The Best of BERT Worlds: Improving minBERT with multi-task extensions

Stanford CS224N Default Project

Julius Hillebrand
Department of Computer Science
Stanford University
juliush@stanford.edu

Abstract

This paper aims to enhance BERT's performance across various NLP tasks, including sentiment analysis, paraphrase detection and semantic textual similarity, amongst others by leveraging the SBERT architecture as proposed by Reimers and Gurevych (2019). It thereby demonstrates SBERT's utility in generating sentence embeddings for a broader range of tasks.

Through a combination of experiments, I manage to significantly improve BERT's scores on the three tasks and enable it to effectively multitask. These improvements are mainly driven by **(a)** cosine-similarity fine tuning, **(b)** using a mean-pooling strategy to generate sentence embeddings, **(c)** concatenating embeddings for paraphrase detection, and **(d)** summing individual losses for multi-task learning (Bi et al., 2022). I find that these adjustments prove to be particularly effective for paraphrase detection and semantic textual similarity, while doing little to improve sentiment analysis.

1 Key Information to include

- Mentor: Heidi Zhang
- External Collaborators: N/A
- Sharing project: N/A

2 Introduction

The *Bidirectional Encoder Representations from Transformers* model (from hereon referred to as "BERT") proved to be an important landmark for the development of foundational models and more widely for the field of NLP. However, while the base BERT model does well with generating contextual word representations (Devlin et al., 2019)—and in our case, sentence embeddings—it can struggle to generalize when tested on multiple downstream tasks, e.g. due to task-specific nuances. Exhibiting strong performance across different tasks is inherently desirable, as well as being cost- and resource efficient.

In this paper, I present modifications to the BERT model that allow it to effectively multitask across the three tasks of sentiment analysis, paraphrase detection and semantic textual similarity (STS). Sentiment analysis classifies opinions expressed in text to determine whether the sentiment towards a specific topic or product is positive, negative, or neutral. Paraphrase detection is the task of determining whether two given text segments express the same meaning, even if they use different wording or structure. Semantic textual similarity involves assessing the degree to which two pieces of text carry the same meaning, ranging from being completely unrelated to being semantically equivalent, even if not identically phrased. Importantly, similarity is measured on a scale, whereas paraphrase detection uses binary labels.

To improve performance, four extensions to the base model are introduced. First, drawing from the work of Reimers and Gurevych (2019), I use cosine-similarity to compare two sentence embeddings for the semantic textual similarity task, firmly improving the multitask-BERT performance. Second, I adjust sentence embeddings during the model's forward passes by using mean pooling of the last hidden state rather than using the hidden state of the [CLS] token. Third, I concatenate embeddings for the specific task of paraphrase detection, using a combination of absolute embedding differences and the element-wise product of the two embeddings to better capture aspects of similarity between the two sentences. Fourth, I implement multi-task learning as described by Bi et al. (2022) to update the BERT embeddings during training, effectively adding together the losses of the three tasks. Taken together, these extensions present a multitask BERT model that significantly improves upon the baseline set by the base BERT model.

3 Related Work

Multiple extensions of the base BERT model have worked to enhance BERT's adaptability and efficiency for a wide array of NLP tasks. By addressing the limitations of the original model through various architectural improvements and training strategies, these developments pave the way for more robust and versatile models capable of generalizing across multiple downstream tasks. Hereby, popular adaptions include domain-specific BERT versions (e.g. BioBERT for biomedical text (Lee et al., 2019)) and models with increased capacity or modified architectures (e.g. RoBERTa ((Liu et al., 2019), XLNet (Yang et al., 2020)). For the three downstream tasks this paper is concerned with, the extensions primarily fall into a third category, mainly involving techniques for task-specific fine tuning.

As such, the approach presented in this paper is grounded in existing work that has been done in this area. While they focused on building siamese and triplet networks for sentence embeddings, Reimers and Gurevych (2019)'s paper on the SBERT architecture has extensively presented the value of using cosine similarity for sentence embedding comparison. The same goes for mean pooling, for which the SBERT paper has been highly influential by demonstrating a practical application of mean pooling within sentence embedding generation. Concatenating absolute embedding differences as well as their element-wise product is inspired by InferSent (Conneau et al., 2017) and Universal Sentence Encoder (Cer et al., 2018), and has also been used in SBERT. Lastly, the modified loss function, whereby I aggregate losses from multiple tasks during the training process, is drawn from Bi et al. (2022), who specifically apply multi-task learning to the context of news recommendation systems, combining the tasks of category classification and named entity recognition (NER) to enhance the system's performance.

4 Approach

4.1 Baseline Architecture

My model follows the base BERT architecture as outlined in the original BERT paper by Devlin et al. (2019). As such, it consists of an embedding layer and 12 encoder layers.

More specifically, BERT first converts sentence input into one of 30,000 different word tokens through a WordPiece tokenizer, which are then further converted into token ids. These ids are put in to the embedding layer, which sums token embeddings, segmentation embeddings and position embeddings to create input embeddings. Within the 12 encoder layers, BERT crucially makes use of the Transformer architecture (Vaswani et al., 2017), consisting of multi-head attention that allows the model to weigh the importance of different words relative to each other within a sentence, an additive and normalization layer with a residual connection, a feed-forward layer and another additive and normalization layer with a residual connection (Figure 1). This architecture enables the model's understanding of the context surrounding each word, using both preceding and following words to generate embeddings.

BERT then outputs the embedding for each word piece of the sentence from the last encoder layer, as well as the [CLS] token embedding, which is the token prepended to the token representation of each input sentence. For our baseline results, I used the [CLS] embedding to predict on the three downstream tasks.

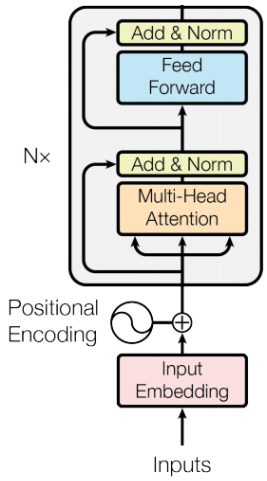

Figure 1: Encoder Layer used in BERT (based on (Vaswani et al., 2017)

Using this default BERT implementation, I added task-specific layers for effective multitasking:

For sentiment classification, I am applying a dropout layer to prevent overfitting, setting dropout probability to 10%. I am then applying a linear layer to transform the embeddings into a vector consisting of five output values, each corresponding to the score for one of the sentiment classes. During training, I am then optimizing for minimizing cross-entropy loss.

For both paraphrase detection and similarity classification, I compute the difference between the two embeddings, apply the dropout layer and then pass the result through a linear layer resulting in one logit (paraphrase detection is a binary classification task, and similarity classification predicts a continuous score). Paraphrase detection then minimizes binary cross entropy loss during training. For similarity prediction, I minimize mean squared error loss.

For the baseline scores, I finetuned the weights training only on the SST dataset, and then evaluated these weights' performance on the three datasets. For all the following extensions, I finetuned by training on all three datasets.

4.2 Cosine Similarity

To improve upon the baseline, the semantic textual similarity prediction was modified to use cosine similarity for comparing two sentence embeddings. Compared to the baseline approach, this extension simplifies the prediction process, not using a dropout or linear layer. It takes the two embeddings, computes their cosine similarity and returns this similarity scaled by a factor of 5 to account for the range of similarity scores in our dataset. Hereby, cosine similarity is defined as follows:

$$similarity = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\max(\|\mathbf{x}_1\|_2, \epsilon) \cdot \max(\|\mathbf{x}_2\|_2, \epsilon)} \tag{1}$$

where ϵ is set at the default of 1e-8 to avoid division by zero.

4.3 Mean Pooling

While the baseline uses the CLS pooling output, I switched to a mean pooling strategy using the last encoding layer's hidden states, i.e. the token embeddings for each input token. Instead of relying on a special token like [CLS], the mean pooled vector effectively condenses the information from all the tokens into a single vector by taking the average of all token embeddings, ignoring the padding tokens through a binary attention mask. Together with 4.2, this yields the architecture shown in Figure 2.

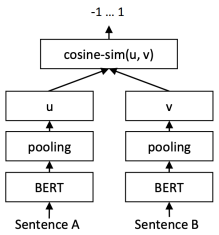

Figure 2: Architecture for STS prediction, adapted from Reimers and Gurevych (2019)

4.4 Multi-Task Learning

I modified the loss function used during training to incorporate the individual losses from each task in one multitask loss function:

$$L = L_{sentiment} + L_{paraphrase} + L_{similarity} \tag{2}$$

Where the individual loss functions remain unchanged from the baseline architecture, i.e. cross-entropy, binary cross-entropy and mean squared error loss, respectively.

4.5 Concatenation Mode

Lastly, I modified the baseline predictor for paraphrase detection by concatenating different representations of the two sentence embeddings u and v. First, I computed both the element-wise difference $|u - v|$ and the element-wise product $u * v$, which can both be seen as different measures of similarity—if two sentences are very similar, the absolute difference between them should be small, and the element-wise product should be large. I then concatenated these two measures and applied a dropout layer with the previous dropout probability of 10%. Further, I passed it through a linear layer, transforming it back to the original size, i.e. the size of the hidden layer, applied a ReLu and then used another linear layer to output a single logit.

5 Experiments

5.1 Data

For sentiment analysis, I am using the Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013). It consists of 11,855 single sentences extracted from movie reviews. Sentiments are labeled as one of negative, somewhat negative, neural, somewhat positive, or positive. I am using a train/dev/test set split with 8,544 `train`, 1,101 `dev` and 2,210 `test` examples.

For paraphrase detection, I am using a subset of the Quora dataset (Quora, 2017), which contains 400,000 question pairs with binary labels indicating whether particular pairs are paraphrases of one another. Here, my split is 141,506 `train`, 20,215 `dev` and 40,431 `test` examples.

For STS, I am using the SemEval STS benchmark dataset (Agirre et al., 2013), consisting of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). I am using a split with 6,041 `train`, 864 `dev` and 1,726 `test` examples.

5.2 Evaluation method

For sentiment analysis and paraphrase detection, I will evaluate accuracy as $\frac{correct\ predictions}{total\ predictions}$.

Due to being computed on a continuous scale, semantic textual similarity will be be evaluated on Pearson correlation of the true similarity values against the predicted similarity values, i.e.

$$\frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

where n is the number of samples, X_i and Y_i are the individual samples of the true and predicted similarity scores, and $\bar{X}$ and $\bar{Y}$ are the mean values of the true and predicted similarity scores.

For evaluating multitask performance across these three tasks, we average the three scores.

5.3 Experimental details

All experiments but the last one were trained for 10 epochs with fine tuning, a batch size of 8, a dropout probability of 0.1, and a learning rate of 1e-5, using an NVIDIA T4.

Overall, I conducted experiments with seven different model configurations, in this order:

Baseline Only training on SST dataset, using the baseline architecture.
Multi-Dataset Training Incorporating training data for paraphrase detection and STS (i.e. Quora and SemEval sets).
Cosine Similarity Adding cosine similarity modification as described in 4.2.
Mean pooling Adding mean pooling as described in 4.3.
Multi-Task Learning Adding multi-task learning as described in 4.4.
Concatentation Mode Adding concatenation as described in 4.5.
More Epochs Running previous best model with 15 epochs instead of 10.

These experiments were done in cumulative fashion, such that, e.g. the Multi-Task Learning configuration also included all the modifications from the experiments before it, i.e. mean pooling, cosine similarity and multi-dataset training.

5.4 Results

Running the aforementioned experiments led to the following results:

Model	Overall dev score	SST dev acc	Paraphrase dev acc	STS dev corr
Baseline	0.494	**0.532**	0.385	0.13
Multi-Dataset Training	0.564	0.503	0.663	0.055
Cosine Similarity	0.704	0.518	**0.799**	0.592
Mean Pooling	0.713	0.52	0.784	0.668
Multi-Task Learning	0.73	0.52	0.738	0.86
Concatenation Mode	**0.75**	0.524	0.793	**0.863**
More Epochs	**0.75**	0.524	0.793	**0.863**

Table 1: Comparing results of different model configurations on the STS, Quora and SST dev sets

Model	Overall test score	SST test acc	Paraphrase test acc	STS test corr
Concatenation Mode	0.749	**0.532**	0.789	0.854

Table 2: Result of the best dev set model on the STS, Quora and SST test sets

The *Concatenation Mode* configuration—which importantly also includes the other modifications presented in section 4—achieves the highest overall dev score. This model proves to be very consistent when applying it to the test set, indicating good generalization from development to testing, and that my chosen architecture, including the dropout layers, were effective at preventing overfitting.

In general, layering on more modifications was able to consistently increase overall dev scores. Training on multiple data sources led to a huge uplift in paraphrase scores. Cosine similarity (and the associated scaling of the logits to fit with our data structure) was very effective at improving STS scores, confirming the results seen in (Reimers and Gurevych, 2019), namely that using cosine similarity is a good predictor for semantic textual similarity in the BERT model. Its introduction also positively affected paraphrase accuracy scores, which I did not expect. Mean pooling further increased STS scores as expected and as laid out in the SBERT paper. Multi-Task learning, while decreasing the scores for paraphrase detection, strongly improved STS scores, showcasing the importance of using a collective loss function when training for multiple tasks. The slight decrease in paraphrase detection scores was counteracted when making use of concatenations for paraphrase detection as described in 4.5. This led to the best model configuration amongst all experiments.

Training duration seems to have little impact on the chosen *Concatenation Mode* model. The identical scores when running the model for 15 epochs instead of 10 suggest that, for this configuration, extending training duration does not yield further improvements, indicating convergence to an optimal level.

Interestingly, the SST dev accuracy does not vary significantly across models, hovering around 0.52. This suggests that improvements in the other tasks do not necessarily translate to better sentiment analysis performance, possibly due to differences in task nature or data characteristics.

6 Analysis

For the three tasks, while the model is generally good at predicting the correct label (or, in the case of STS, a score that is close to the actual one) there are multiple shortcomings. I'm going to look at false positives and false negatives for each of the three tasks.

Semantic Analysis

False positive *It's everything you don't go to the movies for.*
Actual score: 0 | Predicted score: 3

The model has difficulties understanding the negation of "it's everything", most likely viewing this as a positive sentiment, and not connecting it to the negation through "don't".

False negative *If Steven Soderbergh's 'Solaris' is a failure it is a glorious failure.*
Actual score: 4 | Predicted score: 0

Similarly to the false positive, the model fails when a word or expression that is normally used for a particular sentiment, e.g. "failure" in this case, is used in the reverse by adding another word, e.g. "glorious" here.

Paraphrase Detection

False positive
Sentence 1: *How do you override a Honeywell thermostat?*
Sentence 2: *How can I unlock a Honeywell thermostat?*
Actual score: 0 | Predicted score: 1

The model will tend to predict something as a paraphrase if the large majority of the words in a sentence are the same. In cases where one word is different, but this word changes the whole meaning of the sentence, e.g. "override" vs. "unlock", the model will often incorrectly classify the sentences as paraphrase.

False negative
Sentence 1: *Which is best digital marketing course?*
Sentence 2: *I am MBA (Marketing) Student. I want to pursue Digital marketing Course. So where Can I find best course of Digital Marketing? Is there any Institute in Mumbai who are providing the Digital marketing Course?*
Actual score: 1 | Predicted score: 0

The model fails to detect something as a paraphrase if it involves significant rewording and the sentences are of very different length, with one of the two including context that is not relevant to the general meaning of the question, e.g. "I am MBA student".

Semantic Textual Similarity

False positive
Sentence 1: *To reach John A. Dvorak, who covers Kansas, call (816) 234-7743 or send e-mail to jdvorak@kctar.com.*
Sentence 2: *To reach Brad Cooper, Johnson County municipal reporter, call (816) 234-7724 or send e-mail to bcooper@kcstar.com.*
Actual score: 1 | Predicted score: 3.61

Similar to paraphrase detection, the model will tend to predict something as a paraphrase if the large majority of the words in a sentence are the same. For cases like the above, where the sentence structure is similar, but the actual datapoints are not the same (e.g. talking about two totally different people), the model will predict high similarity even though there is very little.

False negative
Sentence 1: *Carney sets high bar to change at BoE*
Sentence 2: *Carney sets high bar to changes at Bank of England*
Actual score: 5 | Predicted score: 2.99

The model fails to recognize abbreviations/acronyms and full names (e.g. "BoE" and "Bank of England") as referring to the same entity. This shortcoming is even more pertinent in short sentences, where the abbreviation makes up a relatively high part of the sentence.

7 Conclusion

This paper explored multiple extensions of the minBERT model in order to fit it for three downstream tasks, being sentiment analysis, paraphrase detection and semantic textual similarity (STS). In particular, I ran different model configurations implementing (1) the training on multiple datasets, (2) cosine similarity fine tuning, (3) a mean pooling strategy for embeddings, (4) multi-task learning during training and (5) a concatenation mode for paraphrase detection. Every extension helped to further improve on the overall baseline scores, with cosine similarity, multi-dataset training and the concatenation mode having the strongest effect, in particular on paraphrase detection and STS. My final model manages to get an overall score of around 0.75 on the test set, and has significantly improved scores on all three tasks.

Given our quantitative analysis, avenues for future work could include weighing certain parts-of-speech more strongly for paraphrase detection and similarity scoring (like in our example, transitive verbs, for instance, can often completely change the meaning of a sentence, more so than auxiliary verbs, so we might want to weigh the difference in transitive verbs more than a differences in auxiliary verbs). Further, pretraining the model on an abbreviation dictionary could help to create embeddings through which, for example, "Bank of England" and "BoE" are treated as the same word.

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.

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*.

Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

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.

Quora. 2017. First Quora Dataset Release: Question Pairs. https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs. [Online; accessed 15-March-2024].

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)*, page 3982–3992, Online. 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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. Xlnet: Generalized autoregressive pretraining for language understanding.