Multi-task Learning using BERT

Stanford CS224N Default Project

Pankhuri Aggarwal and Nina Cruz
Department of Computer Science
Stanford University
pankhuri@stanford.edu and ninacruz@stanford.edu

Abstract

This paper focuses on developing a multi-task model for natural language processing related tasks. The tasks include sentiment analysis, paraphrase detection, and semantic textual similarity at the sentence-level. We implement a minBERT transformer-based model for this purpose and use that as our baseline model. Then, we use a combination of gradient surgery, regularization techniques such as the Smoothness-inducing Adversarial Regularization, and dropout layers to improve our model performance. We evaluate the impact of using these techniques on the model performance for the three listed tasks. On the test set, our best performing model achieves an accuracy of 72.6% for paraphrase detection, accuracy of 53.3% for sentiment analysis, and correlation of 0.427 for semantic textual similarity. The three-task average performance is 0.562 on the test set.

1 Key Information to include

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

2 Introduction

The development of pre-trained models such as BERT has greatly improved performance and training efficiency for many natural language processing (NLP) tasks. Muti-task learning (MTL) has become a popular technique to handle various downstream tasks simultaneously. MTL is able to combat some pervasive problems in NLP tasks such as overfitting and data scarcity (Chen et al., 2021). Though effective, using MTL presents a number of challenges such as which architecture to use, how to train multiple models, how to combine the losses, how to monitor the losses and gradients to improve performance for all tasks, etc. The advantages of MTL make these challenges worthwhile to pursue and motivate us to experiment and find methods to combat these challenges.

In this paper, we use MTL along with a pre-trained minBERT model for three NLP tasks, namely paraphrase detection, sentiment analysis, and semantic textual similarity. We use minBERT as a baseline model and then experiment with a multitude of techniques to improve model performance. The challenge is to manage the different datasets and tasks to be able to develop a multi-task model that performs well on all three tasks. We experiment with different architectures and techniques, such as gradient surgery (Yu et al., 2020) and SMART regularization (Jiang et al., 2021), and evaluate their impact on our model.

This paper first discusses previous work in the field of MTL and highlights some challenges that the task presents. Then, it details the approach used in the development and analysis of the experimental process. The Experiments section provides details about the datasets, the evaluation metrics, and the experimental setup. Following this, the Results section presents the observed results for each experiment. The next two sections focus on the analysis of these results, describe the limitations, and

discuss possible extensions. Finally, the Conclusion provides a summary of the results and highlights the learnings of this paper.

3 Related Work

In their paper, Chen et al. (2021) describe the benefits of using MTL in the field of NLP. MTL has been able to improve performance and reduce training costs by simultaneously training related tasks. It also combats overfitting and data scarcity, which are common problems in natural language processing.

Zhang et al. (2023) discuss the challenge of designing and training a single model for multiple tasks. They determined that task-relatedness is the key to choosing the correct multi-task training method, which is of two types: “joint training" and “multi-step training". According to them, joint training should be used when all tasks can be trained simultaneously while multi-step training should be used when one’s task output is used for another task. In this project, we use joint training for three separate tasks.

Yu et al. (2020) have also highlighted that MTL poses various optimization challenges such as dealing with multiple gradients. They propose a model-agnostic approach called ‘gradient surgery’, which uses the strategy of projecting different task-specific gradients on the normal plane of each other, to optimize MTL. Their work shows that this approach leads to higher model performance and training efficiency for MTL tasks.

4 Approach

The first part of this project consists of implementing the minBERT model, as detailed in the paper by Devlin et al. (2019). Specifically, we implement the following layers of the BERT transformer: the attention layer, embed layer, and add plus normalize layer. We then implement the classifier, which takes in the pooled BERT representations and outputs a logit of sentiment classes. Finally, in this part of the project, we write the `step()` function in the AdamW optimizer, which is used as the optimization algorithm to minimize loss. The rest of the minBERT implementation is provided as base code from the CS224N Default Project Handout (CS224N, 2023).

Following this, the second part of this project consists of creating an effective multi-task model based on our minBERT implementation. The tasks include sentiment analysis, paraphrase detection, and semantic textual similarity. To generate a baseline, we use our minBERT model and train it on each of these tasks simultaneously. We use cross-entropy loss for sentiment analysis, binary cross-entropy loss for paraphrase detection, and the mean squared error for the similarity task. The total loss is generated by using a sum of these three losses. The baseline results obtained are reported in the Results section.

Our extensions to the baseline minBERT implementation are guided by initial and intermediate task-specific performance and overall performance on our three datasets. Specifically, we add gradient surgery (Yu et al., 2020), try various types of regularization, add layers to underfitting tasks, and implement hyperparameter tuning to finalize our model.

Gradient surgery combines the three gradients obtained from our three datasets by projecting a dissimilar gradient onto the normal gradient of another, thereby decreasing the effect of conflicting gradients. This is demonstrated in the equation below, where g_i and g_j are the gradients of the i-th and j-th task.

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

We implement gradient surgery by integrating existing code from Tseng (2020) into our model.

After implementing gradient surgery, we test several regularization techniques to help mitigate overfitting. For overfitting on the sentiment task, we integrated an implementation of SMART (Smoothness-Inducing) regularization (Jiang et al., 2021) code written by the Open Source AI Research Lab (2019). We experiment with SMART due to its ability to effectively combat overfitting without aggressive updates (Jiang et al., 2021). This has been further detailed in the Appendix.

For overfitting on the similarity task, we experiment with l_2 regularization (integrated into AdamW with weight decay) (Loshchilov and Hutter, 2019a), l_1 regularization, and l_∞ regularization. For the l_2 regularization, we add a weight decay value of 0.05 to the AdamW optimizer. For the other two norms, we adjust similarity loss after calculating it using MSE Loss:

$$\text{loss} = \text{mse\_loss}(\text{sts\_logits}) + 0.05 \cdot ||\text{weights}||_p.$$

To address the issue of underfitting (i.e. low accuracy and correlation on both training and development sets), we add linear layers to underfitting tasks to increase the expressivity of our model.

After combining several of these techniques into a model that produces the best results, we experiment manually with batch size, learning rate, weight decay, dropout rate, the number of dropout layers, and activation functions to pick the hyperparameters that optimize model performance.

We also experiment with implementing class weights in the binary cross-entropy loss for sentiment analysis to test if it improves performance on the task and performance per class. The class weights per class (W_i) are calculated using the total number of observation in the training set (TC) and dividing it by number of classes (5) times the total number of observation per class (C_i) in the training set. This is described in the equation below, for class $i = 0, ..., 4$:

$$W_i = TC/(5 * C_i)$$

This adds higher weights to classes with fewer observations and put more emphasis on these classes (Hasan, 2020).

Finally, we experiment with two activation functions between the linear layers of the similarity task. The ReLU (Rectified Linear Unit) activation is selected because it is computationally efficient, widely used, and often yields great results. However, it suffers from the dying ReLU problem, which causes gradients to become zero and networks to stop learning when inputs are close to zero. Since similarity is measured on a scale of 0 to 5, the dying ReLU problem is a cause of concern. Thus, we also experiment with the LeakyReLU activation which combats this problem (Gharat, 2019).

5 Experiments

5.1 Data

For this project, we use four datasets, obtained from the Default Project Handout (CS224N, 2023) Each of these have been detailed below.

- Stanford Sentiment Treebank Dataset: This dataset consists of sentences from movie reviews, which are parsed into phrases and then labeled as 'negative', 'somewhat negative', 'neutral', 'somewhat positive', or 'positive'. There are 11,855 sentences in total, of which approximately 80% (8,544 + 1,101) are used for training and development and the remaining are used for testing. This dataset is used for the sentiment analysis task. Amongst the labels for training and development dataset , 13% are 'negative', 26% are 'somewhat negative', 19% are 'neutral', 27% are 'somewhat positive' and 15% are 'positive'. Thus, there is a slight imbalance in the classes.
- CFIMDB Dataset: This dataset is made up of binary-labeled movie reviews (positive/negative). In total, there are 2,434 reviews. Approximately 80% (1,701 + 245) of the reviews is used for the purposes of model training and development while the remaining is used for testing. This dataset is used for pretraining some of the final layers of the BERT model (in the minBERT implementation stage). However, it is not used in the multi-task modelling process.
- Quora Dataset: This dataset is made up of question pairs, which are labeled as paraphrases or not paraphrases. There are 400,000 question pairs but the project uses a subset consisting of 202,152 examples. Out of these, approximately 80% (141,506 + 20,215) will be used for training and development while the remaining will be used for testing. This dataset is used for paraphrase detection. Within the training and development datasets, 37% of the labels are 1 and 63% are 0. There is an imbalance in the labels.
- SemEval STS Benchmark dataset: This dataset is made up of sentence pairs with labels on a scale between 0 to 5. Here, 0 indicates sentences are unrelated while 5 indicates that sentences are equivalent in meaning. In total, there are 8,628 sentences in the dataset. Out of these, approximately

80% (6,041 + 864) will be used for training and development while the remaining will be used for testing. This dataset will be used for assessing semantic textual similarity. When rounding the labels for the training and development sets, approximately 12% are 0, 14.5% are 1, 17% are 2, 22.5% are 3, 24% are 4, and 10% are 5. There is a slight imbalance in the labels.

5.2 Evaluation method

For the paraphrase detection task and sentiment analysis task, we use accuracy to evaluate the model performance. Since we observe imbalance in both the datasets, we also report balanced accuracy and macro-averaged F1 scores for the final model for both these tasks. For the sentence similarity task, we evaluate model performance using Pearson correlation of the true similarity values against the predicted similarity values. During the experimentation process, we compare these metrics for the training and development sets to check for underfitting or overfitting in the model.

5.3 Experimental details

Following the implementation of minBERT, we implement each of the methods described in the Approach section as well as combinations of the methods that improved performance on the development set to choose the best performing model. For each of the initial model architectures (before hyperparameter tuning), we use the following default parameters.

learning rate	batch size	number of epochs	dropout probability	optimizer	weight decay
1e-5	8	10	0.3	AdamW	0.0

Figure 1: Default Hyperparameter Configuration

For each task, we use a different type of loss. For sentiment analysis, we use cross-entropy loss as it is a multi-class classification problem. For paraphrase detection, we use binary cross-entropy loss as it is a binary classification task. For semantic textual similarity, we use mean squared error as the outcome (similarity) is a float ranging from 0 to 5.

We run experiments with the following configurations:

- gradient surgery
- gradient surgery and increase semantic textual similarity (STS) dropout probability to 0.5
- gradient surgery and use l_2 regularization by adding weight decay of 0.05 in AdamW
- gradient surgery and l_1 regularization on STS loss
- gradient surgery and l_∞ regularization on STS loss
- gradient surgery and add an extra linear layer (plus activation) to each task
- gradient surgery and increase dropout probability to 0.5 for both STS and SST (sentiment analysis)
- gradient surgery, 0.5 STS and SST dropout probability, and SMART regularization for SST
- final architecture

Our final architecture uses gradient surgery, a single linear layer and a dropout layer with probability 0.3 for paraphrase detection, and a single linear layer and dropout layer with probability 0.5 for sentiment analysis. For semantic textual similarity, there are two linear layers with dropout layers before each with probabilities 0.5 and 0.3 respectively. There is also a ReLU activation layer between the two linear layers. We use l_∞ regularization for this task. See Figure 2 for a depiction of the final architecture.

For hyperparameters, we experiment with different learning rates, batch sizes, weight decay, class weights for loss, and activation functions. This has been described in more detail in the Results section. Our final chosen hyperparameters are: learning rate of 5e-5 (for finetune), batch size of 8, ReLU activation function, and no use of weight decay or class weights. Additionally, we continue using 10 epochs and the AdamW optimizer.

Our training time for different model architectures is fairly consistent. It is approximately 3 minutes and 30 seconds per epoch.

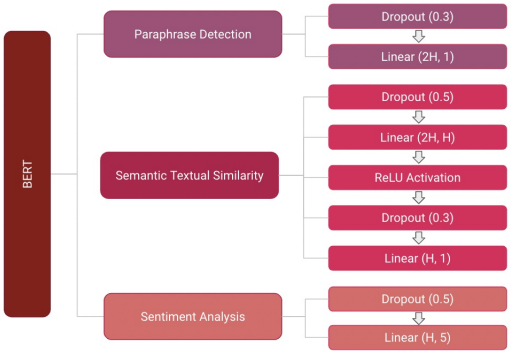

Figure 2: Final Architecture Diagram

5.4 Results

The accuracies and correlation on the training and development sets are reported in Figure 3 from multiple runs of our model with various extensions. Note that “Paraphrase" and “Sentiment" values are accuracies, “sst" refers to sentiment analysis, “sts" refers to semantic textual similarity, and “gs" refers to gradient surgery. Additionally, “final architecture" includes the following: an extra linear and dropout layer and l_∞ for similarity, higher dropout probability (0.5) for both sentiment and similarity, and gradient surgery (see Experimental Details for more details).

	Training Set Accuracy			Development Set Accuracy		
	Paraphrase	Sentiment	STS Correlation	Paraphrase	Sentiment	STS Correlation
Baseline minBERT	0.663	0.337	0.274	0.667	0.347	0.254
gradient surgery	0.756	0.920	0.966	**0.746**	0.494	0.338
gs + increased STS dropout	0.755	0.927	0.961	0.745	0.495	0.362
gs + l_2 regularization	0.711	0.449	0.625	0.704	0.396	0.326
gs + l_1 regularization	0.689	0.470	0.434	0.683	0.423	0.352
gs + l_∞ regularization	0.753	0.927	0.963	**0.746**	0.496	0.363
gs + add one linear layer each	0.703	0.473	0.718	0.700	0.401	0.387
gs + increase dropout (sts & sst)	0.679	0.438	0.398	0.678	0.414	0.338
gs + dropout + SMART(on sst)	0.750	0.273	0.959	0.739	0.258	0.335
final architecture	0.723	0.728	0.873	0.720	**0.513**	**0.396**

Figure 3: Training and Development scores for all three tasks, with experimentation in model architecture. ‘Final architecture’ is defined above.

On the test set leaderboard, we achieved the following results with our final architecture (before tuning hyperparameters):

Paraphrase test accuracy: 0.718, sentiment test accuracy: 0.518, similarity test correlation: 0.363.

Gradient Surgery: Adding gradient surgery improves performance in all three tasks. This is expected because it mitigates the effect of conflicting gradients “cancelling each other out" (Yu et al., 2020). Instead, it projects conflicting gradients in order to make them more similar to each other, thereby allowing for decreases to the loss of both tasks.

Increasing Dropout: As expected, increasing the dropout probability for the semantic textual similarity task reduces the problem of overfitting and increases correlation on the development set. We later increase dropout for sentiment analysis as well, since this task is also overfitting on the training data. In our final architecture, this increased dropout does appear to help increase the development set accuracy for sentiment and similarity tasks.

Regularization: Interestingly, none of our regularization efforts helped to significantly improve performance on the development set. As expected, l_2 and l_1 regularization do help to mitigate

overfitting on the STS task. However, these models do worse than the basic gradient surgery plus increased dropout. The same is true when using SMART regularization for the sentiment task. SMART is designed to prevent aggressive updating and thus, we experiment with it to reduce overfitting (Jiang et al., 2021). However, it does not improve results. The low results for SMART regularization may also be due to the fact that sentiment analysis had two regularization techniques (increased dropout and SMART) being applied together. Using l_∞ regularization did not effectively reduce overfitting, likely because minimizing the l_∞-norm of the weights will not affect most weight values; it does not encourage small values and instead only encourages minimization of one weight at a time (the maximal element) (StClair, 2023). Even so, using l_∞ regularization does noticeably improve performance on the similarity task, so we retain it in our final model. Since the l_1, l_2, and SMART regularization techniques result in worse performance (though l_2 and l_1 do help with overfitting), it is necessary to add expressivity to our model through building more layers.

Adding Layers: To increase the expressivity of our model, we first experiment with adding a second linear layer (with ReLU activation in between) to each of our tasks. This ultimately only improves the score for the similarity task, so we only retain the extra linear layer for similarity in our final architecture.

	Training Set Accuracy			Development Set Accuracy		
	Paraphrase	Sentiment	STS Correlation	Paraphrase	Sentiment	STS Correlation
batch size = 16	0.732	0.776	0.932	0.728	**0.519**	0.353
weight decay = 0.1	0.700	0.408	0.545	0.692	0.365	0.282
wd = 0.1, lr = 5e-6	0.678	0.400	0.438	0.674	0.381	0.347
wd = 0.05, lr = 5e-6	0.730	0.760	0.934	0.728	0.513	0.372
lr = 5e-6	0.730	0.760	0.934	0.728	0.513	0.372
lr = 5e-5	0.735	0.767	0.826	0.728	0.516	**0.438**
lr = 3e-5	0.750	0.858	0.923	0.741	0.490	0.432
lr = 6e-5	0.749	0.857	0.908	0.740	0.485	0.425
lr = 5e-5, weighted loss	0.753	0.899	0.921	**0.744**	0.477	0.426
lr = 5e-5, LeakyReLU	0.759	0.888	0.939	0.744	0.492	0.456

Figure 4: Training and Development scores for hyperparameter tuning with the final architecture. Highlighted row indicates the final model chosen. Here, “wd" indicates weight decay, “lr" indicates learning rate and weighted loss indicates the use of class weights in cross-entropy loss for sentiment analysis. Unless otherwise specified, ReLU is the activation function used.

On the test leaderboard for our final architecture and a learning rate of 5e-5 (the only hyperparameter adjustment), we obtain the following results:

Paraphrase test accuracy: 0.726, sentiment test accuracy: 0.533, similarity test correlation: 0.427.

Additionally, for our final architecture, we also obtain the following metrics on the development set: paraphrase detection F1 score of 0.694 and balanced accuracy of 68.8%, and sentiment classification F1 score of 0.485 and balanced accuracy of 48%. These values are slightly lower than the typical accuracy scores obtained on the development set.

Hyperparameter Tuning:

- Learning rate: In addition to the default learning rate of 1e-5, we test learning rates of 5e-5 and 3e-5 for finetuning, as suggested in the original BERT paper (Devlin et al., 2019). We also try 6e-5 and 5e-6. The higher learning rate of 5e-5 helps to achieve a significantly higher score for STS correlation specifically. Therefore, we choose this learning rate for our final model.
- Weight decay: Apart from the default zero weight decay, we also test weight decay values of 0.1 and 0.05, which, as described in the regularization section, is a form of l_2 regularization for all of the tasks. Weight decay helps with the issue of overfitting but ultimately lowers the overall model performance, even when training for a larger number of epochs. This may be due to lack of expressivity of the model.

- Batch size: Following the suggestions in the original BERT paper (Devlin et al., 2019), we test a larger batch size of 16 (as compared to the default 8). Because this does not improve our overall scores, we do not train further with larger batch sizes.
- Epochs: During training, we observe that model performance did not improve after 6-9 epochs. Thus, we use 10 epochs which is the default.
- Optimizer: We use the default optimizer - AdamW. This is because Adam is computationally faster, inherits the advantages of other optimizers, and performs well for deep learning tasks (Gupta, 2023). AdamW uses an enhanced and optimized way to incorporate weight decay (Loshchilov and Hutter, 2019b). Thus, it is chosen.

After finalizing these hyperparameters, we also experiment with the following:

- Class weights: Due to the imbalance in the STS dataset and the per-class analysis presented in the Analysis section, we try using class weights when calculating the cross-entropy loss for the sentiment analysis task. This is done with the intent of improving accuracy across all classes, as detailed in the Approach section. However, this technique does not significantly improve results and thus, it is not used in the final model.
- Activation functions: In between the two linear layers for the similarity task, we test the ReLU and LeakyReLU activations. The model performs slightly better with the LeakyReLU activation function on the development set. However, on the test set, using ReLU provides better results. Thus, we use the ReLU activation function.

True Label	Paraphrase		Sentiment		Similarity	
	Accuracy	% of Values	Accuracy	% of Values	Accuracy	% of Values
0	0.721	62.5%	0.307	12.7%	0.139	11.7%
1	0.316	37.5%	0.727	26.2%	0.333	11.8%
2	X	X	0.301	20.8%	0.448	16.6%
3	X	X	0.620	25.3%	0.314	24.3%
4	X	X	0.448	15.0%	0.189	24.5%
5	X	X	X	X	0.000	11.1%

Figure 5: Accuracies by label for each task on the development sets. Note that similarity correlation scores are rounded to the nearest whole number. Percent of values refers to percent of datapoints in each task with the specified true label. See Appendix for visualization of these results.

6 Analysis

The final architecture for our model is chosen to maximize the effectiveness of our extensions to the minBERT baseline. There is significant improvement from our baseline model's performance on the development set (paraphrase 0.667, sentiment 0.347, and similarity 0.254) to our final model's performance on the development set (paraphrase 0.728, sentiment 0.516, and similarity 0.438). Notably, our greatest improvements are in the sentiment analysis and similarity tasks; we focus our extensions on these tasks because their baseline scores were much lower than that of paraphrase detection.

Overall, the most significant improvement in model performance resulted from the addition of gradient surgery to mitigate the negative effects of conflicting gradients for the three tasks. This improvement indicates that the gradients for the three tasks likely were conflicting and therefore the baseline without gradient surgery was not able to train effectively for any of the three tasks.

We train the model with several regularization techniques, including dropout layers and l_2, l_1, l_∞, and SMART regularization as discussed in the Results section. We choose to target semantic textual similarity for initial regularization efforts because it was overfitting most severely (see Figure 3). We find that l_2 regularization (increasing the weight decay for the AdamW optimizer), does reduce overfitting for both sentiment and similarity tasks, but ultimately reduces model performance for these two tasks. This may be due to low overall expressivity of the model. However, we also see that

adding a linear layer to each task does not improve the overall performance of our model. It does improve the similarity correlation, suggesting that our initial configuration was not expressive enough for the similarity task. To combat overfitting in the sentiment analysis task, we increased the dropout rate, allowing for better generalization to the development set and test set. We also see that SMART regularization does not improve results for the sentiment analysis task when used in addition to a higher dropout rate. Therefore, we choose the simpler solution of dropout rate as opposed to the more complicated SMART regularization tool for our final model.

Ultimately, through our experiments, we attempt to simultaneously improve the expressivity and decrease overfitting on our three tasks, specifically targeting the low-performing sentiment analysis and similarity tasks. The added expressivity and regularization improve correlation for the similarity task over the baseline, and the higher dropout for sentiment classification allows for higher accuracy scores for this task as well. Interestingly, the paraphrase detection performs best when there are fewer additions to the model (after the initial improvement when implementing gradient surgery). This result indicates that these tasks may be somewhat conflicting. As we see with the implementation of gradient surgery, the gradients may have been conflicting for the three tasks. Although gradient surgery improves this issue significantly, it may still be that performing well on the other two tasks results in somewhat lower performance on paraphrase detection. This is an inherent challenge in multi-task learning. Even so, more focus on improving the model for the paraphrase detection task may allow for increased paraphrase detection accuracy.

We further explore our results by manually calculating per-class accuracies for each task (see Figure 5). This is motivated by the slighlty lower macro-averaged F1 and balanced accuracy scores on our final model compared to the regular accuracy (and correlation) scores obtained. Overall, we see that classes with a higher number of observations have higher accuracies. This is expected, as the model is better able to learn when to predict a class label if it has seen more examples from that class. Therefore, although we would like to see high accuracies in all classes, these results are unsurprising. It is somewhat surprising that class five of similarity (true scores between 4.5 and 5.0) has an accuracy of 0.000. That is, our model never predicts a similarity higher than 4.5, and upon inspection of the predicted values, we see that it never predicts a similarity higher than 4.2. This is an interesting result that merits further study.

7 Limitations and Extensions

Our final model shows significant improvement over the baseline model. However, there are still possible improvements that can be made, specifically to the performance of the sentiment analysis and semantic textual similarity tasks. For the paraphrase detection task, performance can be improved on the class level. Additionally, even after experimenting with multiple techniques, our model shows signs of overfitting on the semantic textual similarity task and the sentiment analysis task. Though incorporating weight decay helped to combat overfitting, it negatively impacted overall model performance. Thus, further work can focus on combating overfitting more effectively.

For the purpose of this paper, we experiment with a multitude of techniques and model architectures. To improve performance, there are more techniques that can be tested. These include but are not limited to pretraining the BERT model on more datasets, using different optimizers (Adagrad, RMSProp, SGD), using cosine similarity techniques, etc. We encourage further work to also focus on improving performance on each class for the different tasks.

8 Conclusion

This project explores the use of extensions to a baseline minBERT model to simultaneously improve performance on three downstream tasks - paraphrase detection, sentiment analysis and semantic textual similarity. Our implementations of various types of regularization, different model architectures, gradient surgery, and hyperparameter tuning are guided by intermediate accuracies and patterns of overfitting on the training sets. We find that gradient surgery greatly improves results for multitask learning, and further slight improvements can be made by adjusting types of regularization, dropout probability, and learning rates. Our final model, which incorporates gradient surgery, increased dropout rates and an extra linear layer, dropout layer, and l_∞ regularization on the similarity task, is able to improve substantially upon the baseline results.

References

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

CS224N. 2023. Cs 224n: Default final project: Minbert and downstream tasks.

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

Snehal Gharat. 2019. What, why and which? activation functions.

Ayush Gupta. 2023. Optimizers in deep learning: A comprehensive guide.

Faiyaz Hasan. 2020. Deep learning with weighted cross entropy loss on imbalanced tabular data using fastai.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2021. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization.

Open Source AI Research Lab. 2019. Smart-pytorch.

Ilya Loshchilov and Frank Hutter. 2019a. Decoupled weight decay regularization.

Ilya Loshchilov and Frank Hutter. 2019b. Decoupled weight decay regularization.

Rachel StClair. 2023. Introduction to vector norms: L0, l1, l2, l-infinity.

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. In *Advances in Neural Information Processing Systems*, volume 33, pages 5824–5836. Curran Associates, Inc.

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

A SMART Regularization Equations

Jiang et al. (2021), present the equation below for Smoothness-Inducing Adversarial Regularization (SMART).

$$\min_{\theta} F(\theta) = L(\theta) + \lambda_s R_s(\theta)$$

Here, λ_s> 0 represents a tuning parameter, $R_s(\theta)$ represents smoothness-inducing adversarial regularizer and $L(\theta)$ represents the loss function.

The equation for the loss function is presented below. Note that $l(\cdot, \cdot)$ represents the loss function for the target task, x_i, y_i denote the sentence embeddings and associated labels respectively, $f(\cdot; \theta)$ is the model and n represents the data points of the target task.

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i, \theta), y_i)$$

The equation for the regularizer $R_s(\theta)$ is presented below. Here, the tuning parameter is $\epsilon > 0$.

$$R_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{||\tilde{x_i} - x_i||_p \leq \epsilon} l_s(f(\tilde{x_i}, \theta), f(x_i, \theta))$$

B Figures

The figures in this section show the results from Figure 5, by task.

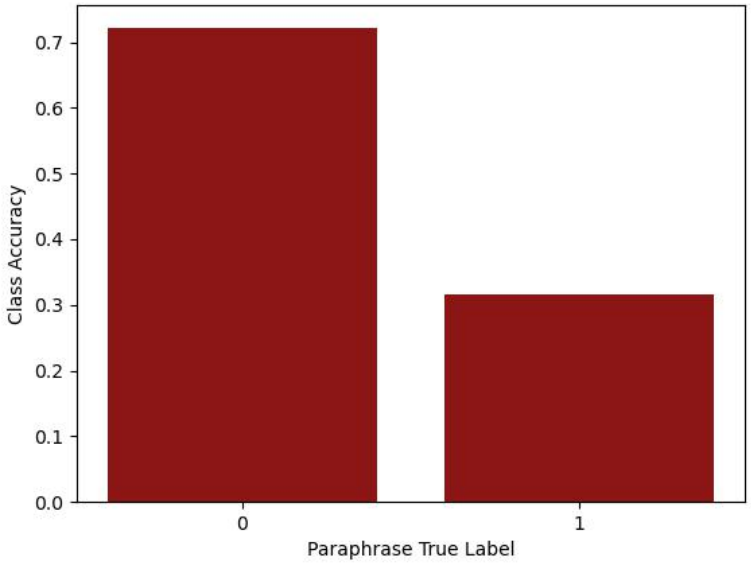

Figure 6: Paraphrase detection accuracy on the development set by true label.

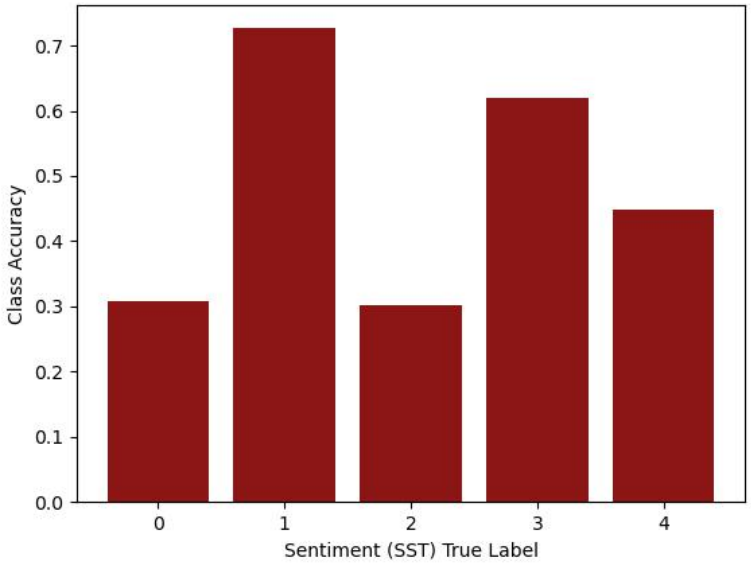

Figure 7: Sentiment analysis accuracy on the development set by true label.

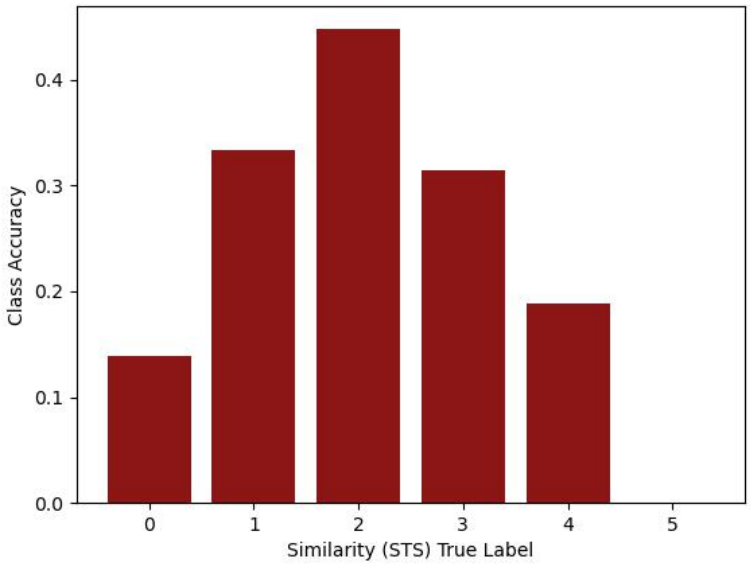

Figure 8: Semantic Textual Similarity accuracy on the development set by true label.

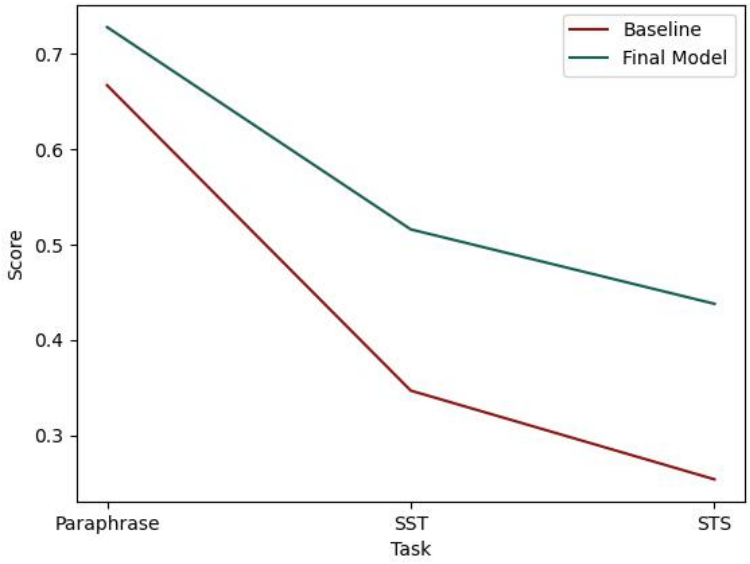

Figure 9: Improvement in Scores from Baseline to the Final Model, per task.