MinBERT and Downstream Tasks

Stanford CS224N Default Project

Wenlong Ji
Department of Statistics
Stanford University
jwl2000@stanford.edu

Abstract

In this project, I implement the important components of the BERT model, including the Multi-head Self-attention mechanism, the Transformer Layer, and the Adam optimizer. Then I explore the power of the minBERT model on three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Extensions are then made to further improve the performance, including multi-task training, loss function balancing, additional finetuning, hyperparameter tuning, the gradient surgery technique, and supervised contrastive learning loss. Overall, my best model achieves a decent performance in all three tasks, the prediction accuracy (correlation) are 0.527, 0.811, 0.646 for sentiment analysis, paraphrase detection, and semantic textual similarity on test datasets, respectively.

Mentor: Timothy Dai, External Collaborators: no, Sharing project: no

1 Introduction

In recent years, deep learning has demonstrated its ground-breaking power in natural language processing. Among various different types of deep learning models, the Bidirectional Encoder Representations from Transformers (BERT) model proposed by Devlin et al. (2018) is often recognized as an efficient and powerful model for a wide range of downstream tasks. The BERT model is a pre-trained model designed to learn deep bidirectional representations from unlabeled text, and one can easily finetune it with a few additional layers to create a state-of-the-art model for downstream tasks.

The purpose of this project is to implement several important components of the BERT model, including the Multi-head Self-attention mechanism, the Transformer Layer, and the Adam optimizer. With the basic model implemented, I will explore its performance on three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. To fully exploit the power of BERT representations in this multi-task setting, several extension techniques are applied to further enhance the performance beyond simple finetuning, including multi-task joint training, loss function balancing, additional finetuning, hyperparameter tuning, the gradient surgery technique Yu et al. (2020), and supervised contrastive learning loss Khosla et al. (2020); Gao et al. (2021).

2 Related Work

BERT is a Transformer-based language model, which uses self-attention to capture the dependence in long sequences. The transformer architecture is proposed in Vaswani et al. (2017), and has been used as the fundamental building blocks of modern language models. Building on top of the original BERT models, there are lots of attempts to make it more powerful and efficient, including model distillation Sanh et al. (2019), weight pruning Gordon et al. (2020), and improved training Liu et al. (2019).

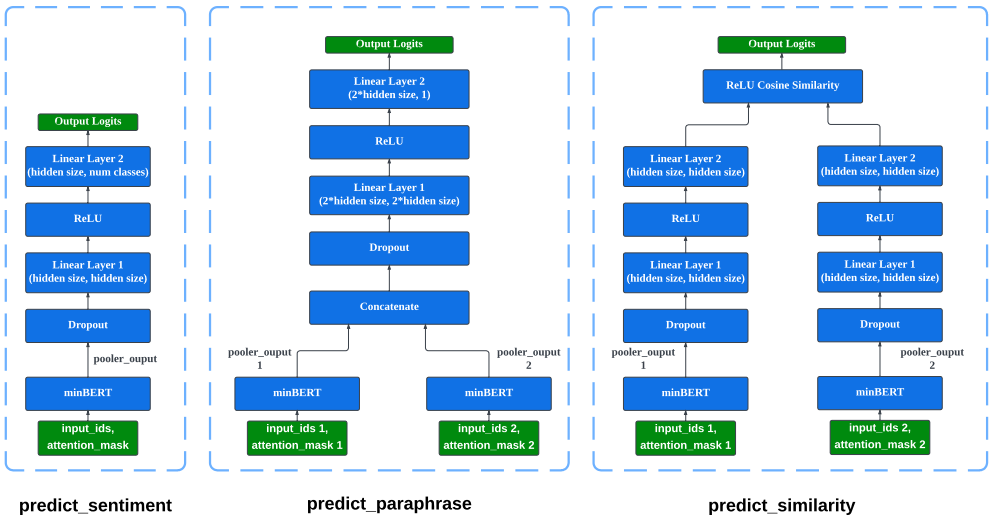

Figure 1: Architecture of task specific heads.

To further enhance the quality of learned features in BERT model, a supervised contrastive learning approach is considered. Supervised contrastive learning is first proposed in Khosla et al. (2020) for image classification tasks as an alternative of standard cross-entropy loss. It is first shown to be powerful in Gao et al. (2021) in semantic textual similarity task with text data. There are many other papers exploring the usage of contrastive learning in NLP. To name a few, Fang et al. (2020); Hupkes et al. (2022); Suresh and Ong (2021) use contrastive learning for text classification tasks; Chen et al. (2021); Qin et al. (2020) use contrastive learning for information extraction tasks; Pan et al. (2021); Vamvas and Sennrich (2021) use contrastive learning for machine translation tasks; and Qian et al. (2022); Su et al. (2022) use contrastive learning for text generation. In general, the contrastive learning framework has been proven to be useful in various NLP tasks.

Training models on multiple tasks is often challenging as different tasks may conflict with each other. To resolve this issue, gradient surgery is proposed in Yu et al. (2020) to project task gradient to each other,

3 Approach

3.1 Model Configuration

To perform the three downstream tasks, I use the pre-trained BERT to produce embeddings of the input, with task-specific additional layers on top of the embeddings to make predictions for each task. The model configuration is shown in Figure 1. In particular, the output logits of the semantic textual similarity task are taken to be the positive cosine similarity between two transformed embeddings, i.e.,

$$\text{Logits} = \max(0, \text{Cosine}(\text{embedding 1}, \text{embedding 2})).$$

This is because two embeddings with cosine similarity equals -1 will be linear dependent in the embedding space, and hence should be considered as strong similarity. Therefore, it is more appropriate to cut off negative similarities and encourage dissimilar pairs to have orthogonal embeddings (i.e., cosine similarity = 0).

For the sentiment analysis task, the labels are 5 classes categorical variables and the loss function is the cross entropy loss. For the paraphrase detection task, the labels are binary categorical variables and the loss function is the binary cross entropy loss. For the semantic textual similarity task, the labels are similarity levels (from 0-5) and the loss function is the mean square error loss between predicted similarity (scaled by 5 since the output logits have a range of [0,1]) and the label.

3.2 Baseline Model

I use the two baseline models, in **Pretrained Baseline** the BERT model is freezed and only the task-specific heads are trained, and in **Finetuned Baseline** both the BERT model and task-specific heads are trained. To perform multi-task learning, in each epoch the model is trained on SST, Para, STS task in turn (i.e., three loops over the three dataloaders separately). At the end of each epoch, I evaluate the model performance on all three tasks and save the model with the best sum performance.

3.3 Extension

To further improve the performance, I explore the following extensions:

3.3.1 Joint Training

The naive multitask training method in the baseline model has two drawbacks:

1. The datasets used in the multitask training have very different sizes (see Table 1 for details). However, due to the memory limit, the maximum batch size is 8, meaning that in each epoch, the optimizer updates about 1000 steps for sentiment analysis and semantic textual similarity tasks and about 20000 steps for the paraphrase detection task. Therefore, the paraphrase detection task dominates the training procedure and potentially hurts other tasks.
2. The model is trained on the three tasks one by one in each iteration, thus the model performance will be biased towards the last task in each epoch.

Motivated by the two drawbacks mentioned above, I implemented a joint training method. In each epoch, the model takes one batch from each dataset, evaluates their loss function and averages them together. Thus for each optimizer step, the loss function is

$$\mathcal{L}_{\text{joint}} = \frac{1}{3}\left(\mathcal{L}_{\text{sentiment}} + \mathcal{L}_{\text{paraphrase}} + \mathcal{L}_{\text{similarity}}\right). \tag{1}$$

This procedure will repeat in one epoch until the largest dataset is exhausted, and the other two datasets will be cycled over and over again in this process. In our later comparison, it will be referred as **Joint Training**.

3.3.2 Balanced loss

Furthermore, since we are using three different types of loss functions for the three tasks, their numerical scales are also different. Therefore, I consider using the balanced loss function

$$\mathcal{L}_{\text{joint}} = \omega_{\text{sentiment}}\mathcal{L}_{\text{sentiment}} + \omega_{\text{paraphrase}}\mathcal{L}_{\text{paraphrase}} + \omega_{\text{similarity}}\mathcal{L}_{\text{similarity}}. \tag{2}$$

where

$$\omega_{\text{task}} = \frac{\mathcal{L}_{\text{sentiment}} + \mathcal{L}_{\text{paraphrase}} + \mathcal{L}_{\text{similarity}}}{\mathcal{L}_{\text{task}}}, \text{for task} \in \{\text{sentiment}, \text{paraphrase}, \text{similarity}\}.$$

These weights are computed under `torch.no_grad()`, and they are only used to balance the scales of different loss functions. In our later comparison, this method will be referred as **Balance**.

3.3.3 Gradient Surgery

The gradient direction computed on different tasks may conflict with each other, therefore multi-task learning may hurt the performance on each task. To address this issue, Yu et al. (2020) proposed to use a gradient surgery technique that projects the gradient of task i g_i onto the normal plane of another conflicting task's gradient g_j via

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

This technique is often refered as projecting conflicting gradients (**PCG**), and it is implemented in pytorch by Tseng (2020). I will directly use this implementation in the code.

3.3.4 Supervised Contrastive Learning

Contrastive Learning Chen et al. (2020); Khosla et al. (2020); Gao et al. (2021) aims to learn effective representation by contrasting different samples, that is, maximizing the agreement between positive pairs, and minimizing the agreement between negative pairs. Positive pairs are usually constructed to be semantically related and negative pairs are usually constructed to be semantically unrelated. In this project, I consider using the supervised contrastive learning for the sentiment analysis task as described below:

Assume we have a dataset $\{x_i, y_i\}_{i=1}^m$, the first step is to generate augmented data $\{x_i^1, x_i^2, y_i\}_{i=1}^m$ by data augmentation (following the suggestion in Gao et al. (2021) I use dropout as the augmentation). In the supervised setting where label information is available, the positive instances for an augmented data x_i^j are all augmented data that is generated from the same class and the negative instances are all augmented data generated from a different class. And the contrastive loss is written as

$$\ell_i = -\log \frac{\sum_{k:y_k=y_i} \sum_{j=1}^2 e^{\mathrm{sim}\left(f(x_i^1), f(x_k^j)\right)/\tau}}{\sum_{k:y_k\neq y_i} \sum_{j=1}^2 e^{\mathrm{sim}\left(f(x_i^1), f(x_k^j)\right)/\tau}} - \log \frac{\sum_{k:y_k=y_i} \sum_{j=1}^2 e^{\mathrm{sim}\left(f(x_i^2), f(x_k^j)\right)/\tau}}{\sum_{k:y_k\neq y_i} \sum_{j=1}^2 e^{\mathrm{sim}\left(f(x_i^2), f(x_k^j)\right)/\tau}};$$

where $f(\cdot)$ is the model that generates the embeddings, sim refers to a similarity measure and $\tau > 0$ is the temperature parameter. In my implementation, I use the supervised contrastive loss implemented by Khosla et al. (2020), and use the minBERT backbone to generate the embeddings. To enhance the training stability, I normalize each $f(x_i^j)$ before computing the loss function. The similarity measure is taken to be the inner produce and the temperature parameter is taken to be $\tau = 0.07$ following the default choice of Khosla et al. (2020). During the training procedure, the loss function for sentiment analysis will be the sum of this contrastive loss and standard cross-entropy loss. In our later comparison, this method will be referred as **CL**.

3.3.5 Additional Training on STS

After comparing my results on dev datasets with top models on the dev leaderboard, I realized that the main bottleneck of my model is on the STS dataset. Therefore, I attempt to improve the performance on the STS dataset by simply adding another round of training on STS datasets at the end of each epoch. In our later comparison, this method will be referred as **additional STS**.

3.3.6 Hyperparameter Tuning

As suggested in the provided code, the dropout technique is crucial to improving the empirical performance. Therefore I also try different choices of dropout probability p to improve the model performance. In our later comparison, this method will be referred as **dropout=p**.

4 Experiments

4.1 Data

Following the instructions, I used the Stanford Sentiment Treebank (SST) dataset and the CFIMDB dataset to perform sentiment analysis for the first part of the project; and I used the SST dataset for sentiment analysis, the Quora dataset for paraphrase detection, and SemEval STS dataset for semantic textual similarity for the second part of the project. Some basic properties of the datasets are listed as follows:

Dataset Name	Train size	Dev size	Test size	Input	Output
SST	8544	1101	2210	Movie review	Categorical labels (5 classes)
CFIMDB	1701	245	488	Movie review	Binary labels
Quora	141506	20215	40431	Question Pair	Binary labels
SemEval STS	6041	864	1726	Sentence Pair	Similarity (0-5)

Table 1: Basic information of datasets being used in the project

4.2 Evaluation method

As suggested in the project instruction, for the sentiment analysis and the paraphrase detection task, I use the classification accuracy as the metric; for the semantic textual similarity task, I use the Pearson correlation as the metric. The Pearson correlation can adjust to different scales in the predicted similarity, and therefore is a more appropriate metric here.

4.3 Experimental details

The details of model configuration have been described in Section 3.1. For the sentiment analysis in part 1, I use a learning rate of 1e-3 for pretraining and a learning rate of 1e-5 for finetuning, the batch size is set to be 8, the number of epochs is set to be 10 and the hidden dropout probability is set to be 0.3. For the multitask training in part 2, if not specially specified, I use a learning rate of 1e-5 and a batch size of 8 to train the model for 10 epochs in total. The dropout probability is set to be 0.2 as a default. For experiments with gradient surgery, I use a batch size of 4 due to the memory limit.

4.4 Results

For the first part of the project, the model performance on the two sentiment analysis datasets is listed in Table 2.

Dataset	Pretrain	Finetune
SST	0.386	0.521
CFIMDB	0.759	0.967

Table 2: Model performance on the two sentiment analysis datasets

For the second part of the project, I tested the performance of the baseline models as well as the combination of several extension techniques on the three tasks, and the performance on dev datasets are listed in Table 3.

Model	SST Dev	Para Dev	STS Dev
Pretrained Baseline	0.351	0.687	0.229
Finetuned Baseline	0.428	0.805	0.685
Finetuned + Joint Training	0.496	**0.810**	0.663
Finetuned + Joint Training + balance	0.507	0.802	0.635
Finetuned + Joint Training + balance + dropout=0.3	0.495	0.807	0.637
Finetuned + Joint Training + balance + dropout=0.4	0.506	0.788	0.659
Finetuned + Joint Training + PCG	0.508	0.802	0.586
Finetuned + Joint Training + CL	0.491	0.723	**0.704**
Finetuned + Joint Training + CL + balance	**0.515**	0.792	0.627
Finetuned + Joint Training + additional STS	0.500	0.782	0.598

Table 3: Model performance on dev datasets of the three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.

Overall the model performance is decent on all three tasks. In general, joint training is found to be helpful to improve the model performance as expected. However, it is out of my expectation that the PCG technique and the contrastive learning loss don't help to improve the overall model performance. It is also surprising to find that training with an additional training round for STS dataset even hurts the model performance.

Then I choose the top three models according to their and evaluate them on Test datasets, the results are summarized in Table 4.

Model	SST Test	Para Test	STS Test	Overall Test Score
Finetuned + Joint Training	**0.527**	**0.811**	**0.646**	**0.721**
Finetuned + Joint Training + balance	0.521	0.803	0.623	0.712
Finetuned + Joint Training + CL + balance	0.517	0.795	0.605	0.705

Table 4: Model performance on test datasets of the three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.

It is very surprising that although the Finetuned+Joint Training model achieve a much lower score on SST dev dataset compared with the other two selected models, its performance on SST test dataset is much better than the other two models, leading to the best overall test score. It suggests that the Joint Training method is perhaps the only technique that can help to improve the model performance in the multi-task setting.

5 Analysis

From the results shown in Table 3, clearly the joint training method outperforms naive training. This coincides with our intuition that training the model on three tasks one by one will bias the model toward specific tasks. Especially, there is a clear improvement on sentiment and paraphrase tasks and a decreased performance on similarity task. This is because the model is purely trained on the similarity task at the end of each epoch and therefore it hurts the performance on the other two tasks.

Following the above intuition, I propose to further improve the model performance on the STS dataset by add an additional round of training. However, this technique will even severely hurt the model performance on the STS dataset. The performance on three dev datasets is shown in Figure 2. It demonstrates that the STS performance indeed drops with more training epochs. It is unclear why this phenomena occurs, a possible explanation could be the optimization trajectory of this training process has bad properties compared with standard Joint Training.

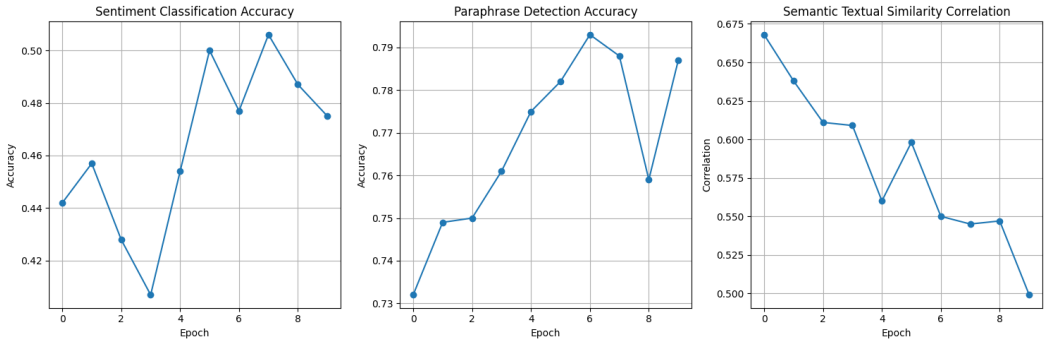

Figure 2: Performance of Finetuned + Joint Training + additional STS model on three dev datasets in each training epoch

The contrastive learning method is found to achieve the best model performance on the SST dataset (Finetuned + Joint Training + CL + balance), suggesting that the supervised contrastive method improve the quality of embeddings. It is remarkable that the Finetuned + Joint Training + CL model achieves the best performance on STS dataset even if the contrastive learning procedure doesn't explicitly use the STS datasets. It suggests that supervised contrastive learning have the ability to learn embeddings that have good transferability to other tasks, but according to the performance on Paraphrase Detection, this improvement is not universal. It is worthwhile to note that supervised contrastive learning typically benefits from large batch size to create enough contrast pairs, for example, Khosla et al. (2020) uses a batch size of 256 while I am using a batch size of 8.

Surprisingly, the gradient surgery technique doesn't work in this setting. A possible explanation is that although the gradient projection reconciles the conflicts of multi-task gradients, it may also

lose information that is beneficial to learn useful features. Therefore, it may not help to improve performance in a multi-task setting. Another potential issue is that due to the memory limit, a smaller batch size is used for PCG training, which may possibly affect the training stability.

In general, I found that the training procedure is very unstable due to its multi-task learning nature and the complexity of the optimization landscape of large language models. It is more reasonable to compare different models via training multiple times with different random seeds. In fact, as I observed in Table 3 and 4, there is a significant discrepancy even between the dev datasets and test datasets. Unfortunately, due to the limits of computational resources, it is impossible to give a thorough investigation in this project.

6 Conclusion

In this project, I evaluate the performance of minBERT embeddings in a multi-task setting. It is shown that a better design of the training procedure can improve the overall performance. However, some commonly used techniques such as gradient surgery and contrastive learning demonstrate limited power in this setting. To enhance the model performance further, a future direction is to use a larger batch size with advanced GPU, and explore the possibility of using unsupervised contrastive learning to make the embeddings more transferable.

References

Tao Chen, Haizhou Shi, Siliang Tang, Zhigang Chen, Fei Wu, and Yueting Zhuang. 2021. Cil: Contrastive instance learning framework for distantly supervised relation extraction. *arXiv preprint arXiv:2106.10855*.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

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*.

Hongchao Fang, Sicheng Wang, Meng Zhou, Jiayuan Ding, and Pengtao Xie. 2020. Cert: Contrastive self-supervised learning for language understanding. *arXiv preprint arXiv:2005.12766*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

Dieuwke Hupkes, Mario Giulianelli, Verna Dankers, Mikel Artetxe, Yanai Elazar, Tiago Pimentel, Christos Christodoulopoulos, Karim Lasri, Naomi Saphra, Arabella Sinclair, et al. 2022. State-of-the-art generalisation research in nlp: a taxonomy and review. *arXiv preprint arXiv:2210.03050*.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673.

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. *arXiv preprint arXiv:1907.11692*.

Xiao Pan, Mingxuan Wang, Liwei Wu, and Lei Li. 2021. Contrastive learning for many-to-many multilingual neural machine translation. *arXiv preprint arXiv:2105.09501*.

Jing Qian, Li Dong, Yelong Shen, Furu Wei, and Weizhu Chen. 2022. Controllable natural language generation with contrastive prefixes. *arXiv preprint arXiv:2202.13257*.

Yujia Qin, Yankai Lin, Ryuichi Takanobu, Zhiyuan Liu, Peng Li, Heng Ji, Minlie Huang, Maosong Sun, and Jie Zhou. 2020. Erica: improving entity and relation understanding for pre-trained language models via contrastive learning. *arXiv preprint arXiv:2012.15022*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022. A contrastive framework for neural text generation. *Advances in Neural Information Processing Systems*, 35:21548–21561.

Varsha Suresh and Desmond C Ong. 2021. Not all negatives are equal: Label-aware contrastive loss for fine-grained text classification. *arXiv preprint arXiv:2109.05427*.

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

Jannis Vamvas and Rico Sennrich. 2021. Contrastive conditioning for assessing disambiguation in mt: A case study of distilled bias.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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.