Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Technology
(An Autonomous Institute affiliated to Savitribai Phule Pune University)

Department of Information Technology

IT-Bulletin

March-2022

'Classifying Long Text Documents Using BERT'

21st Feb 2022

HIGHLIGHTS

Transformer based language models such as BERT are really good at understanding the semantic context because they were designed specifically for that purpose. BERT outperforms all NLP baselines, but as we say in the scientific community, "no free lunch". How can we use BERT to classify long text documents?

What do we want to achieve?

We want to classify texts into predefined categories which is a very common task in NLP. For many years, the classical approach for simple documents was to generate features using TF-IDF and combine it with logistic regression. Formerly we used to rely on this stack at Sinequa for textual classification, and, spoiler alert, with the model presented here we have beaten our baseline from 5% to 30% for very noisy and long documents datasets. This former approach had two main issues: the feature sparsity that we tackled via compression techniques and the word-matching issue that we tamed leveraging Sinequa's powerful linguistic capacities (mainly through our home-grown tokenizer).

Later on, the pandora box of language models (pre-trained on humongous corpora in an unsupervised fashion and fine-tuned on downstream supervised tasks) was opened and TF-IDF be word2vec combined with LSTMs or CNNs, ELMo, and most importantly the Transformer (in 2017: https://arxiv.org/pdf/1706.03762.pdf).

based techniques were not state of the art anymore. Such language models could be word2vec combined with LSTMs or CNNs, ELMo, and most importantly the Transformer (in 2017: https://arxiv.org/pdf/1706.03762.pdf).

BERT is a Transformer based language model that has gained a lot of momentum in the last couple of years since it beat all NLP baselines by far and came as a natural choice to build our text classification.

What is the challenge then?

Transformer based language models such as BERT are really good at understanding the semantic context (where bag-of-words techniques fail) because they were designed specifically for that purpose. As explained in the introduction, BERT outperforms all NLP baselines, but as we say in the scientific community, “no free lunch”. This extensive semantic comprehension of a model like BERT offers comes with a big caveat: it cannot deal with very long text sequences. Basically, this limitation is 512 tokens (a token being a word or a subword of the text) which represent more or less two or three Wikipedia paragraphs and we obviously don’t want to consider only such a small sub-part of a text to classify it.

To illustrate this, let’s consider the task of classifying comprehensive product reviews into positive or negative reviews. The first sentences or paragraphs may only contain a description of the product and it would likely require to go further down the review to understand whether the reviewer actually likes the product or not. If our model does not encompass the whole content, it might not be possible to make the right prediction. Therefore, one requirement for our model is to capture the context of a document while managing correctly long-time dependencies between the sentences at the beginning and the end of the document.

Technically speaking, the core limitation is the memory footprint that grows quadratically with the number of tokens along with the use of pre-trained models that come with a fixed size determined by Google (& al.).

'Classifying Long Text Documents Using BERT '

This is expected since each token is “attentive” [https://arxiv.org/pdf/1706.03762.pdf] to every other token and therefore requires a [N x N] attention matrix, with [N] the number of tokens. For example, BERT accepts a maximum of 512 tokens which hardly qualifies as long text. And going beyond 512 tokens rapidly reaches the limits of even modern GPUs.

Another problem that arises using Transformers in a production environment is the very slow inference due to the size of the models (110M parameters for BERT base) and, again, the quadratic cost. So, our goal is not only to find an architecture that fits into memory during the training but to find one that also responds reasonably fast during inference.

The last challenge we address here is to build a model based on various feature types: long text of course, but also additional textual metadata (such as title, abstract …) and categories (location, authors …).

So, how to deal with really long documents?

The main idea is to split the document into shorter sequences and feed these sequences into a BERT model. We obtain the CLS embedding for each sequence and merge the embeddings. There are a couple of possibilities to perform the merge, we experimented with:

- Convolutional Neural Networks (CNN)
- Long Short-Term Memory Networks (LSTM)
- Transformers (to aggregate Transformers, yes :))

Our experiments on different standard text classification corpora showed that using additional Transformer layers to merge the produced embeddings works best without introducing a large computational cost.

Want the formal description, right?

We consider a text classification task with L labels. For a document D, its tokens given by the WordPiece tokenization can be written $X = (x_1, \ldots, x_n)$ with N the total number of token in D. Let K be the maximal sequence length (up to 512 for BERT). Let I be the number of sequences of K tokens or less in D, it is given by $I = \lfloor N/K \rfloor$.

'Classifying Long Text Documents Using BERT '

Note that if the last sequence in the document has a size lower to K it will be padded with 0 until the K^{th} index. Then if s^i with $i \in \{1, .., I\}$, is the i-th sequence with K elements in D, we have:

$$s^i = (x_{Ki+1}, \ldots, x_{K(i+1)})$$

We can note that

$$X = \bigcup_{i=1}^{I} s^i$$

BERT returns the CLS embedding but also an embedding per token.

Let define the embeddings per token returned by BERT for the i-th sequence of the document such as:

$$CLS^i, y_1^i, \ldots, y_K^i = BERT(s^i)$$

where CLS is the embedding of the special token inserted in front of each text sequence fed to BERT, it is generally considered as an embedding summarizing the full sequence.

To combine the sequences, we only use CLS_i and do not use y. We se t transformers T_1, …, T_t to obtain the final vector to feed to the last dense layer of the network:

$$Z = T_t \circ \ldots \circ T_1(CLS^1, \ldots, CLS^I)$$

where . is the function composition operation.

Given the last dense layer weights $W \in R^{LxH}$ where H is the hidden size of the transformer and bias $b \in R^L$

The probabilities $\boldsymbol{P} \in \mathbb{R}^L$ are given by:

$$P = softmax(WZ + b)$$

'Classifying Long Text Documents Using BERT '

Finally, applying *argmax* on the vector *P* returns the predicted label. For a summary of the above architecture, you can have a look at figure 1.

The architecture above enables us to leverage BERT for the text classification task bypassing the maximum sequence length limitation of transformers while at the same time keeping the context over multiple sequences. Let's see how to combine it with other types of features.

How to deal with metadata?

Oftentimes, a document comes with more than just its content. There can be metadata that we divide into two groups, textual metadata, and categorical metadata.

Textual Metadata

By textual metadata, we mean short text that has (after tokenization) a relatively small number of tokens. This is required to fit entirely into our language model. A typical example of such metadata would be titles or abstracts.
Given a document with *M* metadata annotation. Let

$$CLS_1,\ \dots,\ CLS_M$$

be the CLS embeddings produced by BERT for each metadata. The same technique as above is used to get the probability vector as:

$$P = softmax(WT_t \circ \ \dots \ \circ T_1(CLS_1,\ \dots,\ CLS_M) + b\ )$$

Categorical Metadata

Categorical metadata can be a numerical or textual value that represents a category. Numerical values can be the number of pages whereas textual values can be the publisher name or a geo-location.

A common way to deal with such features is to implement the Wide and Deep architecture. Our experiments showed that results yielded by the deep part of this network were sufficiently good and the wide part was not required.

We encode the categorical metadata in a single cross-category vector using one-hot encoding. This encoding is then passed into an embedding layer that learns a vector representation for each distinct category. The last step is to apply a pooling layer on the resulting embedding matrix.

We considered max, average and min pooling and found that using average pooling worked best for our test corpora.

How does the complete architecture look?

Hope you stuck around until now, the following figure will hopefully make things a lot clearer.

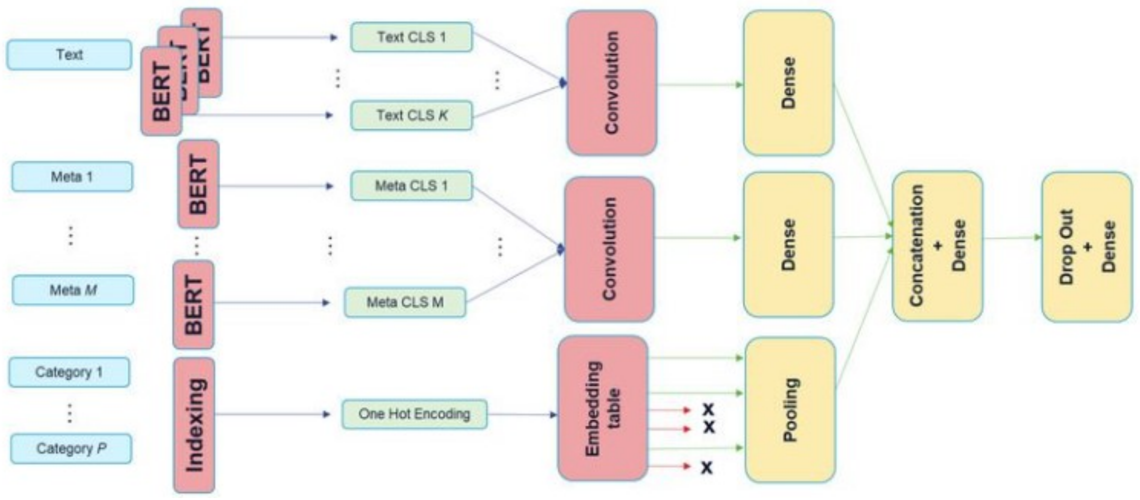


Figure 1

There are three sub-models, one for text, another for textual metadata, and the last one for categorical metadata. The output of the three sub-models is merely concatenated into a single vector before passing it through a dropout layer and finally into the last dense layer with a softmax activation for the classification.

You probably have noticed that there are multiple BERT instances depicted in the architecture, not only for the text input but also for the textual metadata. As BERT comes with many parameters to train, we decided not to include a separate BERT model per sub-model, but instead share the weights of a single model in between the sub-models. Sharing weights certainly reduces the RAM used by the model (enabling training with larger batch-size, so accelerating training in a sense) but it does not change the inference -time since there will still be as many BERT executions no matter whether their weights are shared or not.

'Classifying Long Text Documents Using BERT '

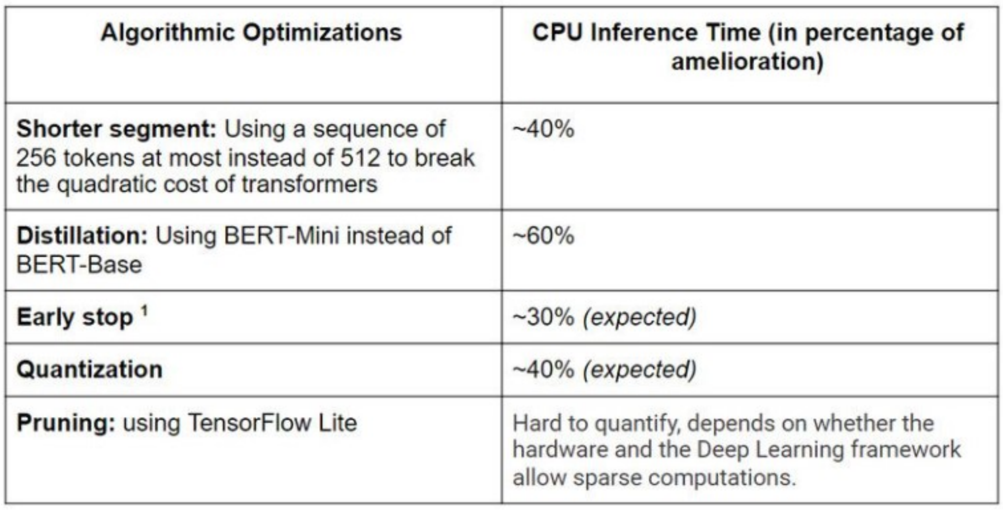
What about inference time?

By now, you must have guessed that including that many invocations of the BERT model do not come for free. And it is true that it is computationally expensive to run inference of such a model. However, there are a couple of tricks to improve inference times. In the following, we focus on CPU inference as this is very important in production environments.

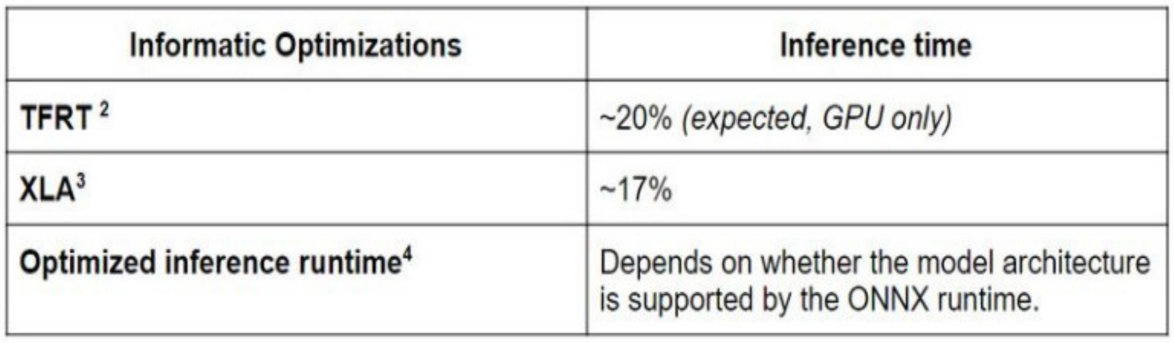
A couple of notes for the conducted experiments:

- We consider a simplified model only containing a text feature.
- We limited the tokens that we used per document to 25,600 tokens which correspond roughly to around 130,000 characters if the document contains English text.

We perform the experiments with documents that have the above described maximum length. In practice, documents have varying sizes and as we use dynamic size tensors in our model, inference times are considerably faster for short documents. As a rule of thumb, when using a document that is half as long reduces the inference time by 50 %.



Algorithmic Optimizations	CPU Inference Time (in percentage of amelioration)
Shorter segment: Using a sequence of 256 tokens at most instead of 512 to break the quadratic cost of transformers	~40%
Distillation: Using BERT-Mini instead of BERT-Base	~60%
Early stop [1]	~30% *(expected)*
Quantization	~40% *(expected)*
Pruning: using TensorFlow Lite	Hard to quantify, depends on whether the hardware and the Deep Learning framework allow sparse computations.



Informatic Optimizations	Inference time
TFRT [2]	~20% *(expected, GPU only)*
XLA[3]	~17%
Optimized inference runtime[4]	Depends on whether the model architecture is supported by the ONNX runtime.

References

1 https://arxiv.org/abs/2006.04152, https://arxiv.org/pdf/2001.08950.pdf

2 https://blog.tensorflow.org/2020/04/tfrt-new-tensorflow-runtime.html

3 https://www.tensorflow.org/xla?hl=fr

4 https://medium.com/microsoftazure/accelerate-your-nlp-pipelines-using-hugging-face-transformers-and-onnx-runtime-2443578f4333

‘Olympics 2022: IoT’s Role in Olympic Timekeeping’

8th February 2022

Long-time Olympic sponsor Omega is once again bringing its technology to he Games with its latest AI capabilities

Written by Liz Hughes

Timekeeping is vital to the Olympic Games and the 2022 Winter Games is no exception.

Long-time Olympic sponsor Omega is once again bringing its technology to the Games with its latest artificial intelligence (AI) capabilities seen from the Olympic beach volleyball courts at the Tokyo summer games to the downhill skiing gates and speed-skating finish lines in Beijing.

Omega uses motion sensing and positioning technologies to continue to revamp how it measures what’s happening in each Olympic event. The company unveiled those technologies at the 2018 Pyeongchang Winter Olympic Games .

By using a combination of image tracking cameras and sensors worn by athletes, Omega can show the live speed of a bobsled as it races down the course or the live positions in speedskating, giving everyone from commentators to spectators, athletes and coaches information to fully analyze each sport in real-time.

Here’s a look at some of the ways Omega is using its technology at the 2022 Olympic Games:

- **Real-Time Tracking System:** Omega launched its Real-Time Tracking System (RTTS) in 2019. Used in track events, the system allows sensor tags fixed to athlete’s start numbers to communicate with the receivers on the track and send information back to its onsite computers tracking the athlete’s live speed, acceleration, deceleration and distance.
- **The Scan‘O’Vision MYRIA:** Omega’s Scan‘O’Vision MYRIA photo finish camera can record up to 10,000 images per second. That knowledge arms Olympic judges for official rankings and times at each event. Omega introduced the system in Albertville in 1992.
- **Snowgate Technology:** When alpine skiers break through Omega’s Snowgate, the timing system is automatically activated ensuring precise timing of each race with the technology the company introduced at the 2010 Winter Games

‘Olympics 2022: IoT’s Role in Olympic Timekeeping’

8th February 2022

in Vancouver.

- **Photoelectric Cells:** Speedskating timing is handled by photoelectric cells placed on the finish line. These little red boxes give off beams of light just two to three centimeters above the ice. When a competitor skates across it, the clock automatically captures the finishing time.

The Quantum Timer: Seconds are electronically counted on a Quantum Timer. By offering an enhanced resolution of one-millionth of a second, it’s five times more accurate than previous versions. Omega debuted this technology at the 2012 London Games.

Student Editor: Priti Patil ,Madhuri Jadhav

TOP

‘Key IoT predictions in 2022 – The year enterprises take control’