BERT Multi-Task Cosine Surgery: Applying Cosine Similarity and Gradient Surgery in a BERT Multi-Task Fine-Tuning Setting

Stanford CS224N Default Project

Graciela Smet
Department of Computer Science
Stanford University
gsmet@stanford.edu

Nick Walker
Department of Computer Science
Stanford University
nhwalk13@stanford.edu

Abstract

When optimizing for performance across multiple tasks, separate models could be trained for each or one model could be used to address the tasks simultaneously. Previous literature has shown that learning efficiency and performance can be improved in the latter approach as related tasks are trained together. In this paper, we explore a round robin training implementation of multi-task learning in a BERT fine-tuning setting, attempting to achieve the best development and test accuracy across our three tasks of sentiment analysis, paraphrase detection, and semantic textual similarity (STS). We use the base BERT model parameters without fine-tuning, simply pushing them through untrained linear layers of the appropriate size, to establish a baseline. To perform the different tasks, our model includes separate classification "heads" that use the BERT embedding outputs to ouput a label. We experiment with three model variations in which we add the following to round robin: gradient surgery, leveraging cosine similarity in the STS head, and combining these two approaches. Additionally, we perform hyperparameter tuning experiments within the gradient surgery and cosine similarity variations, and combined the best hyperparameter choices into a singular "holy grail" model for each. Overall, we found that the gradient surgery and cosine similarity approaches greatly improved performance but that the combination of the two led to worse performance. These findings support gradient surgery's success in mitigating the conflicting gradient problem in multi-task settings and cosine similarity's effectiveness as a metric in semantic textual similarity in a BERT fine-tuning multi-task setting. Furthermore, we contribute to existing literature by finding that the combination of the two approaches yields worse results and by finding hyperparameters optimizing the task at hand.

1 Key Information to include

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

2 Introduction

As machine learning engineers have succeeded in creating narrow models that perform well at a singular task, the next frontier has been exploring singular models that perform well at several tasks at once. This is what we aim to do with this paper: generalizing a basic BERT model to perform well in the 3 tasks of sentiment classification, paraphrase detection, and similarity detection, which we attempt to achieve through multitask training with a variety of approaches to determine the best combination of hyperparameters and prediction strategies. While there are a myriad of potential options to improve BERT's performance on such tasks, we chose a very specific subset and created a thorough experiment plan to optimize our performance and develop a strong multitasking strategy. Our general approach involved implementing round-robin multi-task training for 3 separate classification heads and to run fine-tuning to propagate our changes throughout the BERT model so that the embeddings perform better in our tasks. Our motivation behind this approach was to create well-tuned prediction heads that generalize well to our selected tasks. First, we were inspired by related work on cosine similarity to explore a variation of this approach that used cosine similarity in the head of the similarity detection head instead of a linear layer. Then, we were inspired by related work on gradient surgery to explore using this new technique instead of loss sum to update our model's parameters. Finally, we attempted combining these two variations. Throughout this process, we conducted thorough hyperparameter tuning investigations to determine how different parameters affected our performance, noting how these impacts differed between the two variations and attempting to determine the best strategy combining the various techniques we explored.

3 Related Work

With respect to multi-task learning, previous researchers have cited its positive effects for performance and data efficiency (Crawshaw, 2020). Multi-task learning can help related tasks learn together efficiently and is especially helpful for tasks in which not much data is available.

Additionally, researchers have experimented with gradient surgery in multi-task learning and applied cosine similarity approaches within the STS head. Namely, gradient surgery for multi-task learning was explored (Yu et al., 2020) and shown to help solve the issue of conflicting task gradients in the multi-task setting. Since the separate tasks don't necessarily have gradients that all are non-conflicting (i.e. have non-negative cosine similarity), they can work against one another when the model is attempting to learn parameters that will perform well on each task. To solve this, when tasks have gradients that conflict, we project the gradient of each task onto the normal plane of the gradient in the other task so that the conflicting component is removed. This results in less interference between the gradients when the model is learning and we can expect better results. Furthermore, cosine similarity related techniques were explored in (Reimers et al., 2019) in which the STS task was approached through the use of CosineEmbeddingLoss in which sentences that are equivalent have a cosine similarity of 1 and those that are unrelated have a cosine similarity score of 0.

Looking at this previous work in which multi-task learning was shown to be effective and gradient surgery and CosineEmbeddingLoss separately improved deep learning NLP tasks, we explore approaches that combine these. In our first improvement over the baseline, we use round robin in a multi-task setting involving our three tasks of interest, taking inspiration from the previous work showing multi-task learning to be effective. With respect to the work regarding cosine, we don't leverage CosineEmbeddingLoss, but in our first model variation we do use cosine similarity with MSE loss within the STS head to enjoy some of the same performance benefits reported in the previous work cited above when cosine similarity was involved in the form of CosineEmbeddingLoss. Meanwhile, in our second variation we apply gradient surgery on top of round robin to rederive the benefits found in the original paper. Finally, we take inspiration from all of our approaches inspired by these separate works and combine them in our third model variation.

4 Approach

Our approach to the problem of improving base BERT's performance on involved experimenting with different kinds of heads for the 3 tasks, different approaches to gradients, and adjusting our approach as we went to reflect the most successful strategies.

- **General Approach: Linear Heads With Round Robin Multi-Task Training**
  For our first approach, we implemented the 3 task heads using almost identical approaches. The sentiment classification was the blueprint: generating embeddings using BERT, applying a dropout layer, then using a linear layer to output logits for the 5 sentiment classes. The paraphrase and similarity detection heads both followed a similar strategy to handle the two-sentence input and output a singular logit: we began by concatenating the two input_ids and the two attention_masks, then generated the embedding using BERT, applied dropout, and finally used a linear layer to output a singular logit. We used cross-entropy loss for the sentiment classification head, binary cross-entropy loss for the paraphrase loss, and MSE loss for the similarity loss. In each training epoch, we evaluated a batch from each of the 3 dataloaders, added together the 3 losses, then took a step, evaluating our model on the dev sets in each epoch to determine which model to save.
- **Variation 1: Introducing Cosine Similarity to the Semantic Text Similarity (STS) Head**
  Next, we experimented with introducing cosine similarity into the semantic textual similarity heads instead of using a linear layer. For this approach, we gathered BERT embeddings for the two input_ids and the two attention_masks separately, then calculated the cosine similarity and clamped its output using ReLU. Finally, we scaled this output by 5 to match the expected label outputs. As before, our gradient was updated after each set of batches. We experimented first with using this approach on both sentiment and paraphrase heads but quickly discovered it only provided benefits in the sentiment prediction, and so centered our experiments around this. We then experimented with adding learnable linear and nonlinear layers after fetching the embeddings and before calculating the cosine similarity to see if these extra parameters improved performance, with increasing epochs, and with different dropout rates.
- **Variation 2: Introducing Gradient Surgery**
  Next, we experimented with using gradient surgery to update our parameters instead of the loss sum used in the previous approaches. We used Wei-Cheng Tseng's public GitHub repo to implement this approach. Here, all the classification heads remained using the linear layer approach: gathering a single embedding (by concatening two sentences when necessary), applying dropout, and applying a linear layer to output the correct number of logits. Similarly to approach 2, we also experimented with increased epochs, dropout parameter tuning, and adding learnable linear and nonlinear layers before doing the final classification to see if extra parameters increased performance.
- **Variation 3: Combining STS Head Cosine Similarity and Gradient Surgery**
  Finally, we experimented with combining both variations 1 and 2 into a singular model that combines the new cosine similarity approach for STS with the gradient surgery technique to evaluate combined performance.

5 Experiments

5.1 Data

We used three datasets to train and evaluate our model, one for each task in our BERT model, which were all provided in the default starter code:

- We used the Stanford Sentiment Treebank (SST) dataset to fine-tune and test our model for sentiment classification. Each example in this dataset is a single sentence with a discrete label from 0 to 5 to reflect how negative (0) or how positive (5) the sentence is.

- We used the Quora Dataset to fine-tune and test our model for paraphrase detection. Each example in this dataset is a pair of two sentences labeled discretely with a 0 or a 1 to reflect whether the sentences are/aren't paraphrases of one another.
- We used the SemEval STS Benchmark Dataset to fine-tune and test our model for sentence similarity. Each example in this dataset is a pair of two sentences labeled with a continuous label from 0 to 5 to reflect how similar (5) or dissimilar (0) the sentences are.

5.2 Evaluation method

We decided to use a simple qualitative evaluation method to assess the performance of our various models on the three tasks: the overall accuracy on the dev set of each included dataset. For sentiment classification and paraphrase detection, we calculated our accuracy using the number of labels we correctly predicted, and for similarity detection we calculated accuracy using the Pearson correlation coefficient.

5.3 Experimental details

We ran a series of experiments guided by the first two variations detailed above. Within these variations, we explored hyperparameter tuning through adjusting the number of epochs, the dropout rate, and the amount of added neural network layers within the classification heads. To isolate these parameters for exploration, for each of the approaches we did a series of experiments freezing all parameters but one to discover how we might improve further on our approaches.

For the epoch parameter, we experimented with running 10 and 20 epochs in each model variation. For the dropout parameter (used in our prediction heads), we experimented with dropout rates of 0.3, 0.5, and 0.8 for each model variation. Finally, for additional learnable layers (we defined one "learnable layer" as applying a linear layer, a ReLU, then a dropout layer), we experimented with adding 1 or 2 learnable layers to each model variation.

After finding which parameters performed the best in each variation, we created a "holy grail" for each model variation using the best parameters to see if we could outperform the baseline.

Note that we did not perform these experiments on the third model variation (combining cosine similarity and gradient surgery) because we found that it consistently performed far worse than the other variations. This is discussed shortly below in "Variation 3: Combining the Cosine Head and Gradient Surgery."

5.4 Results

Baseline Approach

For our baseline approach, we set up the linear heads used in the general approach, left them untrained, and simply predicted the labels for the three tasks using the base BERT embeddings. As expected, BERT performed very poorly with the untrained prediction heads, but we will use these accuracies as our baseline to compare how our approach and variations were able to provide performance increases.

Results			
dev SST	dev para	dev STS	Overall
0.144	0.547	-0.019	0.224

Variation 1: Introducing Cosine Similarity to the Semantic Text Similarity Head

Hyperparamater tuning: Changing dropout rate:

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	0.3	**0.501**	**0.831**	0.676	0.669
10	None	0.5	0.498	0.830	**0.700**	**0.676**
10	None	0.8	0.490	0.828	**0.700**	0.672

Adding the cosine head and training the model was able to immediately improve the performance of our model, as expected. A 0.5 dropout rate likely did the best as it was enough to provide a regularizing effect in the heads while still capturing much of the information from the previous layers. The marginal performance differences between dropout rates is likely because the dropout is only occurring over the relatively shallow classification head neural layers.

Hyperparamater tuning: Changing epochs

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	0.3	**0.501**	0.831	0.676	0.669
20	None	0.3	0.498	**0.857**	**0.726**	**0.693**

By running more epochs, we were able to increase our performance over the previous best established by the dropout parameter tuning. This was what we expected as our model had more opportunities to fine-tune both the BERT parameters and the classification head parameters for the tasks at hand. Overfitting was avoided due to dropout layers and we saw the expected effect of our neural network leveraging the additional training to perform even better. However, we were surprised to see the magnitude of the increase being so great. This is likely due to the high number of parameters needing time to be learned and the relatively low epoch count of 10 didn't provide a sufficient amount of updating to fully leverage the many parameters.

Network tuning: Adding learnable layers
One LL = an added linear layer, ReLU layer, and dropout layer

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	**0.3**	0.501	0.831	0.676	0.669
10	1 LL in STS head	0.3	0.501	**0.845**	**0.793**	**0.713**
10	1 LL in all heads	0.3	**0.511**	0.834	0.669	0.671
10	2 LL in all heads	0.3	0.505	0.841	0.535	0.627

Adding extra learnable layers afforded yet another significant performance improvement. We were surprised that adding a learnable layer to only the STS head was able to yield such a large improvement, which we suspect is because the cosine head in this variation currently has no learnable parameters. In adding this, we allow our model to adjust both the embeddings and the parameter in this layer to increase performance. We suspect that the experiments with even more learnable parameters suffered either from an insufficient number of epochs or from overfitting.

Holy Grail model
For the holy grail model for this approach, we took what we learned from the variable experiments and ran a model for 30 epochs with a learnable layer added to the STS head and with a dropout rate of 0.5 since these were the most effective parameter combinations for this variation.

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	1 LL in STS head	0.5	0.515	0.880	0.558	0.651

We were disappointed to find that our Holy Grail experiment performed worse than the previous best performance for this variation, which achieved when running 10 epochs with extra learnable parameters on the STS head. This is likely because increasing the number of epochs caused overfitting in our data. Additionally, this could be because our method of taking the best hyperparameters from different experiments strips them of the context in which they performed so well. Meshing these things together isn't guaranteed to provide better results, demonstrating the complexity of neural network hyperparameter tuning. Nonetheless, we were still able to see which hyperparameters performed best on their own and leave this as a helpful starting point for future researchers.

Variation 2: Introducing Gradient Surgery

Hyperparamater tuning: Changing dropout rate

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	0.3	0.500	0.853	**0.864**	0.739
10	None	0.5	0.504	**0.855**	0.860	**0.740**
10	None	0.8	**0.515**	0.838	0.862	0.738

Immediately upon implementing gradient surgery with the linear heads, we were able to surpass the previous best accuracies achieved by the first variation. We found again that impact of dropout rate on performance was marginal but nevertheless optimized when set to 0.5, which we expected and believe is because of similar reasons to the hypothesis outlined in variation 1.

Hyperparamater tuning: Changing epochs

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	0.3	0.500	0.853	**0.864**	0.739
20	None	0.3	**0.509**	**0.870**	0.861	**0.747**

Once again, running the experiment for more epochs allowed us to realize a significant improvement in our overall model performance. We were surprised by how quickly the gradient surgery approach was improving upon the original variation, and we again suspect that allowing our model to train for longer allowed it to better adjust the embeddings and learnable layers. While we didn't necessarily see improvements across all individual task performances, we can't exactly determine how the different tasks are affected since we are training multiple tasks at once and increases in performance on one task can be relatively greater than those of others. However, gradient surgery helps prevent conflicting gradients which likely allowed for even better results over the additional epochs.

Network tuning: Adding learnable layers
One LL = an added linear layer, ReLU layer, and dropout layer

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	0.3	0.500	0.853	0.864	**0.739**
10	1 LL in all heads	0.3	**0.505**	0.842	**0.866**	0.738
10	2 LL in all heads	0.3	0.488	**0.865**	0.764	**0.739**

Through these experiments, we observed that changing the number of learnable parameters only offered marginal performance improvements, despite us believing that more layers would allow the network to perform even better. We suspect that either these added layers haven't had enough time to learn and that more training time will improve the performance or that different approaches than those in this paper must be tried (i.e. we've nearly maximized performance for this particular experiment).

Holy Grail model
For the holy grail model for this approach, we took what we learned from the variable experiments and ran a model for 30 epochs and with a dropout rate of 0.5. Since the model performed similarly whether we added no additional parameters or if we added 2, we ran the Holy Grail model with 2 additional layers on each, hoping to see the additional epochs help train the additional learnable parameters.

Variables			Fine-tune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
30	2 LL in all heads	0.5	0.501	0.870	0.869	0.747

We were slightly disappointed to find that, while our holy grail model performed very well, it did not surpass the original best score that was achieved when running 20 epochs during the epoch training test. Perhaps this suggests that we simply need more epochs to push past the plateau, and future work might explore running for more epochs or attempting ensembling with randomized restarts, as we may be stuck in a local minima.

Variation 3: Combining the cosine head and gradient surgery

Variables			Finetune Results			
epochs	Head NN Layers	dropout	dev SST	dev para	dev STS	Overall
10	None	0.3	0.518	0.375	0.171	0.355

We were surprised to see that combining the two variations resulted in a model with abysmal performance scores. Because of these initial results, we were uninspired to continue down this route and we leave this space open for further exploration. We suspect that the poor performance was related to the gradient from the STS head being affected/corrupted from the non-PyTorch methods applied within that head to the cosine similarity, like the scaling by 5. Since cosine similarity was used, it's also possible that there was a disconnect between the output of this head (being related to cosine similarity) and the actual gradient that had only to do with the linear layers applied before the cosine similarity was computed.

6 Analysis

Through our thorough experiments for each approach, we were able to generate some key takeaways based on the hyperparamater values to better understand how such changes affected our network. The variation of our model that used gradient surgery was overall the most effective, achieving our highest accuracy scores and placing us relatively high on the multitask performance leaderboards for the dev and test sets of the Default Final Project.

While this performance was exciting, we also learned much about our how our model works through its successes and failures. In both of our variations, increasing the number of epochs had a massive

impact on the performance of the model, whereas changing dropout rate had minimal effects. The first effect is likely because more epochs allows our model to gain a better understanding of the semantic meaning of the training data and has more opportunities to fine-tune the BERT and classification head parameters, while the second effect is likely because various dropout rates would have been sufficient for preventing overfitting here and thus that once we include dropout in the first place, changing its rate has little effect. For the cosine similarity experiments, adding additional learnable parameters in our prediction heads only yielded performance results when added to the similarity predictor, which is likely because the cosine head alone has no learnable parameters in its most basic state, thus adding extra layers allows both the embeddings and the layer parameters to update to increase performance. On the other hand, adding learnable layers to the gradient surgery variation did not have large effects on outcome. This could be because there weren't enough epochs or parameters (we only added a maximum of 2 layers) for the network to take advantage of.

Finally, the combination of the two variations failing shows that simply stacking various techniques on top of each other doesn't always work and our model is not responsive to this.

7 Conclusion

The main findings of our project were discovering the successes and limitations of various combinations of cosine similarity/gradient surgery strategies and hyperparameter adjustments. Through our experiments, we found that we achieved the highest performance on an extended (with classification heads for each task) BERT model that was trained round-robin style using gradient surgery across 3 different datasets and using linear prediction heads with dropout rates of 0.5 over 20 epochs.

In addition to creating a BERT model that performed very well on the 3 tasks, our research and experimentation uncovered some important information about hyperparameter tuning. We discovered that, in a model like this with shallow prediction heads, the selected dropout rate had little effect on the performance regardless of other strategies involved. Through our experiments with the cosine head, we discovered the importance of having learnable parameters in the prediction heads, uncovered by the large performance increase when we added a linear and a non linear layer to the cosine head. Overall, we learned of the importance of high numbers of epochs to give the model a sufficient amount of time to train as well as the power of gradient surgery as an approach for maximizing multitask performance.

Another important takeaway was that combining these two variations (to produce the third variation) did not work, demonstrating that different techniques in neural networks that independently improve performance can't be blindly combined without careful tuning.

Avenues for future work include further explorations of hyperparameter combinations, since we discovered through our Holy Grail models that parameters that performed well in isolated conditions didn't necessarily perform well when merged together. Future research might also include further exploration/tuning of the 3rd variation, which we were unable to deeply explore but could potentially yield high performance with similar hyperparameter tuning.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

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

[4] Wei-Cheng Tseng. Weichengtseng/pytorch-pcgrad, 2020.

[5] Rich Caruana. *Multitask Learning*, pages 95–133. Springer US, Boston, MA, 1998.

[6] Asa Cooper Stickland and Iain Murray. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671, 2019.

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

[8] Chunjie Luo, Jianfeng Zhan, Lei Wang, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. *CoRR*, abs/1702.05870, 2017.

[9] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *CoRR*, abs/2009.09796, 2020.