Linformer : Self-Attention with Linear complexity

Lucas Henneçon & Matthieu Neau & Killian Coustoulin

Introduction

The Linformer [1] is a model based on the Transformer, created to reduce the high computational cost of Self-Attention. Self-Attention allows Transformers to perform very well on tasks related to language understanding, but it requires a lot of computation. Specifically, its complexity grows as $\mathcal{O}(n^2)$ with the length of the input sequence.

To solve this problem, other models like the Reformer and Sparse Attention have been proposed. These reduce the complexity to $\mathcal{O}(n\log(n))$ and $\mathcal{O}(n\sqrt{n})$ respectively, but they often lose performance as a trade-off.

The Linformer's authors, however, claim to reduce the complexity to $\mathcal{O}(n)$ while keeping performance as good as the original Transformer. They achieve this by using a simple method to approximate Self-Attention with low-rank projections, making it much more efficient for long sequences.

Self-Attention and Complexity

The Transformer architecture is based on the attention mechanism. As introduced in the original paper *"Attention is all you need"* [2], the self attention module computes a contextualized embedding of the entry sequence using three linear transformations of the initial embedding X : the queries Q, the keys K and the values V. Each of these matrices is of size $n \times d$ where n is the sequence's length and d is the embedding dimension. In the case of multi-head attention with h heads, we obtain h triplets of matrices $Q_i, K_i, V_i, i \in \{1, ..., h\}$. Each of these matrices is of size $n \times d_h$. Each head computes a matrix:

$$Z_i = \text{softmax} \underbrace{\left(\frac{Q_i K_i^\top}{\sqrt{d_k}} \right)}_{A_i} V_i \tag{1}$$

To compute the matrix A_i, we multiply Q_i of size $n \times d_h$ by K_i^T of size $d_h \times n$. This matrix multiplication requires $n^2.d_h$ operations which results in a time complexity of $O(n^2.d_h)$, which is equivalent to $O(n^2)$ if we assume d_h constant and much smaller than n. Computing the scaled softmax and multiplying by V_i also involve operations over a $n \times n$ matrix which keeps the the complexity at $O(n^2)$.

In terms of space complexity, Keys, Queries and Values are of size $n \times d_h$ and the attention matrix A_i is of size $n \times n$ which requires $O(n^2)$ in memory in total under the same assumptions.

Linformer Solution

The paper introduces a theorem based on the Johnson–Lindenstrauss lemma proving that the matrix:

$$P_i = \frac{Q_i K_i^\top}{\sqrt{d_k}} \tag{2}$$

can be approximed by a low rank matrix $\tilde{P}$ of rank $\sim log(n)$. The idea is that the size of the attention matrix can be reduced without loosing too much information. A first method would be to perform an

SVD decomposition of each self-attention matrix, however it would increase the number of computations.

The paper introduces another method: multiply K_i and V_i (which are of size $n \times d_h$) by E_i and F_i (Matrix of size $k \times n$) to project them into lower-dimensional spaces with k independent of n. The transformed matrices $E_i K_i$ and $F_i V_i$ are now of size $k \times d_h$. We obtain:

$$\tilde{Z}_i = \underbrace{\text{softmax}\left(\frac{Q_i(E_i K_i^\top)}{\sqrt{d_k}}\right)}_{n \times k} \underbrace{F_i V_i}_{k \times d_h} \tag{3}$$

The calculation of the attention matrix now is in $O(n \times k \times d_h)$, which is equivalent to $O(n \times k)$ under the same assumptions as before.

The paper introduces a second theorem showing that Z_i can be approached by $\tilde{Z}_i$ and that k can be chosen independently of n (k depends only on d_h) for n large enough. That is why they consider the complexity to be in $O(n)$.
The attention matrix being of size $n \times k$, its space complexity is $O(n \times k)$ which can be approximated to $O(n)$ by the same arguments.

In the paper, the projection matrices are learned. They also introduce parameter sharing techniques to improve efficiency. The parameters of the matrix E and F can be shared across heads of the same layer (Headwise sharing) or shared across all the layer (Layerwise sharing). Furthermore, Instead of computing two projection matrix E and F, it is possible to use a single matrix (E=F) for both projections. However, they show that these different techniques lead to similar results.

Different values of k for different heads and layers can be chosen. As shown in the paper, deeper layer tend to have lower rank, so a lower value of k could be chosen.

Implementation and Results

The experiment was carried out on a classification task. We choose the IMDB reviews Dataset which contains 50000 reviews, either positive or negative. Since the reviews are quite long on average, it is a good choice to evaluate the computation time gain of the Linformer.

We implemented a Transformer model from scratch that has the ability to switch between classic self-attention and Linformer self-attention. As discussed earlier, the main difference between the Linformer and the Transformer models lies in the projection of the matrix K and V, of size $n \times n$ to a lower dimension $k \times n$ during the attention module.

We tried different strategies for the projection : Head-shared, Layerwise shared, one or two projection. Empirically, the method that worked the best was the Layerwise sharing strategy with two distinct projections. To implement it, we simply introduce E and F like so in the classification head:

```
if Linformer_Mode:
# Linear projection matrices for K and V (Layer Wise Sharing strategy)
    self.E = nn.Parameter(torch.empty(max_len, k_dim))
    self.F = nn.Parameter(torch.empty(max_len, k_dim))
```

Once again, several strategies can be used for the projection. We could have used nn.Linear, nn.Parameter, a convolutional layer etc..
Howerver, it was empirically observed that using nn.Parameter with this initialization method gives the best results.

Then, we can apply these projections inside the self-attention module :

```
# Project K and V using E and F
K = (K.transpose(-2, -1) @ E).transpose(-2, -1)
V = (V.transpose(-2, -1) @ F).transpose(-2, -1)
```

Using K.transpose(-2, -1) put the n dimension in last position, then the projection is applied to the last dimension and compress it from a size n to a size k.

Masking was also introduced so that non-masked Tokens do not attend to masked Tokens inside the Self-Attention module. However, the usual strategy of masking the score matrix before the softmax do not apply anymore, since this score matrix is now of dimension $k \times d$.
This is not discussed at all in the paper, so instead, we opted to mask the matrix V and K directly :

```
# Apply mask
if attention_mask is not None:
        mask = attention_mask[:, None, :, None]  # (B, 1, seq_len, 1)
        K = K * mask
        V = V * mask
```

Now, let's move on to the results of the experiments. First, we performed the classification task with a 22M parameters model, which yielded a 89.5% accuracy for both the Transformer and Linformer models. (SOTA pre-trained models have a 96% accuracy on this dataset). Then, we performed an ablation study (with a smaller model) using several values of n and k to see the Linformer model performance and computation time compared to the Transformer model.

n	Model	Accuracy (%)	Computation Time (s)
256	Transformer	86.20	139.49
	Linformer $k = 64$	86.33	135.02
	Linformer $k = 128$	85.67	135.94
512	Transformer	87.82	176.58
	Linformer $k = 64$	88.01	154.11
	Linformer $k = 128$	87.98	158.00
	Linformer $k = 256$	87.73	166.86
1024	Transformer	86.88	336.45
	Linformer $k = 64$	87.02	196.72
	Linformer $k = 128$	49.94	203.99
	Linformer $k = 256$	87.96	221.55
	Linformer $k = 512$	87.43	261.74

Table 1: Comparison of Accuracy and Computation Time for Different Models and Sequence Lengths

We can see that in terms of performance, the Linformer is doing well compared to the Transformer. Now, when it comes to computation time, applying the Linformer to a dimension of $n = 256$ is mostly irrelevant and one could argue that it is also the case if $n = 512$.
The reduction in computation time for $n = 512$, which are presented in the Linformer paper, could not be replicated here.
Now, when it comes to $n = 1024$, we can see that the difference in computation time is much more interesting. For a value of $k = 256$, the Linformer achieves similar result as the Transformer which is 1.5 times longer to train ! (Notice that the Transformer underperformed on this set and it is discussed in the next section)

Difficulties and limitations of the experiment

The results presented above are not very accurate. We used a larger validation set than usual (25000 of the 50000 reviews), but these results still suffer from a lot of variance. A better approach to validate the performance of the Linformer would have been to perform K-fold cross-validation technique. However, the cost of such an operation would be computationally quite high, but, using that method, we would have gotten results that are much more accurate and trustworthy.

There has also been some difficulties concerning the Linformer itself. Indeed, since the Linformer uses these new learnable projections E and F, it can sometimes not learn at all for one or two epochs. It will start learning eventually, but it can sometimes struggle to start.

We tried several different methods of initialization or types of projections (nn.Linear or nn.Parameter), different values for the learning rate and weight decay but it does not matter. Sometimes the Linformer can start learning right away, but the newly introduced E and F projections may lead to unstable gradient or suboptimal learning early on.

Of course, such an issue only arises because this model had no pre-training at all. If it was pre-trained, even for a few epochs, we could suppose that the E and F projection would already be somewhat more efficient than the random initialized one, and the model would be learning the right parameters for the classification right away.

Critique of the Paper

Self-Attention as a Low-Rank Theorem

The central theorem of the paper posits that self-attention matrices exhibit a low-rank structure. However, this result is derived solely from the Johnson-Lindenstrauss lemma, which is a general statement about dimensionality reduction and does not exploit any specific properties of self-attention matrices. Consequently, framing this as a theorem specific to self-attention might be seen as overstated. More importantly, the theorem does not unveil any intrinsic structure unique to self-attention matrices. Summarizing the information encoded in self-attention matrices into smaller matrices remains an active area of research.

Furthermore, similar results can be observed in randomly generated matrices. For example, we generated random matrices of size $n = 512$ with entries sampled from a uniform distribution on $[0, 1]$. The cumulative explained variance at $k = 128$ is comparable to what the authors report for self-attention matrices, with an explained variance of approximately 0.91. However, it is important to note that the structure of such random matrices is likely very different from that of self-attention matrices. This is not universally true for all types of random matrix: repeating the experiment with entries sampled from a standard normal distribution resulted in much less pronounced behavior (see Fig. 1).

The qualitative analysis of increasing explained variance in higher layers is particularly compelling and reminiscent of the behavior observed in CNNs, where deeper layers aggregate more specific features. Notably, in higher layers, the explained variance reaches up to 96% using the first 128 dimensions, which is significantly higher than the results for random matrices. This observation highlights the structured nature of self-attention matrices, distinguishing them from purely random counterparts.

Critique of the Experiments

The authors made a commendable choice to evaluate their method not only on pre-training tasks but also on fine-tuning tasks, demonstrating performance comparable to architectures employing traditional self-attention.

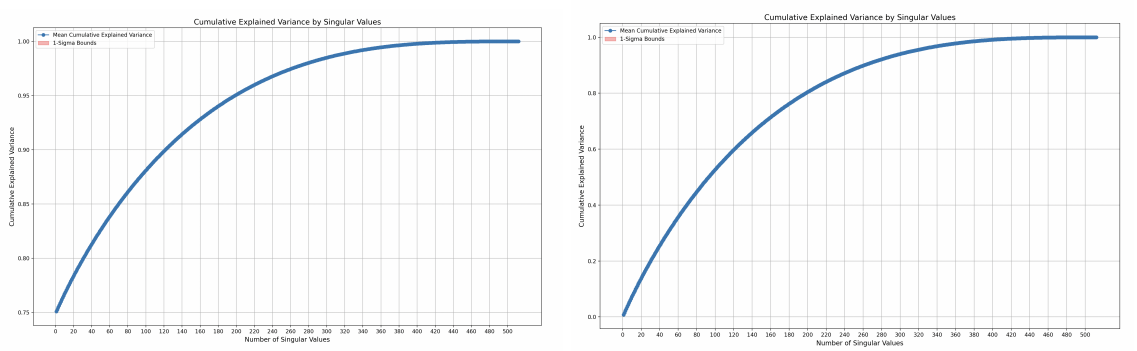

Figure 1: Cumulative explained variance for random matrices: (left) matrices generated with entries from a uniform distribution on $[0, 1]$; (right) matrices generated with entries from a standard normal distribution.

However, we note that some of the performance curves in Figure 3 are truncated, particularly for the two top plots, where it appears that the perplexity of the standard self-attention model is still decreasing. Extending these training curves could reveal a small but potentially significant gap in performance between the two approaches. Providing fully extended curves would allow for a more comprehensive comparison and further validate the results presented in the paper.

Lastly, it is worth noting that the sequence length was only pushed to $n = 4096$, which is only an order of magnitude above $k = 256$, when the paper claims in its second theorem that the Self-Attention matrix rank is equal to $k = \mathcal{O}(\log(n))$

Our opinion on the paper

Overall, we enjoyed working on this paper as it proposes a solution to one of the major bottlenecks in transformers today, which is the quadratic cost in the sequence length of attention. The methodology is clear and it provides rigorous experiments along with mathematical proofs motivating them.

It also gave us the opportunity to take a deeper look at the Self-Attention mechanism with original approaches like Big Bird [3] or the Reformer [4].

AI usage disclosure

GPT-4 was used as a technical assistance and for concepts clarification throughout the project.

References

[1] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[3] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062, 2020. URL `https://arxiv.org/abs/2007.14062`.

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