Enhancing minBert Embeddings for Multiple Downstream Tasks

Stanford CS224N Default Project

Donald Stephens
Department of Computer Science
Stanford University
Stanford, CA 94305
dsteph@stanford.edu

Abstract

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based model that generates contextual word embeddings. Starting with a minimal implementation of BERT, called minbert, I implemented Multi-head Self-Attention and the Transformer Layer, including a portion of the Adam stochastic optimization method. My goal was to make enhancements to obtain robust and generalizable embeddings that perform well in multiple downstream tasks: sentiment analysis, paraphrase detection and semantic textual similarity. The completed base implementation achieved a 38.5% accuracy, 37.5% accuracy and -0.074 correlation respectively (overall average of 0.229) on holdout datasets of the aforementioned tasks. After further pre-training on task specific data by training on a masked language model objective, fine-tuning using cosine embedding loss, applying a learning rate decay schedule, and hyperparameter tuning, my final model achieved a 52.6% accuracy, 59.3% accuracy and 0.418 correlation respectively (overall average of 0.512) on the same datasets, a 124% increase over the base implementation.

Keywords: bert pre-training, bert fine-tuning, masked language modeling, cosine embedding loss, learning rate schedule, sentiment analysis, paraphrase detection, semantic textual similarity

1 Introduction

My base model was a minimal implementation of Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018), called minbert. The model contains 12 Encoder Transformer Layers each consisting of multi-head attention, followed by an additive and normalization layer with a residual connection, a feed-forward layer, and a final additive and normalization layer with a residual connection. The transformer architecture follows Figure 1 and is based on work in "Attention Is All You Need" (Vaswani et al., 2017).

I completed implementations of the Multi-head Self-Attention and the Transformer Layer, including a portion of the Adam stochastic optimization method. I based my implementation of step function for Adam optimization on the efficient version of Algorithm 1 as outlined in "Adam: A Method for Stochastic Optimization" (Kingma and Ba, 2014).

1.1 Multi-head Self-Attention

Attention functions take query vectors and a set of key-value pair vectors and map them to some output denoting the compatibility of the query with the given keys. Multi-head attention can be thought of as a group of attention functions running in parallel, and they allow the model to jointly attend to information from different representations, at different positions. Single attention heads typically do not allow this joint behavior. We summarize the mathematical formulas for Self-Attention and Multi-head Attention as described in (Vaswani et al., 2017):

$$\text{Attention}\left(Q, K, V\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \quad (1)$$

$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V,\right) \quad (2)$$

$$\text{MHA}\left(Q, K, V\right) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O \quad (3)$$

1.2 Adam Optimizer

Adam is a method for stochastic optimization that "computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients"(Kingma and Ba, 2014). I used a variation of Adam, called AdamW, where the weight decay is performed only after controlling the parameter-wise step size - the regularization term is outside of the moving averages and proportional to the weight itself.

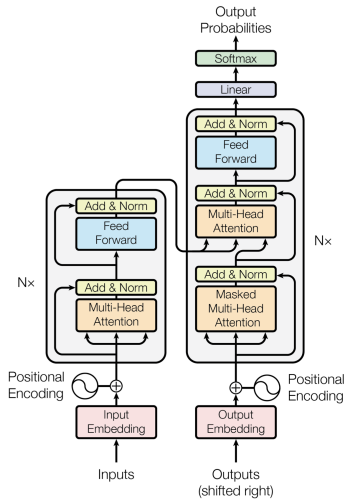

Figure 1: Transformer Architecture

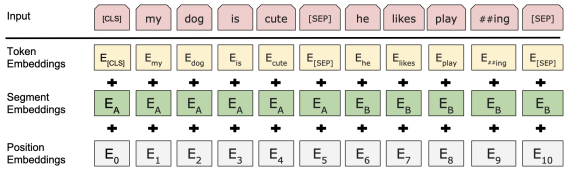

Figure 2: Bert Input Representation and resulting Embeddings

2 Downstream Tasks and Goal

The goal of the research was to explore and understand was to obtain robust and generalizable embeddings that perform well in multiple downstream tasks. The tasks were:

1. Sentiment Analysis: The task of classifying text to understanding which affective state (e.g., positive, negative, neutral) is expressed.

2. Paraphrase Detection: The task of finding paraphrases of texts in a large corpus of passages. At it's essence, this task seeks to determine whether particular words or phrases convey the same semantic meaning. The ultimate outcome would be binary (i.e., yes one is a paraphrase of other, or no they are not).

3. Semantic Textual Similarity: The task seeks to capture the notion that some texts are more similar than others by measuring the degree of semantic equivalence. To account for the degree of semantic equivalence there is a scale from 0 (denoting not related) to 5 (denoting they have the same meaning).

3 Prior Work

"BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers" (Devlin et al., 2018). The pre-trained BERT can be used for a variety of downstream tasks. Such tasks can be accomplished by adding a task (or multi-task) head on top of BERT, to create a task (or multi-task) output layer.

To better ensure success in the downstream tasks, one would either enhance the task layer or make the embeddings from BERT more robust. Such enhancements can be incorporated at either the pre-training stage (i.e., making the embeddings more robust) or the fine-tuning stage. Later in this document I discuss which enhancements I experimented with. In this section, I provide a cursory background on the motivations for experimenting with specific pre-training or fine-tuning approaches.

Researchers Sun, Qiu, Xu and Huang (Sun et al., 2020) provided an overview of various approaches to fine-tune BERT for text classification downstream tasks. The contributions of their work include:

1. The authors proposed a general solution to fine-tune a pre-trained bert:
    - Futher pre-train BERT on within-task training data or in-domain data (see Figure 3)
    - Fine-tune BERT with multi-task learning objectives and data
    - Fine-tune BERT for a specific task

2. The authors investigated pre-processing of text methods, layer selection, layer-wise learning considerations and low-shot learning problems

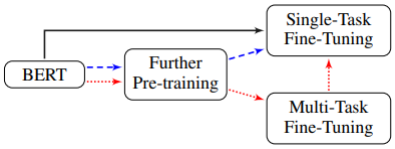

Figure 3: General Ways for fine-turning BERT

Their work introduced some very good ideas on practical considerations for improving the model on the given tasks. Unfortunately, given the short (<4 weeks) amount of time and compute resources,

I decided it was more prudent to be tactical and focus my efforts largely around pre-training on multi-task data, and some straight forward approaches to fine-tuning (impacting relatively more of the task layer for at least 2 of my downstream tasks).

Reading the work of Reimers and Gurevych in their development of Sentence-BERT (SBERT) (Reimers and Gurevych, 2019) they achieved success computing similarity scores between embeddings for each of two source sentence inputs. Though the authors incorporated a siamese network architecture, their work did show strong success with using a similarity measure like cosine similarity. As such, I chose to use a similar concept for the paraphrase and semantic text similarity downstream tasks of my project. Additionally, their work also motivated me to fine-tune using a cosine embedding loss function.

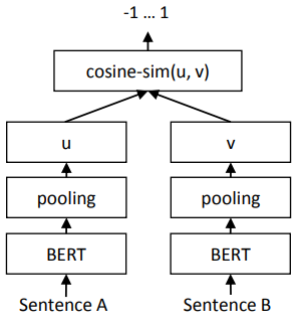

Figure 4: Computing Similarity Scores in SBERT during Inference

4 Data and Summary Statistics

4.1 Development and Evaluation Data

4.1.1 Data for Sentiment Analysis

I used an excerpt of the Stanford Sentiment Treebank (STS) dataset. The original dataset consists of 11,855 single sentences extracted from movie reviews. The dataset was parsed with the Stanford parser and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges. Each phrase has a label for the following:

1. Negative
2. Somewhat Negative
3. Neutral
4. Somewhat Positive
5. Positive

The labels previously described represent varying levels of positive, negative, and neutral affective states. Below I provide a distribution of the train and validation split, along with a distribution of the composition of labels.

Type	Count	Percent
Train	8,544	88.6%
Validation	1,101	11.4%
	9,645	

Label	Meaning	Train (n)	Train (%)	Val (n)	Val (%)
0	Negative	1,092	13%	139	13%
1	Somewhat Negative	2,218	26%	289	26%
2	Neutral	1,624	19%	229	21%
3	Somewhat Positive	2,322	27%	279	25%
4	Positive	1,288	15%	165	15%
		8,544		1,101	

4.1.2 Data for Paraphrase Detection

Below I provide a distribution of the train and validation split, along with a distribution of the composition of labels.

Type	Count	Percent
Train	141,498	87.8%
Validation	20,212	12.5%
	161,170	

Figure 5

Label	Meaning	Train (n)	Train (%)	Val (n)	Val (%)
0	No	89,225	63%	12,627	62%
1	Yes	52,273	37%	7,585	38%
		141,498		20,212	

4.1.3 Data for Semantic Textual Similarity

Below I provide a distribution of the train and validation split, along with a distribution of the composition of labels.

Type	Count	Percent
Train	6,040	87.5%
Validation	863	12.5%
	6,903	

Figure 6

Label	Train (n)	Train (%)	Val (n)	Val (%)
0 (Not Related)	1,043	17%	136	16%
1	974	16%	108	13%
2	1,041	17%	160	19%
3	1,586	26%	235	27%
4	1,108	18%	167	19%
5 (Same)	288	5%	57	7%
	6,040		863	

Figure 7

5 Model

5.1 Sentiment Analysis Classification

5.1.1 Inference

Using the final BERT embedding, which is the hidden state of the [CLS] token, I added an classification head composed of a linear layer and a dropout layer. The linear layer is based on 768 neurons, plus a bias neuron, and 5 output units. The dropout layer is used as regularization method to reduce overfitting. Using a dropout layer could result in a lower training performance metric value, but should allow better generalization to any holdout development sets.

$$h = 768$$
$$c = 5$$
$$\vec{y} = \vec{x}A^T + \vec{b}$$

$$p = 0.1$$
$$dropout = \frac{1}{1-p}$$

5.1.2 Pre-Training

To enhance the quality of the word embeddings, I implemented a masked language objective which I used to further pre-train BERT. I started with an existing pretrained bert model (called "bert-base-uncased"), and further pre-trained using the sentiment training data. This process masked 15% of the training data (excluding PADDED AND MASKED tokens), attempted to predict the masked data, then updated the weights and embeddings based on the cross entrophy loss function.

5.2 Paraphrase Detection

5.2.1 Inference

For each pair of sentences given from a dataset, the model creates one contextualized embedding for each sentence. For the interference layer, I calculate a cosine similarity score between the two sentence embeddings and apply a sigmoid transformation function on the cosine similarity.

$$\text{similarity} = \frac{x_1 \cdot x_2}{\max(\|x_1\|_2 \cdot \|x_2\|_2, \epsilon)}$$

5.2.2 Pre-Training

To enhance the quality of the word embeddings, I implemented a masked language objective which I used to further pre-train BERT. I started with an existing pretrained bert model (called "bert-base-uncased"), and further pre-trained using the paraphrase training data. This process masked 15% of the training data (excluding PADDED AND MASKED tokens), attempted to predict the masked data, then updated the weights and embeddings based on the cross entrophy loss function.

5.2.3 Fine-Tuning

5.3 Semantic Textual Similarity

5.3.1 Inference Head

For each pair of sentences given from a dataset, the model creates one contextualized embedding for each sentence. For the interference layer, I calculate a cosine similarity score between the two sentence embeddings and return that score as the logit from the model.

5.3.2 Pre-Training

To enhance the quality of the word embeddings, I implemented a masked language objective which I used to further pre-train BERT. I started with an existing pretrained bert model (called "bert-base-uncased"), and further pre-trained using the semantic similarity training data. This process masked 15% of the training data (excluding PADDED AND MASKED tokens), attempted to predict the masked data, then updated the weights and embeddings based on the cross entrophy loss function.

5.3.3 Fine-Tuning

To improve the quality of the prediction, I added the Cosine Embedding Loss as part of the fine-tuning state, the formula is:

$$\text{loss}(x, y) = \begin{cases} 1 - \cos(x_1, x_2), & \text{if } y = 1 \\ \max(0, \cos(x_1, x_2) - \text{margin}), & \text{if } y = -1 \end{cases}$$

6 Experiments

6.1 Performance Enhancement Strategy

To better ensure success in the downstream tasks, one would either enhance the task layer or make the embeddings from BERT more robust. Such enhancements can be incorporated at either the pre-training stage (i.e., making the embeddings more robust) or the fine-tuning stage. As a tactical strategy (given the short timeframe to work on the project) I focused primary on further pre-training of BERT.

6.2 Further Pre-Training - Masked Language Model

To enhance the quality of the word embeddings, I implemented a masked language objective which I used to further pre-train BERT. I started with an existing pretrained bert model (called "bert-base-uncased"), and further pre-trained using the the downstream multi-task training data.

This process masked 15% of the training data (excluding PADDED AND MASKED tokens), attempted to predict the masked data, then updated the weights and embeddings based on the cross entrophy loss function. This provided a significant boost to the quality of the embeddings. The masked language objective provided the most impact boost to all of the performance enhancement experiments.

6.3 Fine-Tuning - Cosine Embedding Loss

The cosine embedding loss measures the loss given input tensors x_1 and x_2 and a Tensor label y with values 1 or -1. This is used for measuring whether two inputs are similar or dissimilar, using the cosine similarity, and is typically used for learning nonlinear embeddings or semi-supervised learning.

6.4 Learning Rate Schedule

Instead of using a static learning rate, and though the AdamW optimizer does have some adjustment to the learning rate, I added an exponential learning rate decay scheduler to the optimizer.

$$\begin{aligned} \text{learning rate}_0 &= 0.001 \\ \text{gamma} &= 0.96 \\ \text{learning rate}_{\text{epoch}} &= \text{learning rate}_0 * \text{gamma}^{\text{epoch}} \end{aligned}$$

Figure 8 provides a visual illustration of the exponential decay schedule.

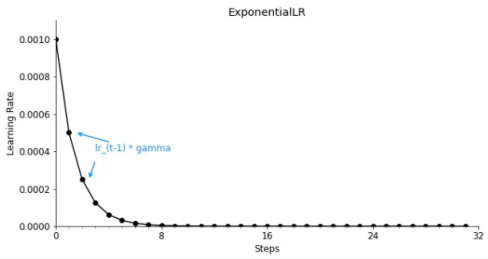

Figure 8: Top

6.5 Hyperparameter Tuning

I employed a standard grid search for hyperparameter tuning. But I did not change or review considerations for changes to the number of layers in the model architecture nor the number of hidden nodes.

$$\begin{aligned} \text{learning rate} &= 0.001 \text{ for pre-training} \\ \text{learning rate} &= 0.0002 \text{ for fine-training} \\ \text{learning rate} &= 0.0003 \\ &\quad \text{for pre-training using masked} \\ &\quad \text{language} \\ \text{gamma} &= 0.96 \\ &\quad \text{for the exponential learning} \\ &\quad \text{rate decay schedule} \\ \text{batch size} &= 64 \end{aligned}$$

7 Results & Analysis

The completed base implementation achieved a 38.5% accuracy, 37.5% accuracy and -0.074 correlation respectively (overall average of 0.229) on holdout datasets of the aforementioned tasks.

After further pre-training on task specific data by training on a masked language model objective, fine-tuning using cosine embedding loss, applying a learning rate decay schedule, and hyperparameter tuning, my final model achieved:

- 52.6% accuracy for the sentiment task
- 59.3% accuracy for the paraphrase detection task
- 0.418 correlation for the semantic text similarity task

The final model achieved an overall average score of 0.512, a 124% increase in performance over that of the overall average score the base implementation.

Below is a table designed to illustrate changes to performance throughout experimentation:

#	Model / Training Change	SST Dev Accuracy	Paraphrase Dev Accuracy	STS Dev Correlation	Overall Dev Score	Percentage Increase
1	Base Implementation	0.385	0.375	-0.074	0.229	
2	Adjusting learning rate	0.410	0.375	-0.074	0.237	3.64%
3	Mapping cosine similarity to predefined similarity classes	0.410	0.376	0.019	0.268	13.22%
4	Attempting to balance data across predictive classes	0.397	0.376	0.019	0.264	-1.61%
5	Masked Language Model	0.500	0.424	0.280	0.401	52.02%
6	Masked Language Model + Learning Rate Decay	0.500	0.599	0.424	0.508	26.50%

The details behind the 3rd row above is explained via the following mapping rules. I attempted to map the cosine score to varying levels of similarity classes:

$$\text{logit} = \begin{cases} 0, & \text{if similarity} < 0.16667 \\ 1, & \text{if similarity} < 0.33334 \\ 2, & \text{if similarity} < 0.50001 \\ 3, & \text{if similarity} < 0.66668 \\ 4, & \text{if similarity} < 0.83335 \\ 5, & \text{otherwise} \end{cases}$$

I attempted to map the cosine score to varying levels of yes / no in terms of paraphrases as well:

$$\text{logit} = \begin{cases} 0, & \text{if similarity} < 0.5 \\ 1, & \text{otherwise} \end{cases}$$

The completed base implementation achieved the results in Figure 9 in the BERT + Adam Dev Board.

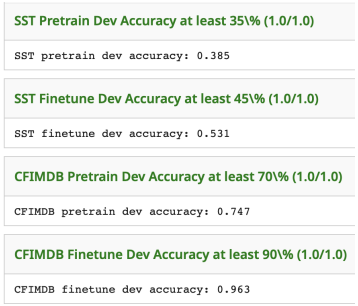

Figure 9: BERT + Adam Dev Board

Authorship Statement

I'd like to extend a big thanks to Dr. Christopher Manning, inaugural Thomas M. Siebel Professor in Machine Learning, Department of Linguistics and Computer Science, Stanford University; and the Teaching Assistant staff of CS224N Natural Language Processing with Deep Learning. Their suggestions and guidance were helpful and appreciated. Any opinions, findings, and conclusions expressed in this document are that of the authors and do not necessarily reflect the views of Stanford University.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to fine-tune bert for text classification?

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.