BERTer Multi-task Fine-tuning for Sentence-Level Tasks

Stanford CS224N Default Project

Andrew S. Chen **Danny S. Park** **Stanley J. Yang**
Department of Computer Science
Stanford University
{aschen1, danspark, sjayyang}@stanford.edu

Abstract

In our project, we implement a BERT model with additional pre-training using masked language modeling (MLM), cosine similarity fine-tuning, and multitask fine-tuning with gradient surgery. We attempt to display improved performance in three downstream tasks (sentiment analysis, paraphrase detection, and semantic textual similarity) over a baseline BERT model with linear projection heads. The most notable result was the significant positive impact of cosine-similarity fine-tuning on STS (semantic textual similarity). On the other hand, both MLM and gradient surgery had negligible impact in increasing the ceiling of our model for the downstream tasks. However, when cross-comparing the accuracies in each epoch with our baseline, the gradient surgery model appears to converge much faster overall. With our model, we achieved a top 33% ranking in the test set leaderboard with an overall dev score of 0.707 (individual scores: 0.529 SST test accuracy, 0.747 paraphrase test accuracy, and 0.845 STS test correlation).

1 Introduction

Previously, many research groups designed language models that are unidirectional to utilize different embeddings from pretraining for downstream tasks. For example, with fine-tuning, Radford et al. train on downstream tasks with fine-tuned parameters that were pretrained beforehand (Radford et al., 2018). Bidirectional Encoder Representations from Transformers (BERT), a transformer developed by Google AI researchers in 2018, uses pre-training on deep bidirectional representations through left and right contexts for all words; this was a major improvement over the unidirectional language models for contextual word embeddings (Devlin et al., 2018). Through the conditioning of both left and right contexts, BERT is able to capture complex, deep representations to allow for several downstream tasks such as sentiment classification, summarization, and sentence similarity and (Devlin et al., 2018).

In our work, we first implement a working a minBERT model involving a Transformer architecture with a multi-headed self-attention mechanism. After some pretraining and fine-tuning of our multi-task head on two datasets for initial testing of sentiment classification, we implemented three different extensions for the three downstream tasks: sentiment classification, paraphrase detection, and semantic similarity. To better fine-tune and adjust our BERT embeddings, we did the following extensions: additional pre-training with masked language modeling, cosine-similarity fine-tuning, and multi-task fine-tuning. Our work will showcase the differences in performance among several experiments involving our different extensions.

2 Related Work

In addition to the BERT baseline model developed by Devlin et al., we implemented three extensions to the model for additional downstream tasks.

For additional pre-training extension, our implementation is inspired by the work of Devlin et al. in the original BERT model. Since we want to train our BERT model to extract deep bidirectional representations, we apply masked language modeling (MLM), to mask 15 percent of the word tokens from a piece of text. During pretraining, the model tries to predict the original identity of the masked words.

For cosine similarity fine-tuning, our implementation is inspired by the work of Reimers and Gurevych, who implemented Siamese networks to compare semantically meaningful sentences with cosine similarity (Reimers and Gurevych, 2019). Reimers and Gurevych state that this Siamese BERT architecture maintains the accuracy of the original BERT model while computing the similarity of sentence pairs up to 50x faster.

Lastly, for multi-task fine-tuning, our implementation is inspired by the work of Stickland and Murray, who first suggested the idea to limit the number of parameters needed by training all tasks simultaneously (Stickland and Murray, 2019). With multi-task learning, however, the gradient directions from different tasks may conflict. Hence, Yu et al. developed gradient surgery to counter conflicting gradient directions by projecting the gradient of the i-th task onto the normal plane of the conflicting task's gradient (Yu et al., 2020).

3 Approach

3.1 Main Approach

In our main approach, the model's architecture is a Transformer layer with a multi-headed self-attention mechanism. More thoroughly, multi-headed self-attention applies a scaled dot product consisting of queries and keys of dimension d_k and values of dimension d_v across multiple heads. We pack the queries, keys, and values into matrices Q, K, and V respectively.

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$
$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$\text{where } (head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Parameter matrices: $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

3.2 Baseline

Our baseline model utilizes the main approach mentioned above. With given pretrained weights from our minBERT model, our baseline model uses the main approach to perform sentiment classification on two datasets. For sentiment analysis, we use a single linear projection. For both paraphrase detection and semantic similarity, we concatenate the two embedding vectors and use another linear projection into the appropriate output dimension. An important thing to note is that our baseline includes training on all downstream datasets (SST, paraphrase, and STS). Essentially, we considered that the baseline is the baseline architecture described earlier but training with all of the datasets instead of just the SST dataset. We implemented a round robin training loop that feeds in a batch from each dataset in each iteration. This note is important because SST dataset training was implemented in the default project, but the sentence pair datasets were not included.

For each task, we used a different loss function suited to the output. For the sentiment classification task, we used the CrossEntropyLoss function, which is used for a general classification task. For paraphrase, we used the BinaryCrossEntropyLoss function since it is a binary classification problem. Lastly, for the semantic similarity task, we used MSE loss since the output can naturally be interpreted as a continuous similarity score. We tried CrossEntropyLoss as well for STS but found MSE to be better for this specific task.

3.3 Additional Pre-training

In our approach to masked language modeling (MLM), we mask out 15% of word tokens in a sentence to further pretrain and extract deep bidirectional representations. By predicting the

masked words during MLM, BERT is able to create complex representations between words and their left and right contexts. For the final hidden vectors, which correspond to the masked tokens, we apply the softmax function over the whole vocabulary to predict the word (Devlin et al., 2018). We then replace the BERT baseline head to the multi-task classifier model with our new optimized pretrained weights after the model gets pretrained on the SST, Quora, and STS datasets. We implemented from scratch a novel Dataset class to randomly mask tokens, a new MLM BERT head, a new training loop for pretraining, and a novel evaluation function for the pretrain task.

3.4 Cosine Similarity Fine-Tuning

In implementing cosine similarity fine-tuning, we apply the approach from Figure 1.

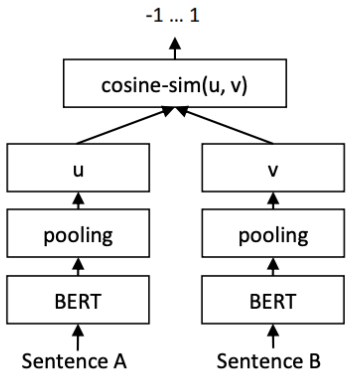


Figure 1: SBERT architecture to compute cosine similarity scores (Reimers and Gurevych, 2019).

With the output representations given by our BERT models, we apply the MEAN pooling strategy (computing the mean of all output vectors from BERT) and use cosine similarity to compute prediction similarity scores (Reimers and Gurevych, 2019). We added a new cosine similarity layer and implemented mean pooling in our code.

3.5 Multitask Fine-Tuning

In our approach to multi-task fine-tuning, we previously used a basic approach to run three different loops for each task: train first on SST, followed by Quora and STS. Afterwards, we used the round robin approach: running only one training loop to train on all three tasks simultaneously, taking a batch from each dataset during each iteration. Thus, the three tasks share one BERT model, reducing the number of additional task-specific parameters (Stickland and Murray, 2019). However, there are concerns over conflicting gradient directions when performing multi-task fine-tuning. Hence, we apply gradient surgery as shown in Figure 2. We implemented round robin training by ourselves and repurposed the gradient surgery approach outlined by the authors of the original gradient surgery paper. In our gradient surgery implementation, we used the PCGrad PyTorch implementation from https://github.com/WeiChengTseng/Pytorch-PCGrad.

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

Figure 2: Gradient surgery: projects the gradient of the i-th task g_i onto the normal plane of another conflicting task's gradient g_i (Yu et al., 2020).

4 Experiments

4.1 Data

After creating our initial BERT model, we pretrained and fine-tuned our model on two datasets: Stanford Sentiment Treebank (SST) (Socher et al., 2013) and CFIMDB.

First, the SST dataset consists of 11,855 single sentences from movie reviews. Since the dataset is parsed, there are 215,154 unique phrases in total, with possible labels of negative, somewhat negative, neutral, somewhat positive, and positive. The training, dev, and testing sets have 8,544, 1,101, and 2,210 examples respectively.

Second, the CFIMDB dataset consists of 2,434 highly polar movie reviews, with possible labels of negative and positive. The training, dev, and testing sets have 1,701, 245, and 488 examples respectively.

After applying each different extension to our initial BERT model, we will train and test with the SST dataset for sentiment analysis, the Quora dataset for paraphrase detection, and the SemEval STS Benchmark dataset for semantic textual analysis. First, the Quora dataset has 400,000 question pairs, with binary labels that indicate if the pair are paraphrases of each other. The training, dev, and testing sets have 141,506, 20,215, and 40,431 examples respectively. Second, the SemEval dataset has 8,628 distinct sentence pairs, with labels from 0 to 5 in terms of increasing sentence similarity. The training, dev, and testing sets have 6,041, 864, and 1,726 examples respectively.

4.2 Evaluation method

We use F1 scores, or testing accuracy, when training and testing on the SST and Quora datasets. For the SemEval dataset, we calculate the Pearson correlation of the true similarity values against the predicted similarity values. For our analysis of gradient surgery, we define a score metric, which is an unweighted sum of the accuracies on the SST and Quora datasets and the Pearson correlation on the SemEval dataset. We interpret score as a convenient way to measure the combined multi-task performance of a model:

$$score = acc(SST) + acc(Quora) + corr(SemEval). \tag{1}$$

4.3 Experimental details

4.3.1 Cosine Similarity

We implemented cosine similarity for the two tasks with sentence pair datasets (paraphrase detection and STS). We experimented with three main configurations: cosine similarity for STS, paraphrase detection, and both STS and paraphrase detection. For all tasks for which we were not using cosine similarity, we kept the baseline prediction head (linear projections with appropriate dimensions and concatenations for tasks with pair datasets). We used the same baseline loss functions: CrossEntropy-Loss for sentiment analysis, BinaryCrossEntropyLoss for paraphrase detection, and MSE loss for STS. Furthermore, for these three configurations, we used the CLS token outputs from our Siamese BERT models as the sentence embeddings to compare using cosine similarity; this is noted as the CLS-token strategy in Reimers and Gurevych (2019). However, we picked the best performing cosine similarity configuration of the three (STS, paraphrase detection, STS + paraphrase detection) and instead applied the recommended MEAN pooling strategy outlined in Devlin et al. (2018) for a fourth and final experiment, which we anticipated to be our strongest result. MEAN pooling uses the mean of all output vectors to derive a fixed-sized sentence embedding instead of the CLS token.

4.3.2 MLM

We implemented the masked language modeling objective for our BERT model for additional pretraining. For the model configuration during pretraining, we used the base BERT head ('bert-base-uncased'). For this additional pretraining task, we ran several experiments varying the number of epochs for pretraining. We ran these experiments specifically because we saw signs of the model overfitting on downstream tasks of the multi-task classifier. Since we are pretraining on the same dataset that we are training our downstream tasks on, this may raise concerns that our model is

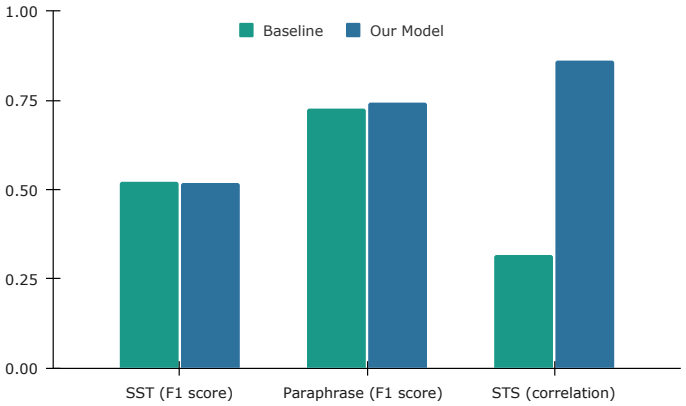


Figure 3: Comparison of baseline and our model

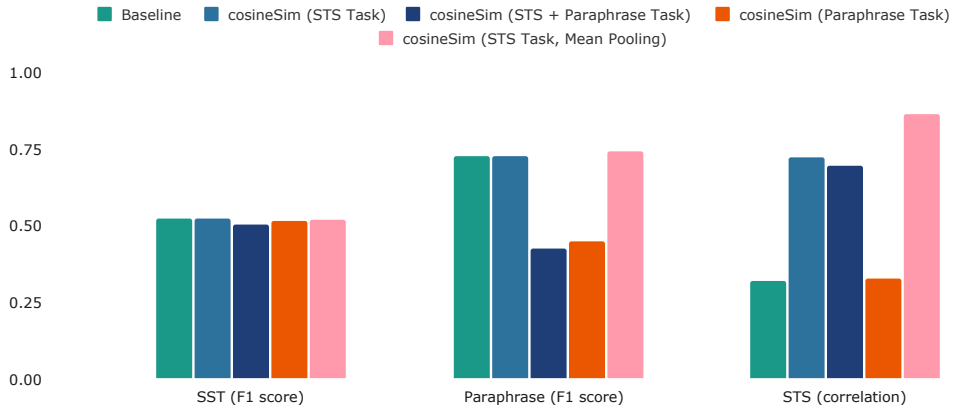


Figure 4: Comparison of different cosine similarity applications

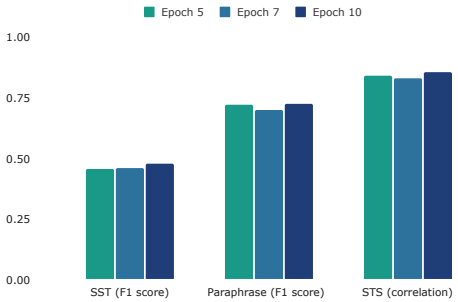
"cheating" and understanding the data distribution too well. For the MLM task, our last linear layer projected onto parameter $vocab\_size$, which has a default value of $30,522$ words. We used the default dropout rate (0.3) and learning rate $(1e-5)$. In testing different loss functions, we found that CrossEntropyLoss was most suitable.

4.3.3 Gradient Surgery

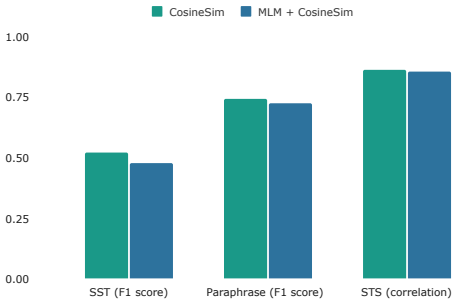
For our gradient surgery experiments, we used our baseline architecture with default hyperparameters (dropout rate = 0.3, learning rate = 1e-5) except for the number of epochs (20 instead of 10), which we chose in order to better analyze convergence. Since gradient surgery is model-invariant, we simply applied gradient surgery to the training loop of our baseline model, comparing it with our baseline model with an unmodified training loop. We then measured (at every epoch) the accuracy for all three tasks individually as well as a score metric that summarized these accuracies on both the training and dev sets in order to gain insight into gradient surgery's potential impact on convergence speed.

4.4 Results

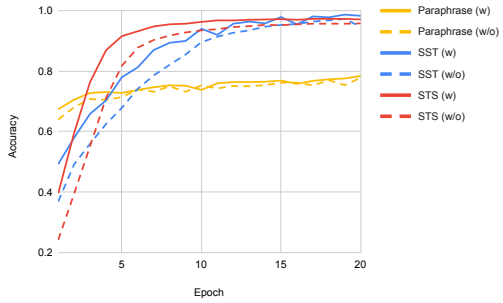
In our baseline architecture, described previously as basic single linear layers, we achieved an accuracy of 0.521 on the SST development set, an accuracy of 0.727 on the paraphrase development set, and a correlation of 0.317 on the STS development set. In our final model, we achieved an accuracy of 0.520 on the SST development set, an accuracy of 0.742 on the paraphrase development set, and a correlation of 0.862 on the STS development set (Figure 3).



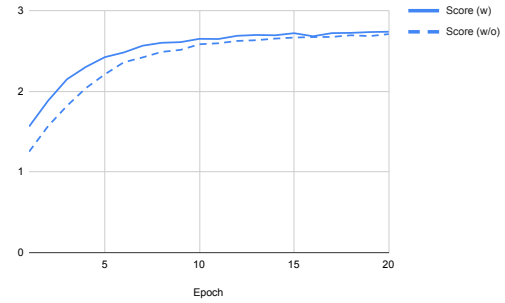
(a) Figure 5: Effect of epochs in MLM pretraining



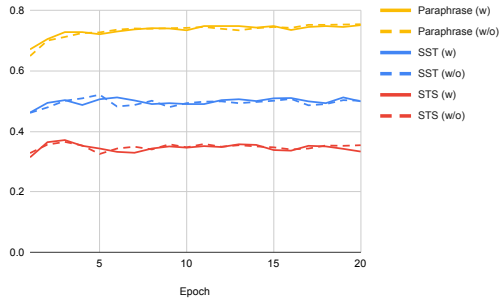
(b) Figure 5: Comparison of model with or without MLM



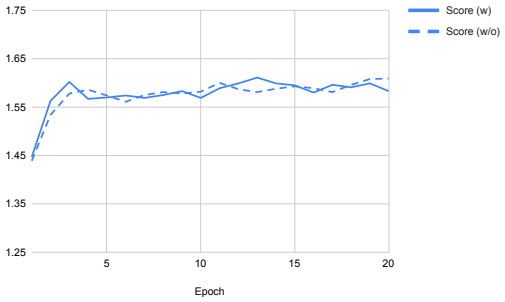
(a) Effect of gradient surgery on accuracy (training)



(b) Effect of gradient surgery on score (training)



(c) Effect of gradient surgery on accuracy (dev)



(d) Effect of gradient surgery on score (dev)

4.4.1 Cosine Similarity

As seen in Figure 4, cosine similarity seemed to have little to no effect on sentiment analysis accuracy, regardless of configuration. Cosine similarity seemed to have a significantly negative impact when applied to paraphrase detection, regardless of whether it was also applied to STS, decreasing the accuracy well below baseline. Yet, cosine similarity appeared to have a significantly positive impact when applied to STS, boosting the accuracy well above baseline. Finally, when we applied the mean pooling strategy (as opposed to naively using CLS tokens as our embeddings) to the configuration in which we only applied cosine similarity to STS (since we found it to be strongest result), the results further improved, particularly for STS.

4.4.2 MLM

On the pretraining set, we saw that pretraining had negative or little impact on the downstream classifications tasks. We tested early stopping by decreasing the number of epochs during pretraining. In Figure 5a, we observe that there was no significant performance difference on downstream classification tasks based on how many epochs we pretrained the MLM head for. In Figure 5b, we

see that there was a slight negative impact on the accuracy when we added MLM to our model. Since this slight negative result was across the board, we postulate that the MLM head had overfit on the downstream classification task data instead of overfitting on a certain task head.

4.4.3 Gradient Surgery

On the training set, gradient surgery resulted in a noteworthy uptick in accuracy on all three tasks particularly before 10 epochs, as seen in figure 6a. This boost in accuracy (at a given epoch) for the gradient surgery model can be observed in figure 6b as well. However, the gradient surgery model performed only slightly better than that without gradient surgery by 10 epochs and negligibly better by 15 epochs. Ultimately, after around 15 epochs, it appears that both models plateau in accuracy/score, although the gradient surgery model appears to converge significantly faster.

On the dev set, gradient surgery appears to have no bearing on accuracy/score beyond 3 epochs in both 6c and 6d. However, within the first 3 epochs, the score in 6d appears to increase more rapidly for the gradient surgery model than for that without gradient surgery–we can once again interpret this as the gradient surgery model converging faster. However, especially since both models appear to plateau within 5 epochs, this uptick in convergence speed is less pronounced on the dev set than on the training set.

5 Analysis

5.1 Cosine Similarity

Cosine similarity, when applied to the STS task, appeared to have a profoundly positive effect on performance. This is hardly surprising because in the STS task, sentences are placed into 6 buckets of increasing similarity, so it makes sense that cosine similarity would be a natural model of sentence similarity that can learn to capture finer-grained notions of similarity or dissimilarity between sentences. Interestingly, cosine similarity resulted in sub-baseline performance when applied to the paraphrase task. While our initial hypothesis was that using cosine similarity to compare sentence embeddings would improve paraphrase detection performance because paraphrases can be classified as "similar" and non-paraphrases can be classified as "dissimilar," we claim that this measure of similarity (or dissimilarity) between paraphrases and non-paraphrases is not granular enough to allow for the effective learning of a similarity space. We also claim that while paraphrasing and similarity may seem related when looking at the binary labels of paraphrase detection (paraphrase = similar, non-paraphrase = dissimilar), it is highly plausible that probabilistic paraphrasing (i.e. a cosine similarity score that falls between 0 and 1) is not nearly as well-defined/well-ordered as probabilistic similarity, which is far more meaningful: cosine similarity scores from 0 to 1 (which are then scaled from 0 to 5) are a logical continuous analog of the discrete similarity buckets in the STS task).

Additionally, mean pooling appeared to further boost the performance of our best-performing cosine similarity model, particularly for STS. This evidence aligns with the results from Reimers and Gurevych (2019), which found MEAN pooling to be superior to CLS pooling across two datasets and several random seeds. However, this paper found MEAN pooling to be only a marginal upgrade over CLS pooling, whereas our MEAN pooling experiment resulted in noteworthy performance improvements. Conneau et al. (2017) conversely found MAX pooling (max-over-time of the output vectors) to perform better than MEAN pooling for the BiLTSM-layer of InferSent, indicating that the optimal pooling strategy may be task-specific. It is not immediately clear to us why MEAN pooling gives our model such a great performance boost over CLS pooling; it is possible that the mean of all output vectors is more sensitive to our fine-tuning procedure than the CLS token.

5.2 MLM

For additional pretraining, we saw that it had a negligible or negative impact on our downstream classification task results. There are a few possible explanations: first, the size of the pretraining dataset we trained our MLM head on could have been too small. The pretrained weights from the original BERT model was trained on the entirety of Wikipedia, so the additional pretraining task would likely require much more data for each dataset for the pretraining to have a significant impact on our downstream tasks. Second, it is possible that the model overfit on the distribution of the

datasets. Since we used the same datasets for pretraining and the downstream classification tasks, the model likely learned the distribution of our SST, paraphrase and STS datasets too well. This finding was interesting to us, since this meant that our MLM pretraining head negatively affected the results of completely different objectives of our three classification heads. Our evidence for overfitting was a training accuracy of 0.94 combined with a dev accuracy of less than 0.5.

To account for overfitting, we incorporated in a dropout layer into the MLM model to regularize the model and tried different strategies such as early stopping (using 5 or 7 epochs instead of 10). Surprisingly, neither method seemed to significantly reduce the overfitting. Lastly, we thought that our token masking strategy could have been incorrect or not strong enough. At first, we implemented a masking strategy that masked a single token in each sentence. When we saw that our results did not improve, we implemented a masking strategy that masked 15% of the tokens. We found that increasing the number of masking tokens also did not have a significant impact on our results. Our next step would have be to implement the full MLM objective task, but given our time constraints, we were unable to carry this out to fruition.

5.3 Gradient Surgery

Gradient surgery resulted in substantial optimization improvements (faster convergence), particularly on the training set; however, it did not lead to better asymptotic performance on either the training or dev sets. This is mostly in agreement with the original results in Yu et al. (2020), which cite the possibility for both efficiency and performance gains. Because gradient surgery aims to mitigate the negative effect of conflicting gradients in multi-task learning settings by reducing destructive interference, it makes sense that gradient surgery speeds up convergence on our training set by streamlining optimization. Less expected but not unusual is the fact that we observe faster convergence on the dev set as well (at least within the first 3 epochs before performance plateaus), which is likely attributed to the faster optimization on the training set enabled by gradient surgery. However, gradient surgery fails to raise the point at which our model plateaus, which is not unusual since gradient surgery is model-agnostic and only directly interacts with the optimization problem at hand–once convergence is reached (which is likely the case when our dev accuracy/score plateaus), gradient surgery provides little ability to better model the data. Since our dev accuracies/score seem to plateau well within 10 epochs, poor optimization hardly appears to be a bottleneck for higher performance, giving gradient surgery little opportunity to improve our end performance. In fact, gradient surgery may increase our model's vulnerability to overfitting.

6 Conclusion

We conclude that small-scale additional pretraining on our downstream tasks may not be able to achieve a higher accuracy on downstream tasks. On the other hand, we found that cosine similarity was highly effective improving our model's performance on the semantic textual similarity task. We found that gradient surgery enabled faster convergence, even if it did not raise the predictive ceiling of our baseline model. In the future, we could address potential overfitting on the MLM head by implementing full MLM model masking further to prevent mismatching between initial pretraining and later fine-tuning and by pretraining our model with larger datasets.

References

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y 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.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.