Improving Self-Attention Mechanisms for Long Document Classification

Cali Rivera, Iris Cheng, Livia Gimenes

Introduction

Transformers, particularly their self-attention mechanism, have revolutionized how sequences of text can be processed for natural language processing tasks. The Transformer architecture solves many of the limitations of traditional RNNs, most notably RNNs' limited long-term memory. However, Transformers' very strength, the fact that they are able to maintain near-perfect long-term memory by considering all words in a given sequence at each stage, has also become their weakness. Since the memory requirements of self-attention scale quadratically with the length of the input sequence, a major architectural limitation of Transformer-based models is that they are unable to process sequences longer than a few thousand tokens without exceeding the limits of reasonable computing resources. This prevents Transformers from being used in applications that require the encoding of longer texts like entire books, large images, and other kinds of data that include a very large number of tokens.

In their paper "Longformer: The Long-Document Transformer", Beltagy et al[1] presented a solution to this limitation. They proposed a novel attention mechanism that allows memory requirements to scale linearly with input sequence length, rather than the quadratic scaling of a traditional Transformer. This mechanism combines a local windowed attention with a task-motivated global attention, and serves as a replacement for the standard self-attention. Their pretrained Longformer model was able to set state-of-the-art results on various long document tasks, easily processing documents with over several thousands tokens.

In this project, we implemented a modified version of this new attention mechanism, namely the sliding window attention pattern, and built a model that employs this mechanism to perform long-document text classification. Particularly, the model was designed to classify truncated book texts by genre. Although this task in particular is relatively trivial, we aimed to use it as an example of the power of this modified attention mechanism in allowing models to work with longer texts. This same mechanism could be applied to numerous more complex tasks, including long document generative tasks such as chapter or book summarization.

[1] Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).

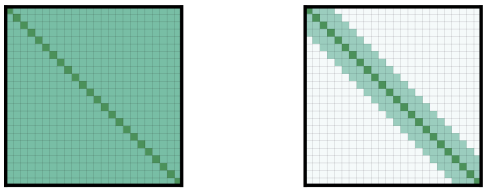
Related Work

As noted above, we are aiming to reimplement the work detailed by Beltagy et al in "Longformer: The Long-Document Transformer". Given that this paper already has supporting open-source code that uses Pytorch, we attempted to implement the system in TensorFlow, and also used a unique dataset when attempting to test our model. Furthermore, very late into our work on the project, we discovered a TensorFlow implementation[2] of this same model. It is important to note that this implementation is quite similar to our own implementation, as both aimed to essentially reimplement the same problem in TensorFlow.

Methodology

For our training data, we used a 10,000 Books and their genres Kaggle dataset[3], a combination of publicly available books from Project Gutenberg and their respective genres scraped from GoodReads. In order to optimize the run time of our model and account for the limited computing resources available to us, we decided to truncate the book data to only include only the first 1000 words of each book. Furthermore, we decided to include only books with only one genre in order to limit the complexity of the model to focus on the attention mechanism as well as reduce the size and run time of the model. With these constraints, our dataset contained the text of 1192 books as well as their corresponding genres, which served as our labels.

As detailed above, the key piece of our implementation was the modified self-attention mechanism. We decided to focus on the sliding window component of the proposed mechanism. As demonstrated in Figure 1, this form of attention operates almost like the convolution process. We define the width of our "window" and only allow each query node to attend to its associated key node, as well as the key node's neighbors within our defined attention. The method relies on the assumption that the most important contextual information for a given word lies in its nearest neighboring words.



(a) Full n^2 attention (b) Sliding window attention

Figure 1: Comparison between tradition self-attention and the the attention pattern we implemented[4]

[2] Hugging Face (2022) Modeling TF Longformer. https://github.com/huggingface/transformers/blob/main/src/transformers/models/longformer/modeling_tf_longformer.py

[3] Russell, M. 10,000 Books and Their Genres *standardized*. https://www.kaggle.com/datasets/michaelrussell4/10000-books-and-their-genres-standardized

[4] Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).

Our overall model architecture follows the details presented by Beltagy et al. Most notably, our model extends the RoBerta architecture, while replacing the traditional attention mechanism with our windowed attention mechanism.

Results

Ultimately, we were able to convert the original Pytorch implementation of the longformer Transformer architecture in Tensorflow. However, several issues arose that prohibited the training of the model. Primarily, because the longformer architecture is built on top of several pre existing architectures, it is therefore dependent on the packages for said components. In most cases, these base architectures were implemented in Pytorch which led to compatibility issues. We attempted to address these matters by instead having our model subclass from tf.keras.layers.Layer as well as Hugging Face's TFTrainer wrapper class[5], made to be compatible with Transformer models built on Tensorflow. Even having employed these workarounds, it became clear that this wrapper class had been deprecated, and a number of methods and attributes necessary for training our model were inaccessible. Attempts to manually configure these features on our end were unsuccessful, being that these classes often comprised thousands of lines of code. Although we were unable to train our Tensorflow-based model, a general idea of its potential performance can be inferred from the range of the longformer's capabilities as well as its performance as compared to other commonly employed Transformer models. Below is an example summary output of a Longformer model built for text summarization for the first chapter of *To Kill A MockingBird*, composed of 8792 tokens[6].

'When Jem is almost 13, he breaks his arm playing with his father Atticus and Jem. They always talk about what really happened to the family when they were kids--they think it was the Finches who moved to Alabama after their ancestor Simon Finch saved up all of his money to build a nice house called Finch's Landing. The story goes that Dill met Boo Radley in Mississippi while spending the summer at his aunt Rachel's house. There are rumors that Boo went out during the night and peered into people's windows; rumor has it that Boo breathed on flowers in the middle of the night. People also say Boo got into a fight with Arthur Radley Jr., but no one ever saw Boo until several years later. Boo ended up getting stabbed by his own father, which made Boo's reputation even more legendary. When Boo died, Calphurnia said Boo would "taunted" the house three more times before he ran away. Jem and Atticus watched as Boo chased after the droopy Radley home'

Challenges

Our main challenge was dealing with the size of the model as well as the size of the dataset. While completing the model, we naively assumed that because the architecture in the paper implemented a solution that processed longer text inputs more efficiently than a standard Transformer, we would not encounter as many issues with time complexity. However, on Google Colab Pro, OSCAR and the CS grid our model timed out multiple times while attempting to train. We tried fine tuning the model by reducing the batch size as well as running the model on a smaller dataset. We scaled back the datasets as much as possible, but it still was not enough for

[5] Hugging Face TFTrainer Class.
https://huggingface.co/transformers/v3.4.0/main_classes/trainer.html#transformers.TFTrainer
[6] Led-base-book-summary model. https://huggingface.co/pszemraj/led-base-book-summary

the model to be able to fully run and train. The Longformer architecture had over 100 million trainable parameters and while Roberta only had 14 million, both of those architectures themselves were too consuming for them to be handled on any resource that we had available to us.

As previously mentioned, a secondary challenge was the occurrence of incompatibility errors between our Tensorflow implementation of the longformer model and the wrapper classes needed to establish the baseline architecture as well as facilitate training. It had come to our attention at this point that the wrapper classes from HuggingFace's Transformer API were deprecated and no longer compatible with the latest version of Tensorflow. We attempted to circumvent this by manually configuring any inaccessible attributes and methods, but the sheer number of required configurations eventually grew too large for a feasible implementation.

Reflection

Ultimately, we were not able to meet our base, target, or stretch goals, although our initial set of goals is not longer relevant, as we scaled down from book text summarization to book text classification due to the complexity of the decoder we would need to implement to perform any sort of generative task. This pivot proved to be necessary, as the decoder architecture involved a number of additional components that would have simply taken too much time for us to both understand and implement correctly. If we were able to start fresh from the beginning of this project, we would have made this pivot much earlier, as we wasted an unfortunate amount of time trying to understand components that we ended up not needing to implement.

Clearly, the fact that we were unable to successfully run our model indicates that it was not entirely a success. However, a key part of our project, the implementation of a sliding window attention mechanism, could not be tested effectively without running our entire model. Consequently, it is difficult to conclude just how much of this component we were able to complete successfully. In spite of this, we feel confident in our implementation of windowed attention, as well as the majority of the other work we did.

If given more time, we would likely pivot once again to a simpler implementation. We got so caught up in perfectly mimicking the code base provided with Beltagy et al's paper that we ended up in way over our heads with subclasses upon subclasses that we did not really understand. Unfortunately, we realized this mistake much too far along in the project timeline, and thus had to make the decision to try to do as much as we could with our current architecture. Given that the key idea we were exploring was just the improved attention mechanism, we could have modified a much simpler Transformer-based model with our windowed attention mechanism.

Although our project was not entirely successful, we certainly learned a significant amount about both the intricacies of self-attention as well as the notable differences between TensorFlow and PyTorch. Initially, we believed that reimplementing the existing PyTorch code base in TensorFlow would be relatively straightforward. As we began attempting to complete this reimplementation, we quickly realized that some of the PyTorch code was organized in a

more object-oriented style, which could not be mimicked identically in TensorFlow. We also did not initially understand just how many dependencies used in the original code base relied on the implementation using PyTorch.

One final takeaway from this project is a better understanding of the limitations of the computational resources available to us. Although we knew that this project would likely be on the more computationally-demanding side, we assumed that since we had access to a Brown CCV account, we would not have to worry about running out of computational resources. Looking back, it is understandable how the scope of our project likely demanded more resources than were available to us.

References

1. Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).
2. Hugging Face (2022) Modeling TF Longformer. https://github.com/huggingface/transformers/blob/main/src/transformers/models/longformer/modeling_tf_longformer.py
3. Russell, M. 10,000 Books and Their Genres *standardized*. https://www.kaggle.com/datasets/michaelrussell4/10000-books-and-their-genres-standardized
4. Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).
5. Hugging Face TFTrainer Class. https://huggingface.co/transformers/v3.4.0/main_classes/trainer.html#transformers.TFTrainer
6. Led-base-book-summary model. https://huggingface.co/pszemraj/led-base-book-summary