# Extending the BERT Model to a Multitask Loss Function Using Gradient Surgery

Stanford CS224N Default Project

**Ali Lasemi**
Department of Mechanical Engineering
Stanford University
alasemi@stanford.edu

## Abstract

This paper presents an extension of the Bidirectional Encoder Representations from Transformers (BERT) model through multitask fine-tuning. The model is first pre-trained for the task of sentiment analysis, and then extended to multitask learning by building task-specific heads on top of BERT. The tasks considered are sentiment analysis, paraphrase prediction, and similarity prediction. The extension proposed is a combined loss function incorporating losses from each task, which is trained using the Gradient Surgery procedure. The data used for sentiment analysis comes from the Stanford Sentiment Treebank (SST) and the CFIMDB dataset, while Quora and SemEval STS Benchmark Dataset are used for paraphrase prediction and similarity prediction, respectively. The results of the experiments are still in progress.

## 1 Key Information to include

- External collaborators (if you have any): N/A
- External mentor (if you have any): N/A
- Sharing project: N/A

## 2 Introduction

In natural language processing, it is often desirable to be able to handle inputs and problems of various types. Modern NLP systems are capable of solving a variety of user-specified problems, instead of being specifically trained for a certain task. An example of this is the GPT-3 language model (Brown et al., 2020). In general, this problem can be challenging since different NLP tasks can be completely unrelated, so model weights trained for one task may not be useful for others.

In this work, the goal is to train an NLP model which is a capable of multitask learning. Specifically, three tasks considered are sentiment analysis, paraphrase detection, and semantic textual similarity (STS). The key ideas of the method are to start with a transformer-based pretrained model, and to build task-specific heads on top of this model which can be trained with task-specific data. Instead of training the heads separately, the heads will all be trained together using a combined loss function along with a technique for avoiding conflicting gradients. This is described in more detail in Section 4.

## 3 Related Work

The Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) model is an important example of a transformer-based pretrained model which can be built upon with task-specific heads. BERT is trained on a large amount of unlabeled text, allowing it to generate a

hidden state representation which can then be passed on to other layers (designed for the specific task) which can be trained separately. The advantage of this is that the BERT weights themselves do not necessarily need to be trained, so training can be simplified to the more computationally efficient task of training just the parameters in the task-specific heads.

Another important advancement used in this work is the concept of combining multiple loss functions, and updating them using gradients that are modified with Gradient Surgery (Yu et al., 2020). The challenge with combining loss functions of unrelated tasks is that the gradients from these losses can have components which are conflicting, which impedes the training and convergence. The Gradient Surgery procedure subtracts the conflicting component of each task's gradient, resulting in a combined gradient which contains only the non-conflicting components.

## 4 Approach

The model being used is the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018). For optimization, the Adam optimization algorithm will be used. First, the model will be tested via pretraining for the specific task of sentiment analysis. Afterwards, to extend the model to multitask learning, the pre-trained weights from the BERT model will be used as an initial condition with which to train task-specific heads built on top of BERT. For this baseline model, three tasks are considered. For the first task of sentiment analysis, the head considered will be to take the embeddings from BERT, apply dropout, and then apply a linear layer to reduce the output size to the number of sentiment classes. For both the second and third tasks of paraphrase prediction and similarity prediction, the head structure is a weighted dot product. This involves computing the embeddings of each of the two input sentences, applying dropout, then computing $\vec{a}W\vec{b}^T$, where $\vec{a}$ and $\vec{b}$ are the two embeddings (post-dropout) and $W$ represents the weights.

To train the task-specific heads that are built on top of BERT, the proposed extension to the model is to train again with the BERT weights held constant, using a combined loss function which sums the losses from each task.

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{SST} + \mathcal{L}_{Para} + \mathcal{L}_{STS} \tag{1}$$

The combined gradients from summing the loss functions may contradict each other, so simply summing the gradients is not the best solution. To improve this, the Gradient Surgery (Yu et al., 2020) procedure will be used to align the gradients together. This algorithm is summarized below, in a slightly different order than the algorithm of the original authors, but with the same results.

1. Loop over each parameter $\theta_k$
   (a) Compute the gradients for each task-specific loss: $\nabla_{\theta_k}\mathcal{L}_{SST}$, $\nabla_{\theta_k}\mathcal{L}_{Para}$, $\nabla_{\theta_k}\mathcal{L}_{STS}$
   (b) Create copies: $\nabla_{\theta_k}\mathcal{L}^c_{SST}$, $\nabla_{\theta_k}\mathcal{L}^c_{Para}$, $\nabla_{\theta_k}\mathcal{L}^c_{STS}$
   (c) Loop over the three original gradients, $\vec{g}_i$:
      i. Loop over the copied gradients in random order, $\vec{g}_j$:
         A. Compute the similarity, $s = \vec{g}_i \cdot \vec{g}_j$
         B. If the similarity is negative, then perform Gradient Surgery:

$$\vec{g}_i = \vec{g}_i - \frac{\vec{g}_i \cdot \vec{g}_j}{||\vec{g}_i||^2} \cdot \vec{g}_j \tag{2}$$

One of the challenges with this implementation is that the above algorithm is applied to each minibatch. Therefore, in order to have the three losses defined for each minibatch iteration, there need to be an equal number of minibatches across all three tasks. This was handled by trimming random examples from the datasets to reduce their size down to the smallest (which happens to be the STS dataset).

The BERT implementation and Adam optimizer have been coded by the author, with a framework provided by the CS224N course staff. The code for the multitask loss function and Gradient Surgery are implemented by the author.

## 5 Experiments

This section begins by describing the datasets used. Next, the evaluation metrics for the model are described. The experimental details and hyperparameter tuning will follow. The section concludes with a report of the results of the model.

### 5.1 Data

The data being used for sentiment analysis comes from the Stanford Sentiment Treebank (SST) (Socher et al., 2013) and the CFIMDB dataset. The SST dataset has 11,855 examples, and the CFIMDB dataset has 2,434 examples. The data used for the paraphrase detection task is from Quora (Quo), and consists of 202,152 examples. Finally, the data used for STS task is the SemEval STS Benchmark Dataset (Agirre et al., 2013), and consists of 8,628 examples. The splits in the dataset are given in Table 1.

Table 1: Dataset sizing and splits across train, dev, and test sets

| | Training Set Examples | Dev Set Examples | Test Set Examples |
|---|---|---|---|
| SST Dataset | 8,544 | 1,101 | 2,210 |
| CFIMBD Dataset | 1,701 | 245 | 488 |
| Quora Dataset | 141,506 | 20,215 | 40,431 |
| SemEval STS Dataset | 6,041 | 864 | 1,726 |

### 5.2 Evaluation method

For the sentiment analysis task, the automatic evaluation metric will be the accuracy, since the sentiment is either positive or negative. Likewise, for paraphrase detection, the evaluation metric will still be accuracy for the same reason. For the STS dataset, the evaluation metric will be the Pearson correlation of the similarity values on the test set.

### 5.3 Experimental details

First, the single-task BERT implementation is tested by training on the sentiment task. In this case, a set of working hyperparameters was a batch size of 8, learning rate of $10^{-3}$ (when training from scratch) or $10^{-5}$ (when finetuning), a dropout probability of .3, and a hidden layer size of $768$. The model with the highest dev set accuracy out of all epochs was chosen as the best model. For this simple single-task test, further searching of the hyperparameter space is not performed; this is reserved for the multitask model.

In the multitask setting, the hyperparameter space is explored more thoroughly, and this was found to have important effects on the result. The hyperparameters which are tuned are the batch size, the learning rate, and the number of hidden layers in the multitask forward layer. With these three hyperparameters, to truly find an optimal combination would require training $N^3$ models (where $N$ is the number of trials used for each hyperparameter). Since the model is expensive to train, this is not efficient, so an alternative method is used where hyperparameters are tuned one at a time in sequence, therefore requiring $3N$ models to be trained. All of the hyperparameter tuning is done with Gradient Surgery enabled.

First, the batch size is tuned, with the learning rate held constant at $10^{-3}$ and using one hidden layer in the multitask forward layer. For the batch size, the focus is mainly on choosing the batch size that is most efficient on the hardware accelerator being used (a NVIDIA A10G GPU from Amazon AWS), since the batch size affects the amount of data processed by the GPU at once. Typically, larger batch sizes are preferred in order to saturate the GPU as much as possible, but in testing it was found that a batch size which is too large also has performance drawbacks. The test was performed by running two epochs at various batch sizes, and averaging the compute time to get a cost per epoch. A table of the timings from this test is given in Table 2. These results show that a batch size of 128 gives the best performance on average, so a batch size of 128 is used for training moving forward.

With the batch size and number of hidden layers held constant, the next hyperparameter to be tuned is the learning rate. Models are trained at various learning rates, and the results of this are shown

Table 2: Computational cost of training for various choices of batch size

| Batch Size | Epoch 1 Time (s) | Epoch 2 Time (s) | Average Time (s) |
|---|---|---|---|
| 32 | 109 | 108 | 108.5 |
| 64 | 99 | 99 | 99 |
| 128 | 95 | 94 | 94.5 |
| 256 | 96 | 96 | 96 |
| 512 | 97 | 96 | 96.5 |

in Table 3. These results show a large difference in the performance metrics between the lower two learning rates versus the higher two learning rates. Even though the higher two learning rates achieved good performance on the paraphrase detection task (over .6, compared to about .38 for the lower learning rates), they performed worse on the other two tasks. Furthermore, the convergence of the loss and performance metrics over epochs is found to be much slower at the two highest learning rates, so in the interest of converging the model as fast as possible, the two highest learning rates are discarded. Between the two lowest learning rates, the lowest rate of $5 \times 10^{-4}$ has better performance on the metrics, so this learning rate is chosen for all models moving forward.

Table 3: Model performance metrics for various choices of learning rate

| Learning Rate | Sentiment Accuracy | Paraphrase Detection Accuracy | STS Correlation |
|---|---|---|---|
| $5 \times 10^{-4}$ | 0.317 | 0.378 | 0.166 |
| $1 \times 10^{-3}$ | 0.297 | 0.379 | 0.122 |
| $2 \times 10^{-3}$ | 0.262 | 0.626 | 0.059 |
| $5 \times 10^{-3}$ | 0.262 | 0.627 | 0.034 |

The remaining hyperparameter is the number of hidden layers used in the multitask forward layer. Models were trained using one, two, and three hidden layers, and the results are shown in Table 4. These results show that using more hidden layers improves the sentiment classification accuracy and the STS correlation, and has a minor effect on the paraphrase detection accuracy. Therefore, the best model used the most hidden layers, so moving forward the model will use three hidden layers in the multitask forward layer.

Table 4: Model performance metrics for up to three hidden layers

| # Hidden Layers | Sentiment Accuracy | Paraphrase Detection Accuracy | STS Correlation |
|---|---|---|---|
| 1 | 0.317 | 0.378 | 0.166 |
| 2 | 0.320 | 0.377 | 0.199 |
| 3 | 0.388 | 0.375 | 0.202 |

### 5.4 Results

For the first portion of this work, where the BERT implementation is tested with both pretraining and finetuning, the results are shown in Table 5. As expected, for both datasets the results of finetuning BERT result in a higher accuracy that pretraining the weights from scratch.

The baseline model uses the multitask heads implemented on BERT, as well as the multitask loss function. The improved model is the same, but with the addition of Gradient Surgery. Both the baseline and improved models use the optimal hyperparameters found earlier (batch size of $128$, learning rate of $5 \times 10^{-4}$, and three hidden layers in the multitask forward layer. The results comparing these two are shown in Table 6. The result shows that Gradient Surgery improves the accuracy for sentiment classification accuracy and STS correlation, and has no notable impact on paraphrase detection. Since it was expected for the performance to improve on all tasks, this is considered to be a partial success. The lack of improvement for paraphrase detection requires further investigation. This may be because the majority of the information used for the paraphrase task is stored in the paraphrase layer parameters, and not much information comes from the multitask forward layer. Since the forward layer is the only one shared between the losses, only those parameters are affected

Table 5: Dev set accuracy for pretraining and finetuning BERT on the SST and CFIMBD datasets

| | Pretrain, SST | Pretrain, CFIMDB | Finetune, SST | Finetune, CFIMDB |
|---|---|---|---|---|
| Dev Set Accuracy | 0.410 | .784 | 0.521 | 0.971 |

by Gradient Surgery. Finally, the results of the model with gradient surgery evaluated on the test set are given in Table 7.

Table 6: Dev set performance metrics on the three tasks for the baseline model and the improved model with gradient surgery

| | Sentiment Analysis Accuracy | Paraphrase Detection Accuracy | STS Correlation |
|---|---|---|---|
| Baseline Model | 0.335 | 0.375 | 0.215 |
| Gradient Surgery | 0.388 | 0.375 | 0.202 |

Table 7: Test set performance metrics on the three tasks for the final model with gradient surgery

| | Sentiment Analysis Accuracy | Paraphrase Detection Accuracy | STS Correlation |
|---|---|---|---|
| Final (Gradient Surgery) Model | .397 | .369 | .193 |

## 6 Analysis

The results shown in Section 5.4 show that Gradient Surgery offers an improvement to the model in terms of key performance metrics. From a qualitative standpoint, it is interesting to try to evaluate in what situations it might make sense to use Gradient Surgery, and in what situations it is more or less useful.

One way to build intuition for this is to repeat the analysis, but with a shallower forward layer in the multitask layer. Repeating the analysis of Section 5.4 but with only one layer in the forward results in the output shown in Table 8. The results are worse across the board (as expected from the experimentation done earlier) but the important difference is that the gradient surgery has only a small improvement in this case. For example, sentiment analysis only improved by .02 points when Gradient Surgery was used, whereas with the optimal 3-layer model, sentiment analysis improved by .053 points when Gradient Surgery was used. This makes sense qualitatively, because the forward layer is the only layer shared between the three tasks, so only the parameters of the linear layer will be affected by Gradient Surgery. A deeper forward layer will have more parameters and potentially more opportunity to have conflicting gradients, so Gradient Surgery becomes more useful in this situation. A shallow forward layer is less likely to overfit the gradients of the task-specific loss functions, and therefore techniques such as Gradient Surgery may not provide as much of a benefit.

Table 8: Dev set performance metrics on the three tasks for the baseline model and the improved model with gradient surgery, but using a shallower forward layer (one layer deep)

| | Sentiment Analysis Accuracy | Paraphrase Detection Accuracy | STS Correlation |
|---|---|---|---|
| Baseline Model | 0.297 | 0.378 | 0.157 |
| Gradient Surgery | 0.317 | 0.378 | 0.166 |

## 7 Conclusion

Summarize the main findings of your project, and what you have learnt. Highlight your achievements, and note the primary limitations of your work. If you like, you can describe avenues for future work.

In this work, the challenges of designing a multitask NLP system were addressed, and a solution strategy was explored. A BERT transformer model with task-specific heads was developed, and using a combined multitask loss function with Gradient Surgery, the model was successfully trained. A sequential survey of hyperparameters was conducted to optimally configure the model according to the evaluation metrics.

In conclusion, the results show that a multitask loss can be successfully optimized using the Gradient Surgery technique. For the final model presented, the Gradient Surgery algorithm improved the evaluation metrics for two out of three tasks considered. Also, a qualitative study of the importance of Gradient Surgery was conducted, which showed that gradient surgery is less useful when there are few parameters shared between tasks. The primary limitations of the work are the relatively low performance seen across the various models trained, which are likely due to parts of the dataset being ignored (to appease a limitation in this work's implementation of gradient surgery, which requires all tasks to have the same number of batches). Directions for future work could include relaxing this requirement to be able to do gradient surgery more effectively with imbalanced dataset sizes.

## References


First quora dataset release: Question pairs. `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`. Accessed: 2023-03-02.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

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.