Scaleformer : an scalable transformer with linear complexity and relative positional encoding

Benoit Favier[1] and Walter Dal'Maz Silva[2]

[1]Phealing
[2]Unaffiliated

September 26, 2021

Abstract

1 Introduction

Vaswani et al. [11] introduced the transformers, a simpler network to model sequence-to-sequence translations tackling new heights of problem complexity. The improvements were due to the non sequential nature of its training process which allowed better parallelization and thus fast training of big models, and due to improved long term dependencies thanks to the intrinsic good gradient flows of the attention mechanism. However the original transformer presented some drawbacks. It has a quadratic complexity with sequences length, what represents a major drawback to its scalability. The positional encoding induce a different initial embedding for the same group of words at different position in the sentence, which is detrimental for generalization. The absolute positional encoding is also detrimental for the extrapolation to sequences longer than training sequences - even for the initially proposed sinusoidal positional encoding.

Ever since, significant research effort have been made to alleviate these problems. Several strategies have been proposed to reduce the quadratic complexity with sequence length, and the relative positional encoding further improved the performances of the model. These improvements have opened the path to the application of transformers to image analysis where scalability and positional invariance are of upmost importance. However most proposed architecture we are aware of only alleviated parts of the issues, by being incompatible with relative positional encoding, having no scheme for masked attention - thus being restrained to encoder-only models, or being complex to use/implement - needing custom operations programmed in CUDA or introducing stochastic methods.

In the present work we propose a complete encoder-decoder transformer model that scales linearly with large sequence lengths, by putting together ideas developped across several works [2, 4, 6, 9–11]. It remains compatible with relative positional encoding, and can be implemented with usual functions of neural network frameworks without requiring custom CUDA code.

2 Background

2.1 The original multi-head attention mechanism

The scaled dot-product attention proposed by Vaswani et al. [11] transforms the vectorial embedding $\vec{Y_i}$ of a token as a function of a sequence of other embedding $\vec{X_j}$. Where $\vec{Y_i}$ and $\vec{X_j}$ are all vectors of size D. A key $\vec{K_j}$ and value $\vec{V_j}$ are attributed to each vector $\vec{X_j}$, and query $\vec{Q_i}$ is attributed to $\vec{Y_i}$. The vectors are obtained by linear projection from dimension D to d using three matrices of learnable parameters. The new vector $\vec{y_i'}$ is a weighted sum of the $\vec{V_j}$. The weights are scores of matching between the query $\vec{Q_i}$ and the keys $\vec{K_j}$, calculated as the dot product between the two vectors. They are also *softmaxed* to sum up to 1. The transformation of L_Q vectors $\vec{Y_i}$ as a function of L_K vectors $\vec{X_j}$ can be efficiently computed with matrix multiplications:

$$A = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d}}\right) \times V \tag{1}$$

With Q a matrix of shape (L_Q, d), K a matrix of shape (L_K, d) and V a matrix of shape (L_K, d). The $\sqrt{d}$ at the denominator is a scaling factor used to avoid saturation in the exponential terms of the softmax function.

The multi-head attention performs h different projections into spaces of dimension $d = D/h$. The resulting vector $\vec{Y}_i'$ is the concatenation of the h vectors $\vec{y}_i'$ obtained. Thus the embedding dimension is preserved. Using multiple heads was found beneficial by the authors over using a single head of dimension $d = D$.

During training, the cross entropy of the n^{th} predicted token is calculated assuming all previous tokens have been generated correctly. This enables to parallelize training completely without need for recurrence. However as the n^{th} token should not depend of the following tokens, the cells in the upper right corner of the score matrix are set to $-\infty$ such that after the softmax they are equal to 0, and the rows still sums up to 1.

2.2 Improving Transformer scalability with sequence length

The original attention mechanism requires the computation of a score matrix $Q \times K^T$ of shape (L_Q, L_K), with complexity $O(L_Q d L_K)$. If the query and key sequence lengths are multiplied by two, then the memory used and computation time are multiplied by 4. To improve the scalability of the transformer with sequence length, several axis of research have been explored.

Kitaev et al. [7] proposed the Reformer's architecture, which uses an hash-bucketting algorithm to reduce the complexity of the original multi head attention operation from $O(L^2)$ to $O(L \log(L))$.

Dai et al. [3] proposed the Transformer-XL's architecture, which cuts the sequence in segments of length L. The model predicts each stage of the current segment as a function of the previous and current segment. All the segments are computed sequentially with a recurrence mechanism. The complexity is linear with sequence length, but the computation cannot be completely parallelized due to the recurrence mechanism, although more than a RNN, as segments can be computed in one go.

Other publications explored using a sparse attention matrix, such as the Longformer by Beltagy et al. [1] and the Big Bird model by Zaheer et al. [13]. As each token attends to a fixed number of all other tokens, the scalability is improved. These sparse attention models however require custom operations implemented in CUDA.

Some other works propose to modify the attention mechanism to be compatible with linear complexity. The Linformer by Wang et al. [12] projects the key and values onto a smaller sequence length dimension with matrix multiplication. It cannot however generalize to sequences longer than during training, as the weights of the projection for such tokens would be undefined.

Shen et al. [10] proposed to replace the softmax attention score. $A = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d}}\right) \times V$ is changed into $A = \rho(Q) \times \rho(K)^T \times V$. With ρ the softmax function along the embedding dimension. Thanks to matrix multiplication commutativity, the order of the operations can be chosen. If Q, K and V are of shape (L_Q, d), (L_K, d) and (L_K, d) respectively, the complexity of $(\phi(Q) \times \phi(K)^T) \times V$ is $O(L_Q \times d \times L_K)$ whereas the complexity of $\phi(Q) \times (\phi(K)^T \times V)$ is $O\left(max(L_Q, L_k) \times d^2\right)$. The right-side-first operation is linear in complexity with sequence length. The shape of the intermediate result matrix is also changed, allowing to scale the better in memory requirements as well. The original *softmaxed* attention score matrix was giving rows of positive scores that sum to 1. With this change the elements of the score matrix remain positive as $\phi(Q)$ and $\phi(K)^T$ are matrices of positive values, but the rows of the score matrix does not sum up to 1. This work also does not give a linear complexity formulation for masked attention. If the right-side-first scheme is adopted, the attention score matrix $\phi(Q) \times \phi(K)^T$ is never explicitly computed, and can't be masked.

Building on this idea of commutative attention function proposed by [10], Katharopoulos et al. [6] introduced their kernerlized attention function as:

$$A = \frac{\phi(Q) \times \phi(K)^T}{\sum_j \left(\phi(Q) \times \phi(K)^T\right)} \times V \tag{2}$$

The function ϕ is applied element-wise and can be any positive function, for example $\phi(x) = elu(x) + 1$. This attention is row-wise normalized so that all rows of the score matrix are sets of positive weights adding up to one. This preserves the objective of the original softmaxed attention scores, while allowing to perform operations in an optimal order.

The Performer by Choromanski et al. [2] exploits the same idea of a kernelized attention introduced by Katharopoulos et al. [6], with an algorithm that betters approximate softmaxed attention. Most importantly they also give in annex a prefix sum algorithm to perform operations in the right-side-first order while giving the same result as masked left-side-first operation.

Although the author did not specify how to implement it, the only implementations we found of this operation requires custom CUDA code. In this work we will give an implementation of the right-side-first masked operation, with usual functions from neural network frameworks, that remains linear in complexity.

2.3 Alternatives to absolute positional encoding

The original multi-head attention operation introduced by Vaswani et al. [11] was intrinsically invariant by token order permutation. As token position was an important information for sequence to sequence models, they encoded the global position of each token in their embedding. Since then, some modified attention mechanisms, that depend on relative tokens position, have been proposed.

Shaw et al. [9] explored modifying the attention mechanism so that it depends on the relative distance between tokens. A second score matrix that is function of the query and the query/key relative distance is added to the original score matrix that only depends on query/key vector representation. $A = softmax\left(\frac{Q \times K^T}{\sqrt{d}}\right) \times V$ becomes $A = \left(\frac{Q \times K^T + S_{rel}}{\sqrt{d}}\right) \times V$ with S_{rel} of shape (L_Q, L_K) defined as $S_{rel ij} = \vec{Q}_i \cdot \vec{RP}_{clip(i-j,-k,k)}$. Where k is the attention horizon length and $\vec{RP}_n$ is one of $2k+1$ relative positional embedding, vectors of size d. Shaw et al. [9] and Huang et al. [5] observed that introducing this attention scheme improved performances. The naive calculation of this term however has a complexity of $O(L_Q L_K d)$. No algorithm was provided to linearize the complexity.

More recently Liutkus et al. [8] gives a stochastic positional encoding that is linear in complexity with regards to sequence length. However the implementation is complex and its stochastic nature requires that the operations be repeated several times in parallel.

Horn et al. [4] noted that the term $S^{rel} \times V$ can be computed with linear complexity for the case where $RP_{-k} = RP_k$. However this is restraining as the model can't make the difference between tokens before the attention horizon or after.

In this work we will show that the computation of $S^{rel} \times V$ can also be done with linear complexity, without concession.

3 Masked attention implementation

In this section we will detail the prefix sum algorithm proposed by Choromanski et al. [2] for masked kernelized attention, and give an implementation with usual functions of neural network frameworks.

$$A^{masked} = \text{masked}\left(\phi(Q) \times \phi(K^T)\right) \times V \tag{3}$$

In this expression, masked being the operation that set all cells above the diagonal to 0 in a matrix. The naive implementation of this operation has complexity $O(L_Q L_K d)$.

We can change the complexity using the operation proposed by Choromanski et al. [2]. We will derive its formulation here. To start with it, A^{masked} is defined as

$$A^{masked} = S^{masked} \times V \tag{4}$$

which in summation form is expressed as

$$A^{masked}_{ij} = \sum_k V_{kj} \times S^{masked}_{ik} \tag{5}$$

and the elements of S are defined as:

$$S^{masked}_{ik} = \begin{cases} k \leq i & \sum_l \left(\phi(Q)_{il} \times \phi(K)_{kl}\right) \\ \text{otherwise} & 0 \end{cases} \tag{6}$$

Putting these elements together leads to

$$A^{masked}_{ij} = \sum_{k=1}^{i} V_{kj} \times \sum_l \left(\phi(Q)_{il} \times \phi(K)_{kl}\right) \tag{7}$$

which can be reworked as

$$A^{masked}_{ij} = \sum_l \phi(Q)_{il} \times \sum_{k=1}^{i} \left(V_{kj} \times \phi(K)_{kl}\right) \tag{8}$$

In this work we make use of these ideas to implement the calculation of A^{masked} with complexity $O(max(L_Q, L_K) \times d^2)$ without custom GPU code as per the algorithm 1.

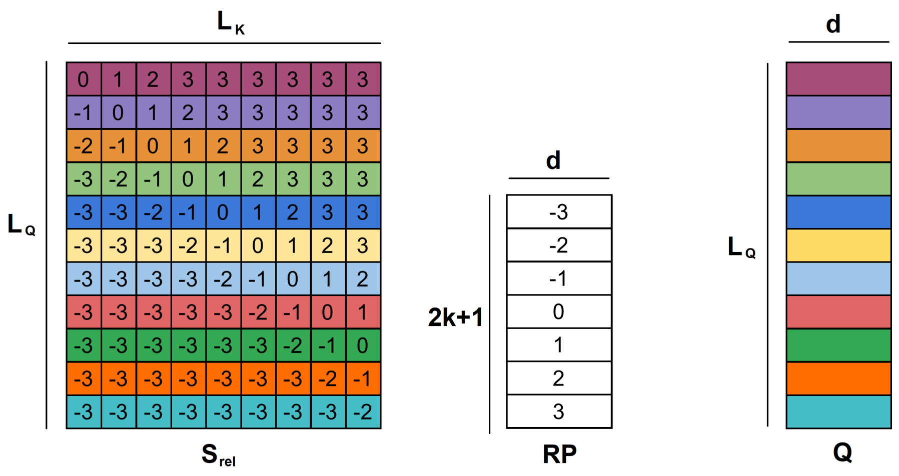


Figure 1: S_{rel} calculation

```
Algorithm 1: calculation of $A^{masked}$ with linear complexity
    Data: $\phi(Q)$, $\phi(K)$, $V$ tensors of shape $(L_Q, d)$, $(L_K, d)$, and $(L_K, d)$
    Result: $A^{masked}$ tensor of shape $(L_Q, d)$
    $\text{UNROLLED}_{kjl} := V_{kj} \times \phi(K)_{kl}$ tensor of shape $(L_K, d, d)$
    $\text{RIGHT} := \mathit{cumsum}\,(\text{UNROLLED}, \text{dim} = 0)$
    if $L_K < L_Q$ then
        $\text{LAST}_{i,j,k} := \mathit{RIGHT}_{L_K - 1, j, k}$ tensor of shape $(L_Q - L_K, d, d)$
        $\text{RIGHT} := \mathit{concatenate}\,(\text{RIGHT}, \text{LAST}, \text{dim} = 0)$
    else if $L_K > L_Q$ then
        $\text{RIGHT} := \text{RIGHT}_{[:L_Q,:,:]}$
    end
    $A^{masked}_{ij} = \sum_k \left(\phi(Q)_{ik} \times \text{RIGHT}_{ikj}\right)$
```

4 Linear complexity RPE implementation

In this section we will detail how the relative positional encoding proposed by Shaw et al. [9] can in fact be computed with linear complexity. The S_{rel} matrix of scores is computed as $S_{relij} = \vec{Q_i} \cdot \vec{RP}_{clip(i-j,-k,k)}$. In figure 1 the colors represent the index of the query and the number the index of the relative position.

Finally we calculate $A_{rel} = S_{rel} \times V$. Each row of the A_{rel} matrix is a

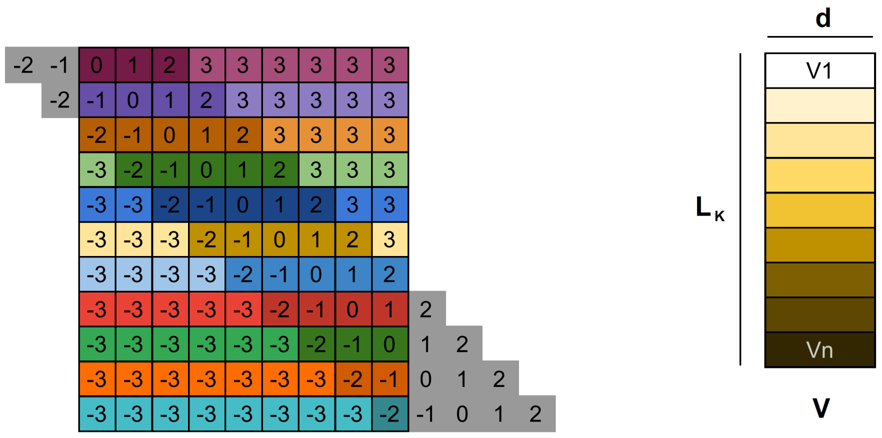


Figure 2: A_{rel} calculation

weighted sum of the value vectors V_i. Each row of S_{rel} is a set of weights.

One can observe in figure 2 that some weights are repeated several times in the S_{rel} matrix. Calculating the whole matrix can be avoided by instead calculating all possible weights only once, with complexity $O\left(L_Q \times d \times (2k+1)\right)$). As illustrated in figure 3, the matrix multiplication can then be decomposed into a sum of three terms: the lower triangle, the diagonal and the upper triangle respectively.

- A set of weights that multiply a accumulated sum of value vectors (complexity $O(max(L_Q, L_K))$)
- An element-wise multiplication between two tensors (complexity $O(L_Q \times (2k-1) \times d)$)
- A set of weights that multiply a accumulated sum of value vectors (complexity $O(max(L_Q, L_K))$).

Thus the attention can in fact be computed with linear complexity if we get rid of the softmax function.

The memory used can be further reduced by observing that the right side of the second element is a strided view of a padded copy of the value matrix. Moving of one cell along the first dimension is equivalent to moving along one cell of the second dimension.

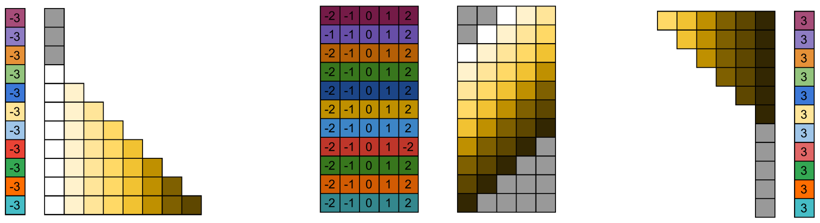


Figure 3: A_{rel} simplified calculation

The calculation can be implemented as follow, using zero-based slice notation for compactness:

- $\phi(Q)$, $\phi(V)$ and RP, are matrices of shape (L_Q, d), (L_K, d) and $(2k+1, d)$ respectively
- The dot product of each query and each relative position embedding is calculated as $weights = \phi(Q) \times RP^T$
- The weights of the horizon before and after are $H_{before} = weights_{[:,0]}$ and $H_{after} = weights_{[:,-1]}$
- The weights of the diagonal are defined as $window = weights_{[:,1:-1]}$. For the masked case, columns on the right of the k^{th} columns are set to 0 $(window_{[:,k+1:]} = 0)$
- The left term is calculated as $left = H_{before} * align(lower, min(max(0, L_Q - k), L_K))$ with $lower$ the concatenation of:
  - a tensor of zeros of shape $(min(k, L_Q), d)$
  - $cumsum(V, dim = 0)$
- The middle term is calculated as $middle = diagonal \odot strided$ with $strided$ the strided view of:
  - a tensor of 0 of shape $(max(0, k-1), d)$
  - the value matrix V
  - a tensor of shape $(max(0, L_Q - L_K), d)$

- The right term is calculated as $right = 0$ for masked case, or for bidirectional case, $right = H_{right} * reverse$, with reverse the concatenation of:
  - a tensor of 0 of shape $(L_Q - max(0, L_K - k), d)$
  - $sum(V_{[inf,sup]}, dim = 0) - cumsum(V_{[inf,sup-1]}, dim = 0)$ with $inf = k - 1$, $sup = min(L_Q + k, L_K)$
- The resulting attention function is $A = left + middle + right$

with:

- $*$ the element-wise multiplication
- $\odot$ the element-wise multiplication followed by sum along second dimension
- all concatenations done along the first dimension

5 Linear Scalable Transformer model

As stated earlier, the proposed model proposes the replacement of the scaled dot-product attention from original Transformer architecture by a kernelized attention with relative positional encoding

The proposed model replaces the scaled-dot-product-attention by a kernelized attention with RPE. Following the observations of Shaw et al. [9] that accumulating absolute positional encoding with relative positional encoding yield no benefits, the positional encoding is also removed - although for some specific applications it might be beneficial to maintain it. The algorithm used to calculate each term is chosen dynamically to occupy the least memory depending on the sequence lengths and embedding dimensions - as memory usage is easier to evaluate precisely than execution time.

In this work we have chosen the following formulation, with $\phi(x) = elu(x) + 1$.

$$A = \frac{\left(\phi(Q) \times \phi(K^T) + S^{rel}\right)}{\sum_j \left(\phi(Q) \times \phi(K^T) + S^{rel}\right)} \times V \tag{9}$$

This is essentially a combination of two terms: the kernelized attention proposed by Katharopoulos et al. [6], and the relative positional encoding proposed by Shaw et al. [9]. The left term is the score matrix of shape (L_Q, L_K), with a denominator which scales all rows so that they sum to 1. For the sake of the implementation, the multiplication must be distributed as:

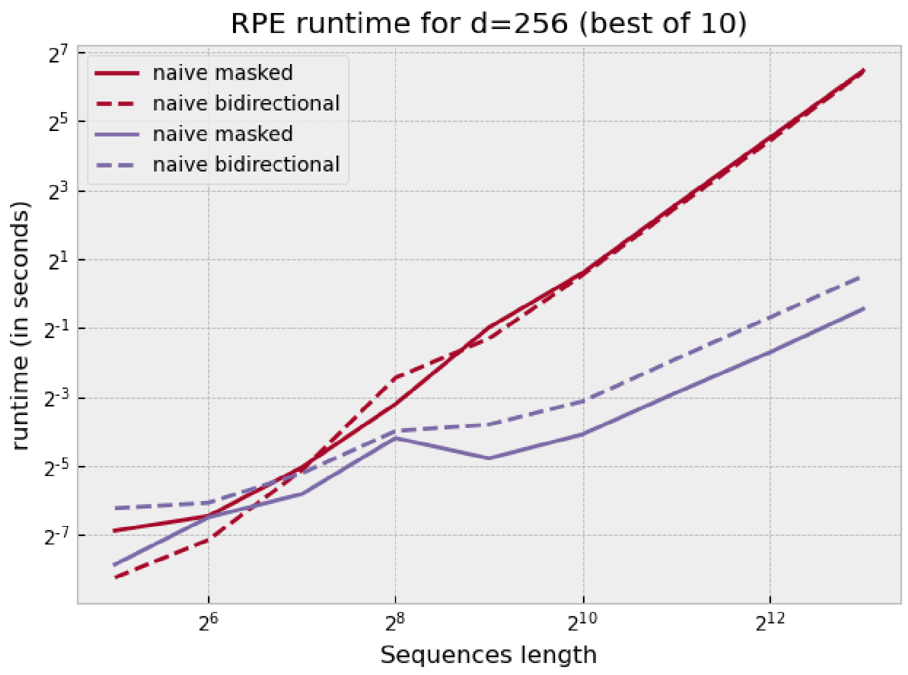
$$A = \frac{\big(\phi(Q) \times \phi(K^T) \times V\big) + \big(S^{rel} \times V\big)}{\sum_j \left(\phi(Q) \times \phi(K^T)\right) + \sum_j \left(S^{rel}\right)} \tag{10}$$

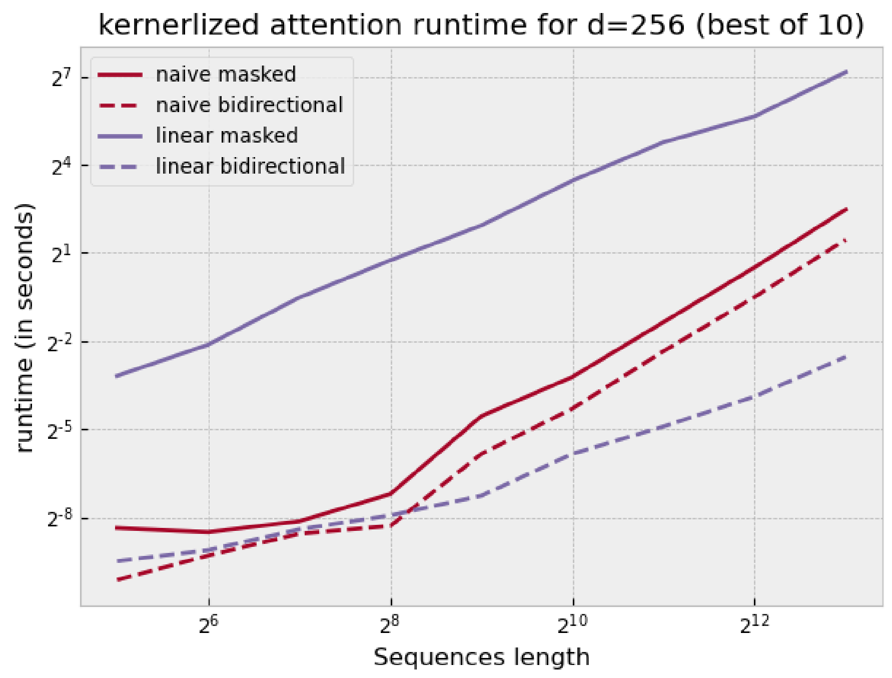
The denominator can be easily calculated by applying the (naive or linear complexity) algorithms with V replaced by a matrix of shape $(L_K, 1)$ full of 1.

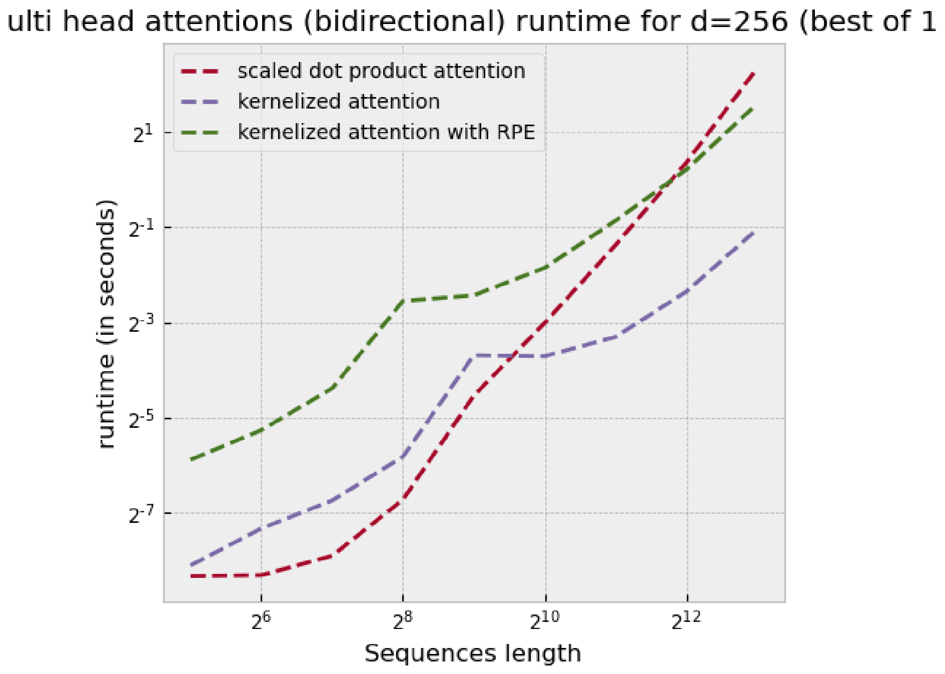
For each case (masked/bidirectional) the algorithm is chosen between naive and linear complexity to occupy the least memory.

- for the masked $Q \times K^T \times V$ term, the memory occupied by the naive algorithm is $L_Q L_K$ while the linear complexity algorithm occupies $d^2 \times max(L_Q, L_K)$
- for the bidirectional $Q \times K^T \times V$ term, the memory occupied by the naive algorithm is $L_Q L_K$ while the linear complexity algorithm occupies d^2
- for the $S_{rel} \times V$ term (masked and bidirectional), the memory occupied by the naive algorithm is $L_Q L_K$ while the linear complexity algorithm occupies $L_Q \times (2k + 1 + 4)$

6 Results


RPE runtime for d=256 (best of 10)
naive masked
naive bidirectional
naive masked
naive bidirectional
runtime (in seconds)
Sequences length
2^7
2^5
2^3
2^1
2^{-1}
2^{-3}
2^{-5}
2^{-7}
2^6
2^8
2^{10}
2^{12}


kernerlized attention runtime for d=256 (best of 10)
naive masked
naive bidirectional
linear masked
linear bidirectional
runtime (in seconds)
Sequences length


ulti head attentions (bidirectional) runtime for d=256 (best of 1
scaled dot product attention
kernelized attention
kernelized attention with RPE
runtime (in seconds)
Sequences length

7 Conclusion

In the present work an easily implemented algorithm to compute Shaw et al. [9] relative positional embedding with linear complexity has been presented. An implementation of Choromanski et al. [2] prefix sum algorithm that doesn't requires custom CUDA code (while maintaining linear complexity) was also presented.

These two elements allowed to define a kernelized attention function with relative positional encoding, that can be computed with linear complexity with regards to sequence length. The proposed model presents linear scalability with sequence length, can be implemented out-of-the-box in neural network framework, and is adapted to sequence to sequence problems instead of being restricted to encoders only models.

References

[1] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL `https://arxiv.org/abs/2004.05150`.

[2] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2021. URL `https://arxiv.org/abs/2009.14794`.

[3] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019. URL `https://arxiv.org/abs/1901.02860`.

[4] Max Horn, Kumar Shridhar, Elrich Groenewald, and Philipp F. M. Baumann. Translational equivariance in kernelizable attention, 2021. URL `https://arxiv.org/abs/2102.07680`.

[5] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer, 2018. URL `https://arxiv.org/abs/1809.04281`.

[6] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL `https://arxiv.org/abs/2006.16236`.

[7] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. URL `https://arxiv.org/abs/2001.04451`.

[8] Antoine Liutkus, Ondřej Cífka, Shih-Lun Wu, Umut Şimşekli, Yi-Hsuan Yang, and Gaël Richard. Relative positional encoding for transformers with linear complexity, 2021. URL `https://arxiv.org/abs/2105.08399`.

[9] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018. URL `https://arxiv.org/abs/1803.02155`.

[10] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities, 2020. URL `https://arxiv.org/abs/1812.01243`.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL `https://arxiv.org/abs/1706.03762`.

[12] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020. URL `https://arxiv.org/abs/2006.04768`.

[13] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021. URL `https://arxiv.org/abs/2007.14062`.