Fine-tuning minBERT for multi-task prediction

Stanford CS224N Default Project

Ishita Mangla
Department of Computer Science
Stanford University
imangla@stanford.edu

Abstract

Large language models like BERT are increasingly being adapted for a range of tasks, from protein and RNA sequence design to threat and malware detection in cybersecurity. Therefore, it is important to explore ways to fine-tune robust multi-task models. In this project, I complete an implementation of the minBERT model and the AdamW optimizer and then leverage BERT's pre-trained embeddings and fine-tune for multi-task prediction for sentiment analysis, paraphrase detection, and semantic textual similarity (STS). I develop multitask learning baselines and then explore gradient surgery, BERT embedding pooling strategies, cosine-similarity fine-tuning, rich feature learning through shared layers, and different loss functions to improve multi-task prediction. My best performing model uses Mean-Pooling of the contextualized word embeddings that BERT outputs, combined with cosine-similarity fine tuning and gradient surgery. This model achieves an overall test score of 0.719, with an accuracy of 0.538 for sentiment classification, 0.727 for paraphrase classification, and 0.780 for semantic textual similarity (STS) prediction.

1 Key Information to include

- Mentor: Timothy Dai
- External Collaborators (if you have any): N/A
- Sharing project: No

2 Introduction

BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018) is a Transformer-based deep learning model that has made significant advancements in the natural language processing (NLP) domain. It possesses the capability to grasp context bidirectionally, enabling it to evaluate the relationships between words in both directions within an input text and produce contextual word embeddings. Unlike conventional unidirectional models like RNNs or LSTMs, BERT's bidirectional structure allows it to capture subtle language semantics by comprehending contextual cues from both preceding and succeeding words concurrently. Additionally, the self-attention mechanism allows BERT to grasp long-range dependencies in text data, resulting in superior performance across various NLP tasks. It is pre-trained on the Toronto BookCorpus and English Wikipedia and thus has a wealth of knowledge — its weights and fine-tuning capabilities enable transfer learning across diverse tasks and domains.

Fine-tuning BERT for various downstream tasks is an active area of research. More specifically, multi-task fine-tuning is challenging because different tasks may have different objectives, which can lead to conflicting gradient directions that may impact the overall performance of the model as well as task-specific performance. Furthermore, differences in dataset sizes for different tasks can lead to imbalanced results and exploring combinations of loss functions for different tasks can be time and resource-expensive.

In this paper, we explore multi-task fine-tuning approaches as laid out by Bi et al. (Bi et al., 2022), which involves combining the losses for all tasks together and backpropagating them. We build on this by exploring an improvement using Gradient Surgery (PCGrad) as proposed by Yu et al. (Yu et al., 2020). We also explore methods like cosine-similarity fine-tuning and mean pooling as outlined in Sentence-BERT (Reimers and Gurevych, 2019), which significantly improves the performance of the STS task. We also experiment with other strategies like sharing layers between the paraphrase detection and STS tasks, as well as adding cosine similarity features to the paraphrase detection embeddings.

3 Related Work

In this section, we will first review the multi-task learning paper as proposed by Bi et al. (Bi et al., 2022). In this approach, the loss for each task is simply added together and backpropagated. In the case of our three tasks, it can be represented as:

$$L_{\text{total}} = L_{\text{task1}} + L_{\text{task2}} + L_{\text{task3}}$$

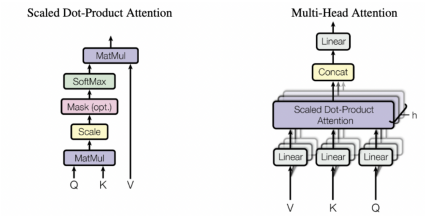
However, this setup encounters the conflicting gradient problem as mentioned earlier. Gradient Surgery (PCGrad) as proposed by Yu et al. (Yu et al., 2020) counters this issue by altering the gradients through projection onto the normal plane of conflicting task gradients. This helps remove interfering components and allows more constructive interactions between tasks. In gradient surgery, the gradient of the i-th task g_i is projected onto g_j, which is the conflicting task. This can be represented as:

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

We will also review cosine-similarity fine-tuning and mean pooling. The SBERT (Reimers and Gurevych, 2019) framework presents a multifaceted approach to enhance the capabilities of BERT and RoBERTa models in generating fixed-sized sentence embeddings. At the core of this enhancement lies the integration of different pooling strategies, including the utilization of the CLS-token, computing the mean of output vectors (MEAN strategy), and determining the max-over-time of output vectors (MAX strategy). It employs siamese and triplet networks for fine-tuning, aiming to produce semantically meaningful embeddings that can be compared using cosine similarity. Fine-tuning SBERT on the STS benchmark using the regression objective function results in competitive performance, with a slight improvement observed when pre-training on NLI data before fine-tuning on STSb, particularly benefiting BERT cross-encoder models.

4 Approach

The backbone of all the following architectures is BERT (Devlin et al., 2018). The BERT architecture consists of 12 Encoder Transformer layers. In Transformer layers, multi-head self-attention (Vaswani et al., 2017) involves performing scaled-dot product operations across various attention heads simultaneously, which allows the model to encode multiple relationships and nuances for each word and learn rich contextual embeddings. For all our experiments, we just use the pre-trained BERT and fine-tune it for our downstream tasks.

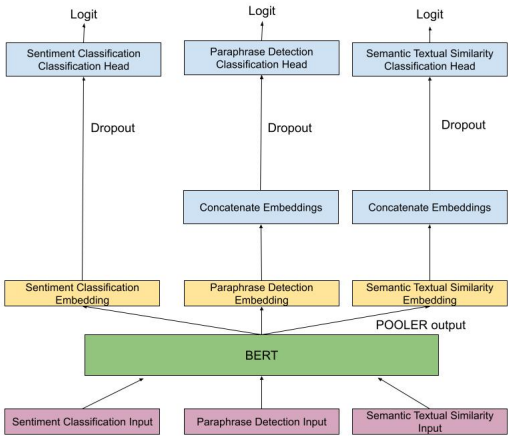


[Figure 1: This is Figure 2 from (Vaswani et al., 2017)]

4.1 Model architectures

4.1.1 Baseline model

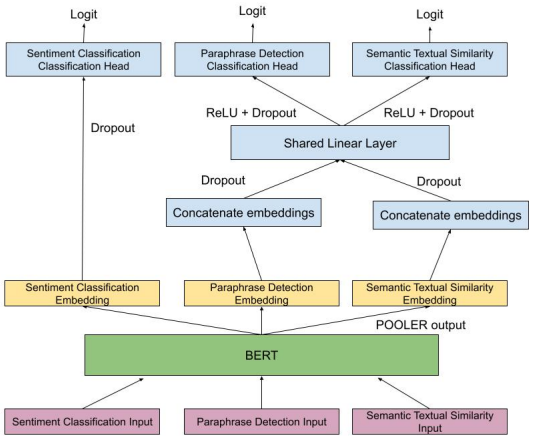
The baseline model involved adding 3-task specific prediction heads for each task, that took in the 'CLS' token output. We use this as the baseline to compare our other experiments to. The model architecture for the baseline model is given below.



[Figure 2: Model architecture for baseline experiments]

4.1.2 Shared layers between tasks

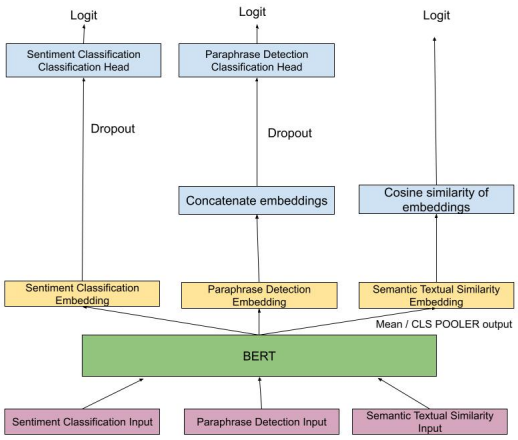
Given the similar nature of our tasks, we implement a shared layer mechanism for the STS and Paraphrase Detection tasks. In this approach, we pass the individually concatenated STS and Paraphrase Detection embeddings through a shared linear layer and then through task-specific layers.



[Figure 3: Model architecture for shared layer experiments]

4.1.3 Embedding cosine similarity

We implement a cosine similarity mechanism for the STS Detection task. In this approach, we pass the STS and Sentiment detection embeddings through a dropout layer combined with task-specific linear layers and return the cosine similarity of the STS embeddings as the logit output.



[Figure 4: Model architecture for cosine similarity experiments]

4.2 Loss functions

4.2.1 MSE Loss

We use the cosine-similarity fine-tuning approach as outlined in SBERT (Reimers and Gurevych, 2019). In order to compare the cosine-similarity, we compare the binarized STS labels (into 0 and 1) with the cosine-similarity logits. The MSE loss can be given by:

$$\text{MSE Loss}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

4.2.2 Cross Entropy Loss

For the paraphrase detection and some STS tasks, we pose our problem as a binary classification problem with the following loss function:

$$\text{BCE with Logits}(z, y) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\sigma(z_i)) + (1 - y_i)\log(1 - \sigma(z_i)))$$

For the sentiment classification task, we pose our problem as a multi-class classification problem with the following loss function:

$$\text{Cross-Entropy Loss}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

5 Experiments

5.1 Data

In order to implement multi-task learning, we have 3 datasets for each of the sentiment analysis, paraphrase detection, and semantic textual similarity tasks. The breakdown of the datasets is as follows:

5.1.1 Quora Dataset

The Quora Dataset is used for paraphrase detection and contains 400,000 question pairs with labels indicating whether instances are paraphrases of one another. There are 141,506 training samples, 20215 validation samples, and 40431 test samples. It has a binary output (0 for not true and 1 if true).

5.1.2 SemEval STS Benchmark

The SemEval STS Benchmark Dataset is used for semantic textual similarity analysis and contains a total of 8628 sentence pairs with labels ranging from 0 to 5, where 0 denotes that a sentence pair is unrelated and 5 means equivalent meaning. There are 6041 training samples, 864 validation samples, and 1726 test samples.

5.1.3 Stanford Sentiment Treebank (SST)

The Stanford Sentiment Treebank (SST) Dataset is used for sentiment classification and contains a total of 11,855 sentences rated on a scale of 5 sentiments, ranging from negative to positive. There are 8544 training samples, 1101 validation samples, and 2210 test samples.

We also did an initial experiment where we limited the size of each dataset to the size of the smallest dataset (STS dataset with 6041 training samples), but concluded with an experiment that increasing the dataset size improves performance, especially for sentiment classification. Therefore, we create 3 datasets of 10000 samples each per task. We keep the dataset sizes equal in order to iterate over all 3 datasets together for our multi-task learning setup. Since the STS and Sentiment Prediction datasets have less than 10000 samples each, we resample each of these datasets using random choice with replacement. We limited the dataset size to 10000 samples because of compute constraints as well as the smaller size of the STS and sentiment prediction datasets.

5.2 Evaluation method

We are given some initial baselines for single-task sentiment classification from the SST and CFIMDB datasets in the project handout. The CFIMDB dataset is not relevant to this task since we are only evaluating on SST for our task. The baselines are as follows:
Pretraining for SST: Dev Accuracy: 0.390 (0.007)
Finetuning for SST: Dev Accuracy: 0.515 (0.004)

Since the paraphrase detection and sentiment analysis are classification tasks, we use the accuracy from the ground truth and predicted labels as an evaluation metric. For the semantic textual similarity task, we use the Pearson correlation metric as used by the original SemEval (Agirre et al., 2013) paper.

5.3 Experimental details

All experiments were performed for 10 epochs, with a learning rate of 1e-5, and with a dropout rate of 0.3 for all experiments that had dropout layers. The models were trained on a T4 GPU on GCP.

5.3.1 Baseline experiment — 6041 training samples

For the baseline experiment, we used the model architecture as outlined in 4.1.1. After obtaining the embedding from BERT, we passed the embeddings through a dropout and task-specific layer, which took the logits and applied CrossEntropyLoss for Sentiment Classification, BCEWithLogits for Paraphrase Detection, and MSELoss for Semantic Textual Similarity (STS). We implemented straightforward multi-task learning and we summed up the losses and backpropagated them after every batch. For this experiment, we chose the length of each dataset to be the length of the shortest training dataset, which was the STS dataset with 6041 samples.

5.3.2 Baseline experiment — 10000 training samples

For the baseline experiment, we used the model architecture as outlined in 4.1.1. The experiment setup in terms of input and loss function is the same as 5.3.1. For this experiment, we wanted to explore how increasing the dataset size would improve performance, so we create 10000 samples from each dataset. We see an improvement in performance for every task, so we continue all further experiment with 10000 samples each.

5.3.3 Shared layers experiment

For the shared layer experiment, we used the model architecture as outlined in 4.1.2. After obtaining the embedding from BERT, we passed the embeddings for STS and Paraphrase detection through a dropout layer and shared layer, and then dropout + task-specific layers. We take the logits and applied CrossEntropyLoss for Sentiment Classification, BCEWithLogits for Paraphrase Detection, and BCEWithLogits for Semantic Textual Similarity (STS). We normalize the STS ground truth logits to 0 and 1 to enable using the BCEWithLogits function. Any logits <= 2 are set to 0 and logits > 2 are set to 1. We implemented straightforward multi-task learning and we summed up the losses and backpropagated them after every batch. For this experiment, we create 10000 samples from each dataset.

5.3.4 Gradient surgery + cosine-similarity experiment

For the cosine similarity experiment, we used the model architecture as outlined in 4.1.3. After obtaining the embedding from BERT for each task, we directly take the cosine similarity of the STS embeddings and output that as a logit. We normalize the STS ground truth logits to 0 and 1. Any logits <= 2 are set to 0 and logits > 2 are set to 1. We then apply MSELoss to the output cosine similarity and the normalizes logits.

For the sentiment classification task, we take the logits and applied CrossEntropyLoss and BCE-WithLogits for Paraphrase Detection. We implemented multi-task learning with gradient surgery and create 10000 samples from each dataset.

5.3.5 Gradient surgery + cosine-similarity + mean pooling experiment

Up to this point, we have been using the CLS token embedding output from the BERT embedding. For this experiment, we Mean Pool the sentence representation that is output by BERT instead of only taking the CLS token embedding. Mean pooling has also shown the best results in the SBERT (Reimers and Gurevych, 2019) paper on the STS benchmark dataset.

For the cosine similarity experiment, we used the model architecture as outlined in 4.1.3. The rest of the experiment setup in terms of loss functions and inputs is the same as 5.3.4.

5.3.6 Gradient surgery + cosine-similarity (Paraphrase & STS) + mean pooling experiment

Since we saw a significant change in STS accuracy after implementing cosine similarity (+0.20) and mean pooling (+0.3), we wanted to perform a final experiment with incorporating cosine similarity into the paraphrase detection task. In order to do so, we used the model architecture as outlined in 4.1.3. After obtaining the embedding from BERT for each task, we calculate the cosine similarity for the embeddings for STS and Paraphrase detection. We directly output the cosine similarity for STS as a logit. We normalize the STS ground truth logits to 0 and 1. Any logits <= 2 are set to 0 and logits > 2 are set to 1. However, for paraphrase detection, we append the cosine similarity to the concatenated embeddings and pass it through the task specific head after applying dropout.

We then apply MSELoss to the output cosine similarity and the normalized logits. For the senti-ment classiciation task, we take the logits and applied CrossEntropyLoss and BCEWithLogits for Paraphrase Detection. We implemented multi-task learning with gradient surgery and create 10000 samples from each dataset.

5.4 Results

All the scores in the table reported below were obtained by submitting to the dev leaderboard. Therefore, these are the scores on the dev set as evaluated on the leaderboard.

Since we have a limited number of submissions for the test leaderboard, we submit the best-performing Cosine-sim features + PCGrad + mean pooling experiment and achieve a test accuracy of 0.538 for the SST sentiment task, 0.727 accuracy for the paraphrase detection task, and a Pearson test correlation 0.780 for the semantic textual similarity task. **The overall test score is 0.719**.

The paraphrase and semantic textual similarity tasks have sufficiently high scores, but the sentiment accuracy significantly underperforms as compared to the other two tasks even after changing the

Model type	Training samples	Sentiment dev acc	Paraphrase dev acc	STS dev corr	Overall dev acc
Baseline	6041	0.485	0.706	0.356	0.628
Baseline	10000	0.501	0.721	0.354	0.633
Shared layers + PCGrad	10000	0.485	0.728	0.342	0.628
Cosine-sim + PCGrad	10000	0.511	**0.735**	0.583	0.679
Cosine-sim + PCGrad + mean pooling	10000	0.499	0.726	**0.798**	0.708
Cosine-sim features + PCGrad + mean pooling	10000	**0.526**	0.726	0.795	**0.716**

Table 1: Results Table for dev leaderboard

pooling strategy. The strength of the paraphrase and STS tasks significantly impacts the overall test score. An interesting observation is that PCGrad (Gradient Surgery) does not significantly impact the scores, which is understandable since the three tasks are significantly related.

6 Analysis

Overall, the biggest difference was observed in the performance of the STS task after incorporating cosine similarity and mean pooling. An intuitive understanding of why mean pooling works better is that CLS token is not an especially robust representation for tasks that BERT has not been fine-tuned on since it gives the overall representation for an input sentence and therefore averaging over the entire embedding will allow for more effective feature extraction.

The lowest difference in performance was observed in the sentiment classification task. This is interesting because the single-task baseline that was provided for sentiment classification on the dev set was 0.515. Therefore, the single-task and multi-task performance was similar, which suggests that the sentiment classification performance was mostly independent of the problem setup i.e. single-task and multi-task. From a qualitative standpoint, it is possible that since sentiment classification is not a binary classification task (like Paraphrase detection), it is harder for BERT to pick up on the subtleties in sentiments. One experiment that would be insightful would be to binarize sentiment labels and then evaluate the performance of BERT on those.

7 Conclusion

We achieved an overall test accuracy of 0.719 for all three tasks. However, paraphrase detection accuracy generally decreased a little bit as STS accuracy increased with the cosine similarity approach. The multi-task system prefers to perform very well on one task, and how to concurrently improve the performance of all tasks is yet to be explored. In an environment with less stringent compute and time constraints, it might be worth exploring if approaches like using a larger dataset for fine-tuning, pre-training, or transfer learning improve downstream performance. While we achieve satisfactory performance, research into multi-task prediction using models like BERT, especially for diverse tasks, is an open and rapidly evolving problem.

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.

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. Mtrec: Multi-task learning over bert for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669.

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*.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

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.