BERTology: Improving Sentence Embeddings for Multi-Task Success

Stanford CS224N {Default} Project

Kyuil Lee
Department of Computer Science
Stanford University
kyuil@stanford.edu

Abstract

Training BERT to simultaneously perform well on multiple downstream tasks can be challenging due to the lack of data for each task as well as conflicting objectives between the different tasks. For this paper, we focus on three downstream tasks - sentiment analysis (SST), paraphrase detection (PARA), and semantic textual similarity (STS). We explore various methods that can avoid the problem of conflicting objectives while learning to perform well on all three tasks simultaneously. We first explore various schemes to share weights while freezing different parts of the model in order to avoid conflicting training objectives. We then also explore methods to improve the general quality of our sentence embeddings, including contrastive learning and using a triplet network structure. We find that sharing weights with freezing has limited success, while improving sentence embeddings via triplet network structures works best. We report on our results on the three downstream tasks.

1 Key Information to include

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

2 Introduction

Given the limited labeled data for our downstream tasks, we need to rely on transfer learning to achieve good performance. Having a larger pretrained model that is also trained on more data, such as BERT-large or GPT3+ is one such solution to our problem. The problem constraint we have for this paper is to limit the pretrained model size to the smaller BERT-base model, while also achieving the best performance we can on the downstream tasks. We explore various ways to maximize the effectiveness of our limited dataset, as well as ways to augment the base model by improving the overall quality of the sentence embeddings generated by the model.

The downstream tasks are sentiment analysis (SST), paraphrase detection (PARA), and semantic textual similarity (STS). For SST, the goal is to predict an integer label from 0 to 4 given a sentence, which signifies the sentiment of the sentence from negative to positive. In paraphrase detection, we output a binary value indicating whether the input pair are paraphrases of one another. Finally, for STS, we output a similarity score (float) between the input pair between 0 and 5, indicating how similar we perceive the two sentences to be.

To deal with the limited size of our training data, especially for SST and STS, which contain less than 10,000 examples each, we experiment with various architectures where weights are shared among the three tasks in the hopes that such a structure would allow the model to leverage the data for the other

tasks, as well as its own. For example, we try to share weights between STS and PARA since these two tasks are related in that they try to measure the similarity between a pair of sentences. We also try to avoid conflicting learning objectives between the different tasks by designing an architecture where we first finetune the model for PARA and STS, and then freeze the weights when training the task head for SST. We also try an architecture where we convert the SST problem into a STS problem, where we sample 500 examples from the training data for SST (100 for each label from 0 to 4), and then calculate the similarity score between the input sentence and each sentence in this 'ensemble' of sampled sentences, and throw the result into a linear layer. We provide as information how similar the input sentence is to sample points with known labels and use that to make a prediction on the class label. Overall, we find such methods to have limited success.

In order to deal with the lack of data, we also finetune the model on two external datasets (SNLI Bowman et al. (2015) and MNLI Williams et al. (2018)) in order to increase the general quality of our sentence embeddings before doing any training on downstream tasks. We try multiple objectives, including contrastive learning Gao et al. (2021), regression with cosine similarity, as well as a triplet learning objective detailed in Reimers and Gurevych (2019) where we try to minimize the distance between similar sentences and maximize the distance between negative sentences. We find that the triplet learning objective works best.

3 Related Work

Sun et al. (2019) describes general strategies where one can finetune a BERT model for sentence classification tasks. Sun also describes a multi-task scheme, where the underlying BERT model is shared while each task head gets an additional linear layer to morph the output into one suitable for each task.

Chen et al. (2021) gives a holistic overview of multi-tasking strategies, including parallel and hierarchical methods, some of which inspired a few of the models presented in this paper. Chen describes various methods to share partial layers in the model across different tasks to achieve better results.

Gao et al. (2021) and Reimers and Gurevych (2019) give ideas on tuning sentence-level embeddings using the SNLI and MNLI datasets using different training objectives. Gao et al. (2021) describe a contrastive learning method, where similar pairs count negatively while negatively related pairs count positively towards the overall loss. This loss is calculated at the batch level, where negative pairs are dynamically generated by paring up unrelated sentences together as pairs within the batch, whereas positive pairs come from the training data. Reimers and Gurevych (2019) gives a triplet training objective that does something similar, which minimizes the distance between the pair of sentences that are positively related and maximizes the distance between sentences that are negatively related. However, this triplet comes straight from the training data instead of dynamically adding negative data from the batch, and also the formulation is a bit different, as explained in later sections.

4 Approach

4.1 Baseline

Our baseline consists of fine-tuning BERT-base on the three individual tasks simultaneously. At each epoch, we go through all of the training data for each task, and when that is finished, move onto the next task.

Each task contains a task head that is unique to the task. The SST head contains a dropout layer and a linear layer to convert the vector embedding to a 5-dimensional logit, which is then used to calculate the cross-entropy loss against the true labels. The PARA head uses the same underlying BERT-base to convert the two input sentences into respective embeddings, concatenates them, and then applies a dropout and linear layer to output a single logit, which is then used for binary cross-entropy loss. Finally, STS head similarly converts the two input sentences into embeddings, concatenates them, and then applies a dropout and linear layer to generate a single number, which is then passed into mean-squared-error loss (MSE loss).

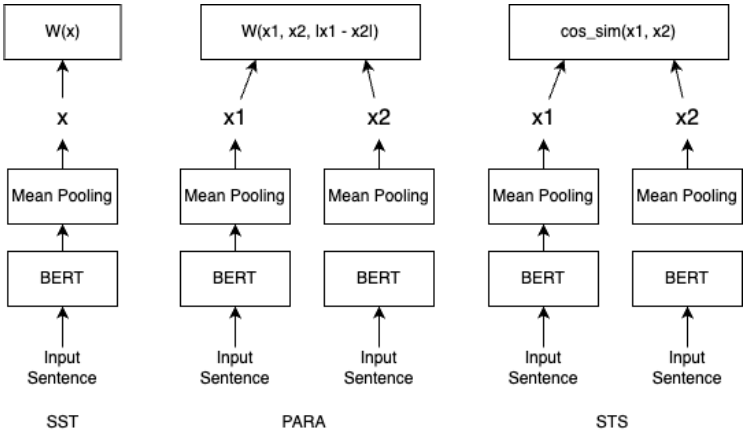

Figure 1: Model architecture after updating STS and PARA heads

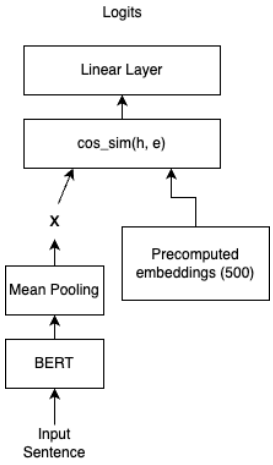

Figure 2: Forcing SST to hierarchically depend on STS

4.2 Sharing Weights between STS and PARA

We had the intuition that STS and PARA were inherently similar tasks in that they try to predict the similarity of the two input sentences. Therefore, we try an architecture where both STS and PARA share 2 linear layers interleaved with dropout layers and RELU activation, after which each have a different final linear to output the final value. We train the model by training all three tasks in each epoch, where we go through all of the training data for each task, one at a time.

4.3 Modifying STS and PARA heads

4.3.1 Using Cosine Similarity for STS

Looking at a few previous papers such as Reimers and Gurevych (2019), we found that using cosine similarity for STS may drastically improve our model's performance. We simplify the STS head by simply outputting the cosine similarity of the input embeddings. Since the data consists of similarity scores between 0 to 5, we scale the scores during training to be between 0 and 1 in order to match the output of cosine similarity. We scale to $[0, 1]$ instead of $[-1, 1]$ because empirically it gave us better results. After applying cosine similarity, we still use MSE loss between the cosine similarity value and the scaled true similarity score.

4.3.2 Modifying the PARA head

We also found that instead of just concatenating the two input embeddings to pass to the linear layer, passing it extra information that captures some relationship between the two input embeddings helped improve the result for PARA. We try the following two approaches, inspired by Reimers and Gurevych (2019): concatenate the absolute element-wise difference between the two input embeddings to the two input embeddings to pass to a linear layer, and concatenate the cosine similarity of the two input embeddings to the two input embeddings.

$$o = W(u, v, |u - v|) \tag{1}$$

$$o = W(u, v, \cos\_\text{sim}(u, v)) \tag{2}$$

4.4 Freezing weights for SST

We found that while the PARA and STS results were quite good using the cosine similarity and PARA head approaches described above, the STS results did not improve much from the baseline. We also

found that training SST in conjunction with STS and PARA caused the PARA and STS scores to detriment significantly. Therefore, we first try training STS and PARA in conjunction, after which we freeze the weights to train SST. Since the underlying weights were frozen, we modified the SST head to be much more complex, with 4 additional BERT layers added on top of the existing BERT, as well as an additional dropout and linear layers to output logits.

4.5 Forcing SST to hierarchically depend on STS

Another approach we tried was to essentially convert the SST problem into a STS problem by sampling 500 examples from the SST training data, 100 from each possible label, and comparing the similarity between the input embedding and the embeddings for these 500 data points. The intuition was that by knowing how close the input is to data points with known labels, we could gain a better sense of the label of the input. The embeddings for the 500 data points were pre-computed, and loaded into the model for efficiency. Finally, the 500 similiarity scores are concatenated with the true labels of the sample data points to pass into a linear layer, which outputs logits. Essentially, this converts a classification problem into one that is simliarity matching with known samples.

4.6 Enhancing sentence embeddings with external datasets

We try enhancing the general quality of the sentence embeddings generated by BERT by fine-tuning the model on two external datasets: SNLI (Bowman et al. (2015)) and MNLI (Williams et al. (2018)). The SNLI and MNLI training data consists of pairs of sentences, labeled as entailment, neutral, or contradiction. We try three different learning objectives for this fine-tuning process.

4.6.1 Regression objective

First, we try a method that is similar to STS training - We assign 0.0 to contradiction, 0.5 to neutral, and 1.0 to entailment for the SNLI and MNLI datasets, and then use cosine similarity paired with MSE loss to train the data, similar to STS.

4.6.2 Contrastive learning

A second method we try is using contrastive learning, inspired by (Gao et al. (2021)). The SNLI and MNLI datasets can be converted into a triplet sentence format where each sentence has both and entailment and a contradiction. This can be done by matching the rows against the given promptID for MNLI or captionID for SNLI. Then, we have the sentence triplets (x_i, x_i^+, x_i^-), where x_i^+ is the entailment and x_i^- is the contradiction for x_i. We implement the following learning objective:

$$-\log \frac{\exp[\text{sim}(h_i, h_i^+)]}{\sum_{j=1}^{N}(\exp[\text{sim}(h_i, h_j^+)] + \exp[\text{sim}(h_i, h_j^-)])} \tag{3}$$

where h_i, h_i^+, h_i^- are the embeddings of x_i, x_i^+, x_i^-, respectively. N is the size of each minibatch. In the process, we modify the system to handle sentence triplets.

4.6.3 Triplet objective from SBERT

A third method inspired by Reimers and Gurevych (2019) is to use a different type of triplet objective, which is designed to maximize the distance between the positively related sentence embeddings (entailments) and maximize the distance between the negatively related sentence embeddings (contradictions).

$$max(||h_i - h_i^+|| - ||h_i - h_i^-|| + \epsilon, 0) \tag{4}$$

For the distance, we use Euclidean distance, and for ϵ, we use the value of 1, as suggested in the paper. The ϵ value ensures that ensures that h_i is at least closer to h_i^+ than h_i^-.

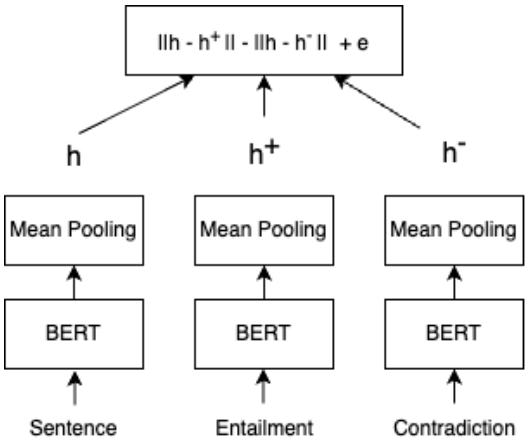

Figure 3: Triplet objective for finetuning sentence embeddings

5 Experiments

5.1 Data

We use the Stanford Sentiment Treebank Socher et al. (2013) for sentiment analysis (SST), the Quora Question Pairs Iyer et al. (2017) for paraphrase detection (PARA), and the Semantic Textual Similarity dataset Agirre et al. (2013) for STS. The SST dataset consists of sentences paired with a sentiment label, ranging from 0 to 5. The Quora dataset consists of sentence pairs with a 0 or 1 depending on whether the sentences are paraphrases of each other. The STS dataset consist of sentence pairs with a similarity score ranging from 0.0 to 5.0.

We also use the MNLI Williams et al. (2018) and SNLI Bowman et al. (2015) datasets for regression, contrastive learning, and tripet objectives described above for improving general sentence-embeddings.

Figure 4: Dataset size

	Training	Dev	Test
SST	8545	1101	2210
PARA	141507	20214	40431
STS	6041	863	1726
SNLI Pair	550152		
SNLI Triplet	151196		
MNLI Pair	392703		
MNLI Triplet	128737		

We omit the Dev and Test set sizes for SNLI and MNLI since they were not used. Only the training data was used to fine-tune the sentence embeddings in one epoch.

5.2 Evaluation method

Both SST and PARA use prediction accuracy as an evaluation metric, since both are classification problems. For STS, we use the Pearson Correlation between the true values and the predicted similarity values, similar to the original SemEval paper Agirre et al. (2013).

Finally, to measure the overall success of the model, we calculate the following formula by normalizing and averaging the three scores:

$$score = \frac{\text{sst\_accuracy} + \text{paraphrase\_accuracy} + (\text{sts\_correlation} + 1)/2)}{3} \tag{5}$$

which is the metric used for the leaderboard.

5.3 Experimental details

There are a total of 9 experiments:

- Baseline (BASELINE),
- Sharing weights between STS and PARA (SHARED),
- Modifying STS and PARA heads with cosine similarity and absolute distance between embeddings (STS_PARA_HEAD_MOD),
- Modifying the PARA head by providing it the cosing similarity between the input embedding pair (STS_PARA_HEAD_MOD_1),
- Freezing weights for SST (FREEZE_SST),
- Forcing SST to hierarchically depend on STS (SST_TO_STS),
- Enhancing sentence embeddings using SNLI and MNLI with the regression objective, where we fine-tune the output further by applying STS_PARA_HEAD_MOD (NLI_REGRESSION).
- Using contrastive learning with SNLI and MNLI, and then following by STS_PARA_HEAD_MOD. (NLI_CONTRASTIVE),
- Uinsg the triplet objective with SNLI and MNLI, and then following by STS_PARA_HEAD_MOD (NLI_TRIPLET).

I provided some code words for the experiments above so that you can refer to them in the results section.

We use a batch size of 32, a learning rate set to 2e-05, and a dropout rate of 0.1 for all of the experiments, which we empirically found to be the best. During training, we only save the model parameters that give the best dev leaderboard score.

5.3.1 Parallel training

We employ the same training strategy for the following experiments: BASELINE, SHARED, STS_PARA_HEAD_MOD, STS_PARA_HEAD_MOD_1. We go task-by-task per epoch, where we finish going through all of the data for one task before moving onto the next task at each epoch. The order of the tasks is given by SST, PARA, then STS.

5.3.2 Partial freezing

For FREEZE_SST and SST_TO_STS, we first train the model on PARA and STS tasks, achieving the best dev score we can get. We follow up by freezing all of the non-SST weights (including BERT-base), and then train the model on SST data.

5.3.3 Finetuning sentence embeddings with NLI

For the experiments involving SNLI and MNLI datasets to enhance sentence embeddings, we first train the model sequentially on the SNLI and MNLI data for one epoch. Afterwards, we train the model in the same strategy as in the STS_PARA_HEAD_MOD experiment. It takes around 2 hours to go through one epoch of the NLI data on a NVIDIA Tesla P100 GPU.

5.4 Results

The results are described in 5 and 6.

As you can see, the NLI_TRIPLET experiment performed the best with a total dev score of 0.784. It performed better than all of the other experiments on all fronts.

For the test leaderboard, the NLI_TRIPLET experiment obtained the following results:

The results were better than expected for NLI_TRIPLET, and showed that the triplet objective described in Reimers and Gurevych (2019) worked well in creating generalizable sentence embeddings that were easily extended to downstream tasks. The change that had the biggest impact was modifying the head for STS to cosine similarity, which saw a jump from 0.328 to 0.721. This strategy continued

Figure 5: Summary of Experiments on Development Set

	SST	PARA	STS	DEV_TOTAL
BASELINE	0.479	0.634	0.347	0.596
SHARED	0.513	0.772	0.328	0.650
STS_PARA_HEAD_MOD	0.480	0.752	0.721	0.697
STS_PARA_HEAD_MOD_1	0.482	0.748	0.726	0.698
FREEZE_SST	0.420	0.775	0.758	0.691
SST_TO_STS	0.358	0.775	0.758	0.671
NLI_REGRESSION	0.501	0.823	0.850	0.750
NLI_CONTRASTIVE	0.490	0.790	0.802	0.727
NLI_TRIPLET	0.532	0.881	0.880	0.784

Figure 6: NLI_TRIPLET on Test Set

	SST	PARA	STS	TEST_TOTAL
NLI_TRIPLET	0.525	0.876	0.862	0.777

to work well for the rest of the experiments. In fact, having cosine similarity as the head seems to help the performance on other tasks as well, since it makes the overall model a bit simpler.

6 Analysis

The methods which finetuned all of the weights without freezing did better than the ones that partially froze weights. For FREEZE_SST and SST_TO_STS, only training on PARA and STS first allowed the model to perform better on those two tasks. However, the separate head for SST failed to generalize well to dev sets, even though it was able to achieve a high training accuracy (0.9+). Overall, we've found that such task-specific freezing methods allowed us to train a smaller subset of the tasks during the fine-tuning without freezing (PARA and STS), which allowed us to get a higher score on these two tasks. However, the overall score was lower since our performance for SST was too low.

Our most interesting approach, SST_TO_STS, failed to perform well. It seems that the problem was too complex for such an ensemble-like method to work well (ensemble of preselected embeddings). One thing that we would have liked to try was to simply use all of the training data for SST as the ensemble of pre-computed embedding vectors, after which we simply return the label of the most closely related vector from the ensemble after performing cosine-similarity with the input vector. We briefly tried with 500-1000 data points in the ensemble, but it did not work yet. It might be interesting to see how the model performs with a larger ensemble.

Another interesting fact that we found was that scaling the true labels for STS to 0 to 1 instead of -1 to 1 performed better. This was initially strange, because cosine similarity naturally falls between -1 and 1. We reasoned that it may be because it is easier for the model to represent a negative relation as a perpendicular vector rather than an anti-parallel vector.

Bringing in external datasets (SNLI, MNLI) seem to overall enhance the performance of the model, regardless of the training objective used.

Overall, for larger datasets (PARA, SNLI, MNLI) - most of the training gains come from 1-2 epochs and no more. As soon as the training went over those limits, the model started overfitting heavily. It goes to show that when the datasets are large, we do not need many training epochs to learn a general representation of the data.

7 Conclusion

Summarize the main findings of your project and what you have learned. Highlight your achievements, and note the primary limitations of your work. If you'd like, you can describe avenues for future work.

We have performed 8 significant experiments, that required custom model architectures as well as varying training strategies. Out of all experiments performed, fine-tuning general sentence

embeddings with the triplet objective seems to work best. The experiments where we used an external dataset seem to enhance to model overall for all tasks involved, and combined with the modifications to the various task heads, we were able to gain a significant boost to our performance on the development set.

We would have also liked to experiment with more fine-grained experimentation strategies, such as dynamically choosing which task to focus on over other tasks, which could help balance the losses across all tasks. Another thing we would like to try in the future would be to finetune the sentence embeddings using all three objectives that we tried at the same time (regression, contrastive learning, and triplet objective). We are curious if such a method would result in a better generalizable sentence embeddings.

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, Atlanta, Georgia, USA. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Shijie Chen, Yu Zhang, and Qiang Yang. 2021. Multi-task learning in natural language processing: An overview. *CoRR*, abs/2109.09138.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. 2017. First quora dataset release: Question pairs. `https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs`.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew 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, Seattle, Washington, USA. Association for Computational Linguistics.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.