Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Sparse factorization of square matrices with application to neural attention modeling

Ruslan Khalitov[1], Tong Yu[1], Lei Cheng, Zhirong Yang[*]

Norwegian University of Science and Technology, Norway

ARTICLE INFO

Article history:
Received 26 October 2021
Received in revised form 17 March 2022
Accepted 14 April 2022
Available online 22 April 2022

Keywords:
Matrix factorization
Sparse
Neural networks
Attention modeling

ABSTRACT

Square matrices appear in many machine learning problems and models. Optimization over a large square matrix is expensive in memory and in time. Therefore an economic approximation is needed. Conventional approximation approaches factorize the square matrix into a number matrices of much lower ranks. However, the low-rank constraint is a performance bottleneck if the approximated matrix is intrinsically high-rank or close to full rank. In this paper, we propose to approximate a large square matrix with a product of sparse full-rank matrices. In the approximation, our method needs only $N(\log N)^2$ non-zero numbers for an $N \times N$ full matrix. Our new method is especially useful for scalable neural attention modeling. Different from the conventional scaled dot-product attention methods, we train neural networks to map input data to the non-zero entries of the factorizing matrices. The sparse factorization method is tested for various square matrices, and the experimental results demonstrate that our method gives a better approximation when the approximated matrix is sparse and high-rank. As an attention module, our new method defeats Transformer and its several variants for long sequences in synthetic data sets and in the Long Range Arena benchmarks. Our code is publicly available[2].

1. Introduction

Many machine learning models include one or more square matrices, such as the kernel matrix in Support Vector Machines (Cortes & Vapnik, 1995) or Gaussian Process, the affinity matrix in graph or network representation, and the attention matrix in Transformers (Vaswani et al., 2017b). In many machine learning problems, the solution space is also over square matrices, for example, graph matching and network architecture optimization.

Obtaining a full $N \times N$ matrix can cause infeasible storage and computational cost if N is large. For example, a double-precision kernel matrix of the typical `MNIST` handwritten digit data set ($N = 70{,}000$) requires about 36.5G memory. In another example, we need to calculate eight quintillion float numbers to fill a full attention matrix of a human DNA sequence with about 3.2 billion base pairs.

Therefore we need economical surrogates to approximate the full square matrices at a large scale. Matrix factorization is a widely used approach that approximately factorizes the square matrix to some cheaper matrices. In conventional matrix factorization, the factorizing matrices must be low-rank, for example, in Truncated Singular Value Decomposition (TSVD) and Nyström approximation (Drineas & Mahoney, 2005; Williams & Seeger, 2001). However, these conventional approaches do not work well if the square matrix in the approximation is not low-rank.

This paper proposes a novel approximation method called Sparse Factorization (SF), where the approximated square matrix is factorized into some full-rank sparse matrices. Using the Chord protocol (Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001) to specify the non-zero entry positions,[3] we can match an $N \times N$ full matrix with $\log N$ factorizing matrices, where each contains $N \log N$ non-zero entries and thus in total $N(\log N)^2$ non-zeros. Therefore the approximation is economical when N is large.

We tested the new approximation method on a variety of synthetic and real-world square matrices, with comparison to the most accurate low-rank approximation method, TSVD. We find that given the number of non-zero entries, SF wins over TSVD for approximating sparse square matrices with unknown non-zero positions.

We exploit the above advantage of SF to design a novel neural attention model. We parameterize the mapping from input data to the non-zero entries in each factorizing matrix with neural networks. The parametric SF thus replaces the conventional scaled

[*] Corresponding author.
E-mail address: zhirong.yang@ntnu.no (Z. Yang).
[1] Equal contribution.
[2] https://github.com/RuslanKhalitov/SparseFactorization.

[3] Non-zero entries are those stored, including both non-zeros and explicit zeros.

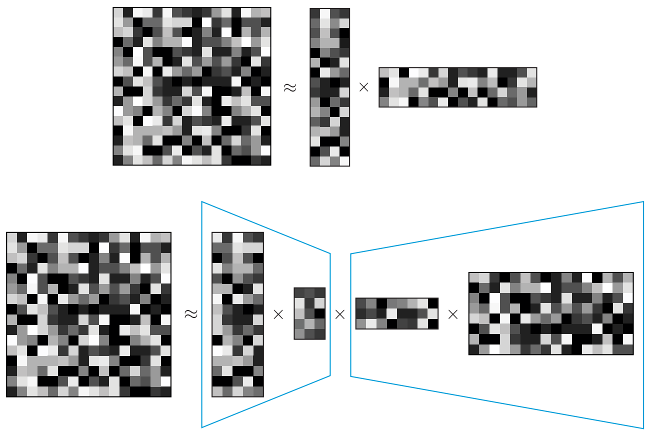


Fig. 1. Examples of low-rank matrix dense matrix factorization. Each small square represents a matrix entry. Top is a two-factor factorization and the bottom is a four-factor factorization. The trapezoids illustrate the "bottleneck", i.e. grouping the factorizing matrices according the place with the smallest rank.

dot-product attention. We find that only one such block of SF attention already defeats Transformer and its several state-of-the-art variants on very long synthetic sequence classification and on the public Long Range Arena benchmark tasks.

The remaining of the paper is organized as follows. We first briefly review the matrix approximation based on low-rank factorization in Section 2. In Section 3, we present the machine learning formulation of sparse factorization of square matrices. The parametric formulation using neural networks is proposed in Section 4. In Section 5, we present the experimental settings and results. Conclusion and future work are given in Section 6.

2. Low-rank matrix factorization

The conventional matrix approximation is to factorize the large square matrix X into a few low-rank matrices, for example $X \approx \widehat{X} = WH$ or $X \approx WSH$ with $W \in \mathbb{R}^{N\times r}$, $S \in \mathbb{R}^{r\times r}$, $H \in \mathbb{R}^{r\times N}$ and $r \ll N$. There are different tri-factor kind decomposition, for example, Nyström decomposition (Williams & Seeger, 2001), where W and H are calculated using normal kernel function, and S is learned, and CUR decomposition (Mahoney & Drineas, 2009), where W contains r columns of X, H contains r rows of X, and S is learned.

The approximation error can be measured by a certain divergence, for example the squared Frobenius norm: $D(X \parallel \widehat{X}) = \|X-\widehat{X}\|_F^2 = \sum_{ij}\left(X_{ij} - \widehat{X}_{ij}\right)^2$. According to Eckart and Young (1936), minimization of $\|X - WH\|_F^2$ over W and H has a closed-form solution using Truncated Singular Value Decomposition (TSVD). Denote Λ the diagonal matrix with the largest r singular values. The corresponding left and right singular vectors as columns form matrices U and V, respectively. Then the minimum appears by setting $W = U$ and $H = \Lambda V^T$, where we can move any scaling from W to H or vice versa.

Matrix factorization with more than two factors cannot achieve a lower approximation error than TSVD. In general, we write $\widehat{X} = \prod_{m=1}^{M} W^{(m)}$, where $W^{(m)} \in \mathbb{R}^{r_m,r_{m+1}}$ with $r_1 = r_{M+1} = N$. We can always reduce a multi-factor case to the two-factor form by grouping $L = \prod_{m=1}^{m'-1} W^{(m)}$ and $R = \prod_{m=m'}^{M} W^{(m)}$, where $m' = \arg\min_m \left(\{r_m\}_{m=2}^{M}\right)$ and $X \approx LR$. It is required that $r_{m'} \ll N$ when approximating large square matrices. Otherwise, the factorizing matrices are still too large. The grouping at the bottleneck is illustrated in Fig. 1 (bottom).

There are low-rank matrix factorization methods with certain constraints on the factorizing matrices, for example, nonnegative matrix factorization (Lee & Seung, 1999) and binary matrix factorization (Slawski, Hein, & Lutsik, 2013; Zhang, Li, Ding, & Zhang, 2007). However, the constraints limit the solution space and thus usually lead to a higher approximation error. Some other methods employ some penalty terms on the factorizing matrices, for instance, the Sparse Coding (Donoho, 2006) or Funk matrix factorization (Funk, 2006). Despite their particular applications, these methods generally give a higher approximation error than TSVD (in terms of Frobenius norm) due to compromise to the extra penalties.

3. Sparse factorization

As we have seen, the performance of LRMF is capped due to the low-rank constraint. Its performance is poor if the approximated square matrix is far from low-rank, i.e., the sum of the few largest eigenvalues does not dominate the matrix trace.

Here we propose a new approximation method that is free of the low-rank constraint. Our method implements the approximation with a number (M) of *square* and *sparse* factorizing matrices. The approximation is still economical if the total number of non-zero entries is much smaller than N^2.

There are many ways to specify the sparse structure (i.e. the non-zero positions). A good specification should guarantee that the product of the factorizing matrices, $\widehat{X}$, should be a full matrix. The condition prevents that the approximating matrix contains some always-zero entries. In addition, we consider a secondary requirement that each factorizing matrix has the same sparse structure, which provides better symmetry in the approximation.

We thus adopt a modified Chord protocol (Stoica et al., 2001) which originates for peer-to-peer distributed hash tables. In the protocol, the indices from 1 to N are organized in a circular graph, where the ith node connects to itself and the $((i+2^k) \mod N)$th nodes with $k = 0, \ldots, K-2$. We set $K = \log_2 N$ in our work. The Chord protocol is illustrated in Fig. 2 (top).

We use the Chord protocol to specify the non-zero positions in each factorizing matrix, where each matrix row has $\log_2 N$ non-zero entries. That is, for $m = 0, \ldots, M$ and $k = 0, \ldots, K-2$

$$W_{ij}^{(m)} \begin{cases} \neq 0 & \text{if } j = i \text{ or } j = (i+2^{k-2}) \mod N \\ = 0 & \text{otherwise.} \end{cases} \tag{1}$$

Thus every factorizing matrix has $N \log_2 N$ stored non-zero entries.

The product of the factorizing matrices corresponds to the connections in the circular graph after multiple hops. We can set the number of factorizing matrices to $M = \log_2 N$, which corresponds to the number of hops, and the resulting matrix product becomes a full matrix with high probability (see Stoica et al., 2001, Theorem 2). In total, there are $N(\log N)^2$ non-zero entries, still much smaller than N^2 for a large N.

The (Chord) Sparse Factorization (SF) can thus be formulated as the following optimization problem:

$$\underset{W^{(1)},\ldots,W^{(M)}}{\text{minimize}} \left\| X - \prod_{m=1}^{M} W^{(m)} \right\|_F^2 \tag{2}$$

where $W^{(m)}$'s are sparse square matrices with non-zero positions specified by the Chord protocol. The approximation scheme is illustrated in Fig. 2 (bottom).

The minimization in Eq. (2) can be implemented by any existing solvers for unconstrained smooth optimization, where the cost of computing the gradient is $O(NMK)$. Once the approximation error is minimized, we obtain the factorizing matrices $W^{(1)}, \ldots, W^{(M)}$. We call the approach non-parametric SF as we directly optimize over the factorizing matrices.

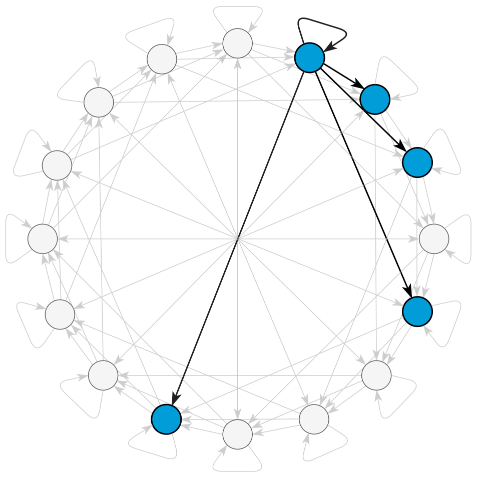
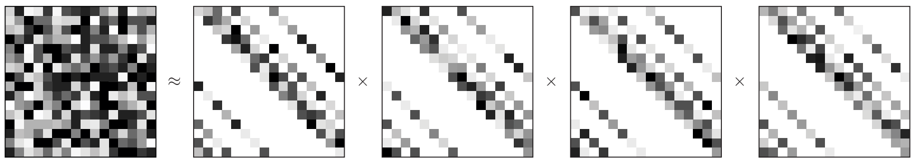



Fig. 2. Illustration of (top) the Chord protocol and (bottom) sparse factorization of a square matrix for $N = 16$. Grayscale squares in the factorizing matrices represent stored entries (non-zeros and explicit zeros), and completely white squares represent non-stored entries (implicit zeros).

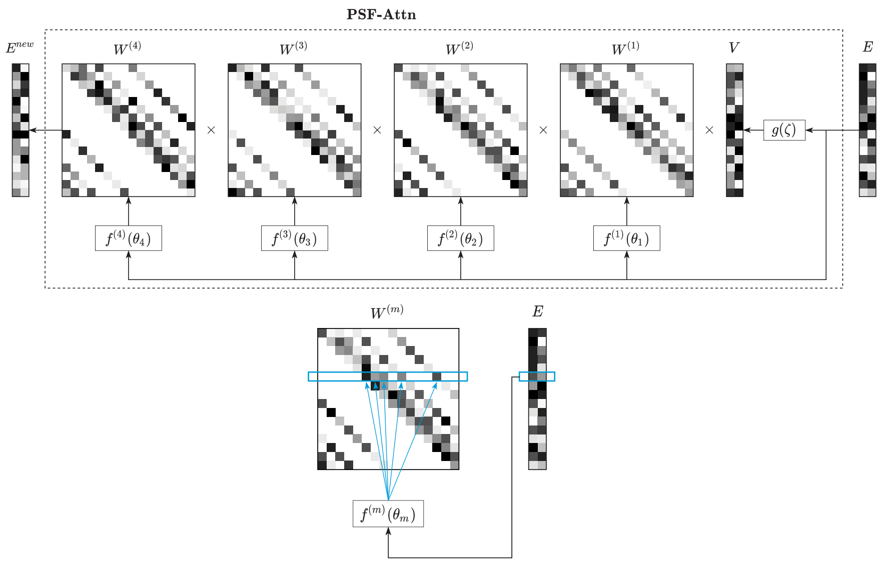


Fig. 3. Illustration of (top) the parametric transformation from current embedding E to new embedding E^{new} based on Sparse Factorization and (bottom) setting of MLP output to non-zero entries in the corresponding sparse factorizing matrix.

4. Parametric sparse factorization

Besides direct optimization over the factorizing matrices, we can consider the mapping from vectorial input data to the factorizing matrix entries because each node in the Chord protocol has the same out-degree. The mapping as a component can endow the model (1) generalization to newly coming data and (2) representation learning by transforming the current embedding representation to a new embedding.

We present a transformation model as a concrete example to illustrate the idea. It is a transformer-like model (see Fig. 3 top), where we replace the scaled dot-product attention with a product of sparse square matrices. The matrix product provides an approximation to a full non-normalized attention matrix.

One block of such Parametric Sparse Factorization Attention (PSF-Attn) transforms data in the current embedding E to a new embedding $E^{\text{new}} = \text{PSF-Attn}(E)$. There is a number of MLPs in the block, where the MLP $f^{(m)}$ with parameters θ_m takes $E_{i:}$ (the ith row of E) as an input and returns the non-zeros in the ith row of $W^{(m)}$, for $i = 1, \ldots, N$ and $m = 1, \ldots, M$. That is, for $k = 2, \ldots, K$,

$$W_{ij}^{(m)} = \begin{cases} \left[f^{(m)}(E_{i:}; \theta_m)\right]_1 & \text{if } j = i \\ \left[f^{(m)}(E_{i:}; \theta_m)\right]_k & \text{if } j = (i + 2^{k-2}) \mod N \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Similar to transformers, we use another MLP g with parameters ζ to convert an E row to the corresponding V row.

The new model can work as a building block in representation learning frameworks. For example, if we define an objective function $\mathcal{J}$ over E^{new}, the learning can be formulated as the following optimization problem:

$$\underset{\theta_1,\ldots,\theta_M,\zeta}{\text{minimize}} \quad \mathcal{J}(\text{PSF-Attn}(E;\theta_1,\ldots,\theta_M,\zeta)), \tag{4}$$

where the optimization can be implemented with back-propagation and a gradient based algorithm such as Adam (Kingma & Ba, 2014). It is also straightforward to stack multiple PSF-Attn blocks to implement deeper learning.

The PSF-Attn method has two advantages. First, the original transformer and many of its variants (e.g. Choromanski et al., 2020; Katharopoulos, Vyas, Pappas, & Fleuret, 2020; Tay et al., 2021; Wang, Li, Khabsa, Fang, & Ma, 2020) are built upon scaled dot-product, which essentially employ low-rank matrix factorization. They may not work well if the attention is intrinsically high rank. In contrast, all factorizing matrices in our method are full-rank, and so is their product. Therefore PSF-Attn does not suffer from the low-rank bottleneck constraint. Second, our model does not require softmax over a large square matrix, avoiding many computational difficulties. Removing the softmax also endows more freedom in mixing the V rows because the mixture can go beyond the convex hull.

Our method also differs from the previous multi-layer sparse attention approaches (e.g. Child, Gray, Radford, & Sutskever, 2019; Correia, Niculae, & Martins, 2019; Li et al., 2019) because they still use scaled dot-products. PSF-Attn can give full attention in one block. For the ith element in the sequence, its attention to other elements can directly be obtained by vector-matrix product $W^{(M)}_{i:}\prod_{m=1}^{M-1}W^{(m)}$.

5. Experiments

We have performed four groups of experiments. In the first group, we studied which types of square matrices are more suitable for our method. In the second group, we verified the scalability of the new attention model PSF-Attn on synthetic sequences up to tens of thousands positions. Next, we compare our method with several Transformer variants on a DNA sequences classification task. Finally, we tested PSF-Attn for the Long Range Arena benchmark data sets to see its performance in real-world practice. The first group of experiments was run on a standard PC with an Intel Core i9 CPU. The other groups were run on a Linux server with one NVIDIA-Tesla V100 GPU with 32 GB of memory.

5.1. Exploring real-world matrices for SF

There are many types of square matrices. TSVD is the best for approximating matrices close to low rank in terms of F-norm. For non-parametric SF, we performed an empirical study (1) to show that SF can supersede TSVD in F-norm by using the same number of non-zeros and (2) to identify the types of square matrices particularly suitable for SF.

Given a square matrix, we used the Matlab `fminunc` optimizer to solve the problem in Eq. (2), where the non-zero entries in the factorizing matrices are initialized to random numbers between $[K^{-1}, K^{-1}+10^{-2}]$. The total number of non-zero entries for SF and TSVD are $N(\log_2 N)^2$ and $2Nr+r$. For a fair comparison, we set the $r = \lceil(\log_2 N)^2/2\rceil$ in TSVD such that it has nearly the same number and no fewer non-zeros than SF. We first used 256×256 grayscale images as approximated matrices because we can directly see them. Fig. 4 (left) shows six typical square images, where we provide the resulting TSVD and SF approximation errors in F-norm below each image (more examples in the supplemental document).

Table 1
Approximation errors in F-norm by using TSVD and SF for different types of square matrices.

Data type	Data name	TSVD	SF
Dense graph	AuraSonar	**8.54e+00**	8.68e+00
Dense graph	Protein	1.17e+01	**1.09e+01**
Dense graph	Voting	**8.07e−04**	1.71e+01
Dense graph	Yeast	3.72e+01	**3.61e+01**
Network	Sawmill	3.24e+00	**1.03e+00**
Network	Scotland	5.90e+00	**3.76e+00**
Network	A99 m	1.47e+01	**1.01e+01**
Network	Mexican Power	3.85e+00	**1.71e+00**
Network	Strike	2.73e+00	**1.04e+00**
Network	Webkb Cornell	6.98e+00	**4.80e+00**
Network	Worldtrade	8.65e+04	**4.47e+04**
Surface mesh	Mesh1e1	1.87e+01	**9.82e+00**
Surface mesh	Mesh2e1	**2.48e+02**	3.47e+02
Surface mesh	OrbitRaising	9.37e+01	**8.35e+01**
Surface mesh	Shuttle Entry	2.73e+03	**1.86e+03**
Surface mesh	AntiAngiogenesis	5.85e+01	**3.29e+01**
Covariance	Phoneme	**2.80e+01**	5.27e+01
Covariance	MiniBooNE	**1.04e+00**	6.36e+03
Covariance	Covertype	8.22e−02	**1.90e−02**
Covariance	Mfeat	**1.11e+03**	4.01e+05
Covariance	OptDigits	**3.28e+01**	7.01e+01
Covariance	PenDigits	4.00e+02	**1.87e+02**
Covariance	Acoustic	1.36e−02	**1.11e−02**
Covariance	IJCNN	5.24e−02	**3.03e−02**
Covariance	Spam Ham	1.07e−01	**4.97e−02**
Covariance	TIMIT	**9.64e+01**	1.56e+02
Covariance	Votes	4.00e−01	**1.70e−01**

The `chess` image (matrix) is close to low-rank because the black-and-white chessboard is two-rank. Therefore TSVD performs better as expected. TSVD also works well for the `Lena` and `apple` images because TSVD tends to preserve low-frequency information in images. In contrast, TSVD is not as good as SF for the other three images that contain rich high-frequency details such as lines and corners, which indicates a matrix type where SF can defeat LRMF.

To further verify this, we computed the gradient magnitudes of the images (shown in Fig. 4 right). In this way, the intensities in constant areas become zero, and the remaining non-zeros are mainly high-frequency details. We can see that SF gives a lower approximation error than TSVD for all such matrices, which confirms that SF is more advantageous for approximating matrices with rich high-frequency details.

In summary, the results show that the TSVD performance does not cap SF by using the same number of non-zeros for approximating square matrices. The winning cases indicate that SF is often better than LRMF when the approximated matrix is (1) sparse, (2) intrinsically high-rank, or (3) containing rich high-frequency details.

Besides square images, we have also compared TSVD and SF on several other types of square matrices. Table 1 shows the comparison results. The data types include affinity matrices of dense graphs (dense graph), affinity matrix of sparse networks (network), affinity matrix of surface mesh over 3D objects (surface mesh), and covariance matrix of vectorial data (covariance). We can see that for dense graph and covariance types, sometimes TSVD is better while sometimes SF can win. SF wins for most cases for surface mesh because the mesh networks probably do not have a low-rank structure. SF wins all data sets in the network type, which indicates that SF is more effective for approximating sparse matrices. We give the details of the data sets in the supplemental document.

5.2. Approximation to large attention matrices

Here we test whether PSF-Attn is scalable to approximate large attention matrices. We have used two synthetic data sets

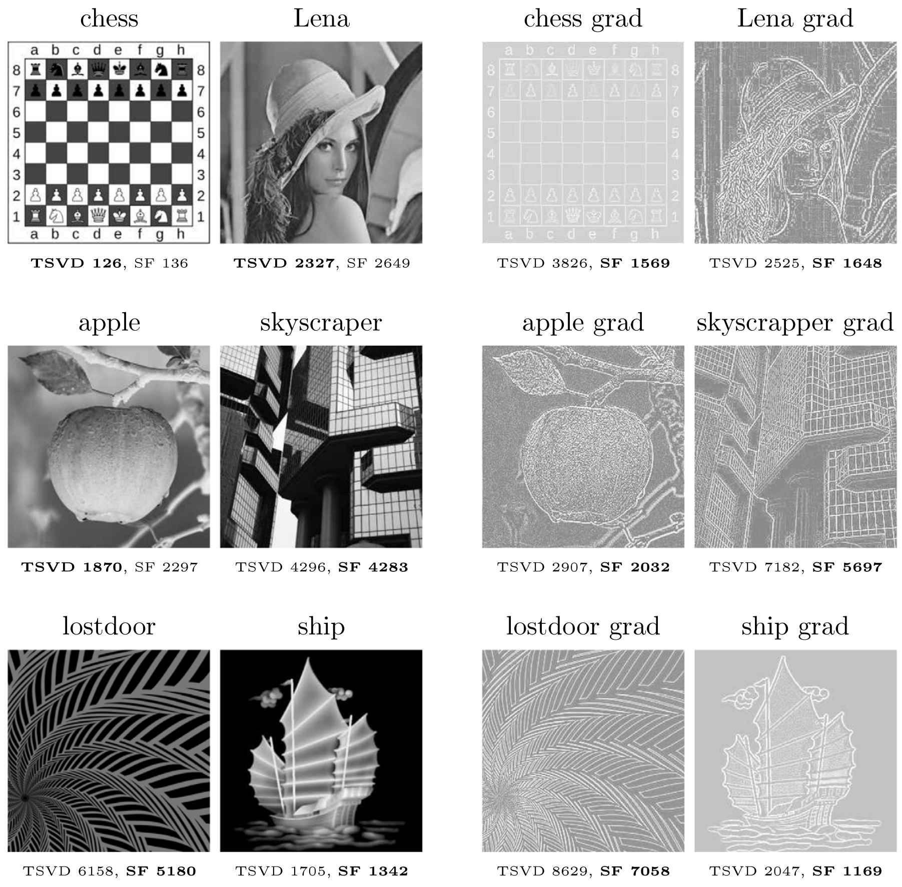


Fig. 4. Example square matrices: (left) original images and (right) the gradient magnitude images (displayed after histogram equalization for better visibility). Approximation errors by using TSVD and SF with the same number of non-zeros are shown below the images. Boldface font indicates the winner for each case.

composed of long sequences for supervised learning tasks. A similar experimental setup was used by Hochreiter and Schmidhuber (1997) for scalability tests. The details of the data sets and tasks are given below:

- *Adding Problem*. This is a sequence regression task. Each element of an input sequence is a pair of numbers (a_i, b_i), where $a_i \sim U(-1, 1)$, $b_i \in \{0, 1\}$, $i = 1, \ldots, N$. We generated signals at two randomly selected positions t_1 and t_2 such that $b_{t_1} = b_{t_2} = 1$ and $b_i = 0$ elsewhere. The learning target is $y = 0.5 + \frac{a_{t_1} + a_{t_2}}{4}$. For example, an input sequence $[(0.1, 0), (-0.4, 1), (0.3, 0), (-0.2, 0), (0.7, 1)]$ will have the learning target $y = 0.575$. Unlike Hochreiter and Schmidhuber (1997), we do not restrict the t_1 and t_2 choice to make the task more challenging. That is, the relevant signals can appear either locally or at a great distance from each other. In evaluation, a network prediction $\hat{y}$ is considered correct if $|y - \hat{y}| < 0.04$.
- *Temporal Order*. This is a sequence classification task. A sequence consists of randomly chosen symbols from the alphabet $\{a, b, c, d, X, Y\}$, where the first four are noise symbols. Each sequence has two signal symbols, either X or Y, which appear at two arbitrary positions. The four target classes correspond to the ordered combinations of the signal symbols (X, X), (X, Y), (Y, X), and (Y, Y). For example, an input sequence $[b, a, c, b, X, a, a, Y, b]$ should be classified as Class 2.

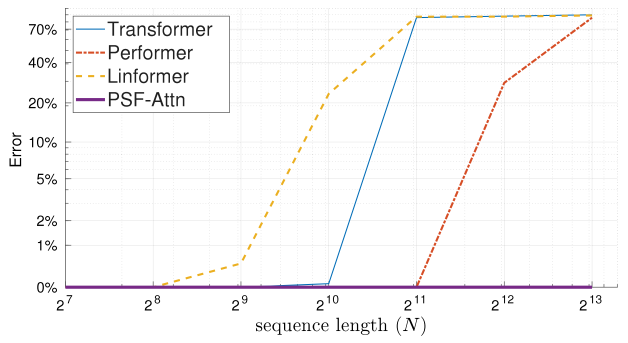
We prepared data of different sequence lengths for each problem. We started with $N = 128$ and gradually increased the length by the factor of two, up to $N = 2^{14}$. For every sequence length, we generated 200,000 training and 5000 testing instances.

We compared PSF-Attn with several popular methods based on scaled dot-product attention (referred to as X-former architectures). The first two, Linformer (Wang et al., 2020) and Performer (Choromanski et al., 2020) have also claimed to be scalable attention-based architectures. For completeness, we also included the original Transformer (Vaswani et al., 2017a). We have used their open-source PyTorch implementations.[4]

We followed standard cross-validation techniques to tune the main hyperparameters, such as the number of layers and heads, dimensionality of the token embedding, and query/key/value dimensions. For the Temporal Order problem, we directly fed the data instances to the embedding layers. For the Adding problem, the input data was only two-dimensional, and one of them was real-valued. Directly using such a low-dimensional embedding space would lead to poor attention. We augmented the dimensionality with a linear layer to assure sufficient freedom for the scaled dot-products in the X-former architectures. All the models were optimized using the Adam optimizer (Kingma & Ba, 2014) with the learning rate of 0.001 using batch size of 40.

The results are shown in Fig. 5. For the Adding problem, we see that all models work fine for short sequences (100% for $N \leq 256$). The X-former methods, however, turn worse or even useless when the sequences are longer. Linformer has an error rate of 0.48% for $N = 512$ and 86.18% for $N = 1024$, which is nearly as bad as random guessing (92%). Transformer becomes problematic (84.48% error) when $N \geq 2048$. Performer starts to get wrong when $N = 4096$, giving only 71.76% accuracy, and when $N =$

[4] Available at https://github.com/lucidrains/performer-pytorch and https://github.com/lucidrains/linformer.



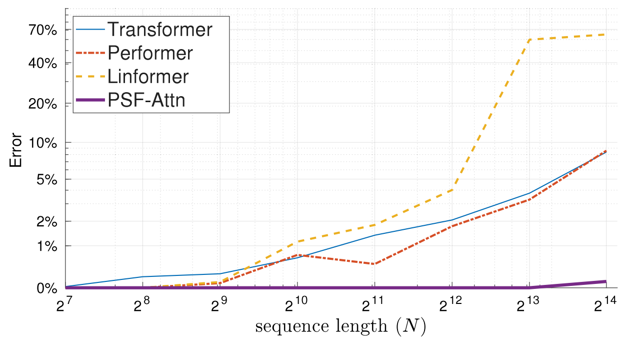


Fig. 5. Error percentage of PSF-Attn and the X-formers for (top) the Adding problem and (bottom) the Temporal Order problem with increasing sequence lengths.

Table 2
Area under the ROC curve (ROC-AUC) for the compared methods on the genome classification task.

Model	ROC-AUC
Transformer	85.42%
Linformer	86.30%
Performer	85.40%
PSF	**90.04%**

8192, its prediction becomes almost random guessing. In contrast, PSF-Attn achieves 100% accuracy for all the tested lengths.

A similar pattern holds for the Temporal Order problem. When $N \leq 512$, all compared models give perfect or nearly perfect predictions. With longer sequences, for example when $N = 4096$, the error rates of Transformer, Performer, and Linformer become 2.06%, 1.76%, and 4.02%, respectively. When $N = 16{,}324$, their error rates become 8.38%, 8.60%, and 64.22%, respectively. In contrast, PSF-Attn achieves 100% accuracy for $N \leq 8192$, and 99.89% accuracy for $N = 16{,}324$.

In summary, our method has better scalability than the three attention models based on scaled dot-products in terms of lower learning errors. PSF-Attn can still achieve nearly 100% prediction accuracy for an attention matrix size up to tens of thousands.

5.3. Genome classification

It is known that long-range interaction commonly exists in genome sequences (Avsec et al., 2021). Here we used a genome data set DDcDNA containing Complementary DNA (cDNA) sequences of dogs and donkeys[5] Each sample is composed of a set of four symbols: A, C, G, T. The task is to classify each cDNA sequence to its organism (dog or donkey). We removed the sequences shorter than 5k and thus got 6251 sequences for dogs and 2335 for donkeys. The mean length for all sequences is 6985. All the sequences are padded or truncated to a fixed length of 16,384. We used ROC-AUC as the comparison metric due to the class imbalance.

We compare the performance of PSF-Attn with Transformer, Linformer, and Performer on this task. The model hyperparameters were chosen according to the same procedure in Section 5.2. The results are shown in Table 2. We can see that Transformer and its variants show a similar mediocre performance. In contrast, PSF-Attn scores 90.04% ROC-AUC on DDcDNA, much better than the runner-up (Linformer) by 3.7 percentage points, which suggests that PSF-Attn can classify long DNA sequences more accurately.

5.4. Long range arena public benchmark

Next we provide the experimental results on Long Range Arena (LRA), a publicly available benchmark for modeling long sequential data (Tay et al., 2020). We select four tasks from LRA that cover various data types and demand flexible reasoning abilities of tested models.

- *ListOps*. This is a sequence classification task for measuring the ability of models to identify and parse hierarchically constructed data (Nangia & Bowman, 2018). We used an enlarged version of the original ListOps, with a max sequence length up to 2000 and tree depth up to 10 (Tay et al., 2020). Each element in a sequence can be an operator, a digit, and a left or right bracket. The brackets define lists of items. Each operator, MAX, MIN, MED, and SUM_MOD, takes the items in a list as input and returns a digit, where MED means median and SUM_MOD means summation followed by modulo 10. An example sequence [MAX 6 [MED 3 2 2] 8 5 [MIN 8 6 2]] has ground truth answer 8. A prediction is correct if an output value of a neural network matches the ground truth label. For good predictions, a model should access all sequence elements and identify the proper parsing structure.
- *Text Classification*. This is a binary sentiment classification task constructed from the IMDb reviews (Maas et al., 2011). Given a review as a sequence of characters, the goal is to classify it as positive or negative. Due to the character-level representation, the sequences are much longer than the word-level version used in conventional language modeling. We truncated or padded every sequence to a fixed length ($N = 4000$).
- *Image Classification*. This task is to classify images into one of ten classes. The images and class labels come from the grayscale version of CIFAR10. Each image is flattened to form a sequence of length 1024. Unlike conventional computer vision, the task requires the predictors to treat the grayscale levels (0–255) as categorical values. That is, each image becomes a sequence of symbols with an alphabet size of 256. Two example images and their class labels are shown in Fig. 6.
- *Pathfinder*. This is a binary classification task on synthetic images, which is motivated by cognitive psychology (Linsley, Kim, Veerabadran, Windolf, & Serre, 2018). Each image (size 32×32) contains two highlighted endpoints and some path-like patterns. Similar to the Image Classification task, the predictors must treat the pixels as categorical values and flatten the image to a sequence of length 1024. The task is to classify whether there is a path consisting of dashes between two highlighted points. Two example images and their classes are shown in Fig. 6.

[5] Downloaded from the Ensembl genome browser https://www.ensembl.org/.

Table 3
Classification accuracies by the compared methods for the four LRA tasks. A dash ("-") means the result is absent in the corresponding paper. For PSF-Attn, we present the mean (μ) and standard deviation (σ) across multiple runs in the $\mu \pm \sigma$ format.

Model	ListOps $N = 2000$	Text $N = 4000$	Image $N = 1024$	Pathfinder $N = 1024$
Transformer (Tay et al., 2020)	36.37	64.27	42.44	71.40
Transformer (Zhu et al., 2021)	37.13	65.35	–	–
Transformer (Xiong et al., 2021)	37.10	65.02	38.20	74.16
Sparse transformer (Tay et al., 2020)	17.07	63.58	44.24	71.71
Longformer (Tay et al., 2020)	35.63	62.58	42.22	69.71
Linformer (Tay et al., 2020)	37.70	53.94	38.56	76.34
Linformer (Zhu et al., 2021)	37.38	56.12	–	–
Linformer (Xiong et al., 2021)	37.25	55.91	37.84	67.60
Reformer (Tay et al., 2020)	37.27	56.10	38.07	68.50
Reformer (Zhu et al., 2021)	36.44	64.88	–	–
Reformer (Xiong et al., 2021)	19.05	64.88	43.29	69.36
Performer (Tay et al., 2020)	18.01	65.40	42.77	77.05
Performer (Zhu et al., 2021)	32.78	65.21	–	–
Performer (Xiong et al., 2021)	18.80	63.81	37.07	69.87
BigBird (Tay et al., 2020)	36.06	64.02	40.83	74.87
Linear transformer (Tay et al., 2020)	16.13	65.90	42.34	75.30
Transformer-LS (Zhu et al., 2021)	38.36	68.40	–	–
RFA-Gaussian (Peng et al., 2021)	36.80	66.00	–	–
Nyströmformer (Zhu et al., 2021)	37.34	65.75	–	–
Nyströmformer (Xiong et al., 2021)	37.15	65.52	41.58	70.94
PSF-Attn	**38.85**±0.1	**77.32**±0.3	**45.01**±0.2	**80.49**±0.1

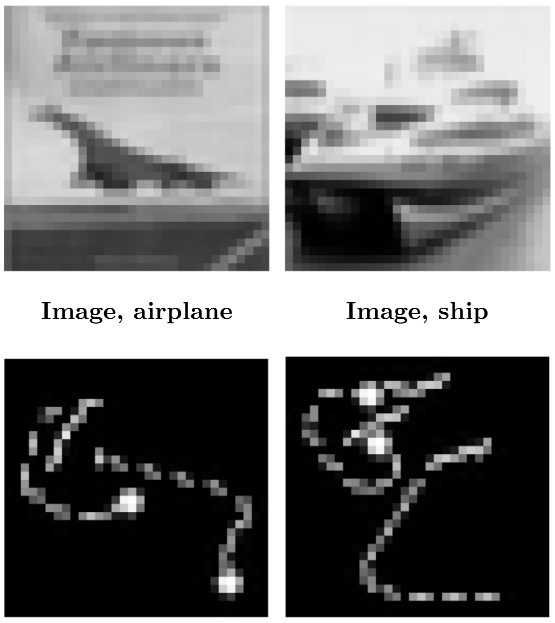


Fig. 6. Example matrices: (top two) from the Image Classification and (bottom two) from the Pathfinder tasks.

We have tested PSF-Attn in the above LRA learning tasks, where the hyperparameters in PSF-Attn were again tuned by cross-validation. No external data was used for pre-training. We ran PSF-Attn four times with a different random seed for each task. The mean and standard deviation across the multiple runs are reported in Table 3.

For comparison, we quote the prediction accuracies reported for many X-former methods in the literature, including Transformer (Vaswani et al., 2017b), Sparse Transformer (Child et al., 2019), Longformer (Beltagy, Peters, & Cohan, 2020), Linformer (Wang et al., 2020), Reformer (Kitaev, Kaiser, & Levskaya, 2020), Performer (Choromanski et al., 2020), Linear transformer (Katharopoulos et al., 2020), BigBird (Zaheer et al., 2020), Transformer-LS (Zhu et al., 2021), RFA-Gaussian (Peng et al., 2021), and Nyströmformer (Xiong et al., 2021). If a method has different implementations, we quote all variants and their results. We exclude results that rely on external data for pre-training.

We see that PSF-Attn wins all tasks by giving the best classification accuracy among all compared methods. Such strong cross-task wins suggest that our method usually provides better attention approximation than those based on scaled dot-products.

Remarkably, our method has substantially improved the state-of-the-art in the Text and Pathfinder tasks. For Text, PSF-Attn achieves 77.32% accuracy, compared to the runner-up 68.4% by Transformer-LS. Our method wins by 80.49% accuracy for Pathfinder, which gains about 5% higher than the best X-former (Linear Transformer 75.3%). The significant improvement brought by PSF-Attn is probably because the two tasks involve sparse attention matrices.

We also investigated whether the approximating attention $\widehat{X} = \prod_m W^{(m)}$ is meaningful by visualization. Fig. 7 shows an attention vector (absolute values) of token ["CLS"]. Tokens in the review having more weight are highlighted. We see that PSF-Attn has a good performance in capturing sparse attention and identify the relevant words.

Next, we visualize the attention values for four instances from the Pathfinder task. We extracted the attention matrix $\widehat{X}$ at test time and applied the following visualization procedure. We define the attention vector of the ith point as the ith row in $\widehat{X}$. We averaged the attention vectors of the endpoints and their direct neighbors and then reshaped the average vector to 32×32 for visualization.

The resulting visualizations are shown in Fig. 8. Good attention towards correct classification should trace the path between the endpoints and neglect non-relevant high intensities in the original image. It is clear PSF-Attn performs pretty well in this respect, where its attention values highlight mostly the relevant parts of the connecting path. In contrast, Transformer mainly highlights

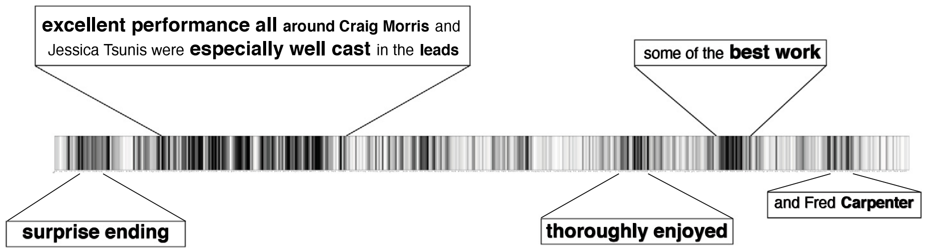


Fig. 7. Attention vector visualization of a positive review in Text Classification. Darker cells have larger absolute values.

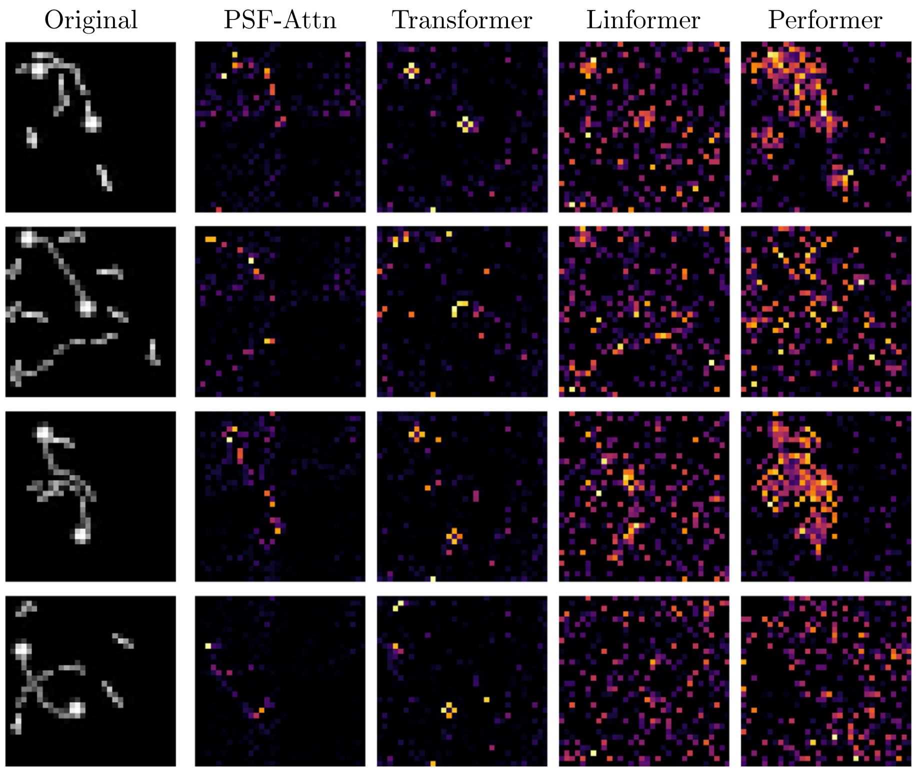


Fig. 8. Comparison of the attention maps in the Pathfinder task by using PSF-Attn, Transformer, Linformer, and Performer. The first column of images shows the original instances, and the other columns are heat maps of the attention values of the corresponding models. The procedure for visualization is the same for all the methods.

the area around the endpoints without the path connecting them. For Linformer and Performer, sometimes their attention maps basically smooth the original image intensities, including the irrelevant parts, and sometimes there is simply no pattern. More instances are provided in the supplemental material.

6. Conclusion

We have proposed a new approximation method called Sparse Factorization for square matrices by using a product of sparse matrices. Given the same budget of non-zero numbers, our method has shown superior performance over conventional low-rank matrix factorization approaches, especially in cases where the approximated matrix is sparse, high-rank, or contains high-frequency details. We have also given parametric design and performed an empirical study on the classification of long sequential data. As the critical attention component, our method has demonstrated clear wins over the conventional scaled dot-product transformer and its several variants in terms of scalability and accuracy.

We have employed the Chord protocol to fix the non-zero positions in the sparse factorizing matrices. Later we could consider the other predefined protocols or even adaptively learned protocols for the sparse structure. In this work, we have considered approximating unconstrained square matrices. In the future, we could investigate the approximation to more specific matrices such as non-negative, stochastic, symmetric, or semi-definite matrices. As a result, we could extend the method for further applications such as large-scale graph matching, Gaussian Process, or network structure identification.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was financially supported by The Research Council of Norway, Grant Number 287284. We acknowledge for using the IDUN computing cluster (Själander et al., 2019).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.neunet.2022.04.014.

References

Avsec, Ž., Agarwal, V., Visentin, D., Ledsam, J. R., Grabska-Barwinska, A., Taylor, K. R., et al. (2021). Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, *18*(10), 1196–1203.

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150.

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., et al. (2020). Rethinking attention with performers. arXiv:2009.14794.

Correia, G. M., Niculae, V., & Martins, A. F. (2019). Adaptively sparse transformers. arXiv preprint arXiv:1909.00015.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Donoho, D. L. (2006). For most large underdetermined systems of linear equations, the minimal l_1-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics*, *59*(6), 797–829.

Drineas, P., & Mahoney, M. W. (2005). On the nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, *6*, 2153–2175.

Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, *1*(3), 211–218.

Funk, S. (2006). Funk matrix factorization. Winner solution of the Netflix Prize, https://sifter.org/~simon/journal/20061211.html.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning* (pp. 5156–5165). PMLR.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Kitaev, N., Kaiser, Ł., & Levskaya, A. (2020). Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451.

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by nonnegative matrix factorization. *Nature*, *401*, 788–791.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., et al. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in neural information processing systems, vol. 32*.

Linsley, D., Kim, J., Veerabadran, V., Windolf, C., & Serre, T. (2018). Learning long-range spatial dependencies with horizontal gated recurrent units. In *Proceedings of the 32nd international conference on neural information processing systems* (pp. 152–164).

Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies* (pp. 142–150).

Mahoney, M. W., & Drineas, P. (2009). CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, *106*(3), 697–702.

Nangia, N., & Bowman, S. R. (2018). Listops: A diagnostic dataset for latent tree learning. arXiv preprint arXiv:1804.06028.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., & Kong, L. (2021). Random feature attention. arXiv preprint arXiv:2103.02143.

Själander, M., Jahre, M., Tufte, G., & Reissmann, N. (2019). EPIC: an energy-efficient, high-performance GPGPU computing research infrastructure. arXiv preprint arXiv:1912.05848.

Slawski, M., Hein, M., & Lutsik, P. (2013). Matrix factorization with binary components. In *Advances in neural information processing systems, vol. 26*.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, *31*(4), 149–160.

Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., & Zheng, C. (2021). Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning* (pp. 10183–10192). PMLR.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., et al. (2020). Long range arena: A benchmark for efficient transformers. arXiv preprint arXiv:2011.04006.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017a). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017b). Attention is all you need. In *Advances in neural information processing systems, vol. 30*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. arXiv:2006.04768.

Williams, C., & Seeger, M. (2001). Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems, vol. 13*.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., et al. (2021). Nyströmformer: A nyström-based algorithm for approximating self-attention. arXiv preprint arXiv:2102.03902.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., et al. (2020). Big bird: Transformers for longer sequences. In *NeurIPS*.

Zhang, Z., Li, T., Ding, C., & Zhang, X. (2007). Binary matrix factorization with applications. In *Proceedings of international conference on data mining* (pp. 391–400).

Zhu, C., Ping, W., Xiao, C., Shoeybi, M., Goldstein, T., Anandkumar, A., et al. (2021). Long-short transformer: Efficient transformers for language and vision. arXiv preprint arXiv:2107.02192.