Near-Infinite Sub-Quadratic Convolutional Attention

Stanford CS224N Custom Project

James Poetzscher
Department of Computer Science
Stanford University
jamesp25@stanford.edu

Abstract

Multi-head self-attention is an incredibly powerful and expressive architecture, however its quadratic nature renders it impractical for longer context-lengths which are increasingly in-demand. We introduce a novel sub-quadratic auto-regressive attention architecture that can scale to arbitrarily long-sequence lengths in time linear in the sequence length. Specifically, we combine sliding-window attention with one, or potentially multiple, convolutions to compress the sequence length (and potentially self-attention layers on the compressed sequence before further convolutions) before the crucial cross-attention layer between the output of our sliding window attention and the compressed sequence. Our architecture is designed for auto-regressive tasks. We also introduce a second approach, which is a more complex extension of the first, that while infeasible to evaluate ourselves is likely the better architecture for especially long sequence lengths. We evaluate our model on a next-character prediction task using the Shakespeare dataset and find that our model largely matches and in some cases exceeds the performance of standard self-attention. Our results suggest our architecture is a viable alternative to standard multi-head self-attention, and further evaluation with larger models and especially larger context lengths would be a logical next step.

1 Key Information to include

- Mentor: Soumya Chatterjee

2 Introduction

The massive improvements in deep learning over the past 7 years have been driven in large part by the Transformer architecture introduced by Vaswani et al. (2023). The Transformer architecture benefits in large part due to the expressiveness enabled by the multi-headed self-attention layer. Specifically, in a sequence of length N, all N tokens can attend to each other, enabling tokens to identify global relationships between themselves and other important tokens Child et al. (2019). This expressiveness however, incurs significant computational cost, as requiring every token in a sequence to attend to all tokens in the sequence means the multi-headed self-attention computation is quadratic in sequence length. While for most reasonable context lengths this remains relatively computationally efficient, for particularly long contexts, which have become increasingly desired for their ability to fully capture larger documents, full self-attention is too computationally expensive to be currently viable.

Numerous alternatives to the default multi-head self-attention mechanism have since been proposed, including linear attention mechanisms, sparse attention mechanisms and a myriad of other alternatives. Currently, however, the default attention mechanism remains in use across most current models, with alternatives proving unable to match the expressiveness and output quality of the baseline.

We introduce a novel attention architecture that combines sliding window attention with a convolution to reduce the initial sequence size before using the output of the sliding window attention layer to perform cross-attention to query the compressed sequence and

3 Related Work

Numerous alternatives to the basic self-attention mechanism have since been proposed. These alternatives largely fall into one of two groups. The first approach is to do away entirely with the Softmax function and instead approximating it through a kernel function such as an ELU or a Taylor polynomial which enables the computation to scale linearly in sequence length. These methods have generally been unable to match the performance of Transformers, particularly as models scale up Arora et al. (2024); Katharopoulos et al. (2020). One rationale for why linear attention mechanisms perform relatively poorly is that the approximations are simply too imprecise, particularly outside of a tight range and the probability distribution which the Softmax generates isn't accurately captured by the linear approximation (Arora et al., 2024). Moreover, while the linear attention mechanisms are, in theory, much faster than quadratic self-attention, recent improvements in the algorithmic implementation of self-attention, notably the Flash-attention architecture, which don't change the underlying attention computations in any way but rather compute the same attention mechanism in a more efficient manner have meant that proposed linear models, and other attention variants which don't benefit from these improvements to quadratic attention's computational efficiency, lose much of their relative run-time superiority in practice (Arora et al., 2024).

The second broad class of approach to improving the efficiency of attention is to somehow limit the number of tokens each token attends to. These attention alternatives, in many cases known as sparse attention mechanisms, limit the number of tokens each token can attend to, making the matrix representation of the N^2 token to token relationships sparse rather than dense as in standard attention (Child et al., 2019). These sparse matrix alternatives have not yet managed to match the performance of standard self-attention. While in theory, most sparse patterns enable tokens to indirectly communicate, the direct communication in standard attention proves much more powerful and is better able to capture crucial global relationships that sparse attention fails to capture. Moreover, many of these sparse attention alternatives, such as BigBird, and Longformer—which uses sliding window attention, a technique we also implement—are not designed for auto-regressive contexts which is a more difficult task given the challenges involved in masking (Zaheer et al., 2021; Beltagy et al., 2020).

The architecture that seems to most resemble our own is the Hierarchical Transformer which computes a series of fully quadratic attention layers before compressing the sequence through pooling, computing self-attention on the hierarchy of compressed sequences and then upsampling until eventually reaching the full size of the input. (Nawrot et al., 2022). The primary overlap is in the use of compressed sequences—parituclarly with our 2nd approach which, like theirs, enables multiple convolutions. Our approach largely differs, from theirs however, in that we avoid full-self attention all together, both in the full sequence, and, in approach two, in the multiple compressed sequences, and use sliding window-attention throughout. Moreover, we use cross-attention to connect the sliding window output and the most compressed input rather than a series of upsampling layers. Their architecture also largely consists of full length layers with the compression interspersed in only a single layer. Our architecture, meanwhile, makes the layer itself a combination of the sliding window attention and the compression plus cross-attention ensuring that every layer is efficient.

4 Approach

We propose a novel architecture that enables sub-quadratic attention with a superior global focus compared to existing architectures, even better performance given equivalent context length compared to standard transformers and, crucially, attention over an arbitrarily large sequence of tokens. We manage all of this in a decoder model, meant for auto-regressive tasks—a significantly harder challenge given the difficulty of masking a compressed sequence.

We focus exclusively on modifying the self-attention layer, initially proposed in Attention is All You Need (Vaswani et al., 2017). Specifically, the input to a standard self-attention layer is a tensor of shape B, N, D, representing batch size, sequence length, and embedding dimension. Rather than simply taking the input, generating Query, Key and Value vectors from it, and then computing standard self-attention across the entire sequence, which has run-time exponential in sequence length, we instead choose a window size (M) which should be smaller than the sequence length if trying to achieve more efficient run-time than standard self-attention. We first compute sliding-window attention over the length of the input sequence, generating a new output layer in which every token in

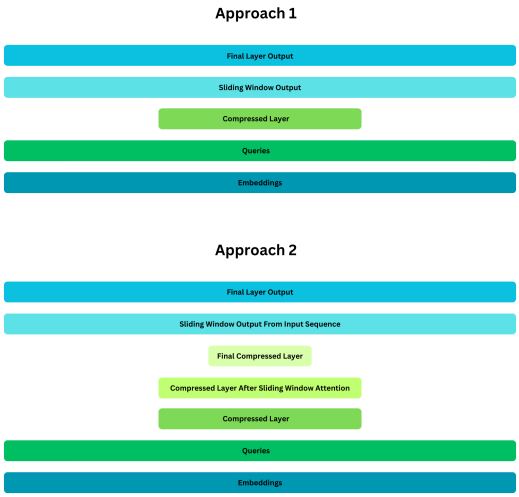


Figure 1: The two approaches we introduce as novel attention architectures

the output has the context of the M previous tokens (as opposed to in a non auto-regressive setting such as an encoder where tokens can have bi-directional context). However, while sliding-window attention is a popular method, used in existing large models like Mistral (citation needed), its approach struggles with capturing relationships over distances in the input sequence larger than M. Sliding window attention on it's own increases the amount of past context a given token can "see" each layer by the window size. This, ultimately, is not a solution to true global attention over particularly long sequences where increasing the past context of a token by window size each layer leads to long-term relationships taking incredibly long-paths and losing core information and also extending the model depth-wise by a factor of N/M. We, instead, view sliding window attention as simply the best way to take attention over a fixed-window size, as opposed to a means to get strong sub-quadratic results on global attention tasks. To handle global attention, we pair sliding window attention with the following novel architecture. We break the input sequence into groups of M tokens, and for each group, take a convolution over the entire size M resulting in an output size of 1, with the embedding dimension unchanged. Doing this for all N/M groups of M tokens in the sequence results in a new layer of N/M tokens. Here we outline two approaches. The first approach is to limit the number of convolutions—that is the number of times we compress the current sequence to 1. We simply take the input sequence, compute sliding window attention over it with a window size of M, then take N/M convolutions which compress groups of size M down to groups of size 1 and finally for the compressed sequence of size N/M we then compute cross-attention with the output of the sliding window-attention. Specifically, the sliding window attention output which we just computed as size equivalent to that of the input sequence. The tokens in this layer are used to query the entire compressed N/M sequence from which we derive the keys and values. In this approach, we are essentially computing global attention but over a sequence that is compressed. This compression, which occurs through a convolution, naturally loses some of the local resolution that self-attention over the entire N sequence length would capture, but this is where the sliding-window attention comes in; sliding window-attention over window M « N enables us to capture local resolution in great detail. Then for global attention, there are, in any given sequence of M tokens, only going to be a select few which are relevant globally. The goal is for the convolution to intelligently capture tokens which may be relevant to global attention, a task it is well suited for, unlike actually capturing global dependencies. In this way, the convolution acts as a filtering mechanism. The compressed sequence of size N/M should, then, contain the globally relevant tokens. We then use the size N output from the sliding window attention computation, which captures the input sequence's local dependencies, and have it query the keys and values generated from the N/M compressed sequence. We mask this cross-attention computation, such that tokens in positions which correspond to the Xth group of M tokens can only attend to tokens N/M that were generated from convolutions on the 1st to the (X-1)

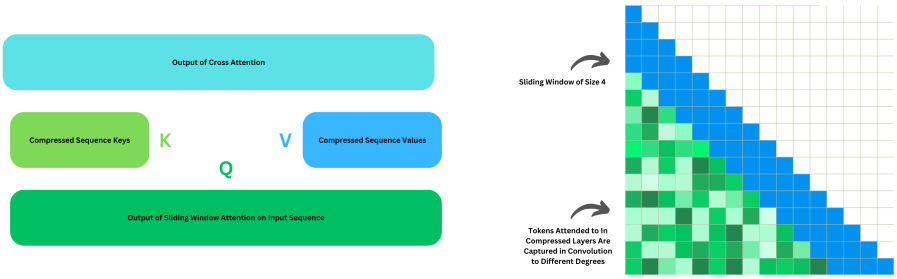


Figure 2: The details of the cross-attention mechanism between the compressed sequence and the output from the sliding attention later, and the matrix representation of what tokens are attended to in the sliding layer at each position (blue) and which tokens are attended to with differing strengths in the cross-attention with the compressed sequence (green)

group of M tokens. This means the first M group doesn't attend to any tokens in the N/M group and this is not an issue as in standard attention they would attend, only to tokens within their M group, which of course they still manage to do through the sliding window attention.

The second approach, largely resembles the first approach, but allows for attention over even longer input sequences. Specifically, rather than only taking one convolution as described above, we now can, if necessary, take a convolution then take sliding window attention over the compressed sequence capturing more of the global dependencies (but not all) from the original input sequence and using the output of sliding window attention over this compressed input to compute a new convolution over. We repeat this process until we reach a compressed sequence of desired size over which we finally compute our cross-attention with the output from the initial sliding window attention over the input sequence. We believe this method represents an ideal way to handle enormously large sequences. If we were to simply take a series of convolutions over the input sequence to compress the sequence to a satisfactory size over which we could then cross-attend, the relevant global information the convolutions are intended to capture from their group of M may get lost as we attempt to compress the information down by so many orders of magnitude. Instead, after each convolution, we want to understand precisely which tokens from the new compressed layer are actually important and which aren't—the convolution ideally captures all possibly relevant tokens but many of these may not in fact be relevant given the larger context. Incorporating sliding window attention over the newly compressed sequence enables us to extract relevant information from the compressed sequence such that the subsequent convolution doesn't blindly compress down information but is instead only compressing a pre-filtered group, still of size M. This approach adds depth to the model instead of width, however, it adds depth logarithmically as a function of the convolution group size M. Given M also represents the window size, it should still be quite large and for virtually every practical scenario there should only be one or, in very rare cases, two convolutions actually required to compress the input down to a manageable size. Theoretically, however, this second approach can scale to incredibly long contexts—and as N grows relative to M our architecture approaches run-time linear in N. We only evaluate our first approach as for any reasonably sized context length and fixed window size—which includes anything we could reasonably train—the second approach will terminate after one compression and the two approaches are therefore the same. However, we recommend that larger models adopt the second approach as it can handle all inputs that the first approach can and simply introduces more flexibility for particularly large contexts.

We design our architectures to function in auto-regressive contexts such as next token prediction in the context of language modeling. This introduces a number of challenges into designing sub-quadratic architectures, and as mentioned above, many existing architectures are designed for encoders and other non-auto-regressive architectures. Specifically, the challenge arises in the convolutional step to compress our input. We must ensure that in the case of approach 1, where we perform one convolution and then a cross-attention with the output of the sliding window attention layer, that the sliding window output which queries the compressed sequence can't access tokens that may contain information involving tokens which come after them in the sequential ordering. The final M tokens in the input sequence will not be attended to in the cross-attention layer because for all but the final token, a convolution over this window will contain tokens which come after in the sequential ordering.

We remove this final M sized window from our input sequence before computing the convolution. Then, the first M tokens in the output layer of our sliding attention also cannot attend to any part of the compressed sequence so we ignore these tokens and create a custom mask, which, rather than featuring all 1s in the lower triangle as is common in auto-regressive masks, instead repeats each row N/M times giving multiple rows with only a singular 1 in the left most position followed by multiple rows with two 1s and so on.

As an example to highlight how our model architecture functions we imagine a sequence the self-att with additional sliding window local attention to compress the input down to a desired size. For a sequence consisting of 1 Trillion input tokens, standard quadratic attention would have a run time of $O(10^{24})$ not even accounting for dimension size. In our proposed approaches however, we achieve significant run-time improvements. For a fixed window size (M) of 1,000, our first approach would compute sliding window attention over the entire sequence—which has run time $O(10^{24})$ (again, ignoring dimension size), before then taking 1 billion convolutions over the M sized groups resulting in a new sequence of size 1 billion which we compute cross-attention with giving run time of $O(10^{21})$—a 1,000x speedup. Our second approach, particularly suited for extreme cases like these, achieves even more drastic performance improvements. We would, for the first layer, compute sliding window attention over windows of 1,000 giving this run time of $O(10^{24})$. Then we compute 1 billion convolutions and repeat the sliding window attention, whose run time is now only $O(10^{12})$. We repeat twice getting sequences of length 1 million and finally length 1,000 over which we then compute the cross-attention whose run time is equivalent to that of the initial sequences sliding window attention—$O(10^{15})$. This is on the order of a 10^9 speed-up over standard attention. We also suggest that as sequences get incredibly long, the ability to distribute attention weights (logits from the softmax) over all tokens in the sequence becomes increasingly difficult, and convolutions that intelligently shrink the input sequence very well could actually improve performance.

Such sequence lengths are obviously far beyond anything we could actually test and evaluate but we highlight that our approach should theoretically scale to such sequence lengths—which may be required in actual applied scenarios eventually—better than existing architectures.

The reason we recommend using the same size sliding window size M as the convolutional group size M is because this ensures every token can attend to, in either the first layer or the final convolutional layer, some representation of every token it would attend to in standard attention. The final token in our convolution is masked, as even tokens, except for the very last one, in the last M group can't be sure that the convolution of their entire M group does not contain tokens to the right of them within their group. However, these tokens are still attended to precisely as in standard attention through the sliding window attention

Our approach prioritizes expressiveness, as opposed to computational efficiency at all cost. For the vast majority of input sequences, a reasonably chosen fixed window size will essentially compute global attention over the entire input, thereby only adding to the expressiveness of the model through the cross-attention and convolutional layer. In scenarios where context length is particularly long, our model then enables efficiency gains while maintaining incredibly strong expressiveness and performance—perhaps actually achieving superior performance to standard transformers.

5 Experiments

5.1 Data

We evaluate our models' performance on the Shakespeare dataset. The dataset is roughly 1MB—40,000 lines—of text found in Shakespeare's writing. The text is processed using a character level tokenizer which has a vocabulary size of 65. We use Andrej Karpathy's implementation of a decoder-only model trained on this Shakespeare corpus for all data processing as well as for the baseline model and the training implementation Karpathy (2023).

5.2 Evaluation method

We evaluate our model and various different hyperparameter trials against a baseline decoder-only model derived from Karpathy's implementation. We use Cross-Entropy as our loss function, which produces a loss when we predict the next character incorrectly. We evaluate our architecture against the baseline through comparing validation loss every 500 iterations, and we determine our validation

loss by averaging loss over 150 iterations so as to reduce noise. We also generate a chunk of text—generated one character at a time, as is our training task—following each training run, enabling us to qualitatively evaluate our model performance and ensure that the loss coincides with the quality of output.

5.3 Experimental details

The focus of our experiments was to evaluate our architecture described in approach 1 against a default architecture for which we again used Karpathy's implementation of the decoder-only architecture. Our architecture remains identical to the default architecture in all areas besides the attention heads such that we exclusively evaluate the relative performance of our attention variant against standard multi-head self-attention. We specifically modify our fixed window size in the experiments to determine how our model performs relative to the baseline, and how this is governed by the fixed window size—the key variable at play.

We run two broad classes of experiment. The first trains a smaller model of roughly 275,000 parameters over 5,000 iterations. We compare the baseline self-attention mechanism using a context length of 1,024 against two models using our own attention architecture. One uses a fixed window size of 256, the other uses a fixed window size of 16. For this experiment we use a learning rate of 1e-3, a batch size of 16, an embedding dimension of 64, 4 attention heads and 4 layers.

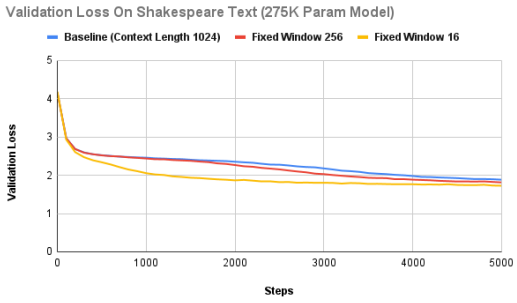
In our second experiment we compare the relative performance of our architecture to the baseline using a much larger model. We train over 2,500 steps. We had to shorten the total context limit given compute constraints. The baseline self-attention mechanism uses a context length of 256. We again train two models using our own architecture to compare. The first uses a fixed window size of 64, the second uses a fixed window size of 16. For this experiment we use a learning rate of 3e-4, a batch size of 64, an embedding dimension of 384, 6 attention heads and 6 layers.

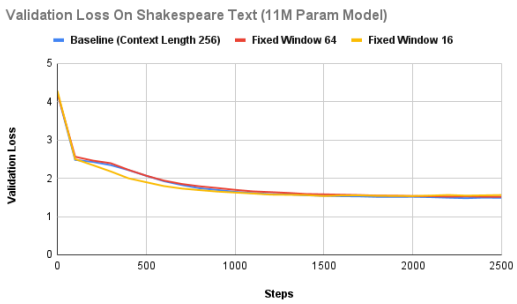
We use the Adam Optimizer across all experiments. We trained our model on a single NVIDIA A100.

5.4 Results

We found that our architecture performed very well relative to the baseline attention in our experiments.

We highlight our results below:





Specifically, we find that for the smaller model, the shortest fixed window size outperforms our larger fixed window size and both outperform the baseline attention mechanism. For the larger model, we see that the baseline model fractionally outperforms our largest fixed window size of 64, which itself slightly outperforms our smallest fixed window size of 16. The results on the larger model are more in-line with our expectations, while the results on the smaller window likely suggest that the smaller fixed window size works well with smaller models unable to represent as much global information.

6 Analysis

A key takeaway with our architecture is that the relationship between the fixed window size and the size of the compressed sequence is a key factor in our model's performance remaining relatively consistent across the various window sizes and context lengths. Given a sequence of length N and a fixed window size of M, our convolutional layer compresses our sequence of length N to one of length $\frac{N}{M}$. For example, with a context length of 1,024 and a fixed window size of 256, our compressed sequence is of size 4. Whereas, with a shorter fixed window size of 32, our compressed sequence is of size 32. This relationship between fixed window size and compressed size is such that a larger fixed window size, which enables the sliding window attention to attend to more past context, has a much smaller compressed sequence (that is, the sequence is more compressed). Meanwhile, shorter fixed window sizes, where the model is losing out on more past context, have larger compressed sequences which should retain more information from the input sequence to be carried into the cross-attention layer. Essentially, whatever information is lost in the sliding window attention—whether this is a lot of information or very little—is made up proportionally by the size of the compressed sequence.

7 Conclusion

We introduced a novel sub-quadratic attention architecture that largely matched or exceeded the performance of baseline-attention. Our results highlight the potential utility of our architecture, particularly in large models meant to handle especially long context-lengths. We also suspect applications of this architecture will work well across multi-modal inputs, particularly vision. The inductive biases injected into the model architecture are fairly weak—the convolutional layers do impose a locality bias but this is counteracted by the global attention across the lower resolution layers which preserves strong global attention. The weak bias towards locality generally holds across a wide range of tasks and multimodal inputs. Language, of course, has a local bias while still maintaining a significant demand for global attention—it's not surprising then, that our model performed extremely well on language tasks. Vision, however, enforces local biases to an even greater extent, explaining the continued success of CNNs relative to transformers—unless a transformer is incredibly large, it tends to focus all attention layers globally ignoring the crucial local relationships present in image data Raghu et al. (2022). Combining convolutions with self-attention in the manner we present will lead to greater focus on local relationships than default transformers while maintaining the global attention of transformers, which obviously surpasses what is possible in CNNs.

Ultimately, however, we were limited by compute, meaning we were unable to test much larger models, and crucially, unable to test over longer sequence lengths. This represents a natural next step for determining the viability of our architecture.

References

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. 2024. Simple linear attention language models balance the recall-throughput tradeoff.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

Andrej Karpathy. 2023. Nanogpt. GitHub.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models.

Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. 2022. Do vision transformers see like convolutional neural networks?

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2021. Big bird: Transformers for longer sequences.