Fine-tuning BERT for the Medical Domain

Michael Benitah
260187637

Myriam Laiymani
260923313

Humam Seddik
260751294

Abstract

Ever since attention mechanisms were introduced into neural networks, transformers have presented a major breakthrough in text-comprehension tasks because those attention mechanisms allow the network to efficiently learn context. BERT (Bidirectional Encoder Representations from Transformers) pushed this even further by capturing context in both directions and using extensive unsupervised pre-trained weights. In this project, we experiment with a BERT model loaded with pre-trained weights for general domains and then fine-tune that model in an effort to improve text comprehension in a more specific (medical) domain.

1 Introduction

BERT (Bi-directional Encoder Representations from Transformers) models have achieved excellent results in text comprehension tasks such as sentiment analysis, named entity recognition, sentence completion, question answering or sentence completion. There are may flavours of BERT and they have all been pre-trained with huge amounts of unlabelled text and can be readily applied to many domains in general.

We asked ourselves, however, if these models would perform equally well in more specific domains. For example, if a physician uses BERT text completions to write an e-mail about the diagnosis of a patient, would it work just as well as if it were the e-mail between two friends meeting for lunch? And if not, could we improve it by fine-tuning the model to a specific domain?

In this project, we concentrate on one specific task: predicting randomly masked words within a sentence. Masked language model (MLM) is closely related to the way BERT is pre-trained. Although this task is not as useful in our everyday lives as sentence completion or question answering, it was chosen because its results could be easily measured using automatic evaluation metrics such as accuracy and score, without relying on human evaluation or more subjective metrics, as tasks like sentence completion would require.

Thus, we fine tune the model with text from a specific domain (i.e. Medical) and then compare the score of the masked text prediction obtained with the pre-trained and the fine-tuned models.

2 Related work

Attention mechanisms and transformers were first introduced by (Vaswani et al., 2017). They presented an encoder-decoder structure that dispensed with the use of recurrent networks, creating a simpler structure that could be parallelised. This idea proved to be useful in models that followed. Models like ELMo (Peters et al., 2018) presented a breakthrough over previous models because they offered contextualised word embeddings, contrary to models such as word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014) where the word *bank* has always the same embedding no matter if it is part of the expression *river bank* or *bank account*. Models like OpenAI GPT (Generative pre-training) (Radford et al., 2018) provide pre-trained weights from huge amounts of unsupervised data.

3 Method

BERT (Devlin et al., 2018) builds on those ideas by capturing context in both directions. ELMo would have no problem in creating an embedding for *bank* in the sentence *She was seen walking by the river bank,* but its embedding may be compromised in the sentence *She went to the bank to make a deposit,* as it would only capture context to the left of the word *bank* and would not take into account the relevant word *deposit*. It is easy then to

see the advantages that BERT introduces, pushing the state-of-the-art even further on many classic Natural Language Processing problems like question answering, GLUE multi-task learning, text classification and Named Entity Recognition. In fact, BERT has achieved state-of-the-art results in eleven different NLP tasks. (Devlin and Chang, 2018)

In more details, BERT uses the encoder part of the transformer architecture and stacks them. The decoder part of the transformer was initially intended for translation. For the purposes of embedding, only the encoder part is required.

One of the main improvements over OpenAI GPT was the introduction of bidirectionality: every token has self-attention with the ones appearing before and after in a sentence. The encoding of tokens is done using the whole context. Using bidirectionality in training a language model is not feasible with OpenAI GPT due the way the model is trained: simply predicting the next word. The genius of BERT was to change the definition of the problem itself, training on a masked language model and next sentence prediction, enabling the use of the whole context.

Pre-trained weights were provided with the Transformers package but they were obtained from unlabelled data over different pre-training tasks. For fine-tuning, the model is initialised with pre-trained weights and all of those weights are fine-tuned with labelled data from the downstream task. So, if the task is question-answering, question-answer pairs would be provided to the fine tuning process. If the task is next sentence prediction, then pair of sentences A and B would be provided. In our case, we simply provided text data, which the model regards as degenerate text-$\varnothing$ sentence pairs. Apart from the output layer, both training and fine-tuning use the same architecture (see Figure 1).

3.1 Data set

We used PyTorch transformers implementation of HuggingFace (Wolf et al., 2019). They are based on Google AI's implementation (Devlin et al., 2018), originally done in TensorFlow. We chose the simple BERT model for masked text prediction with pre-trained weights `'bert-base-uncased'`. This model was trained on a corpus combining BookCorpus (800 million words) and English Wikipedia (2.5 billion words). The tokenizer uses a vocabulary size of 37 000 WordPiece tokens.

We fine-tuned the model using a GPU in Google Colab using the default hyperparameters and the data set provided by (Dernoncourt and Lee, 2017). This data set, mostly used for sequence classification, consists of approximately 200 000 abstracts of randomised controlled trials, totalling 2.3 million sentences. They always follow the same pattern with each line in the data set being tagged with one of the following: OBJECTIVE, METHODS, RESULTS, CONCLUSIONS. However, we limited our training data to 10 MB in order to reduce the time required for fine-tuning. Fine-tuning provided us with a new set of weights that were subsequently used in our experiments.

3.2 Procedure

We proceeded as follows:

1. We gathered five short paragraphs from each of the following:
   - articles on CBC News website.
   - articles on `Medscape.com` website, specific for medical news.
   - the same data set used for fine-tuning (this, however, is a test set disjoint from the training and validation sets used for fine-tuning).
2. For each one of the paragraphs, we ran a code that would randomly mask 10% of the words and then try to predict them using the pre-trained model.
3. For each one of the paragraphs, we did that five times, discarding the results that presented the lowest and the highest accuracy in order to control for outliers.
4. For the remaining three results we noted the corresponding accuracy and score and calculated an average.
5. We then repeated the same experience, this time using the fine-tuned weights for predicting the masked words.

4 Results

Detailed results can be found in Table 2 in the Appendix. Table 1 summarises results by averaging group of files. We see the fine-tuned test set performed better than with the pre-trained weights but

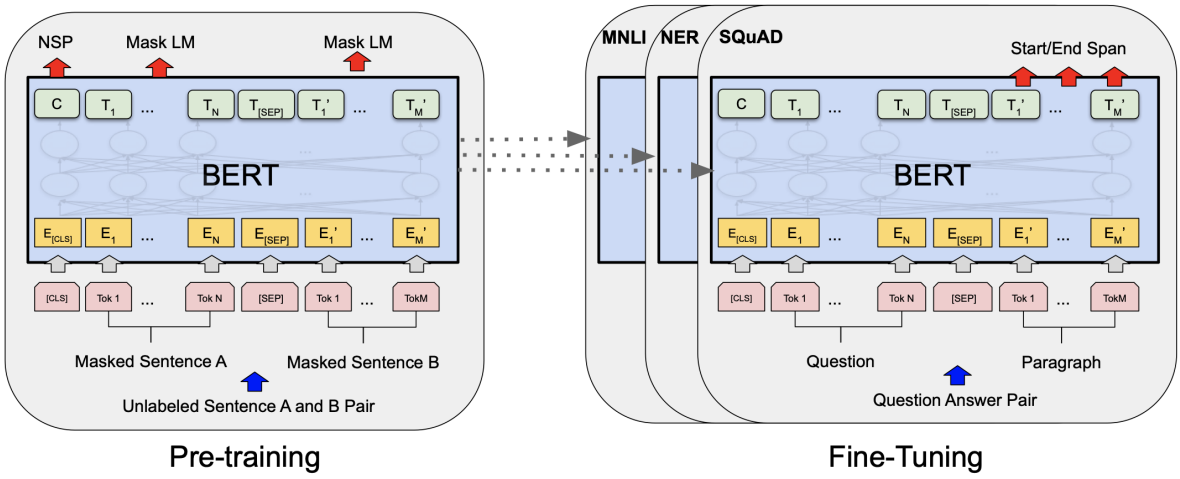

Figure 1: Apart from the output layer, training and fine-tuning BERT share the same architecture. Compared to pre-training, fine-tuning is relatively inexpensive. Still, it took us approximately 3 hours to fine-tune a 10 MB file on Google Colab. Figure reproduced from (Devlin et al., 2018).

we had a mild deterioration of score with Medscape and CBC articles.

After fine-tuning, the score over the test data set improved by 0.80 points, an increase of 5.5%. This may seem modest, but it is worth nothing that we fine-tuned only on 10 MB of additional text.

Even though our ultimate goal is to achieve perfect accuracy, we realise accuracy can be a lottery. Synonyms are perfectly acceptable but would deteriorate accuracy. Likewise, consider the sentence: *She went to the market to buy* $[\texttt{MASK}]_1$ *and* $[\texttt{MASK}]_2$. Say $[\texttt{MASK}]_1$ is *tomatoes* and $[\texttt{MASK}]_2$ is *onions*. If both values were switched, no one would mind and yet we would have 0% accuracy. This is why we eliminated outliers by only keeping three measurements. And this is why we also measure scores.

Scores are, in fact, the output of the model itself. For each of the 37 000 tokens in the vocabulary, the model assigns a score by means of a softmax function. The higher the score, the higher the probability of that word being the correct one. But synonyms or words that belong to a certain class like *onions* and *tomatoes* would score just as high.

5 Discussion and conclusion

Although we were hoping that fine-tuning the model would also improve prediction for Medscape articles, we still believe that fine-tuning presents great advantages as we could verify with the test data set. So why did it not help with Medscape articles?

First, the data set used was very specific. It is primarily used for sequential sequence prediction and is derived from reports on randomised clinical trials. Medscape articles, on the other hand, are much more generic, and even if they all relate to some medical subject, their structure more closely resembles that of a news article. We believe that if we are to achieve excellent results predicting Medscape text, then we would have to fine-tune the model with data from similar websites, focusing on generic medical articles rather than scientific experiments.

	Accuracy	**Score**
pre-trained		
CBC	61.20%	13.64
Medscape	62.71%	14.57
Test	65.71%	14.62
Fine-tuned		
CBC	66.92%	13.15
Medscape	57.04%	14.01
Test	67.53%	15.42

Table 1: Fine-tuned model improves accuracy and score for the test data set paragraphs but not for Medscape articles.

Second, although the data set provided by (Dernoncourt and Lee, 2017) is huge, for the purposes of this project, we only used a tiny little portion of it. Fine-tuning took three hours to complete on Google Colab's GPU for our 10 MB file. We believe we could have achieved even better results had we trained on a larger portion of the data set and run the process for longer or on a more pow-

erful machine.

What is interesting to note, however, is that fine-tuning on a particular data set did not substantially deteriorate performance of the more general domains, because the pre-trained weights are not being entirely rewritten by the fine-tuning process but are rather being adjusted.

In a certain way, we were trying to achieve better performance for a particular task by overfitting on that specific data, even at the expense of worse generalisation. Fortunately, this effect was mild and the model still generalises well.

Statement of contributions

Myriam Laiymani came up with the project idea, chose the task (masked text prediction) that we would be using for running our experiments and created the first draft of the code. Humam Seddik improved the code for masking a given percentage of words and obtaining the prediction scores. He also collected domain-specific data and parsed that text for tuning the model. Michael Benitah worked on the code structure, model fine-tuning on Google Colab and devising and running the experiments. Everyone contributed to the report.

References

Franck Dernoncourt and Ji Young Lee. 2017. Pubmed 200k rct: a dataset for sequential sentence classification in medical abstracts.

Jacob Devlin and Ming-Wei Chang. 2018. Open sourcing bert: State-of-the-art pre-training for natural language processing.

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*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL* `https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf`.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

A Appendices

Table 2 presents the average of the three prediction results of each paragraph after discarding the lowest and highest accuracy to control for outliers.

B Supplemental Material

The accompanying ZIP file contains the required code to run the predictions but it is possible that you may have to install additional components such as PyTorch, the transformers package, regex or sacremoses. To run the code with the pre-trained weights, use: `python COMP550_Project.py <text_file> --percent_words 10%`. The flag `-w` can be added to use fine-tuned weights. The fifteen short paragraphs used for running the experiments can be found in the `tests/` subfolder. You may use other text files as long as it is short enough to keep within the 512-token limit.

The data sets used for fine-tuning and testing can be found in the `finetunedata/` subfolder. The complete data set can be downloaded from GitHub at `https://github.com/Franck-Dernoncourt/pubmed-rct`. The code used for fine-tuning is provided under `transformers/run_lm_finetuning.py` but you may prefer running it from the Google Colab notebook provided here.

Text	pre-trained		Fine-tuned	
	Accuracy	**Score**	**Accuracy**	**Score**
CBC01	58.33%	12.24	62.50%	11.55
CBC02	66.67%	13.76	80.95%	14.26
CBC03	72.22%	15.12	63.89%	14.10
CBC04	60.61%	13.66	60.61%	12.23
CBC05	48.15%	13.42	66.67%	13.59
Medscape01	61.11%	13.60	50.00%	14.15
Medscape02	66.67%	15.39	70.00%	15.68
Medscape03	64.12%	14.46	56.86%	12.49
Medscape04	63.89%	14.42	52.78%	13.86
Medscape05	57.78%	14.96	55.56%	13.88
Test01	76.07%	14.68	81.20%	17.63
Test02	64.58%	14.70	61.46%	14.59
Test03	55.56%	14.28	63.25%	15.01
Test04	67.46%	15.08	78.57%	16.37
Test05	64.86%	14.37	53.15%	13.51

Table 2: Average accuracy and score for the pre-trained and fine-tuned models. The averages are calculated on the three remaining measures after discarding the results from the lowest and highest accuracy.