Fine Tuning Multi Downstream Tasks based on BERT with Gradient Surgery

Stanford CS224N Default Project

Jiwen Chen
Department of Mechanical Engineering
Stanford University
`jiwchen@stanford.edu`

Abstract

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based model that generates contextual word representations. Such representation could be utilized to multiple downstream tasks, including sentiment analysis, paraphrase detection and semantic textual similarity analysis. In order to create more robust and semantically-rich sentence embeddings, it is important to fine-tune the BERT weights with task-relative data efficiently. In the project, we have tested different fine-tuning techniques for improving the model performance on the aforementioned three downstream tasks. These tests include changes in the architecture (dense layer size, Siamese network), in the optimization step (train separately or together, whether to apply gradient surgery or not), and in the training setting (whether to reshuffle data after each epoch, batch size, regularization). After comparison between different tests, under the limit of AWS g5.2xlarge instance, the best performance is achieved with (1) Add Siamese network and cosine similarity for semantic textual similarity task (2) Pass embeddings for two sentences with their difference into the dense linear layer for the paraphrase task (3) Multi-task training with gradient surgery (4) Reshuffle dataloader after each training epoch (5) Apply l2 regularization (6) Train under batch size 2 for sentiment and similarity task, and batch size 32 for paraphrase task. The model achieves 52.1% accuracy for sentiment task, 82.8% accuracy for paraphrase task and 0.820 Pearson Correlation for semantic textual similarity task on the dev set, and 52.4%, 82.7% and 0.792 on the test set.

1 Key Information to include

- Mentor: Hans Hanley
- External Collaborators (if you have any): /
- Sharing project: /

2 Introduction

With the appearance of ChatGPT and GPT-4 into public, language models have been attached more importance worldwide. In recent years, pre-trained language models such as BERT (Bidirectional Encoder Representations from Transformers) have revolutionized the field of natural language processing (NLP). With the transformer architecture and training on large amounts of text data, BERT is able to learn deep, contextualized representations of words and sentences [1]. One of the most important areas where BERT could be easily utilized is the downstream tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity. Sentiment analysis involves determining the sentiment (positive, negative, or neutral) of a given text, while paraphrase detection aims to identify whether two texts have similar meanings. Semantic textual similarity, on the other hand, involves determining the degree of similarity between two texts.

While BERT has shown great promise for a wide range of NLP tasks, the performance of using pretrained weight directly on the downstream tasks may not be optimal. This is because the model is trained on a general corpus of text and may not have learned specific features or nuances that are relevant to a particular task. To overcome this challenge, fine-tuning the BERT weights based on the task-related data is crucial. For example, in sentiment analysis, fine-tuning BERT on a corpus of labeled sentiment data can lead to better performance than using the pre-trained BERT weights directly. Similarly, for paraphrase detection and semantic textual similarity, fine-tuning BERT on specific pairs of data can result in more accurate and relevant results. Due to the lack of task-specific data compared with general corpus of text, fine-tuning efficiently is the key to achieving better performance. To improve the fine-tuning efficiency, choosing a proper method to generate logits and selecting a proper loss function is important. A related work is Sentence BERT that applies Siamese networks [2].

While fine-tuning on task-specific data can improve the performance of the single task, it may not be the most efficient way to use a pre-trained model for multiple tasks. Therefore, a multi-task learning approach can be used to train the same BERT weights to perform multiple related tasks simultaneously, with the only difference on the embedding heads. As a result, the method promises more efficient use of computational resources, improved generalization, and reduced risk of overfitting. For efficient multi-task training, Round Robin approach and gradient surgery could be referred [3].

3 Related Work

The downstream tasks involved in the project include paraphrase detection and semantic textual similarity, where in both tasks a pair of sentence inputs is needed. Sentence-BERT (Sentence Embeddings using Siamese BERT-Networks) is proposed to fine-tune a BERT model on a sentence-pair classification task using a Siamese network architecture [2] Specifically, the model takes two input sentences, and the Siamese network learns to encode these sentences into fixed-length vectors that capture their semantic similarity. The method has proven advantages over traditional methods, and has achieved state-of-the-art performance on several benchmark datasets [2]. The limitation of Sentence-BERT includes the need of large amount of data and higher computation time.

For training multi-task, gradient surgery is a recent technique for improving the quality of gradients during the training process. The method involves modifying the gradients of one task so that there is no conflict components along the direction of the gradient of the other tasks [3]. Gradient surgery has been proven to speed up the training process since the modified gradient is in the same direction as the updating gradient of each task. One limitation of the method is that gradient surgery can sometimes lead to sub-optimal results or introduce new issues, such as gradient noise or instability [3].

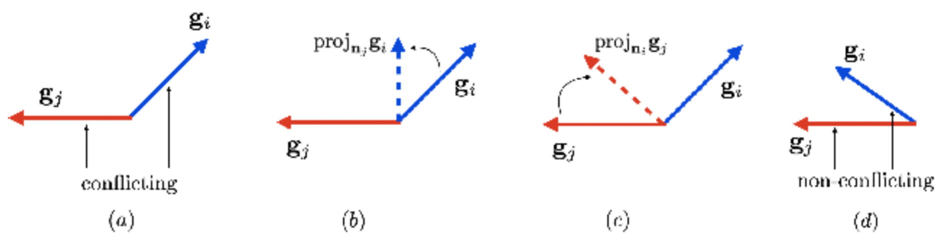

Figure 1: Conflicting and Non-conflicting Gradients in Gradient Surgery.[3]

Despite the limitations, Siamese network and gradient surgery are still two promising techniques to test in the project.

4 Approach

Compared with BERT, the default project introduces minBERT, a small-scale version of BERT that includes 12 BERT layers (encoder transformer layer), as the backbone of the project. As the elementary component of the model, each BERT layer includes a multi-head attention layer, followed by an add&norm layer, a feed forward layer and another add&norm layer. Multi-head self-attention layer allows the model to jointly attend to different aspects of the input sequence simultaneously, making it more robust and effective in capturing complex patterns in the data [1].

The input of the minBERT model is embedded as the sum of token embeddings and position embeddings. The token embeddings are a learnable map from individial input ids into vector representation of size 768, and the position embeddings are also learnable for each 512 positions in a given input. The beginning of a sentence input is a [CLS] token. The corresponding minBERT output of the [CLS] token is sent to a fully connected linear layer (pooler layer) and a tanh activation function. The pooler output is utilized as the semantic representation of the sentence. The overall architecture of the model is shown in the minBERT part in the figure below.

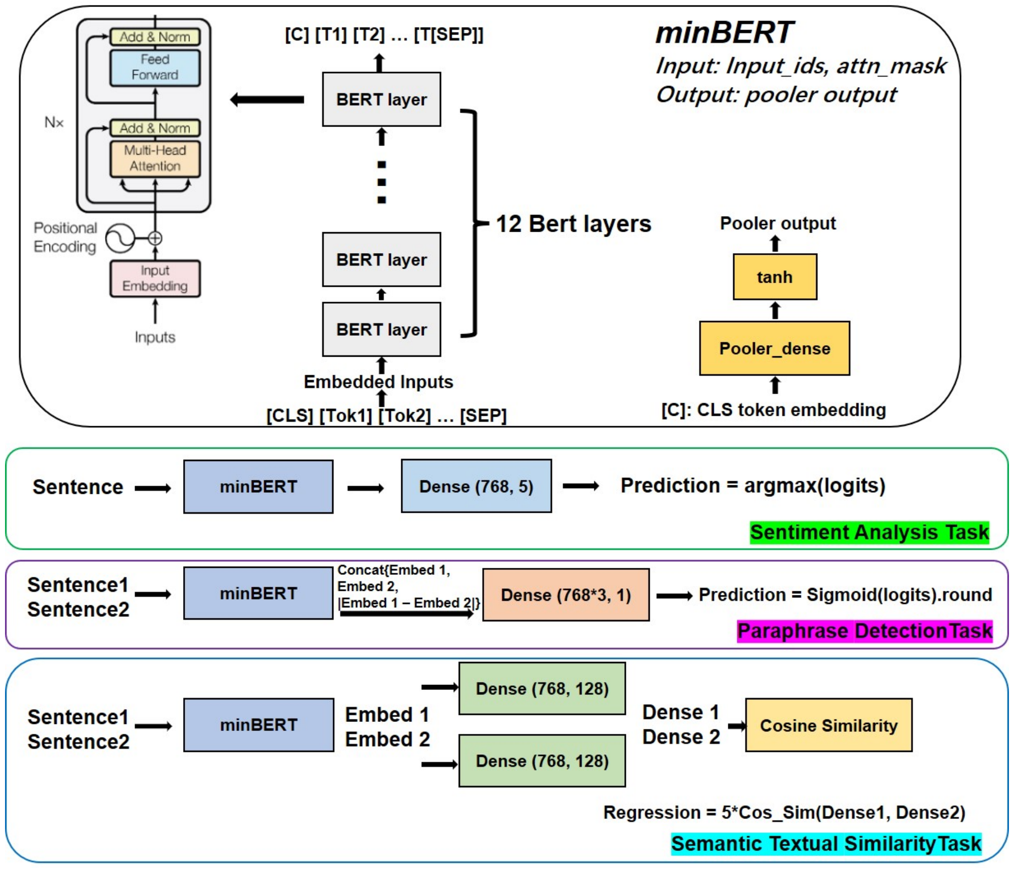

Figure 2: Overall Model Architecture.

Besides the transformer, we have designed different head architectures for the three downstream tasks: sentiment analysis, paraphrase detection and semantic textual similarity. For sentiment analysis, the labels are an integer between 1 to 5. Therefore, we designed a simple linear layer with 5-channel output and use argmax function on the logits for prediction. Besides, cross-entropy loss is applied since the task is a multi-class classification problem. For paraphrase detection, the labels are either 0 or 1. We concatenated the embeddings for the input pair and their embedding difference as the input, and got a single-value output through a linear layer. The logits are passed through a sigmoid function and round to 0 or 1 as the prediction. Since the task is a binary classification problem, we applied binary cross-entropy loss. For the semantic textual similarity task, we applied Siamese network with shared weight to output a 128-channel tensor for each sentence and applied cosine similarity between two 128-channel tensors to predict the similarity score. The regression score is calculated as five times the cosine similarity score so that the range maps the labels (0-5). MSE loss is applied for this task.

To better investigate how each addition in the model could influence the performance, we train the minBERT and different heads under five different settings:

1. Only use dense layer for three tasks: for sentiment analysis, we use a dense linear layer of size (768, 5), for paraphrase detection, we contatenate two sentence embeddings and use a dense layer of size (768*2, 1), and for semantic texutal similarity task, we use the same dense layer architecture as the paraphrase task. We use round robin approach but train three tasks separately. This method is used as baseline.

2. Based on 1, we change the layer for sts task: directly apply cosine similarity between two embeddings output from minBERT.

3. Based on 2, we apply Siamese network (dense layer with size (768, 128)) between the minBERT output and cosine similarity calculation for the sts task. We also concatenate the difference between two embeddings for the paraphrase task and feed into a dense layer with size (768*3, 1). Different from two tests before, we train all the tasks together as a multi-task training process for test 3.

4. Based on 3, we apply gradient surgery during the training.

5. Based on 4, we reshuffle the dataloader after each training epoch and add a 0.001 weight on l2 regularization to avoid overfitting.

We plan to train minBERT and the different heads from the given uncased pretrained model under each of the five settings for 10 epochs, and compare the performance on the dev set to investigate how each addition in the model changes the model behavior.

5 Experiments

We conducted the experiments mentioned in the Approch Section above, and compare the dev accuracy for model selection on three tasks: sentiment analysis, paraphrase detection and semantic textual similarity.

5.1 Data

For the sentiment analysis task, we are using Stanford Sentiment Treebank (SST) dataset. SST consists of 11,855 single sentences extracted from movie reviews and the sentiment is labelled in five categories from negative (1) to positive (5). SST data are split into 8,544 train examples, 1,101 dev examples, and 2,210 test examples [4].

For paraphrase detection task, we are using Quora dataset. Quora dataset consists of 400,000 question pairs and the detection label is classified as either 0 or 1. We are using a subset of Quora dataset, and is split into 141,506 train examples, 20,215 dev examples, and 40,431 test examples [5].

For semantic textual similarity task, we are using SemEval STS Benchmark Dataset. STS dataset consists of 8,628 different sentence pairs with varying similarity from 0 to 5. STS dataset is split into 6,041 train examples, 864 dev examples, and 1,726 test examples [6].

5.2 Evaluation method

For evaluation, we are using accuracy for sentiment and paraphrase task since they are both classification problems. We are using Pearson Correlation between logits and labels for the similarity task since it is a regression problem.

5.3 Experimental details

The model settings for the experiment have been described in detail in the end of the Approch Section. For the hyperparameters, we apply 1e-5 as the fine-tuning learning rate. Due to the limitation of RAM on the instance, we are using batch size 16 for sst and sts dataset, batch size 32 for paraphrase dataset for any methods that do not involve gradient surgery, and are using batch size 2 for sst and sts dataset, batch size 32 for paraphrase dataset for methods that apply gradient surgery. After training 10 epochs, we compare the accuracy (or Pearson Correlation) on the dev set, and select the best method. With that best method, we train for 20 epochs to deliver the submission for the default project.

5.4 Results

The 10-epoch training performance on the sst dataset for the sentiment analysis task (except the baseline) is shown in the figure below:

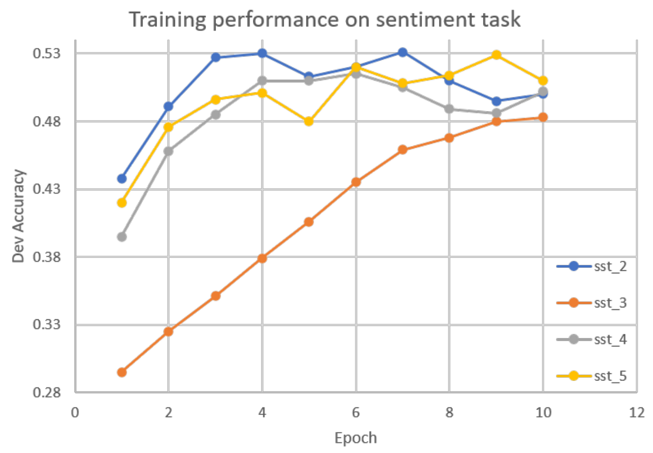

Figure 3: Training Performance on Sentiment Task, Evaluated on the Dev Set

In the result we can see, training separately on different tasks (method 2) learn fastest, but both method 2 and method 5 converges to an accuracy (53%) during the 10-epoch training. It can be assumed that the accuracy is limited by the architecture of the BERT and head itself.

The 10-epoch training performance on the quora dataset for the paraphrase detection task (except the baseline) is shown in the figure below:

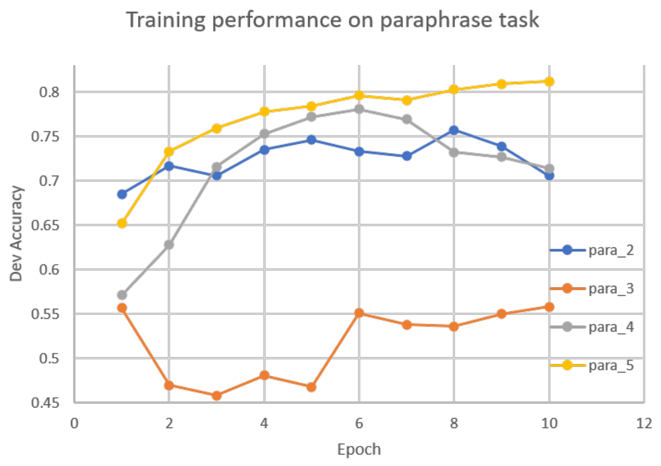

Figure 4: Training Performance on Paraphrase Task, Evaluated on the Dev Set

In the figure we can see that method 5 generates most stable and best performance (over 80%). Compared between method 2 and method 4 (method 2: train separately, method 4: train together with gradient surgery), it is hard to tell whether training together helps the performance. Compared between method 4 and method 5 (method 4: without reshuffle data loader, method 5: reshuffle + l2 regularization), it is obvious that when reshuffle the dataloader and have the model access to more data helps training dramatically.

The 10-epoch training performance on the sts dataset for the semantic textual similarity task (except the baseline) is shown in the figure below:

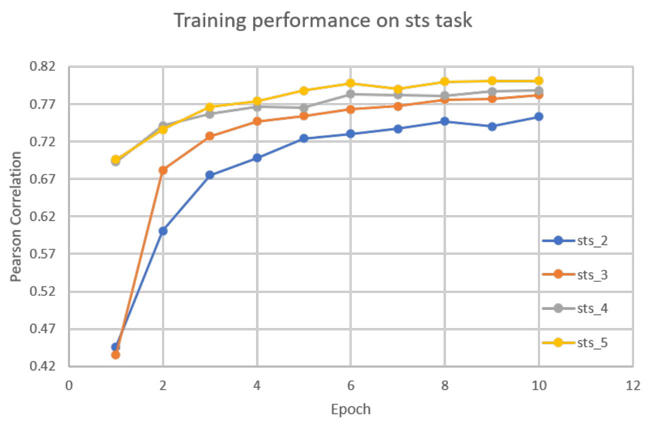

Figure 5: Training Performance on Paraphrase Task, Evaluated on the Dev Set

In the figure we can see that method 5 generates best performance. Compared with method 2 and method 3 (train separately vs train together), we can observe that training tasks together helps the similarity task. One reason for that is paraphrase detection and similarity task share similarity and fine-tuning on the large paraphrase detection dataset benefits training on the sts task. Compared between method 3 and method 4, we can observe the influence of gradient surgery that speeds up the training. Compared between method 4 and method 5, we can also observe getting access to more and random paraphrase data + extra l2 regularization helps improve the model performance.

To quantitatively look into the overall performance comparison, Table 1 below records the dev set accuracy from the best model trained within 10 epochs for each method:

Table 1: Dev Set Evaluation on Five Methods for the Best Model among 10 Epochs Training

Model	Sentiment	Paraphrase	STS
Baseline	50%	65%	35%
(2) Add cos simiarity	51%	75.7%	76.3%
(3) Add Siamese, train together	48.3%	55.8%	79.9%
(4) Add gradient surgery	51.5%	78.1%	80%
(5) Add reshuffle data + l2 reg	**52.9%**	**80.9%**	**81.7%**

The results show that method 5 does well in all the three downstream tasks involved in the project. Therefore, we trained method 5 for 20 new epochs for model submission. We submit the result of 52.1% (SST), 82.8% (Para), 82.0% (STS) for dev set and 52.4% (SST), 82.7% (Para), 79.2% (STS) for test set.

6 Analysis

In the project, we have tested the influence of cosine similarity, Siamese network, training as a multi-task problem, gradient surgery, data reshuffling and regularization. The result shows that each method has a promising improvement on at least one of the downstream tasks tested in the project. Comparing Baseline with method 1 in Table 1, we can observe that adding cosine similarity instead of using a single dense layer for the STS task dramatically improve the performance from 35% to 76.3%. Comparing method 2 and method 3, although multi-task training decrease the performance on paraphrase and sentiment task, the addition of Siamese network helps improve STS furthermore. Comparing method 3 and method 4, we can observe how gradient surgery helps the training of multi-task problem and the model performance for all the tasks exceeds the performance when training separately. Comparing method 4 and method 5, we can observe how reshuffling dataloader to have the training acess more data in the paraphrase detection task helps, and how adding extra l2 regularization helps on all three tasks.

However, due to the limitation of the architecture (Siamese layer size limited by the RAM of the instance), and the limitation of STS data, the STS task performance does not seem to improve after around 12 epochs of training. Therefore, to further improve the model performance, a more powerful gpu instance, a better architecture design, and more data on STS dataset are needed.

7 Conclusion

In the project we have implemented BERT layer and minBERT architecture, and conducted comparison tests between five different methods differing in model architecture, optimization step and training settings. The tests are conducted on the SST, Quora and STS datasets, to separately solve the task of sentiment analysis, paraphrase detection, and semantic textual similarity analysis. After comparison between different tests, under the limit of AWS g5.2xlarge instance, the best performance is achieved with (1) Add Siamese network and cosine similarity for semantic textual similarity task (2) Pass embeddings for two sentences with their difference into the dense linear layer for the paraphrase task (3) Multi-task training with gradient surgery (4) Reshuffle dataloader after each training epoch (5) Apply l2 regularization (6) Train under batch size 2 for sentiment and similarity task, and batch size 32 for paraphrase task. The model achieves 52.1% accuracy for sentiment task, 82.8% accuracy for paraphrase task and 0.820 Pearson Correlation for semantic textual similarity task on the dev set, and 52.4%, 82.7% and 0.792 on the test set after training for 20 epochs. The result shows that each method (cosine similarity, Siamese network, training as a multi-task problem, gradient surgery, data reshuffling and regularization) has a promising improvement on at least one of the downstream tasks tested in the project. Limited by the architecture (Siamese layer size limited by the RAM of the instance), and the amount of STS data, the training model stops improving the performance on the STS task after a few epochs. As a result, for future work, a more powerful gpu instance, a better architecture design, and more data on STS dataset are needed.

References

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

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

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

[4] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 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, 2013.

[5] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th annual research colloquium of the UK special interest group for computational linguistics*, pages 45–52, 2008.

[6] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * 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, 2013.