BERT for Sentiment Analysis, Paraphrase Detection and Semantic Textual Similarity with Cosine Similarity

Stanford CS224N Default Project

Debolina Paul
Department of Statistics
Stanford University
deblinap@stanford.edu

Abstract

BERT, which stands for Bidirectional Encoder Representations from Transformers (Devlin et al., 2018), is a language model based on the transformer architecture that produces contextual word representations. For the initial phase of this project, sentiment analysis was conducted on two datasets - Stanford Sentiment Treebank (SST) and CFIMDB. In the extended phase of the project, a multitask implementation of minBERT was trained to perform sentiment analysis, paraphrase detection, and semantic textual similarity tasks simultaneously on the SST, Quora and SemEval datasets. In single tasks as well as multitask, BERT is experimentally shown to perform effectively on all datasets.

1 Key Information to include

- Mentor: N/A
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018) is a model that uses the transformer architecture to create contextual word representations. It is a significant advancement in contextual word embeddings, large language models, and foundational models since it incorporates bidirectional word representations. The transformer architecture enables BERT to understand the relationships between words and capture long-range dependencies, making it a powerful language model.

The first challenge in the project was to do sentiment analysis using BERT. “Sentiment Analysis" is the task of understanding the polarity of a given text, i.e. whether the text is positive, negative or neutral. In this project we are doing sentiment analysis using BERT on two movie review datasets, namely, the *Stanford Sentiment Treebank*[1] (SST) dataset and the CFIMDB[2] dataset. To do sentiment analysis, a pretrained original BERT model, both with and without finetuning, has been used. The mean accuracies for the `dev` split are reported for both of these datasets after training on the `train` splits.

For the extension I implemented the multitasking via BERT suggested in the handout[3]. BERT performs multitasking by jointly training the model on multiple tasks simultaneously. BERT has a shared encoder that can be fine-tuned for different downstream tasks, such as sentiment analysis,

[1] `https://nlp.stanford.edu/sentiment/treebank.html`

[2] `https://ai.stanford.edu/~amaas/data/sentiment/`

[3] `http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf`

paraphrase detection, and semantic textual similarity (STS). During multitask training, BERT is optimized to minimize a weighted sum of the loss functions of all the tasks. The weights can be fixed or learned dynamically based on the performance of each task. This way, BERT can learn to share and transfer knowledge across tasks, which can improve its overall performance and reduce the need for task-specific architectures. Moreover, BERT can benefit from pretraining on large amounts of data using an unsupervised objective, such as masked language modeling (MLM) or next sentence prediction (NSP). This pretraining allows BERT to learn general language representations that can be adapted to specific tasks with fine-tuning. The multitask fine-tuning of BERT can further enhance its ability to capture complex linguistic phenomena and improve the robustness of its embeddings.

The main objective of the extension of the project is to investigate methods for constructing strong embeddings that can demonstrate high performance across a diverse set of tasks, rather than being limited to just one. The multitasking mainly consists of three tasks implemented by the BERT -

- Sentiment Analysis: Predicting sentiment scores of sentences.
- Paraphrase Detection: Predicting whether a sentence pair are paraphrases of each other.
- Semantic Textual Similarity: Predicting the similarity of two input texts.

The *Stanford Sentiment Treebank* (SST), *Quora*[4] and SemEval Agirre et al. (2013) Benchmark datasets have been used for evaluating the multitask results for sentiment analysis, paraphrase detection and semantic textual similarity (STS) respectively.

3 Related Works

The prevailing models for sequence transduction rely on sophisticated recurrent or convolutional neural networks that comprise both an encoder and a decoder. The most successful models incorporate an attention mechanism that links the encoder and decoder. Vaswani et al. (2017) introduced a network structure called the Transformer. This architecture is founded entirely on attention mechanisms and does not require the use of recurrent or convolutional layers, which proved to achieve significant improvement in language modelling tasks.

Based on this attention mechanism, Devlin et al. (2018) proposed the BERT architecture, which has been used successfully in several language processing tasks such as sentiment analysis, text classification etc. Several language modelling techniques have been proposed in the literature such as Howard and Ruder (2018); Ziegler et al. (2019) and so on.

4 Approach

In this project, we approached two tasks using BERT for the given datasets. The first is the Sentiment Analysis for the given movie review datasets where we do classification on BERT's output to positive, negative and neutral sentiments. The second approach is extension for additional datasets for multitasking on sentiment analysis, paraphrase detection and semantic textual similarity.

4.1 BERT for Sentiment Analysis

BERT can perform sentiment analysis by fine-tuning its pre-trained language model on a sentiment analysis task. This involves adding a classification layer on top of BERT's output, which predicts the sentiment label (e.g., positive, negative, or neutral) of a given input text. The classification layer consists of a fully connected layer followed by a softmax function, which computes the probability distribution over the sentiment labels.

Tokenization of input is a critical step that converts text into a format that can be processed by the model. The first step is to split the raw text into words using a word tokenizer. In BERT, the word tokenizer used is WordPiece, which splits words into sub-words based on their frequency in a large corpus of text. BERT requires the addition of special tokens at the beginning and end of each sequence of tokens. The [CLS] token is added at the beginning of the sequence, and the [SEP] token is added at the end of each sentence or document. Unknown words are assigned a special [UNK]

[4] `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

token. The next step is to convert each word or sub-word into an integer index using a vocabulary file. The vocabulary file contains a mapping of each word or sub-word to a unique integer index. Since all sentences are required to be of the same length, long sentences are truncated and short sentences are padded with a [PAD] token. Positional embeddings, that are learned for each of the 512 positions in a given BERT input are utilized to encode the position of different words within the input. The input embeddings that are utilized later in the model are the sum of the token embeddings, the segmentation embeddings, and the position embeddings.

Multihead Self Attention The 12 Transformer architecture for the base BERT makes it possible to parallelize the ML training efficiently. Transformers in BERT consist of two linear-transformations and a ReLU activation layer, but the most important part it uses is the multi-headed self-attention mechanism, which is responsible for sentiment analysis. It is a scaled dot-product attention which is applied across multiple heads. Each head takes as input queries and keys of dimension d_k, and values of dimension d_v. The dot product of the query with all keys is computed, and each result is divided by $\sqrt{d_k}$. A softmax function is applied to obtain weights on the values, and the final output is obtained by multiplying the values with the weights. BERT performs this attention function on a matrix of queries, keys, and values, and computes the multi-head attention by concatenating the output of each attention head and multiplying it with a parameter matrix W^O.

The multi-head attention is computed as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ are parameter matrices.

The Multihead attention layer of BERT helps in sentiment analysis by allowing the model to attend to different parts of the input sequence simultaneously. This enables the model to capture the most important parts of the input that are relevant for predicting the sentiment of the text.

Masked Language Model (MLM) enables bidirectional learning from a given text by masking a word in a sentence and forcing BERT to bidirectionally use words on both sides to predict the masked word. For BERT, a random 15% of the tokenized words are hidden during training and BERT's job is to correctly predict the masked words. In 80% of these cases,

4.2 BERT for Multitasking

BERT (Bidirectional Encoder Representations from Transformers) performs multitasking by jointly training on multiple tasks using a shared set of parameters. This is achieved using a multi-task learning (MTL) approach Liu et al. (2019), where a single model is trained to perform several tasks simultaneously.

Mathematically, the MTL approach can be formulated as follows:

Given a set of N tasks, $T_1, T_2, \ldots, T_N$, and a training set for each task, $D_1, D_2, \ldots, D_N$, the objective is to minimize the following loss function:

$L_{\text{total}} = \sum_{i=1}^{N} \lambda_i L_i$

where λ_i is a weighting coefficient that determines the importance of each task, and L_i is the task-specific loss function for the ith task.

The task-specific loss function L_i is typically defined as the cross-entropy loss between the predicted labels and the ground truth labels for the task. For example, in sentiment analysis, L_i would be the cross-entropy loss between the predicted sentiment labels and the true sentiment labels.

During training, BERT is optimized to minimize the total loss, L_{total}, by updating the shared set of parameters that are used for all tasks. This allows BERT to learn task-specific features that are useful for multiple tasks and improves its ability to perform well on different tasks simultaneously.

In summary, BERT performs multitasking by jointly training on multiple tasks using a shared set of parameters, which are optimized to minimize a total loss function that combines the losses of all tasks with different weighting coefficients.

5 Experiments

The experiments are conducted as follows.

5.1 Data Description

The first dataset used in this project is the *Stanford Sentiment Treebank*[5] (SST) dataset. The dataset is comprised of $11,855$ individual sentences that were extracted from movie reviews. To prepare the dataset for analysis, the Stanford parser was used to parse the text, resulting in a total of $215,154$ unique phrases extracted from the parse trees. Each phrase in the dataset was analyzed by three human judges and assigned a label among the following: *negative, somewhat negative, neutral, somewhat positive,* or *positive*. The dataset has been split into three subsets for training, development, and testing namely, `train` ($8,544$ datapoints), `dev` ($1,101$ datapoints) and `test` ($2,210$ datapoints) respectively. It has been used for sentiment analysis as a single task and under multitasking.

The second dataset used for this project is also a movie review dataset, called the CFIMDB[6] dataset. The CFIMDB dataset contains $2,434$ movie reviews that are highly polarized and have been labeled as either *negative* or *positive*. It is worth noting that some of these reviews consists of multiple sentences. The dataset has been divided into three subsets for training, development, and testing namely, `train` ($1,701$ datapoints), `dev` (245 datapoints) and `test` (488 datapoints) respectively.

The Quora[7] dataset contains $400,000$ pairs of questions, labeled to indicate whether they are paraphrases of each other. A split of the dataset is provided, which includes $141,506$ training examples, $20,215$ development examples, and $40,431$ testing examples. The dataset has been used for evaluating paraphrase detection in the multitask extension.

The SemEval STS Benchmark dataset Agirre et al. (2013) contains $8,628$ pairs of sentences with varying degrees of similarity, ranging from 0 (unrelated) to 5 (equivalent meaning). The dataset is commonly used for evaluating algorithms for semantic textual similarity. The dataset is split into $6,041$ training examples, 864 development examples, and $1,726$ testing examples. This dataset has been used for paraphrase detection under multitasking via BERT.

5.2 Evaluation method

We have evaluated the performance of BERT sentiment analysis based on accuracy. The BERT is first trained on the training datasets and then we try to estimate the labels of the dev and test datasets. To evaluate the efficacy of the classification, we use the metric *accuracy*, which is defined as the fraction of the correctly classified datapoints.

5.3 Experimental details

We have run the baseline for the report, i.e. sentiment analysis for SST and CFIMDB datasets using the starter code provided for the project. For pretrain, we used `epochs` to be 20, `hidden_dropout_prob` to be 0.3 and the rest of the parameters as default. For finetune, we used 20 epochs keeping the rest of the parameters as default. For multitask, all the default parameters have been used with number of epochs being 10.

For the extension to additional downstream task, we extended it for multitask classifier. For training, I used SST, Quora and STS datasets together. Since the SST dataset is a multiclass dataset with 5 classes, we use the cross entropy loss for training. For the Quora dataset, the labels are binary indicating whether or not two sentences are paraphrases of each other and hence I used the binary cross entropy loss. For the SemEval dataset, the labels indicate varying similarity between sentence pairs on a scale from 0 (unrelated) to 5 (equivalent meaning) and hence for this, I used the Mean Squared Error (MSE) loss. Finally, all the three different loss are added together and minimized for training. For optimization, I have used Adam Optimizer based on Decoupled Weight Decay Regularization Loshchilov and Hutter (2017) and Adam: A Method for Stochastic Optimization Kingma and Ba (2014).

[5] `https://nlp.stanford.edu/sentiment/treebank.html`

[6] `https://ai.stanford.edu/~amaas/data/sentiment/`

[7] `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

5.4 Results

The results obtained for the Sentiment analysis on SST and CFIMDB datasets for both pretrain and finetune have been summarized in Table 1. The results show that BERT is accurate in classification when finetuning have been used. This is because finetuning uses the labeled data for training as opposed to the pretraining.

Dataset	Method	Train Accuracy	Dev Accuracy
SST	Pretrain	0.337	0.316
SST	Finetune	0.984	0.517
CFIMDB	Pretrain	0.600	0.653
CFIMDB	Finetune	1.000	0.967

Table 1: Accuracy obtained for training and deviation datasets for Sentiment Analysis

The results obtained for multitask is summarized in Table 2 for different tasks. Comparing the result for sentiment analysis with pretraining, we can see that BERT is almost as efficient in multitasking as in handling a single task. To increase the efficacy of BERT, we can maybe further pretrain the model or use finetuning.

Task	Dataset	Train Accuracy	Dev Accuracy
Sentiment analysis	SST	0.310	0.307
Paraphrase Detection	Quora	0.631	0.625
Textual Similarity	STS	0.235	0.233

Table 2: Training and Dev Set Accuracy for Multitask via BERT evaluated on different datasets

6 Analysis

The obtained results for Sentiment Analysis show that the accuracy obtained for the SST dataset for both pretrain and finetune is much lower than that obtained for CFIMDB dataset. This may be due to the fact that analysis is much easier when there are just 3 classes compared to 5 because trying to understand whether a review is strongly positive or moderately positive is a lot harder than trying to understand if the review is positive vs neutral. Also, for both datasets finetune works much better than pretrain because we are finetuning pre-trained representations on a specific downstream task, such as sentiment analysis, or text classification. This allows the model to learn task-specific features and improve on the performance.

Further improvements can be made to increase the efficacy of BERT in multitasking. One such method is to pretrain the model even more, which involves training the model on a large amount of data before fine-tuning it on a specific task. Another method is to use finetuning, where the model is trained on a specific task and then fine-tuned on related tasks to improve its overall performance. Both methods have been shown to improve the performance of deep learning models like BERT, and could potentially enhance its multitasking capabilities even further.

7 Conclusion

In conclusion, from the experiments, we can see that BERT is very efficient in a lot of Natural Language Processing tasks. The results obtained for Sentiment Analysis indicate that finetuning performs better than pretraining on datasets. This is because finetuning involves training the model's pre-trained representations on a specific task such as text classification or sentiment analysis, allowing the model to learn task-specific features and improve its overall performance.

To further improve the multitasking capabilities of BERT, there are several methods that could be used in future. One such method is to increase the amount of pretraining, which involves training the model on a large dataset before fine-tuning it on a specific task. Another approach is to use finetuning, where the model is trained on a particular task and then fine-tuned on related tasks to improve its overall

performance. Both techniques have been demonstrated to enhance the performance of deep learning models such as BERT and could potentially improve its multitasking capabilities even further.

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.

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*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

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.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.