Applying Multitask Fine-Tuning to Sentence-BERT

Stanford CS224N Default Project

Alvin Ayuyo
Department of Computer Science
Stanford University
ayuyo17@stanford.edu

Abstract

This project aimed to improve the performance of the minBERT model on sentence-level NLP tasks by incorporating techniques from Sentence-BERT (SBERT) and applying multitask fine-tuning. SBERT modifies pre-trained transformer models like minBERT to derive semantically meaningful sentence embeddings useful for tasks like semantic textual similarity and paraphrase detection. Multitask learning has shown promise in enhancing model capabilities by jointly training on multiple tasks. The focus of this project is implementing the SBERT model architecture and objectives based on the paper "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." Preliminary results on the default tasks (SST) show performance close to the provided baselines for fine-tuning miniBERT. However, I have made a few changes in the approach suggested in the paper, an approach that has led to better results. I have used the classification objective function described in the paper for paraphrase prediction with the slight change of using a Sigmoid instead of a Softmax classifier. Moreover, I have used the regression objective function described in the paper for similarity prediction. However, I have replaced the cosine similarity calculation with a linear layer, otherwise, everything else holds. Furthermore, I have fine-tuned the pre-trained minBERT model on the three datasets that come with the default project.

1 Key Information to include

- Mentor: Olivia Lee
- External Collaborators (if you have any): No
- Sharing project: No

2 Introduction

The main aim of this project is to improve the performance of the default minBERT project on sentence-level NLP tasks. SBERT modifies pre-trained transformer models like minBERT to derive semantically meaningful sentence embeddings useful for tasks like semantic textual similarity and paraphrase detection. The tasks are as follows.

1. Sentiment prediction - which involves predicting the correct label for a review. That is whether it is positive, negative, or lies somewhere in between. For the purposes of this project, there are five possible labels for a sentence (0 - negative, 1- somewhat negative, 2- neutral, 3- somewhat positive, 4- positive).
2. Paraphrase prediction – This task entails determining whether two given sentences are restatements or paraphrases of each other. The model needs to understand the semantic similarity between the two sentences and identify if they convey the same meaning, albeit with different wording.

3. Similarity prediction – This task involves assessing the degree of similarity between two given sentences. The model needs to quantify how similar or different the sentences are in terms of their meaning and semantic content.

The main challenge in this project is to accomplish all three tasks using a single model. This is a non-trivial task because these tasks are inherently asking different questions of the model, and the only similarity that exists is that the second and third tasks each involve considering two sentences. To address this challenge, the proposed approach is to employ multitask fine-tuning, where the model is trained on all three tasks simultaneously during the same training cycle.

Multitask learning involves optimizing the model's performance on multiple tasks simultaneously by combining their respective loss functions. However, this approach is not always beneficial, as the gradient directions of different tasks may conflict with one another. Yu et al. recommend a technique called Gradient Surgery, which projects the gradient of the i-th task gi onto the normal plane of another conflicting task's gradient gj. This technique aims to mitigate the potential conflicts between gradients and facilitate more effective multitask learning.

The paper proposes various methods for achieving multitask fine-tuning, such as adding together the losses on the tasks and using the combined loss to evaluate the model's performance. The authors explore different approaches to effectively combine the tasks and leverage the strengths of multitask learning while mitigating potential drawbacks, such as conflicting gradients and task interference.

3 Related Work

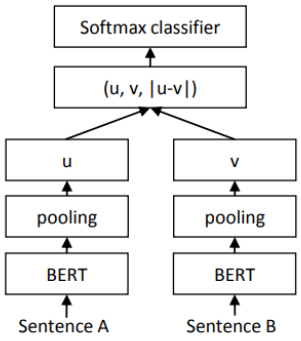

Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

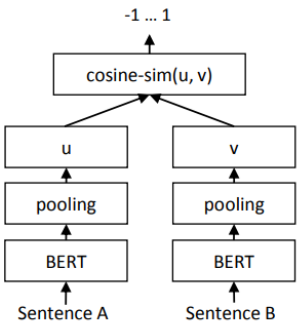

Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

The main paper Reimers and Gurevych (2019) that inspired this project presents Sentence-BERT (SBERT), a modified version of pre-trained transformer networks like BERT and RoBERTa, designed to address the limitations of BERT in tasks involving sentence-pair regression and semantic similarity search. While BERT excels in various NLP tasks, it requires inputting both sentences into the network for each comparison, leading to substantial computational overhead. The authors emphasize that tasks such as finding the most similar sentence pair in a large collection or identifying the most similar question in datasets like Quora can be computationally intensive with BERT. For instance, comparing 10,000 sentences would require approximately 65 hours due to the vast number of possible combinations.

To tackle this issue, the authors propose SBERT, which employs Siamese network structures to derive semantically meaningful sentence embeddings. These embeddings enable efficient

cosine-similarity comparisons, reducing the computational effort from 65 hours with BERT to around 5 seconds with SBERT. Furthermore, SBERT's embeddings are demonstrated to outperform other state-of-the-art methods, such as averaging BERT outputs or utilizing the [CLS] token for sentence embeddings.

The authors aim to solve the problem of computational inefficiency in tasks like semantic similarity comparison, clustering, and information retrieval via semantic search by leveraging SBERT. This approach enables the use of BERT for tasks that were previously infeasible due to computational constraints. Moreover, SBERT's fine-tuning process takes about 20 minutes and yields superior results compared to other sentence embedding methods, highlighting its efficiency and effectiveness across various NLP tasks.

Multi-Task Learning over BERT Qiwei Bi (2022)for News Recommendation paper, there are two methods proposed for improving the performance of multitask fine-tuning.

1. For their specific situation, they optimize the loss function of the main task, category classification, and NER task simultaneously, which derives the final loss function.

$$\mathcal{L}_{\text{MTRec}} = \mathcal{L}_{\text{Main}} + \mathcal{L}_{\text{Category}} + \mathcal{L}_{\text{NER}}.$$

2. Multi-task learning is not always beneficial, since there may exist gradient conflicts among different tasks. The problem means that the gradient directions of different tasks form an angle larger than 90◦ and thus harm each other. To alleviate this issue, Yu et al. Tianhe Yu (2020) propose a technique called Gradient Surgery (GS) that projects the gradient of the i-th task gi onto the normal plane of another conflicting task's gradient gj.

$$\mathbf{g}_i = \mathbf{g}_i - \frac{(\mathbf{g}_j \cdot \mathbf{g}_i)}{\|\mathbf{g}_j\|^2} \cdot \mathbf{g}_j.$$

In my implementation, I am optimizing all three tasks separately and using the total loss only as a way to understand the performance of the model. I use this approach as opposed to the proposed one to prevent the model from overfitting the training data. Moreover, I have not addressed the gradient concern, this could be an interesting direction to take in future improvements.

4 Approach

1. I have initialized 3 linear layers, one for each task, and a dropout layer as shown below.

```
self.dropout = torch.nn.Dropout(config.hidden_dropout_prob)
self.sentiment_classifier = torch.nn.Linear(config.hidden_size, config.num_labels)
self.paraphrase_classifier = torch.nn.Linear(3 * config.hidden_size, 1)
self.similarity_regressor = torch.nn.Linear(2 * config.hidden_size, 1)
```

2. In the forward function, I am using MEAN pooling on the last hidden state. This is the same approach used in the paper that inspired this project. This is what is returned by the function.
3. In predict sentiment function, I get embeddings from the forward function, apply dropout to it then pass it through the sentiment classifier layer defined above to get the logits.
4. In the predict paraphrase function, I get the embeddings for both batches. I then concatenate the embeddings of the first batch and those of the second batch to the difference between the two batches. I then apply a dropout before passing it through the paraphrase classifier layer. This is a method that is described in the motivating paper for classification tasks but I have used it for paraphrase prediction instead with convincing results.

5. In the predict similarity layer, I get batch embeddings for both batches. I then concatenate them before applying dropout. I then pass it through the similarity regressor layer. This differs from the approach in the paper that uses cosine similarity instead.
6. In the train multitask function, I fine-tune with all three datasets, that is Stanford Sentiment Treebank (SST) dataset, Quora Dataset, and SemEval STS Benchmark Dataset, in that exact order every epoch. To predict sentiment I use the SST dataset and optimize cross entropy loss. For, predict paraphrase I use the Quora dataset and optimize binary cross entropy with logits. To predict similarity I use the SemEval dataset and optimize mean square error loss, this is the chosen loss function in SBERT for use with cosine similarity.
7. A benchmark I was aiming for was to get performance on SST dataset comparable to part 1 of the project while still improving on the two other tasks.

5 Experiments

This section contains the following.

5.1 Data

To predict sentiment I use the SST dataset. For, predict paraphrase I use the Quora dataset. To predict similarity I use the SemEval dataset. All these are datasets provided in the default project.

5.2 Evaluation method

To predict sentiment I optimize cross entropy loss. To predict paraphrase optimize binary cross entropy with logits. To predict similarity I optimize mean square error loss, this is the chosen loss function in SBERT for use with cosine similarity. Binary cross entropy loss with logits is a combination of two functions commonly used in binary classification tasks:

1. Sigmoid Activation Function: The sigmoid function maps real-valued inputs to the range (0, 1), essentially turning logits (raw prediction scores) into probabilities.
2. Binary Cross Entropy Loss: Measures the difference between the predicted probabilities and the actual binary labels.

In deciding when to save a model, I used the dev accuracies across all three tasks. That is to say, I only saved a model if the dev accuracies for all three tasks were an improvement over the accuracies of an already saved model if any had been saved. I got the dev accuracies by using the function

```
model_eval_multitask(sst_dev_dataloader, para_dev_dataloader, sts_dev_dataloader, model, device)
```

5.3 Experimental details

Report how you ran your experiments (e.g., model configurations, learning rate, training time, etc.) Below is some information about the training phase.

1. All training was done for 10 epochs.
2. The average training time for one epoch was about 21 minutes, it was taking close to 3 hours to train on all 3 datasets.
3. Below are some configurations I used when running the training. These configurations are appended at the end of the line

```
python3 multitask_classifier.py
```

```
--fine-tune-mode full-model --lr 1e-5 --batch_size 64 --use_gpu
--fine-tune-mode last-linear-layer --lr 1e-5 --batch_size 64 --use_gpu
```

4. I used batch size 64 to improve the training time of the model.

5. Moreover, I also applied the cosine similarity approach to both the predict paraphrase and predict similarity tasks. I also applied the classification objective visualized in Figure 1 to both paraphrase and similarity tasks.

5.4 Results

Below is a table showing the results of various configurations.

- Classification objective is the approach demonstrated in Figure 1
- Cosine similarity is the approach demonstrated in Figure 2 and is the motivating paper's preferred approach.
- Mixed objective means using classification objective for predict paraphrase and cosine similarity for predict similarity
- Best approach is the result of following the setup I have described in this paper.

Configuration	SST	Quora	SemEval
Classification Objective	0.207	0.368	-0.020
Cosine Similarity Objective	0.409	0.689	-0.308
Mixed Objective	0.387	0.743	0.160
Best Approach	0.480	0.889	0.338

Table 1: dev accuracies and correlation for SemEval

6 Conclusion

Summary of Findings
This project aimed to enhance the performance of the minBERT model on sentence-level NLP tasks by incorporating techniques from SBERT and applying multitask fine-tuning. The key findings are:

1. The multitask learning approach, with specific modifications, achieved better results than traditional single-task training methods.
2. The implementation of different loss functions for each task contributed to the model's improved performance, particularly in paraphrase and similarity predictions.
3. The implementation of different loss functions for each task contributed to the model's improved performance, particularly in paraphrase and similarity predictions.
4. Multitask fine-tuning with Gradient Surgery as a future direction could potentially address the gradient conflict issue and further boost performance.

Achievements

1. Implemented a multitask learning model using SBERT-inspired techniques, resulting in improved performance on paraphrase prediction.
2. Demonstrated the viability of multitask fine-tuning, achieving satisfactory results across all three tasks.

Limitations

1. The model still struggles with nuanced sentiment predictions and abstract similarity assessments.
2. Gradient conflicts during multitask training were not fully addressed, potentially limiting the model's performance.

Future Work Future improvements could focus on:

1. Incorporating Gradient Surgery to better handle gradient conflicts during multitask learning.
2. Experimenting with advanced pooling techniques and different network architectures to enhance sentence embeddings.

3. Conducting extensive robustness testing to ensure the model's generalizability and fairness across diverse datasets.

By addressing these limitations and exploring new directions, future iterations of this project could yield even more robust and accurate NLP models for multitask learning.

7 Ethics Statement

While this project focused on technical model development, it's important to consider the ethical implications of language models and their potential misuse. Semantic understanding capabilities could be leveraged for disinformation, generation of biased or offensive content, or invasion of privacy through analyzing personal text data.

Mitigation strategies should involve robustness testing to identify and address biases, transparency around model capabilities/limitations, and institutional oversight regarding high-stakes applications. It's also crucial to consider environmental impacts from the immense computational resources required for large language model training.

Ultimately, this highlights the need for a multistakeholder approach - bringing together technologists, ethicists, policymakers, and impacted communities - to navigate the challenges of democratizing natural language AI capabilities responsibly.

References

Lifeng Shang Xin Jiang Qun Liu Hanfang Yang Qiwei Bi, Jian Li. 2022. Mtrec: Multi-task learning over bert for news recommendation.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *https://arxiv.org/pdf/1908.10084*.

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