Sliding Window with Global Attention – Longformer

Final Course Project Report – EN 705.743 ChatGPT from Scratch

Punita Verma

Introduction

Large Language Models (LLMS) are inherently sequential. Text is segmented and encoded into sequences of tokens that are used to train a model. Parallel computing resources previously struggled with handling and training on copious amounts of texts and sequential computation was necessary. The discovery of the transformer model[1] and associated self-attention mechanism by Vaswani et al. (2017) was a groundbreaking and pivotal moment for LLM training. Multi-Headed Self Attention (MHA) allowed for the powers of parallel computing and GPU processing to be exploited while training sequential data and large language models and preserving long-range dependencies amongst the data. Early originations of well-known LLMs today were possible due to the efficiency and compute power transformers and the attention mechanism unearthed.

One significant drawback of this architecture, however, is the quadratic scaling quality of the transformer attention mechanism. As sequence length increases, the time, computation, and memory requirements of the transformer also increase. This makes it inherently difficult to train models on longer sequences of tokens due to increasing resource requirements. The Longformer model[2], developed by Beltagy et al. (2020), overcomes this scaling limitation with the innovation of an attention pattern that scales linearly with sequence length. The attention mechanism used by the Longformer model does not cover the entire token space, but rather uses a sliding window pattern, dilation, and some global attention nodes to develop learnings between tokens.

In this project, we implement the sliding window attention with global attention nodes in the GPT LLM architecture. The goal of this implementation is to demonstrate the linear complexity of the improved attention model as contrasted with the quadratic complexity of the original transformer model. The expectation of this is to notice a linear increase in time and memory requirements from shorter to longer training sequence lengths for the Longformer model implementation. In comparison, the original Transformer model implementation will yield a quadratic increase in time and memory requirements for the same increases in sequence lengths.

The key results of this experiment showed that, as expected, the Baseline Transformer model exhibited increasing time and memory demands with longer sequences, consistent with its quadratic complexity. In contrast, the Longformer model demonstrated more efficient scaling, particularly in terms of time, thanks to its linear attention mechanism.

<u>Background</u>

The attention mechanism in an LLM is used to relate tokens to each other. This is an important aspect of how LLMs can work. Key patterns, contextual information and associations between tokens can be derived, learned on, and encoded.

Transformer Attention Mechanism

The transformer uses a set of vector matrices to achieve this. The components used to generate the attention values are the Queries (Q), Keys (K), and values (V). These vectors represent the textual inputs, their contexts or references, and the information being retrieved, respectively. In the proposed *self-attention* mechanism of the Transformer and what is used in the GPT model, these vectors refer to the same source and are thus called *self-attention.*

$$Attention(Q,K,V) = soft\max\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

To compute the attention scores, we use the attention equation above. First the Query vector is multiplied by the transpose of the Key vector. This result is then scaled by a factor of the dimension of the model in order to normalize. Finally, we apply a SoftMax to normalize the attention scores. This is done to ensure that any single token does not have an outsized impact on the computation. The SoftMax product is multiplied by the Values to get the final attention matrix. Self-attention is also called scaled-dot product attention due to its equation.

Taking this a step further, the Transformer applies *Multi-Head* self-attention (MHA). MHA allows the model to jointly attend to information from different representation subspaces at different positions simultaneously. Outputs from the different heads are concatenated and linearly transformed to project a final value:

$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_h)W^O$$

where

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right).$$

W_Q^i, W_K^i, and W_V^i are weight matrices specific to each attention head.

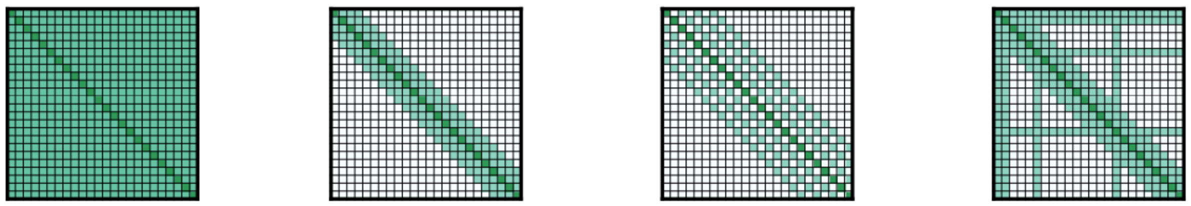
(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure from orignal Longformer paper (2)

The Transformer attention mechanism results in an attention matrix of size *NxN* (Figure A), where *N* is the sequence length. The time complexity for computing this attention is $O(N^2)$, meaning that as the sequence length increases, the computation time and memory requirements grow quadratically. This scaling makes it difficult to handle long sequences, as the resources needed increase dramatically with longer inputs.

Longformer Attention Mechanism

The Longformer improves upon this attention mechanism by introducing sliding window attention and global attention pattern. These innovations address the quadratic scaling problem, allowing the model to handle longer sequences with linear complexity.

Sliding Window

Longformer implements a sliding window attention (Figure B), which limits each token's attention to a fixed-size window of nearby tokens. Instead of attending to every token in the sequence, a token may only focus on a window of *window_size* tokens around it. This approach creates *local attention,* meaning tokens are only concerned with nearby tokens, which is usually enough for most language tasks. Because of this local scope, the Longformer's attention computation scales *linearly* with sequence length, making it efficient even for longer sequences.

Dilation

Dilated sliding attention (Figure C) adds gaps between the tokens that a single token attends to within the sliding window. This expands the model's view of the sequence without increasing computational load. Instead of attending to every consecutive token within the window, a token may only attend to every second or third token, which helps the model capture patterns that are further apart within the sequence, all while maintaining the efficiency of the sliding window.

Global Attention

While sliding window attention efficiently captures local dependencies, certain tasks require some tokens to look at the entire sequence for context. For instance, tasks like summarization or question answering rely on specific tokens to access global information. *Global attention* (Figure D) solves this by designating a few key tokens as *global attention tokens*. These tokens can attend to every other token in the sequence, and vice versa, allowing them to capture global context.

By combining sliding window attention, dilation, and global attention, the Longformer strikes a balance between computational efficiency and capturing both short and long-range dependencies. These innovations make it particularly well-suited for tasks that involve long sequences, where traditional Transformers struggle.

Methods

Dataset

The *Wikitext-103 dataset* was used for all experiments. This dataset consists of high-quality, large-scale textual data, making it well-suited for training and evaluating language models. To test the hypothesis across varying sequence lengths, the dataset was partitioned into three distinct sequence lengths: 256, 512, and 1024 tokens. These sequence lengths were chosen based on my estimation of the machine's capability to handle these inputs efficiently, balancing the need for meaningful long-range dependencies with the computational constraints of the available hardware.

Each sequence was padded to ensure uniformity, which allowed the models to process inputs with a consistent length. Padding was applied to shorter sequences, and longer sequences were truncated, if necessary, to maintain the fixed sequence length for each partition.

GPT Model Implementation

An implementation of a GPT architecture is used, tailored to integrate sliding window attention and global attention for efficient handling of longer sequences. The *CustomMHA* class, which defines the MHA mechanism, implements two versions of the attention pattern, one for each model, the *Baseline* transformer model and the *Longformer* model. For the Baseline attention pattern, self-attention is used out-of-the-box. For Longformer attention pattern, a sliding window pattern is implemented. In this case, each token only attends to a fixed-size window of neighboring tokens, significantly reducing the computational load.

Masking

To further refine the attention mechanism, attention masks are used. Masks are crucial for ensuring that certain tokens do not attend to irrelevant tokens or padding. This is particularly important in the Longformer model, where the sliding window restricts the attention span. For the Longformer attention pattern, a mask is applied to restrict attention to a fixed number of nearby tokens. This is done by creating a sliding window mask that zeroes out attention weights outside the window, ensuring that each token only attends to its neighboring tokens within the defined window size.

In addition to the sliding window, certain tokens are assigned global attention. For these tokens, a mask is applied that allows them to attend to every other token in the sequence, ensuring that global context is captured. This mask is combined with the sliding window mask to create a hybrid attention pattern, where most tokens attend locally, and a few key tokens have global reach.

The baseline model for these experiments was based on the GPT architecture. A Transformer architecture is used with self-attention layers. The architecture included multiple layers of attention, feed-forward networks, and normalization. For the Longformer variant, I implemented the sliding window attention and global attention mechanisms, modifying the attention layers within the GPT architecture. This allowed the Longformer to process longer sequences more efficiently, while retaining the core structure of the original GPT model.

Training Experiment Runs

All experiments were run using Google Colab. Due to time and resource constraints, each experiment was trained for 5 epochs. For each of the three sequence lengths (256, 512, 1024), experiments were run for both the Baseline transformer model and the Longformer model, resulting in a total of 6 experiments across 30 epochs.

Initial attempts to run the experiments on a CPU resulted in impractical runtimes (approximately 52 hours for a single experimental epoch). Consequently, after trial and error, I upgraded to an A100 GPU for training on datasets with sequence lengths of 256 and 512 tokens. The batch size for these experiments was set to 32. For the dataset with a sequence length of 1024 tokens, training was conducted using T4 GPUs with high RAM capacity. However, I had to experiment with different batch sizes for the 1024 token sequence due to memory constraints and excessive runtimes. Despite these adjustments, I was unable to successfully complete the training for the 1024 token sequence due to memory constraints and excessive runtimes.

Custom CUDA Kernel

The Longformer paper by Beltagy et al. (2020) discussed a crucial limitation with mainstream deep learning frameworks such as PyTorch and TensorFlow: they do not natively support efficient sparse attention mechanisms like the sliding window and global attention patterns. As such, the authors of the Longformer paper had to develop a custom CUDA kernel to efficiently implement these attention patterns. Following this approach, I explored the implementation of a custom CUDA kernel to overcome bottlenecks when training on longer sequences. The intent behind this custom kernel was to optimize the GPU computations specifically for the sliding window and global attention mechanisms. Traditional GPU implementations struggle with handling the dynamic masking required for the Longformer model, especially when scaling to longer sequences. The custom CUDA kernel was designed to address these issues by optimizing memory access patterns and minimizing redundant computations in the attention mechasnism.

I leveraged the publicly available Longformer repository from the Allen Institute for AI as a foundation for training the 1024 sequence-length dataset. Here, the compiled binary for a custom CUDA kernel implemented in Tensor Virtual Machine (TVM). Unfortunately, despite attempts to implement the custom kernel, challenges in kernel optimization and debugging prevented the successful execution of the model for the 1024-token sequences. This bottleneck limited the experiments for the largest sequence length, but it provided valuable insights into the potential scalability challenges of implementing custom attention mechanisms at such a large scale. This leaves room for further exploration of more advanced kernel-level optimizations in future work.

Metrics

To evaluate, the elapsed times for each epoch, total training time, cross entropy loss, and perplexity were tracked.

Elapsed Time per Epoch: The time taken to complete each epoch was tracked to assess the efficiency of the models. Time was recorded for each individual epoch as well as for the overall training run. This data was outputted to a CSV file for analysis.

Total Training Time: Total elapsed time for each training run was recorded, allowing for comparisons of time efficiency across different sequence lengths and models.

Cross Entropy Loss: Cross-entropy loss was tracked as the primary performance metric for the models. This loss measures how well the models are performing in predicting the next token in the sequence. A lower cross-entropy loss indicates better performance.

Perplexity: Perplexity, which is derived from cross-entropy loss, was also tracked. Perplexity provides a measure of how uncertain the model is when predicting the next token. A lower perplexity score indicates that the model is making more confident and accurate predictions.

Results

Elapsed Time

Time Usage Summary		
Model	**Sequence Length**	**Average Time per Epoch (s)**
Baseline	256	2007.8618578910800
Longformer	256	928.546985244751
Baseline	512	692.0159687995910
Longformer	512	922.9168248653410
Baseline	1024	1229.073876810070

Table 1: Time Usage Summary

For the lowest sequence length of 256, both the baseline and Longformer models has similar training times per epoch. The Baseline model had a mean training time per epoch of about **705.7 s** and the Longformer model has a mean training time per epoch of about **902.73 s**. The Longformer model required more time per epoch compared to the Baseline model, likely due to the complexity of the sliding window and global attention mechanisms.

For experiments on the dataset with sequence length 512, Longformer performed more efficiently than the baseline. Longformer had an average training time per epoch of about **1202.82 s**. In comparison, the baseline model had an average training time per epoch of about **1394.91 s**. In this set of experiments on the longer sequence length, the Longformer model performed more efficiently in terms of time per epoch than the Baseline model.

For sequence length of 1024, the Longformer model was unable to be evaluated. The Baseline model however exhibited a significant increase in time per epoch of about 2246.53 s.

These results suggest that while the Longformer model's custom attention mechanism introduces computational overhead for shorter sequences, it scales better as sequence lengths increase, reducing the relative difference in time per epoch.

Memory

Memory Usage		
Model	**Sequence Length**	**Average Memory Usage (MB)**
Baseline	256	79.42109375
Longformer	256	333.665625
Baseline	512	-104.21171875
Longformer	512	535.809375
Baseline	1024	79.56640625

Table 2: Memory Usage Summary

For training on sequences of length 256, The baseline model had an average memory usage of about **2614.66 MB** while the Longformer model has an average memory usage of about **2556.63 MB**. Despite its more complex attention mechanism, the Longformer model demonstrated lower average memory usage for the 256-token sequence.

For experiments on the dataset with sequence length 512, the baseline model had an average memory usage of about **2676.92 MB** while Longformer had an average memory usage of about **2712.45 MB**. The Longformer model's memory usage exceeded that of the Baseline model. This can be attribute to the increased complexity in handling larger sequences with its more complex attention mechanism.

Cross Entropy Loss and Perplexity

Model	**Sequence Length**	**Initial Loss**	**Initial Perplexity**
Baseline	256	9.20900535583496	9986.658203125
Longformer	256	9.209095001220700	9987.5537109375
Baseline	512	9.213370323181150	10030.345703125
Longformer	512	9.216456413269050	10061.3486328125
Baseline	1024	9.210432052612310	10000.9169921875

Table 3: Cross entropy loss and Perplexity metrics

Both baseline and Longformer models show similar loss and perplexity metrics at the beginning of the training. The loss values start around 9.21, with high perplexity values (about 9987 for the Longformer and about 9986 for the baseline). The metrics remain fairly consistent across different epochs for both models, with loss values starting around 9.21 and perplexity around 10030 for the Baseline and 10061 for the Longformer. The Baseline model's initial loss is slightly higher at 9.21, with a perplexity starting at around 10000. These metrics show similar performance for both models across different sequence lengths. The Baseline model however showed slower convergence at longer sequence lengths, underscoring the need for more scalable attention mechanisms like those in the Longformer model for larger text sequence

Conclusions

The Longformer model demonstrates its potential for handling longer sequences more efficiently than the Baseline model, particularly in scaling time requirements in comparisons of the results between training on datasets of sequence lengths 256 and 512. However, this efficiency comes with greater variability in memory usage. This can be attributed to contrainsts in hardware and resources, such as limited RAM. Despite these fluctuations, for tasks involving long sequences the Longformer model's sliding window and global attention mechanisms do offer a promising approach, although further optimizations could improve its performance, especially at larger scales and for average (non-industrial) compute requirements.

Impacts and Future Work

The experiments faced limitations related to hardware resources, such as memory constraints and available compute power. These limitations hindered the ability to fully explore the model's performance on larger sequence lengths, like 1024 tokens, and prevented testing of even longer sequences that could further showcase and contrast the linear scalability of the Longformer attention mechanism compared to the quadratic scalability of the baseline transformer.

If computational and memory limitations were not a concern, future experiments would focus on training the Longformer model on even longer sequences, such as 2048 or 4096 tokens. This would allow for a more pronounced comparison between the Baseline and Longformer models, particularly in their ability to capture long-range dependencies and their scalability across longer text inputs. Experimenting with the sliding window size and adding dilated attention would also provide further insights into optimizing the model's efficiency.
Additionally, successfully implementing a custom CUDA kernel could significantly reduce overhead in memory and time. This would allow for more scalable and efficient training on even longer sequences, making the model more accessible to environments with average computational resources.
Beyond text-based models, I am also interested in exploring the application of the Longformer's attention mechanisms in the vision domain, similar to the ViT or Swin Transformer architecture. A *Vision Longformer* leverages sliding window attention and global attention for tasks like image classification or object detection.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023, August 2). Attention is all you need. arXiv.org. https://arxiv.org/abs/1706.03762
2. Beltagy, I., Peters, M. E., & Cohan, A. (2020, December 2). *Longformer: The long-document transformer*. arXiv.org. https://arxiv.org/abs/2004.05150
3. Allen Institute for AI. (2020, October 26). *Longformer: The long-document transformer.* GitHub. https://github.com/alπlenai/longformer