Implementing BERT for multiple downstream tasks

Stanford CS224N Default Project

Hamad Musa
Department of Computer Science
Stanford University
`hamadm77@stanford.edu`

Abstract

Within NLP, the need for versatile models able to do multiple and diverse tasks simultaneously is increasing at a rapid pace. When one model is able to be adapted to multiple NLP tasks, both memory and time are saved, while simplicity of use for users goes up. For the project, the main goal is to implement a minBERT model which was motivated by the original BERT model. The goal is for the model to be adapted to multiple and diverse NLP tasks. Specifically, the main tasks which the model is required to be proficient in are sentiment analysis, paraphrase detection, and sentence similarity. The main datasets used to train and test the model were the Stanford Treebank Dataset, the Quora Dataset and the SemEval dataset provided to us by Stanford University. For each task, I took a slightly different approach. For paraphrase detection, I found that using two Linear Layers followed by an activation layer was more effecient than using cosine similarity, while for sentence similarity, I experimented with both Euclidean Distance and Cosine Similarity. And here I found that Cosine Similarity yields far better results. In order to improve the embeddings for sentence similarity, I also implemented a contrastive learning approach, which also yielded better results.

1 Key Information to include

- Mentor: Anridhu Sriram

2 Introduction

Within the field on NLP, there have been many remarkable breakthroughs within the last decade, and the field is still expanding at a rapid pace especially with the developent of AI tools such as ChatGPT and so on. One such breakthrough was the BERT model, which stands for Bidirectional Encoder Representations from Transformers. One of the main breakthroughs related to the BERT model was the fact that it was Bidirectional, meaning it processed data in a non-unidirectional way as opposed to previous language models which mostly processed data in a linear fashion.

When it comes to the project, although I will not be implementing a full BERT model, I will be developing what's referred to as a minBERT model, which, as can be deduced from the name, is simply a miniature version of the BERT model. However, the main goal of the project will be to adapt the BERT model for additional downstream tasks. As mentioned in the abstract, these tasks are Sentiment Analysis, Paraphrase Detection and Sentence Similarity. I will be using and manipulating the embeddings from the BERT model to optimize each of these tasks.

The main datasets used to train the BERT model were the SSt dataset for sentiment analysis, the Quora dataset for paraphrase detection, and the SemEval dataset for sentence similarity. I experimented with multiple approaches for the project such as varying the number of linear layers used, using contrastive learning, and cosine similarity. This will be detailed in the approach section below.

3 Related Work

Since the BERT model was a big breakthrough within there is a lot of research surrounding it and its extensions. And I used a lot that research for developing this project. The first research paper I used was "BERT: Pretraining of Bidirection Encoders for Language Understanding" Devlin et al. (2018).It helped me understand the architecture of the BERT model itself, along with the project handout. Most of the implementation of BERT, and even the minBERT project I developed can be traced back to this paper and it's techniques.

Another research paper which was employed in the development of the project is a contrastive learning research paper reference here Gao et al. (2021). Since I had to apply contrastive learning within my project, I would say that this research paper was the one which was of most use to the project when it came to extension part for the three downstream tasks. The research describes the SimCSE approach or Simple COntrastive Learning for Sentence Embeddings. When implementing contrastive learning, a contrastive loss function is used, and the research from this paper was needed when implementing the contrastive loss function for the project.

In addition, I also used the following research paper to look into the implementation of Siamese Networks followed by cosine similarity for my own similarity prediciton function in the project. Reimers and Gurevych (2019)

Finally, I would say in order to guage the importance of the number of layers I was looking into this piece of research by Anthony Sarah and Sharth Nittur Shridar: Sridhar and Sarah (2021). Their work deals with the importance of the intermediate layers for the BERT model implementation. Although not as related to my question, some of the implementation of the project architecture was taken as inspiration from it.

4 Approach

The main goal of the project is to refine the minBERT model for multiple downstream tasks. When it comes to the implementation of the minBERT model itself I didn't change or implement new techniques.

However, the BERT model does serve as the base upon which I built multiple classification heads, one for each of the three downstream tasks. The BERT model itself is responsible for generating the embeddings for the sentences provided. These embeddings will then be further refined by me, and then used for the downstream tasks. Before speaking about the details of how the embeddings were refined and the classification heads for each task, it's important to elaborate on the structure of the BERT model itself.

The BERT model implemented starts by tokenizing the input sentences. These tokens are then passed through an embedding layer. Then at the center of the model are 12 transformer layers. The transformer layers make use of multi-head self-attention which allows the layers to focus on different aspects of the sentences. Then a feed-forward network is then used on top of the output from multi-head self attention to further refine the results. The model then produces embeddings for each token within the sentence.For each sentence, I decided to use the embedding of the CLS token at the start of each sentence.

Now when it comes to each of the downstream tasks, the output embeddings from BERT were manipulated in different ways. For sentiment analysis, I passed the embeddings through a dropout layer, followed by a linear layer which reduced the size of the embeddings to 70. Then the embeddings are passed through another linear layer which reduced the embedding vector to the final output vector of size 5. Each entry in the vector represents a sentiment class which is the final output returned. I used cross entropy loss as the loss function for sentiment analysis. The softmax activation function is built into cross entropy loss, so there was no need to apply it at the end of the linear layer.

However, when it comes to paraphrase detection, again I used a linear layer, but unlike semantic analysis, I used two linear layers instead of one because they yielded better results. The embeddings passed through a dropout layer, a linear layer, a relu activation function, and then again another dropout layer, and a final linear layer. This yielded the best results. For the loss function, since paraphrase deteciton is a binary operation, I used binary cross entropy loss instead.

For sentence similarity, I settled on using a symmetrical network of two linear layers operating side by side. Both outputs from each linear layer were then passed through the cosine similarity function. To be more specific, each embedding of each sentence was individually passed through a dropout layer followed by a linear layer. This reduced the size of each embedding to a vector of size 5. Then the two vectors were compared with cosine similarity. After that a sigmoid function was applied to cosine similarity, and the result of the sigmoid function was multiplied by 5 to get the returned logit into the correct range for sentence similarity (1 - 5).

As for the embeddings from BERT themselves, I refined them using a contrastive learning approach. I would take the embeddings for sentence pairs from BERT and calculate a contrastive loss function which I would add to the total training loss. This way the model would bring similar embeddings closer together in the embedding space to aid with sentence similarity.

5 Experiments

This section contains the following.

5.1 Data

The datasets I'm using are the Stanford Treebank Dataset for the sentiment analysis task, the Quora dataset for paraphrase detection, and the SemEval dataset for Semantic Textual Similarity. The Stanford Treebank dataset consists of single sentences, while the latter two datasets are composed of pairs. That is because the nature of paraphrase detection and sentence similarity entail that two sentence are compared together and so their datasets are composed of pairs. No futher refining was done on the datasets.

5.2 Evaluation method

One of the main evaluation methods is the training loss. But the training loss itself is composed of multiple components. One component is the cross entropy loss from sentiment analysis. Another component is the binary cross entropy loss for paraphrase detection. And the third component is the MSE loss for semantic textual similarity. The last component for the training loss is the contrastive loss for contrastive learning which is used to refine the embeddings. The contrastive loss was calculated and added between each pair of sentences in the SemEval dataset along with the MSE loss.

The other metrics are used are accuracy scores for each of the sentiment analysis and paraphrase detection. For semantic textual similarity, a pearson correlation coeffecient was calculated. An average accuracy score is calculated between all three tasks by averaging the latter three values out. This mean accuracy value, along with the training loss, is the most important value which was used to detect the most efficient model and save it.

5.3 Experimental details

The main things which were between the program trials were the number of layers for sentiment analysis and paraphrase detection. In addition, I experimented between using euclidean distance and cosine similarity for semantic textual similarity. Last but not least, I experimented with using contrastive learning.

Due to limitations in the number of gpu units I had available I couldn't afford to test how varying hyperparameters such as the learning rate, or the dropout probability or so on and so forth, would affect the results. I had to conserve the gpu units for my main tests, which involved contrastive loss, cosine similarity, and varying the number of layers. Therefore, the values of the hyperparameters I'm about to outline were the consistent across all trials.

The BERT model ran for 10 epochs in the the training loops. The learning rate was approximately . For all the dropout layers, the dropout probability was. And the hidden size of the embeddings was 768.

For the first trial of the experiments, I used a dropout layer followed by a linear layer for semantic analysis. The softmax activation function was already built into the cross entropy loss function used. For paraphrase detection, I concatenated the two embeddings, and then proceeded to pass the concatenated matrix into a dropout layer, followed by a linear layer, followed by a ReLU activation function. For sentence similarity, I simply calculated the cosine similarity between the two embeddings. I used the binary cross entropy loss function for paraphrase detection, and the MSE loss function for sentence similarity..

For the second trial, what changed was I used two linear layers instead of one for paraphrase detection. The two embeddings of each sentence were concatenated. And then this concatenated embedding was passed through a dropout layer and a linear layer that reduced the dimensionality of the embedding to a vector of size 512. This vector was then passed through another dropout layer and another linear layer that reduced it's dimensionality to 1, ie a single logit, which was the final output inputted into the loss function. The same process was applied for semantic textual analysis as well - used two linear layers and two dropout layers instead of one dropout followed by a linear layer.

For the third trial, the only changes from the second trial was for sentence similarity. Before implementing cosine similarity, I passed both CLS token embeddings for both sentences into a linear layer. The linear layer reduced the embeddings to vectors of size 5. Then I compared the two vectors with cosine similarity. Since the cosine similarity values are between -1 and 1, I applied the sigmoid function and multiplied by 5 to get the logits into the appropriate range. I also removed the second linear layer from the second trial for semantic textual analysis, and reverted back to using one Linear Layer instead.

For the fourth trial, everything was implemented exactly the same as the third trial, however instead of using cosine similarity followed by a sigmoid function for my activation layer, I used a euclidean distance measure instead. To be specific, I actually used the value:

$$\frac{1}{1+EuclideanDistance(LinearLayer1(Embedding1)+LinearLayer2(Embedding2))}$$

The Linear Layers are really a dropout layer, followed by a linear layer.

For the fifth trial, I implemented contrastive learning. I implemented a contrastive loss function, and added it's loss to the total training loss across each training loop. The contrastive loss function was implemented on every pair of embeddings in the semeval dataset in an unsupervised manner. For each sentence in the pair, I applied a dropout layer to its embedding, then calulated its loss with respect to the original embedding. The same process was repeated for the second sentence pair. And those losses were added to the total training loss.

5.4 Results

Just so it's clear, below are short descriptions of what each trial entailed in a table:

Trial 1	**Sentiment Analysis**	**Paraphrase Detection**	**Semantic Textual Similarity**
1	One Dropout/Linear Layer	One Dropout/Linear Layer	Cosine Similarity
2	Two Dropout/Linear Layers 2	Two Dropout/Linear Layers	Cosine Similarity
3	One Dropout/Linear Layers 2	Two Dropout/Linear Layers	Two Linear layers + Cosine Similarity
4	One Dropout/Linear Layers 2	Two Dropout/Linear Layers	Two Linear layers + Euclidean Distance
5	One Dropout/Linear Layers 2	Two Dropout/Linear Layers	Two Linear layers + Cosine Similarity + Contrast L

Here are the results across all trials. The first trial is the baseline. And the exact methodology for each trial is described above. Across all tasks I used a multi-task classifier, as in one BERT model was used for every task.

Trial	SST Accuracy	Paraphrase Accuracy	STS Corr
Baseline	0.291	0.538	0.019
Trial 2	0.293	0.629	0.019
Trial 3	0.290	0.640	0.207
Trial 4	0.283	0.601	0.095
Trial 5	0.210	0.607	0.215

6 Analysis

6.1 Analysis of Varying Number of Dropout and Linear Layers with both Paraphrase Detection and Semantic Textual Analysis

With both semantic textual analysis, and paraphrase detection, the only changes that were applied from base line were doubling the number of linear layers in trial 2. When it comes to paraphrase detection, the accuracy results shot up significantly due to this change. Although the training time also went up which makes perfect sense since there were more parameters to figure out. However, for semantic textual analysis doubling the number of linear layers did not lead to much improvement in the accuracy scores. My current reasoning for this is that paraphrase detection is a binary task, wherease with semantic textual analysis the model needs to understand the meaning of the words and the sentences themselves. The latter is a much more complex task. Simply increasing the number of layers might still not be enough to capture this complexity. I think I would have done better to focus on refining the embeddings instead of the linear layers for semantic textual analysis as the embeddings are where the semantic meanings of words in relation to each other are stored.

6.2 Analysis of Sentence Similarity Changes

Sentence Similarity is by far where most of the changes to the code were taking place, since the results were worse from the start compared to the other two tasks. The sentence similarity results should be addressed first in my opinion since they were by far the worst of the three. When simply using cosine similarity directly on the embeddings without any use of linear layers beforehand, the correlation coeffectient value for sentence similarity was incredibly low at only 0.019. I'm still not entirely sure why this was the case. Perhaps the embeddings needed for this task conflicted with the embeddings needed for the semantic analysis task. And since only one BERT model across all three tasks, this lead to poor performance. In addition, cosine similarity as a metric might not capture the similarity of two sentences adequately, especially since only the CLS token was used for the embeddings.

When the embeddings were passed through their own dropout layer, followed by a linear layer, and then cosine similarity was applied to them, the sentence similarity results, although still objectively low, improved significantly. I hypothesize this is due to the fact that cosine similarity targets the embeddings only bringing embeddings with higher cosine values closer together. But the actual predict similarity method itself does not do any learning of parameters. With the other methods, the parameters for the linear layers need to be adjusted, but with predict similarity, there are no parameters needing to be adjusted because cosine similarity operates the same way regardless of the embeddings.

I attempted to use a Euclidean Distance measure for the fourth trial to penalize embeddings which are far apart in distance. However, it didn't work nearly as well as cosine similarity. I believe the reason is because direction is a better encoder of sentence meaning than the size of the embedding vector itself. With Euclidean Distance two embeddings might have the exact same direction, but simply differing sizes, and this will increase the training loss.

The final change attempted on Sentence Similarity was contrastive learning. Although it did not lead to significant improvement, there was still noticeable change in the sentence similarity scores, which makes perfect sense since contrastive learning brings positive pairs closer together while pushing negative pairs further apart, further aiding with sentence similarity. However, I did notice that performance with semantic textual analysis dropped during this part, which reduced the overall accuracy. I hypothesize the reason is due to contrastive learning promotes embeddings which help with sentence similarity, but not so much with the sentiment or meaning of the sentences themselves.

7 Conclusion

The main finding of the project would be that simply increasing the number of linear layers does not necessarily lead to better results, especially when the task is complex such as with sentiment analysis, it might be more beneficial to focus on optimizing embeddings instead.

Specifically, if I had to redo the project with more gpu units, I would have definitely decided to put more time into techniques that refine the embeddings such as pretraining and so on and so forth.

Simply by improving the embeddings a lot of the preprocessing for the model would have taken place before the model even uses any of the classification heads for any of the three tasks.

In the future, I was hoping to look into the gradient surgery technique proposed in Yu et al. (2020). Multi-task learning was used across all of the tasks, where one model learns everything at the same time. Although this is fine, I was curious as to how the model would have improved if it learned each task independently. And even multi-task learning is used instead, how much would gradient surgery have improved the results.

References

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*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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.

Sharath Nittur Sridhar and Anthony Sarah. 2021. UNDIVIDED ATTENTION: ARE INTERMEDIATE LAYERS NECESSARY FOR BERT? arXiv preprint arXiv:2012.11881.

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