IASD NLP Project – Linformer Transformer Classifier

[1]Brice CONVERS,
[2] Paul MALET
[3]Shijie TIAN

[1,2,3]IASD, PSL - Paris Dauphine University,
e-mail: [1] brice.convers@dauphine.eu,
[2] paul.malet@dauphine.eu,
[3] shijie.tian@dauphine.eu,

1 Introduction

Transformer models from [1] are widely used for sequence modeling due to their self-attention mechanism, which captures long-range dependencies effectively. However, standard self-attention scales quadratically with sequence length, making training and inference costly for long sequences.

The Linformer from [2] addresses this limitation by approximating the attention matrix as low-rank, reducing complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(nk)$ while maintaining performance. The key idea is to project keys and values into a lower-dimensional space, preserving most of the relevant information.

In this project, we focus on implementing a simplified version of the Linformer encoder and running a small-scale experiment to explore its behavior. Following the experiment guidelines, we do not aim to reproduce the full experimental setup of the paper. Instead, we design a practical experiment that highlights a key aspect of the method, using limited data and model size suitable for our computational resources. The report presents our implementation details, the experiment, and the results, emphasizing understanding and critical evaluation rather than large-scale reproduction.

2 Linformer: Low-Rank Self-Attention

2.1 Motivation and Key Idea

Standard Transformers scale quadratically with sequence length n, since every token attends to all others. Linformer [2] addresses this by observing that the attention matrix is approximately low-rank. By projecting keys and values to a lower dimension $k \ll n$, attention is computed in $\mathcal{O}(nk)$ time while preserving most information.

2.2 Theoretical and Empirical Foundations

The softmax attention matrix P often has most information concentrated in top singular values. Empirical spectra from RoBERTa show higher layers are even "more low-rank." A Johnson–Lindenstrauss argument proves P can be approximated with rank growing logarithmically in n (Theorem 1), justifying factorized attention. Figure 1 illustrates this.

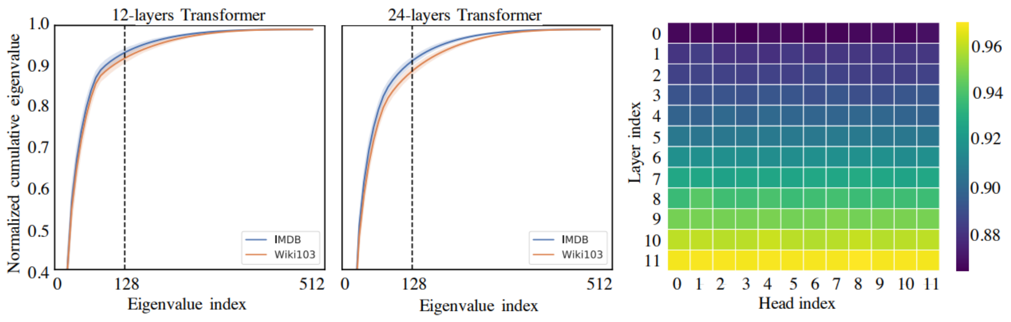

Figure 1: **Spectrum analysis of the self-attention matrix.** Left: normalized cumulative singular values of P. Right: heatmap of cumulative eigenvalues at the 128-th largest eigenvalue across layers and heads.

2.3 Architecture and Variants

Each attention head uses learned projections

$$E, F \in R^{n \times k}$$

to compress keys K and values V. Attention is then computed between queries Q and projected keys, multiplied by projected values. This reduces per-layer time and memory from $\mathcal{O}(n^2)$ to $\mathcal{O}(nk)$. Variants include sharing projections across heads or layers, using non-uniform k, or alternative projections like pooling. Figure 2 shows the architecture and inference times.

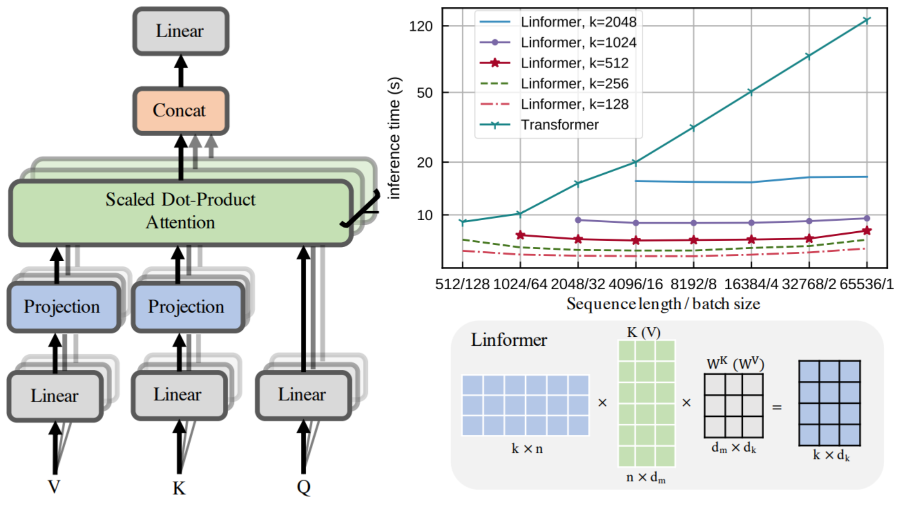

Figure 2: **Linformer architecture and inference time.** Left/bottom-right: multihead linear self-attention. Top-right: inference time vs. sequence length for different Linformer models.

2.4 Results and Efficiency

Linformer, pretrained on BookCorpus and Wikipedia and finetuned on IMDB, SST-2, QNLI, and QQP, achieves performance close to the Transformer baseline with $k = 128$ and slightly surpasses RoBERTa-base with $k = 256$, indicating that performance depends more on the projection dimension k than on sequence length n.

In terms of efficiency on a V100 GPU, Linformer shows significant gains: for $n = 512$ and $k = 128$, inference is $\sim 1.5\times$ faster with $1.7\times$ larger batch size; for $n = 4096$ and $k = 256$, it is $\sim 3.2\times$ faster with $13\times$ less memory; and for $n = 8192$ and $k = 256$, $\sim 5\times$ faster with $26\times$ memory reduction.

2.5 Summary

Low-rank attention allows projecting keys and values to a small k, achieving $\mathcal{O}(n)$ self-attention with Transformer-level accuracy while greatly reducing computation and memory for long sequences.

3 Implementation and Contributions

3.1 Design Rationale

In this project, we focused exclusively on implementing the **Linformer encoder** rather than the full encoder–decoder architecture. This choice was made to simplify the problem while keeping the project interesting and technically relevant. From a computational point of view, using only the encoder reduced training time and memory usage, making it possible to run more experiments within our time constraints. Conceptually, the encoder still retains the main mechanisms of interest (multi-head attention and feed-forward sublayers) and is therefore sufficient to study the Linformer approximation in a controlled setting.

While the encoder and decoder share several similarities, the encoder does not require causal masking, making the implementation simpler while still enabling a meaningful comparison between the Linformer and the standard Transformer.

3.2 Codebase and Adaptation

Our implementation was inspired by the open-source Linformer-PyTorch repository.

However, we extracted only the parts relevant to our work, cleaned the code, and removed features outside the scope of our experiments. We also add particularities and factorize the code as needed. This allowed us to keep the implementation minimal, easy to read, and fully under our control. We ensured a complete understanding of each retained component before integrating it.

3.3 Architectural Features

Linear Attention Projection. We implemented the Linformer idea of projecting the key and value matrices into a lower-dimensional space, reducing the complexity of attention from $O(n^2)$ to $O(n)$. Our main experiments used a fully connected layer for the projection, but we kept the option to use a convolutional layer. Although our tests showed no improvement with convolutions, the option remains available for future work.

Attention Head Sharing. We added the possibility to share attention heads between multiple encoder layers. This feature was not explored experimentally in this work but could be relevant for reducing parameters and improving efficiency.

Masking Mechanisms. We preserved various masking options, including masks on the keys/values and on the embeddings, even though they were not required in our experiments. Keeping them ensures that the implementation remains general-purpose.

Sequential Head Computation. Instead of computing all attention heads in parallel using tensor operations, we process each head sequentially. This approach reduces peak RAM usage but increases training time.

Projection Dimension Decay. We kept the possibility of linearly decreasing the projection dimension across the encoder layers, as suggested in the Linformer paper. This was not activated in our experiments but remains available for further optimization.

3.4 Training and Optimization Enhancements

We built a simple but effective training loop with the following improvements. All these points are our contributions.

- **Multiple activation functions:** support for GELU, ReLU, Swish, and SwiGLU. We used GELU by default due to its smooth activation profile and strong performance in Transformer-based models from [3]
- **Root Mean Square Layer Normalization (RMSNorm)** applied at the end of the attention mechanisum for stability from [4]. Unlike LayerNorm, RMSNorm does not subtract the mean, normalizing only by the root mean square, which can improve convergence and reduce computation.
- **Output processing:** average pooling to reduce sequence dimension, followed by two fully connected layers with an intermediate softmax for probability outputs.
- **Learning rate scheduling:** cosine annealing scheduler from [5] to gradually lower the learning rate and improve convergence.
- **Real-time monitoring:** integration of TensorBoard for visualization of metrics during training.

3.5 Conclusion of Coding Implemetation

The developed Linformer encoder provides a flexible, lightweight, and memory-efficient framework for studying linear attention mechanisms. Focusing on the encoder allowed us to balance computational efficiency with architectural completeness. While some features such as head sharing and projection dimension decay were not fully explored, they remain available for future experiments. This modular implementation can serve as a solid base for both reproduction of existing results and new research on efficient Transformer architectures.

4 Results of our experiments

4.1 Benchmarking Setup and Results

We implemented a benchmarking process to evaluate the performance of the Linformer compared to the Transformer, focusing on training speed, memory usage, and validation accuracy across different sequence lengths.

The tested sequence lengths were 64, 128, 256, and 512, to understand how the models scale with varying input sizes. Each model was trained for 5 epochs using the AdamW optimizer and a learning rate scheduler. During training, the loss and accuracy metrics were recorded for each epoch, and GPU memory consumption was monitored.

Prior to training, we used the BERT tokenizer for preprocessing and divided the dataset into training and validation sets, with a batch size of 64. For visualization, we generated multiple plots to display accuracy trends over epochs, training time against sequence length, memory usage, and speedup ratio comparisons. These plots are stored in the repository and in the appendix.

The benchmarking results revealed distinct differences between the Transformer and Linformer models across the tested sequence lengths:

- **Accuracy:** Linformer's accuracy is higher than Transformer's while training faster. For both models, accuracy decreases between sequence lengths 64, 128, and 256, before increasing at 512. This could be related to the initialization of the models.
- **Training Time:** Linformer requires less time to train compared to the standard Transformer for large sequence lengths, but was slightly slower for small sequence lengths, which we cannot explain.
- **Memory Usage:** Linformer uses slightly more memory than the Transformer, which is not the expected behavior.

Overall, these results align with the original paper's findings: Linformer trains faster than the Transformer and achieves slightly higher accuracy.

5 Limitations of our experiments

Our experimentations have some limits and could be improved. The main limit is the repetition of computation, it would be great to have repeated runs of the training process for each experimental condition. Each model was trained only once for each sequence length, which limits the robustness of our findings because of factors such as model initialization or stochastic elements in training. A more comprehensive study would involve multiple runs to account for randomness and offer averaged performance metrics to increase result reliability.

Our experiments were conducted under computational power constraints, which influenced our choice of sequence lengths and model configurations. As a result, our tests occurred on short sequence lengths, where ideally longer sequences could be explored to better understand model scaling behavior with real-world long-text scenarios. These computational power constraints also make it harder to have repeated runs of the training process.

6 Conclusion

Our project confirms the main promise of Linformer in a small, controlled setting. With our encoder-only implementation and the benchmark over sequence lengths 64–512, we observe that Linformer tends to reach equal or slightly higher validation accuracy than the vanilla Transformer, and it trains faster once the sequence

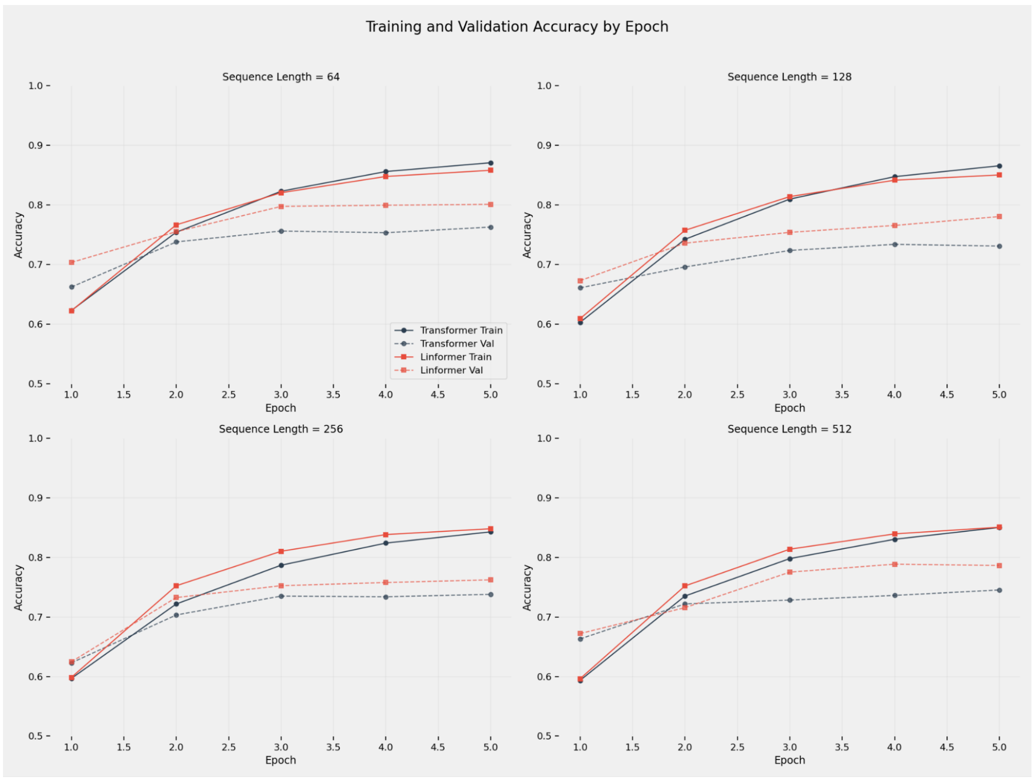

Figure 3: Training and Validation Accuracy by epoch for different n and k.

is longer. At very short sequences the speed advantage is not always visible, and memory usage behaved a bit unexpectedly in our run,. Overall, the trends still match the intuition of linear-complexity attention.

Our concrete contributions are: a minimal, readable Linformer encoder with options for projection strategy and training utilities (RMSNorm, scheduler), plus a lightweight benchmark that compares accuracy, time, and memory across lengths. The limitations are: each condition was trained once, the sequences are short, and some options (e.g., head sharing, projection decay) were not explored. These choices were made to respect the resource budget, but they reduce statistical confidence.

For amelioration: repeat runs to average results, add a tiny ablation on the projection dimension k, and include one compact figure that aligns accuracy/time/memory in the same visual style. This can improve reliability

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: `1706.03762 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1706.03762`.

[2] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, *Linformer: Self-attention with linear complexity*, 2020. arXiv: `2006.04768 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2006.04768`.

[3] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2023. arXiv: `1606.08415 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1606.08415`.

[4] B. Zhang and R. Sennrich, *Root mean square layer normalization*, 2019. arXiv: `1910.07467 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1910.07467`.

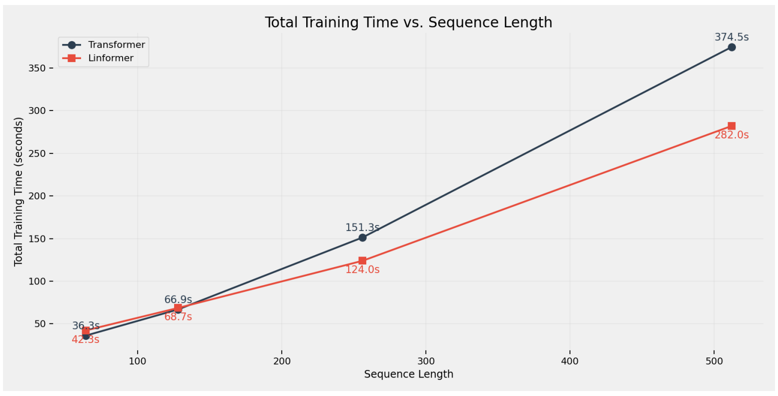

Figure 4: Linformer and Transformer Training time by sequence length.

[5] T. Cazenave, J. Sentuc, and M. Videau, "Cosine annealing, mixnet and swish activation for computer go," in Aug. 2022, pp. 53–60, ISBN: 978-3-031-11487-8. DOI: 10.1007/978-3-031-11488-5_5.