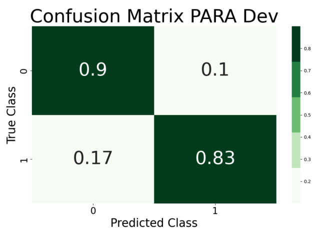
Figure 5: Performance Analysis by Word Commonality

6.2 Analysis of Model Predictions

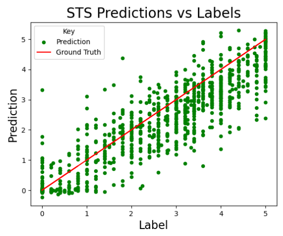
We also assessed the quality of our best model (MultiNLI + GS$_w$ with ConcatB + AL architecture) by generating data visualizations for the relationship between our predictions and labels. For our two discrete classification tasks (SST and PARA), we generated a confusion matrix, which provides a detailed view of the accuracy and errors of our model's predictions. Our analysis revealed that our best performing model consistently achieved high accuracy and maintained a close alignment with the true labels for both the SST and PARA tasks (See figures 6(a) and 6(b)). This observation is supported by the fact that the confusion matrices of our results consistently exhibit a high level of correct predictions, with only a relatively small number of incorrect classifications. For STS, which has continuous outputs, we generated a scatterplot comparing the labels vs. predictions (See figure 6(c)). In this visualization, we see that predictions generally hover around the ground truth labels, while slightly undershooting them.



(a)



(b)



(c)

Figure 6: Confusion Matrices From Best Model

These findings demonstrate the efficacy of our best model architecture and its ability to accurately predict the labels for our objective tasks.

7 Conclusion

In this report, we implemented, trained, and evaluated a series of neural training methods and model architectures on multi-task learning objectives, and we had a variety of findings regarding each of our models. Most importantly, we found that two methods are primarily responsible for increased multi-task performance. First, the concatenation of input sentences before BERT dramatically improves model performance across PARA and STS tasks by allowing BERT to do most of the work. Second, performing Gradient Surgery during training – both using the different batch size (GS_{bd}) and wrap (GS_{w}) round robin approaches – provides considerable gains in performance across all 3 tasks by allowing BERT to learn parameters that favor all 3 tasks simultaneously. Though not as impactful as the former two methods, we also found that using additional layers on each task head made the model more expressive and allowed it to learn more task-specific features. Furthermore, our first training method, additional pretraining on MultiNLI, generally provides small bump in performance for all tasks. Finally, we found that the performance of the final layer fine-tuning varied, providing a small bump in accuracy for some models while in others, the performance decreased.

While we would have liked to conduct more experiments with alternative model hyperparameters, such as learning rate, dropout percentages, and the number of final layers on each task head, we were limited by compute resources and time. Future work for this report includes exploring this hyperparameter experimentation further. Additionally, future research is needed to explore the efficacy of using modular and hierarchical architectures for sharing weights for multitask learning.

References

[1] Shijie Chen, Yu Zhang, and Qiang Yang. *Multi-Task Learning in Natural Language Processing: An Overview*. 2021. arXiv: 2109.09138 [cs.AI].

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[3] Kuntal Dey, Ritvik Shrivastava, and Saroj Kaushik. "A Paraphrase and Semantic Similarity Detection System for User Generated Short-Text Content on Microblogs". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical*

Papers. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 2880–2890. URL: `https://aclanthology.org/C16-1271`.

[4] Mona Diab, Tim Baldwin, and Marco Baroni, eds. *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*. Atlanta, Georgia, USA: Association for Computational Linguistics, June 2013. URL: `https://aclanthology.org/S13-1000`.

[5] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: `1907.11692 [cs.CL]`.

[6] Richard Socher et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. URL: `https://aclanthology.org/D13-1170`.

[7] Orion Weller, Kevin Seppi, and Matt Gardner. *When to Use Multi-Task Learning vs Intermediate Fine-Tuning for Pre-Trained Encoder Transfer Learning*. 2022. arXiv: `2205.08124 [cs.CL]`.

[8] Adina Williams, Nikita Nangia, and Samuel Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. URL: `http://aclweb.org/anthology/N18-1101`.

[9] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf`.

[10] Tianhe Yu et al. *Gradient Surgery for Multi-Task Learning*. 2020. DOI: `10.48550/ARXIV.2001.06782`. URL: `https://arxiv.org/abs/2001.06782`.