MiniBERT: Training Jointly on Multiple Tasks

Stanford CS224N Default Project

Xiyuan Wang
Department of Computer Science
Stanford University
xiyuanw@stanford.edu

Manasven Grover
Department of Computer Science
Stanford University
maanug@stanford.edu

Abstract

Language models have been shown to be expressive and powerful enough to perform well on a variety of NLP tasks, but training a separate model for each desired downstream task is quite inefficient. We aim to boost the performance and efficiency of the $BERT_{BASE}$ model on multiple downstream tasks, namely Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity, by jointly training on these tasks. We also extend joint training by experimenting with an optimization technique: Gradient Surgery, to avoid possible conflicts of parameter gradients coming from different tasks. By benchmarking on single and multi-task training, we show a single model can achieve 90% of the performance attained from training these tasks individually.

1 Key Information to include

- Mentor: Andrew Lee

2 Introduction

The performance of deep learning models has improved rapidly over the past several years on all kinds of downstream tasks. These improvements can come from new architectures, training objectives, or several other sources. However, often times these efforts are focused on one task at a time, using a new instance of the model and training procedure for each task. This is inefficient and unrealistic in the context of deployment, especially as models continue to grow in size. If a developer has a large set of NLP tasks they wish to solve by deploying deep learning models, it would take a great deal of compute and storage to deploy a different model checkpoint for each NLP task. Training a separate model for each task is also very expensive, for example if a developer has their own data they wish to use for finetuning.

Multi-task learning has emerged as a powerful paradigm to address this challenge by training a single model to perform all downstream tasks simultaneously. It enables models to learn shared representations across tasks, leveraging the inherent relationships between them. By jointly optimizing across multiple tasks, multitask learning improves the efficiency of training.

However, problems can occur with multi-task training as well. Conflicting objectives between downstream tasks can cause the multi-task gradients to oppose each other during optimization. For example, a conflict can arise when the model needs to strike a delicate balance between focusing on sentiment-specific cues for sentiment analysis and capturing broader semantic features for semantic textual similarity, which may not always align perfectly.

This optimization challenge as illustrated in Yu et al. (2020) is the problem of conflicting gradients. The paper also hypothesizes three conditions that cause conflicting gradients to hinder learning, provides a solution to mitigate this problem, and test its efficacy. Prior solutions have attempted to modify model architecture to alleviate this problem, however these approaches would be limited to the architecture. Other efforts try to train smaller models on each task separately and later 'distill' the smaller models into a single model. The approach in Yu et al. (2020) is more elegant and well-defined than previous approaches and model agnostic. We provide a thorough walk-through in this paper on how we utilize the intuition of gradient surgery to enable multi-task training with language models and test its performance, as well as an analysis of the result.

In this project, we seek to further investigate multi-task training approaches. To do so, we conduct several experiments with pre-trained embeddings from BERT (Bidirectional Encoder Representations from Transformers) as the representative of modern language models. We fine-tune BERT on three downstream tasks, namely Sentiment Analysis, Paraphrase Detection and Semantic Textual Similarity. We experiment with individual and joint finetuning, as well as the aforementioned optimization technique 'gradient surgery', and compare the performance results. We found that joint finetuning with gradient surgery of BERT plus naive predictions heads can come very close to the performance (within 10%) of individually finetuning on the aforementioned three downstream tasks, while only using a single model. We interpret this as reducing necessary parameters by roughly 33%.

3 Related Work

The language model of focus for this project is BERT (Bidirectional Encoder Representations from Transformers). The BERT model has been shown to produce effective embeddings for a variety of downstream tasks (Devlin et al., 2019), and many efforts have been made to push the performance of BERT higher than the original paper (Liu et al., 2019). Given the large variety of NLP tasks, we wanted to investigate if a single set of BERT embeddings can effectively represent information on multiple tasks simultaneously. Good performance on multiple downstream tasks is a very desirable property of language models, and there has been a breadth of work in this area. For example, the GLUE benchmark from Wang et al. (2018) was designed to evaluate and benchmark the performance of models on a wide range of natural language understanding tasks. The authors' motivation was to create a benchmark that would encourage the development of models that generalize well across different language understanding tasks and domains. In fact, our downstream tasks of choice have some overlap with those in the GLUE benchmark. It's clear from the research that developing models that perform well on various linguistic tasks is an important direction of progress.

Multitask learning and performance is an active area outside the world of natural language processing as well. One such work that emerged from the domain of computer vision is the 'gradient surgery' technique (Yu et al., 2020). This paper introduces the problem of conflicting gradients, where gradients from different tasks point in opposing directions during multitask training. Their solution, called gradient surgery, involves removing the components of each task's gradient that conflicts with another task's gradient. The authors claim their approach is simpler than previous approaches and model agnostic. They benchmarked their method only on vision tasks, however given that this technique is not specific to computer vision models, we believe it may be beneficial in multitask learning for language tasks as well.

Using BERT as the core model, we aim to determine the effectiveness of multitask training approaches, including the gradient surgery method, on multiple NLP tasks.

4 Approach

General Architecture We start by building a simple version of BERT. The tokenization involves converting input sentences into tokens using a WordPiece tokenizer, then into word pieces, finally into embedding ids. Unseen word pieces are assigned as the [UNK] token with padding to ensure equal sentence length. [CLS] is used to prefix sentence embeddings, which was used in training for the downstream tasks. The Embedding Layer combines token, segmentation, and position embeddings to create input representations for subsequent BERT layers, each with a dimensionality of 768. We assert our own implementation of multi-head self-attention mechanisms, to attend to information from different representation subspaces at different positions. Along with additive point-wise feed-forward layers, and additional dropout layers with p_{drop} = 0.1, we complete the BERT construction.

As outlined in the algorithm section in Kingma and Ba (2014), we utilized the *efficient* implementation version for Adam Optimizer. Adam (short for Adaptive Moment Estimation) is an optimization algorithm commonly used in training deep neural networks. It is known for its efficiency and effectiveness in optimizing complex models with large datasets. We collect the default parameters of our implementation of Adam optimizer in Table 1, for reference as the default parameters in the Experiment section.

Name	Symbol	Default Value
Learning Rate	α	1e-3 (pretrain), 1e-5 (finetune)
Decay Rate for first moment estimate	β_1	0.9
Decay Rate for second moment estimate	β_2	0.999
Weight Decay Regularization	λ	0.0

Table 1: Default Values for Parameters used in Adam Optimizer

After producing an implementation of BERT and the Adam optimizer, we designed a simple prediction head for each downstream task. In order for our experiments to concentrate on the effects of multi-task training methods, we decided it was best to first use very naive prediction heads.

For Sentiment Analysis, we take the output embedding of the [CLS] token (this pools the output embeddings of the other tokens as per the BERT paper (Devlin et al., 2019)), apply dropout with default hidden dropout probability p_{drop} = 0.3, and finally apply a linear layer to produce logits for each label of sentiment. We use CrossEntropy loss to train these classifications.

For Paraphrase Detection, we again take the output [CLS] token embedding for each input sentence and apply dropout. We then pass this output to Cosine Similarity as the final output of this prediction head. We feel this is an appropriate choice, since the [CLS] token pools information over the whole sentence, which should allow it to summarize all information in the sentence. If two sentences or questions are similar, these pooled embeddings for those sentences should also be similar, and thus their cosine similarity would be close to 1. The Cosine Similarity also keeps our output in the desired range of 0 to 1, for this task. We use Mean Squared Error to train these predictions.

For Semantic Textual Similarity, we take the same approach for the prediction head of Paraphrase Detection, and multiply the final Cosine Similarity by 5. This gives predictions in the range of 0 to 5 and align with the desired output range for sentence similarity. We also use Mean Squared Error here.

Baseline Since we want to investigate the efficacy of multi-task training methods, we took the model described above, trained it on each task sequentially, and used this as our baseline. We use these metrics to compare against our experiments with multi-task training. Specifically, we first trained the model on sentiment analysis, then trained the same model on paraphrase detection, then finally on semantic textual similarity, evaluating on all tasks after training on each individual task completed. Therefore our baseline consists of three different model states. We collect the corresponding performance in table 2. The finetuning process is executed with learning rate as 1e-5 and trained for 10 epoch on each dataset.

Metrics / Finetuning task	SST Dev Accuracy	Quora Dev Accuracy	SemEval Pearson Score
Sentiment Analysis	0.509	0.412	0.235
Paraphrase Detection	0.274	0.521	0.368
Semantic Textual Similarity	0.290	0.403	0.757

Table 2: Baseline Metrics for Each Finetuned Task. Note on interpretation: Each column shows a particular eval metric after each of the 3 stages of finetuning.

We note that each task has highest performance right after its finetuning stage completes, but decreases after other tasks are finetuned. For example, sentiment analysis accuracy is highest after finetuning on its dataset (0.509), but decreases after finetuning on the paraphrase detection dataset (0.274). We hypothesized that this is due to conflicts in the gradient landscape of these tasks.

Main approach Our strategy for training a single model for all three tasks is to perform multi-task learning. There are two parts to our approach. Firstly, we implement a simple joint training loop in four steps: (1) Load a batch of inputs from each task's dataset. (2) Provide each batch of inputs to the model to compute logits for all three tasks. (3) Compute all three tasks' losses. (4) Backpropagate on the sum of the losses. This ensures the model sees examples for all three tasks at the same time, and makes updates with respect to each task simultaneously. As a side note, since the datasets are not all the same size, for each epoch, we choose to cycle over the smaller datasets until the entirety of the longest dataset has been used.

Secondly, we extend the above joint training loop with an optimizer modification called PCGrad (Project Conflicting Gradients) as suggested in Yu et al. (2020). As quoted from the paper "*the goal of PCGrad is to modify the gradients for each task so as to minimize negative conflict with other task gradients*" (Yu et al., 2020). We believe this approach can allow for better parameter sharing and efficient joint training. We detail this algorithm below.

The first step is to determine if two gradients for individual tasks, $\mathbf{g_i}$ and $\mathbf{g_j}$, are conflicting, which can be done by calculating their cosine similarity, $\cos\phi_{ij}$. If the cosine similarity of the two gradients is negative, then $\mathbf{g_i}$ is modified in the following way:

$$\mathbf{g_i} = \mathbf{g_i} - \frac{\mathbf{g_i} \cdot \mathbf{g_j}}{||\mathbf{g_j}||_2}\mathbf{g_j}$$

If the cosine similarity is positive, the gradients are not adjusted. This will iterate through all tasks' gradient and perform necessary update. During the following optimization step, we will calculate the gradient by summing up the updated gradient for each task. We utilize a PyTorch reference implementation found in *pcgrad.py* from Tseng (2020). Parameters are updated with the new gradient g^{PC}. The full update rule for an arbitrary number of tasks can be found in Figure 1 below.

```
Algorithm 1 PCGrad Update Rule
Require: Model parameters $\theta$, task minibatch $\mathcal{B} = \{\mathcal{T}_k\}$
1: $\mathbf{g}_k \leftarrow \nabla_\theta \mathcal{L}_k(\theta)\ \forall k$
2: $\mathbf{g}_k^{\text{PC}} \leftarrow \mathbf{g}_k\ \forall k$
3: for $\mathcal{T}_i \in \mathcal{B}$ do
4:     for $\mathcal{T}_j \overset{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order do
5:         if $\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j < 0$ then
6:             // Subtract the projection of $\mathbf{g}_i^{\text{PC}}$ onto $\mathbf{g}_j$
7:             Set $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i^{\text{PC}} - \frac{\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2}\mathbf{g}_j$
8: return update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$
```

Figure 1: PCGrad Algorithm (taken from original paper Yu et al. (2020)).

5 Experiments

5.1 Data

Sentiment Analysis is a basic task in understanding whether the expressed opinion in a text is positive, negative, or neutral. *Stanford Sentiment Treebank Dataset* is a dataset composed of 11,855 single sentences. Each sentence has a sentiment integer score ranged from 1 to 5 and can be accessed from here. *CFIMDB dataset* is another dataset composed of 2,434 highly polar movie reviews. Each review has a binary sentiment score 0 or 1, representing positive or negative.

Paraphrase Detection is a task to determine whether particular sentence pairs convey the same semantic meaning. *Quora Dataset* is a dataset composed of 400,000 question pairs, which can be accessed from here. Each question pair have two questions and an indicator whether the two questions are duplicates.

Semantic Textual Similarity is a task that measures the degree of semantic equivalence with a similarity score. *SemEval STS Benchmark Dataset* is a dataset composed of 8,628 different sentence pairs. Each sentence pair contains two sentences and a score of semantic equivalence ranged from 0.0 to 5.0, which can be access from here.

5.2 Evaluation Method

For Sentiment Analysis and Paraphrase Detection, we will use accuracy as the metric to evaluate the model performance. For Semantic Texual Similarity, we will use Pearson Score. As discussed in

Agirre et al. (2013), Pearson Score is more suitable than Spearman score as it accounts for value differences as well as rank differences. It can also capture linear relationship better.

5.3 Experimental details

Hyperparam	**Value**
Pretrained checkpoint	'bert-base-uncased'
Hidden size	768
Encoder layers	12
Attention heads	12
Dropout prob	0.3
Batch size	8
Epochs	10
Learning Rate	1e-5

Table 3: Default Values for all Hyperparameters

For all experiments, we make use of the 'bert-base-uncased' checkpoint from HuggingFace. This allows us to keep our compute usage within a realistic limit, and focus our experiments on finetuning the downstream tasks.

In the first experiment, we apply only our joint training loop to finetune on all three tasks simultaneously (without PCGrad). We sought to determine how a very simple multi-task training implementation compared in performance to the baseline. As detailed in our approach, we load examples from all datasets at the same time, make predictions for each task, sum the losses, and use the combined loss to train the model. Since the Quora question pairs dataset is much larger than the SST and SemEval datasets, examples from the latter are reused until the entire Quora question pairs dataset has been trained on once. This applies to every epoch. We keep several hyperparameters constant throughout all experiments. These can be found in 3.

In the second experiment, we added the PCGrad optimizer to the training procedure to determine its effect on performance. We still use the same training loop from the first experiment to train on all three tasks jointly. It's worth noting that the PCGrad implementation we used makes its adjustments to deconflict the gradients before invoking our Adam optimizer to update parameters as usual with the Adam update rule.

In our third experiment, we increase the size of the prediction heads, while still using PCGrad and joint training. Specifically, for Sentiment Analysis, we still apply dropout, then follow with two linear layers separated by a GeLU function to make classifications. For Paraphrase Detection and Semantic Textual Similarity, we also add two linear layers separated by a GeLU function (there is a separate pair of layers for each task). In both task's head, both of the sentence embeddings (outputted by the BERT encoder) are passed through these layers. Here, we wanted to determine if some additional non-shared weights improved PCGrad's ability to deconflict the gradients for the shared parameters.

For our fourth and final experiment, we take the same model with larger prediction heads as in the third experiment, and split up the training from 3-way fully joint training to pairwise joint training, while still using PCGrad. We train in the order of sentiment analysis+paraphrase detection, paraphrase detection+semantic similarity, semantic similarity+sentiment analysis.

5.4 Results

We summarize our results in 4. Note that the baseline row is the diagonal of 2.

Comparing joint training with the baseline, we see that paraphrase detection accuracy is essentially equivalent, while sentiment analysis accuracy is about 10% less than the baseline and semantic similarity correlation is about 15% less than the baseline. This seems to suggest that our joint training approach is fairly effective and the model is able to share weights somewhat well across the three tasks without any adjustment to the optimization algorithm.

The results from the Joint+PCGrad experiment are interesting. When compared with joint training without PCGrad, paraphrase detection accuracy is unchanged, however performance on the other two tasks decreases slightly. These results are unexpected, since we would expect PCGrad to at best

Experiment	Dataset split	Best SST Acc	Best Para Acc	Best STS Pearson Corr
Baseline	Dev	0.509	0.521	0.757
Joint Training only	Dev	0.466	0.520	0.652
Joint+PCGrad	Dev	0.453	0.520	0.628
Joint+PCGrad+Large Head	Dev	0.468	0.503	0.677
Joint Pairwise +PCGrad+Large Head	Dev	0.485	0.503	0.635
Joint+PCGrad+Large Head	Test	0.504	0.500	0.668
Joint Pairwise +PCGrad+Large Head	Test	0.508	0.501	0.642

Table 4: Evaluation performance by experiment

remove conflicts between parameter updates and speed up learning, or at worst if no conflicts are present, perform the same as joint training without PCGrad. There could be several reasons for this result, which we will discuss in the analysis.

Since the second result moved further away from the baseline, we chose to increase number of non-shared parameters in the prediction head to see if this enables PCGrad to more easily optimize shared weights. There is a slight improvement in sentiment analysis accuracy and semantic similarity correlation, but also a decrease in paraphrase detection accuracy. It's difficult to attribute these changes to both the additional parameters and PCGrad or just the additional parameters alone.

The final experiment, training the tasks in pairs with PCGrad and the larger prediction heads, produced for the most part the same results as the previous experiment, with a slight drop in semantic similarity correlation. It's unclear at this point, whether making updates to the model in subsets of tasks is much worse than making updates with respect to all tasks simultaneously. However, we did note that the total training time of this experiment was much longer than the previous one.

We include the results of the last two experiments on the test split as well, and the deltas in these results match those from the dev split.

6 Analysis

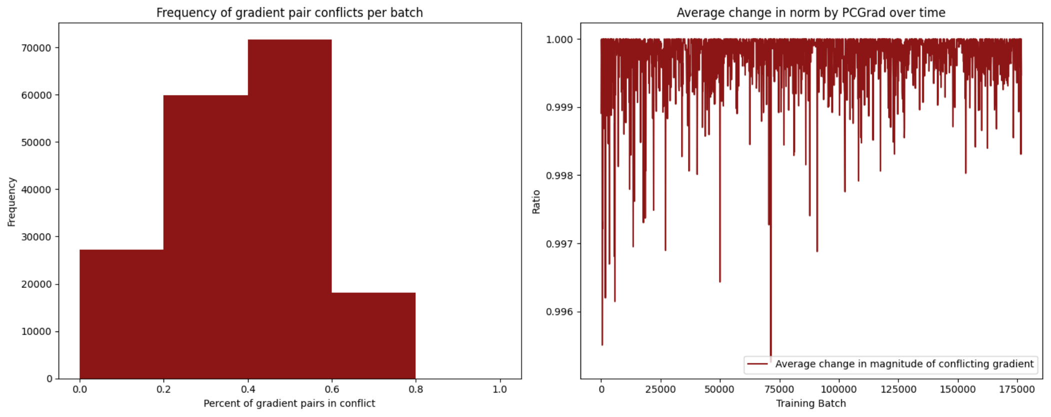

Figure 2: Breakdown of PCGrad's Effect on Gradients

Based on the results of the experiments, it is difficult to claim that PCGrad is significantly improving the joint training procedure. One possibility is that these tasks don't in fact have as many conflicts as we initially believed. Referring back to the PCGrad update rule, this lack of conflicts would be evident if the dot products between pairs of gradients were rarely negative, and therefore the gradients were not being modified very often.

We chose to investigate this possibility by logging the proportion of parameters that had conflicting task gradients each batch, as well as the average change in the norm of parameter gradients each batch. These measurements can be found in 2. We find that usually 20-40% of the parameters have conflicting gradients, however the average ratio of the gradients' norms before and after PCGrad makes its modification is quite small. If there were significant conflicts between a parameters' task gradients (in other words the conflicting component $\frac{g_i \cdot g_j}{||g_j||}$ for tasks i and j was large), we would expect a significant reduction in the L2 norm of the gradient g_i from which PCGrad removed this conflicting component. However, we instead see that the average change in norm is very close to 1, in other words the changes made by PCGrad to the gradients in this scenario are miniscule.

There could be many reasons why PCGrad is not helping significantly here, which opens up many questions for possible future investigation of what conditions increase conflicts between downstream task gradients. It might be the case that the BERT encoder we used is large enough to represent shared information on all three of these tasks. A larger set of downstream tasks might also lead to more changes being necessary to deconflict the gradients.

Based on our experiments in this project, the general applicability of the PCGrad technique to training language models on multiple downstream tasks is inconclusive. On the other hand, we believe our experiments show an overall positive result, that our simple adjustments to jointly train a single multitask BERT model are effective. All experiments that trained jointly on all three tasks came quite close to the baseline, while sharing a single BERT model to do so. It's very likely that with further adjustments like a better choice of loss function or more adjustments to the prediction heads, it would be possible to surpass our baseline model with a jointly trained model.

7 Conclusion

In this project, we embarked on a thorough exploration of multi-task training approaches using pre-trained embeddings from BERT as our cornerstone modern language model. Fine-tuning BERT on three diverse downstream tasks—Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity—we delved into both individual and joint finetuning methodologies, supplemented by the optimization technique known as 'gradient surgery'. Our experiments revealed that joint finetuning, augmented with gradient surgery and employing naive prediction heads, remarkably approaches 90% of the performance levels achieved by individually fine-tuning on the aforementioned tasks, however there is clearly room for future work on using the gradient surgery technique more effectively with NLP tasks. Notably, this strategy enabled us to achieve comparable results while utilizing only a single model, effectively reducing the necessary parameters by 67%. This underscores the efficacy and efficiency of multi-task training with $BERT_{BASE}$ model, presenting a promising avenue for advancing natural language processing tasks.

References

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert:pre-training of deep bidirectional transformersfor languageunderstanding.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. volume abs/1907.11692.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning.