A summary of BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Wael Ben Hadj Yahia and Dorian Hervé

Link to our GitHub code for a multi-class classification problem using BERT

1 Introduction

BERT [1] (Bidirectional Encoder Representations from Transformers) is a language model designed to tackle various Natural Language Processing (NLP) tasks. What makes BERT innovative is its ability to obtain the bidirectional context of words (by *simultaneously* taking into account both the words before and after a given word), leading to a more accurate grasp of their meaning in a sentence. Furthermore, it takes advantage of transfer learning in order to provide a range of task-specialized models at a relatively low computational cost.

2 Context and Related work

2.1 Transformers

Given that, under the hood, BERT is a succession of encoders from the Transformers [2] architecture, we will take the time to go over a concise explanation of its key components.

The motivation behind Transformers is to drop the widely used Recurrent Neural Network (RNN) architectures in language models that not only have trouble with long sequences (exploding or vanishing gradients during retropropagation) but are also hard to parallelize, and replace them solely with an attention mechanism. While there are models that combine these two approaches, Transformers is the first model that relies entirely on an attention mechanism to draw global dependencies between words, leading to a significantly higher parallelization potential.

As seen in Fig.1, a Transformer consists of two parts : an encoder and a decoder. As input, it takes the embedding of each word in the sequence of interest to which we add a positional embedding (to conserve order) using the sine and cosine functions :

$$\text{PE}_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$\text{PE}_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

where pos is the position, i the dimension and $d_{model} = 512$ is the unified output dimension.

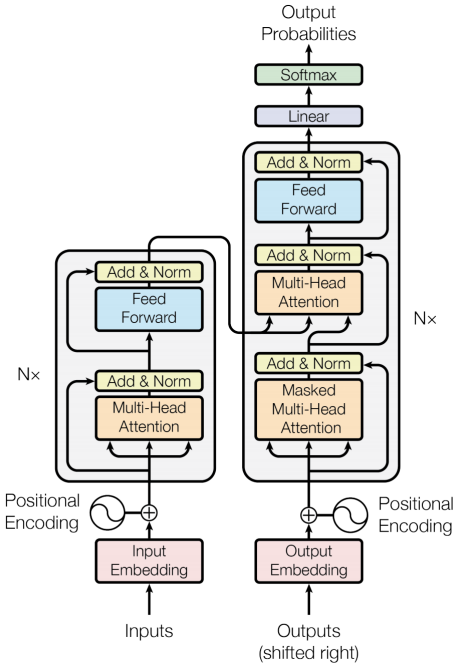

Fig. 1: The Transformer - model architecture

This representation is then used as input to an attention-computing process. An attention function's purpose is to map a query and a set of key-value pairs (all of which are vectors) into an output. In this endeavor, the attention mechanism takes a query Q (a word taken in a vectorial representation), the keys K (all the other words in the sentence), and the associated value V. We can also concatenate all the associated word vectors of a given sequence into equivalent matrices Q, K, and V. The attention computing mechanism can be summarized by the following formula :

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where $\sqrt{d_k}$ is the dimension of K.

When we compute the normalized dot product between the query and the keys, we get a tensor that represents the relative importance of each other word for the query in question. This is a key procedure to infer contextual information leading to a more precise encapsulation of the words' meaning, which is at the origin of the *truly bidirectional* aspect of BERT.

The advantage of this procedure is that, unlike its counterpart in recurrent neural networks, the attention function can be parallelized with respect to each query (i.e each word in the input sequence). This is yielded by the following formula :

$$MultiHead(Q, K, V) = Concat(head_1, \ ... \ , \ head_h)$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ and $W_i^Q \in \mathrm{R}^{d_{model} \times d_K}$, $W_i^K \in \mathrm{R}^{d_{model} \times d_K}$ and $W^0 \in \mathrm{R}^{hd_v \times d_{model}}$ are the projection matrices as defined in [2], and $d_{model} = 512$.

The next block is the Feed-Forward cell, which is a simple feed-forward network that is applied to every one of the attention vectors generated from the previous step (one vector at a time). This process can be parallelized, where each network could transform an attention vector independently from the others. These feed-forward networks are used in practice simply to reshape the attention vectors into the expected input shape of the next block.

For optimization purposes, both the Multi-Head Attention cell and the Feed-Forward cell outputs go through an Add&Norm cell, which consists of the following operation :

$$LayerNorm(x + Sublayer(x))$$

where $Sublayer(x)$ is the the function implemented by the sub-layer itself (self-attention in the Attention cell for example).

Despite the decoder block being absent from BERT's structure, a quick discussion is due. When compared to the encoder, the decoder has a nearly identical structure, but with a few twists : the first cell is a *Masked* Multi-Head Attention cell. Its subtlety resides in the masking of all the words that are *after* each output word at the bottom of the decoder block. Moreover, the next Multi-Head attention block takes the previous cell's output as well the encoder's output, and maps all of the relative attention vectors. For example, in the case of English to French translation, this is where the mapping between English words and French words happens. At the end of the decoder block, a Linear transformation is applied (which is another Feed-Forward layer) in order to transform the result into the subspace of interest (in our example, the desired output shape is the number of words in the French language). The result is then passed into a Softmax layer to transform the result into a distribution, which can then be transformed into the desired output (in our case, we can elicit the predicted French word).

2.2 Other language models

There are other language models that share similar procedures, of which the two most relevant are OpenAI GPT [3] and ELMo [4]. On one hand, in the same way that BERT is a succession of Transformer encoders, OpenAI GPT is a stack of Transformer decoders (which, due to the Masked Multi-Head Attention layer, makes it a left-to-right approach). However, they are both fine-tuning approaches. On the other hand, ELMo uses a concatenation of left-to-right and right-to-left LSTMs (which naturally suffers from the flaws of RNN-based models). Unlike those previous models, ELMo uses a feature-based approach.

3 Contribution

The pre-training of BERT is what gives the model the ability to *understand language* and to grasp the context of words. This is achieved by training on two supervised tasks simultaneously : Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

- Masked Language Modeling : For this task, BERT randomly selects and masks 15% of the training corpus by changing the selected words with a [MASK] token. The goal is to correctly predict the masked tokens.
- Next Sentence Prediction : For this task, BERT takes in two sentences and the goal is to determine if the second sentence actually follows the first sentence.

At the end of the pre-training phase, BERT should be capable of grasping the contextualized meaning of words. Prior to passing text into BERT, the text needs to be encoded in a particular manner (Fig.2). For each word in a sentence, we need to concatenate its Position Embedding (which indicates its position in a sentence), its Segment Embedding (which indicates the sentence (A for First/B for Second) containing the word), and a Token Embedding (the corresponding token of this word in the WordPiece embedding tokens). The aggregation of these three vectors gives us an embedding vector that we use as input to BERT. The role of the Segment Embeddings and Position Embeddings is to preserve the chronological ordering of the text. The fact that all these vectors are fed in simultaneously to BERT makes this step crucial, since language models need the ordering preserved to correctly infer meaning.

3.1 Pre-training

NSP : With evey input passed into BERT, we get a token C (Fig.3) as output, along with the word vectors. C is a binary value indicating whether or not Sentence A follows Sentence B. This task allows the model to understand the *relationship* between two sentences, which isn't captured by MLM. This could be crucial for certain tasks for which this relationship is important, such as Question Answering (QA) and Natural Language Inference (NLI).

MLM : Once we have the output word vectors (all of which are of the same size), they are passed into a fully connected layer output with the same number of neurons as the vocabulary size of the pre-trained embedding model (so 30,000 here since WordPiece is used). Then, a soft-max activation is applied to convert word vectors into distributions, and these

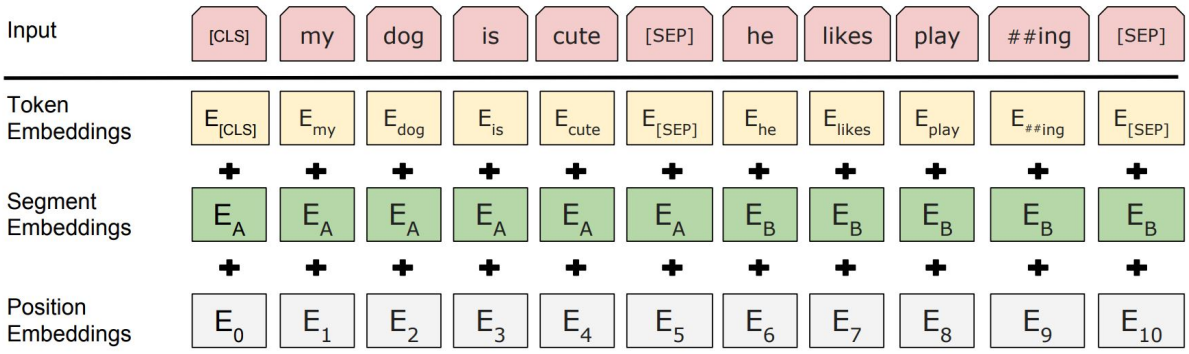

Fig. 2: BERT input embedding representation

distributions are compared with the ones that correspond to the actual words. This process minimizes the cross-entropy loss function between the predicted word and the correct word. Given that these are in a discrete support, the loss takes the following form:

$$L(p, q) = \sum_{x \in \chi} p(x)\, log\, q(x)$$

where p and q are probability distributions on the same support χ, which is the WordPiece embedding values here.

The subtlety of this process is that it only takes into account the loss between the masked words' predictions and the true words (it ignores the words that weren't masked in the computation of the loss). This is done to ensure that the emphasis is on the prediction of masked words, so that the model can improve in this endeavor. MLM is a great way to train a language model in a self-supervised setting (without human-annotated labels).

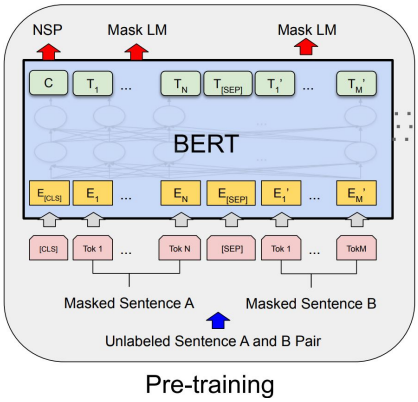

Fig. 3: Pre-training procedure for BERT

3.2 Fine Tuning

The fine tuning phase allows us to construct adapted architectures for a specific NLP task by applying minimal modifications to the pre-training model architecture and parameters. In fact, the only alteration made is the replacement of the output layer with a new set of output layers that can give adequate answers with respect to the specific targeted NLP task (such as Question Answering, Next Sentence Prediction, or Sentiment Analysis). As such, the only parameters learned from scratch are the ones associated with the new output layer, while all of the other model parameters are simply fine tuned. Naturally, the inputs (which aren't necessarily Sentence A followed by Sentence B) must be adapted to the task. For example, for Question Answering, the first sentence contains the question, while the second sentence would be the paragraph containing the answer. The output layers could return a Start/End delimiter that would indicate the position of the answer in the paragraph (assuming that it exists).

4 Discussion

4.1 Strengths

BERT's major strengths can be outlined by two major assets.

First of all, the pre-trained model offers numerous application possibilities at a relatively low computational cost thanks to the flexibility offered by the transfer learning mechanism. The pre-training phase, which gives the model a solid language understanding capability, would have been unattainable to most users due to the high computational power required. The option to capitalize on this pre-trained model by shaping it into a mean to answer a specific NLP task through fine-tuning is what makes it accessible.

Secondly, for each word and at all levels, BERT simultaneously considers the text that is before and after the word. This allows it to better grasp the context surrounding the word, which leads to a more accurate meaning encapsulation. This is different (and more precise) than a sequential concatenation of left context and right context, hence the *truly bidirectional* representation of words.

4.2 Limitations

While BERT brings a substantial contribution to the table in the realm of NLP, there are certain limitations that arise upon implementing the model.

First of all, as a data-dependant deep learning model, the performance of BERT is inherently dependant on the training data used in the fine-tuning phase. Naturally, poor data quality (quality or quantity wise) will reflect on the

performance of the model. Furthermore, at the time of release of this article, the Hugging Face [6] implementation of BERT (the most popular one) does not support long texts as inputs (each tokenized sentence needs to be shorter than 512 tokens). This begs another question: what happens when the text used for fine-tuning is not structured in the same manner as the pre-training text? One could argue that in order to deal with heavily processed text (stemmed and devoid of stopwords for example), it might be advantageous to pre-train BERT on similarly processed text. However, this would be out of the end user's reach. Yet, we could imagine that processing text in this way could be a sound solution to avoid the aforementioned text-length limitation.

Another issue to raise is the hefty cost of BERT's pre-training. The required computational power entails a very pricey procedure (an estimated 7,000 dollars for BERT-BASE). This makes it unattainable to most end-users who will use it under a black box setting. It also translates into a substantial impact on the environment when considering the accumulated computing that went into the development, testing, and tweaking of BERT and all of its size variants. The showcased results show that the increase in the model's size translates into exponentially more complex models (which are consequentially longer to train) while barely offering any relevant enhancement on the performance. In light of the unrewarded gain of complexity that fails to surpass a certain performance threshold, one could argue that at some point, a line must be drawn, and a call be made to settle for a reasonable model.

4.3 Implementation and results

Finally, while the benchmark results on standard NLP tasks seem impressive, there might be some NLP problems for which BERT might not be adapted. In fact, we explored BERT (through the Hugging Face library) as an option to tackle a multi-class classification problem. The data consists of 217,197 samples, where each entry consists of a job description (a text sequence describing the job of a person, with a non-uniform length throughout the dataset), and the associated job category, which is a qualitative variable with 28 possible values. We fine-tuned the model by adding 5 additional layers, and fitted the model on our train sample. The model implemented was Distill-BERT, a lighter version of BERT, and the training took more than 14 hours on an 8-core machine. The metric used is a macro-F1 score (which is simply the average of the F1 scores across the 28 classes). The yielded results were 69% on the validation data, and 59% on the test data.

To put this into perspective, a strategy consisting of applying a Logistic Regression on a truncated TF-IDF matrix gave us 72% accuracy on the test data, while a Linear SVM Classification approach on the same matrix yielded 74% accuracy on the test data. Those strategies were also significantly faster to run (less than 45 minutes for either of those). Needless to say, this performance did not meet its expectations.

This led us to alternative approaches, such as the user-friendly SimpleTransformers [7] library that gives access to Transformer-based architectures with just a few lines of code. Through trial and error, we have found that the model is much more efficient with a longer sequence-length, minimal text pre-processing, fewer added layers, and granted optimal results when trained through 3 epochs approximately (with a clear overfit whenever we trained it for more than 5 epochs). This time, we were able to achieve more than 80% accuracy on the test data, which was far better than the TF-IDF based representations. Both notebooks can be accessed here (Link to our GitHub code), where more details are available.

5 Conclusion

BERT's major contribution consists in offering a flexible language model that takes advantage of bidirectional architectures that capture word representations more precisely. This is integrated during the pre-training phase, during which BERT learns to *understand language* and *sentence relationships* through MLM and NSP. BERT also offers the possibility of taking advantage of the pre-trained model using transfer learning in order to build task-specific models at a relatively low computational cost. This is achieved by learning the parameters of an added output layer (from scratch), and fine-tuning all the other parameters.

Nevertheless, the current implementations of BERT are limited in certain aspects, notably when it comes to large inputs. Furthermore, there is a call to reason regarding the unjustified development of increasingly complex models. Moreover, the previously mentioned job classification problem shows that configuring BERT properly might be a delicate task, despite the model consistently showing promising results on reference NLP tasks.

Finally, it seems that BERT will pave the way for promising research in the expanding NLP domain. In fact, BERT has already motivated fruitful new work, such as Google's XLNET [5], who aims to overcome BERT's major technical flaws, such as its occasional pretrain-finetune inconsistency, as well as its reliance on switching part of the input with masks while neglecting dependency between the masked positions.

References

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

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

[3] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

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

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

[6] HuggingFace, BERT HuggingFace documentation, https://huggingface.co/transformers/model_doc/bert.html.

[7] SimpleTransformers, BERT SimpleTransformers documentation, https://simpletransformers.ai/docs/usage/.