PARAMIXER: PARAMETERIZING MIXING LINKS IN SPARSE FACTORS WORKS BETTER THAN DOT-PRODUCT SELF-ATTENTION

A PREPRINT

Tong Yu, * **Ruslan Khalitov,** * **Lei Cheng, Zhirong Yang** †
Norwegian University of Science and Technology

ABSTRACT

Self-Attention is a widely used building block in neural modeling to mix long-range data elements. Most self-attention neural networks employ pairwise dot-products to specify the attention coefficients. However, these methods require $O(N^2)$ computing cost for sequence length N. Even though some approximation methods have been introduced to relieve the quadratic cost, the performance of the dot-product approach is still bottlenecked by the low-rank constraint in the attention matrix factorization. In this paper, we propose a novel scalable and effective mixing building block called Paramixer. Our method factorizes the interaction matrix into several sparse matrices, where we parameterize the non-zero entries by MLPs with the data elements as input. The overall computing cost of the new building block is as low as $O(N \log N)$. Moreover, all factorizing matrices in Paramixer are full-rank, so it does not suffer from the low-rank bottleneck. We have tested the new method on both synthetic and various real-world long sequential data sets and compared it with several state-of-the-art attention networks. The experimental results show that Paramixer has better performance in most learning tasks.[3]

Keywords parameterization · mixing links · sparse · matrix factorization

1 Introduction

Transformer models have been widely used on many tasks such as text classification [1], text summarization, promoter region prediction [2], and image classification [3]. The main engine in Transformer is the self-attention mechanism, which can work in parallel to mix long-range tokens in a long sequence. This fundamental innovation eliminated the sequential dependency in recurrent neural networks and was used as a building block for many powerful models, such as Bert [4], GPT[5] and Ernie[6].

However, the original self-attention is not scalable because it requires computing and storing all pairwise dot-products, which incurs $O(N^2)$ cost for sequence length N. The scalability issue significantly restricted the application of neural models based on self-attention.

Various methods have been introduced to alleviate the quadratic cost of full attention. Some of them attempt to shorten the sequence length [7, 8], even though much information is lost. Others try to break up the softmax by a certain kernel factorization. Another family of methods sparsify the attention matrix with predefined attention [2, 9, 10, 11, 12]. However, most Transformer variants stick to the dot-product self-attention, of which the expressive power is restricted by the low-rank bottleneck [13] because the dimensionality of the dot-product space is much smaller than the sequence length. Therefore, they cannot accurately model the transformation if the attention is intrinsically high-rank.

This paper proposes a scalable and effective attention building block called Paramixer without dot-product and softmax. Our method directly parameterizes the mixing links in several sparse factors to form an attention matrix, where all

---

factorizing matrices are full-rank. Therefore Paramixer does not suffer from the low-rank bottleneck. We present two ways to specify the non-zero positions in each sparse factor. Both lead to an economical approximation of the full attention matrix, with the computing cost as low as $O(N \log N)$. As a result, our method can easily model very long sequential data.

We have tested Paramixer on various sequence data sets and compared it with many popular self-attention neural networks based on dot-products. The experimental results show that Paramixer gets the best performance on very long sequence tasks, including synthetic data inference, Genome classification, and character-level long document classification. Paramixer also achieves state-of-art accuracy on the public Long Range Arena benchmark tasks.

We organize the rest of the paper as follows. Section 2 investigates dot-product self-attention and its related work. Section 3 introduces the development clue and model architecture of Paramixer. The experimental settings and results are presented in Section 4, and we conclude the paper in Section 5.

2   Related Work

Self-attention (SA) is a building block in neural networks which enables long range interaction between elements in a sequence. The most widely used SA architecture is called Transformer [1], where the self-attention matrix is constructed using scaled dot-product followed by softmax. Given an input sequence of N elements encoded in $X \in \mathbb{R}^{N \times d}$, a self-attention building block calculates a weighted average of feature representations $V \in \mathbb{R}^{N \times d_v}$, where the weights are the result of scaled dot-product of $Q \in \mathbb{R}^{N \times D}$ and $K \in \mathbb{R}^{N \times D}$: $Q = XW_q$, $K = XW_k$, and $V = XW_v$. Then the self-attention in Transformer is

$$\text{Self-Attention}(Q, K, V) = AV, \tag{1}$$

where

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right) \tag{2}$$

and the softmax applies row-wise on the scaled dot-products.

The attention matrix in Eq. 2 is not scalable because it requires computing and storing N^2 attention values, which is infeasible for a large N. The quadratic cost becomes a significant bottleneck when applying self-attention applications for long sequences. Many research teams have proposed Transformer variants to relieve this problem.

The first branch of methods attempts to reduce N. The Linformer approximation [14] uses random projection to reduce the rows of K and V from N to r with $r < N$. However, because $r \propto \epsilon^{-2}$ with ϵ the approximation error bound, Linformer has to use a large r to achieve satisfactory approximation quality. Another method called Enformer [8] shortens the sequence length by convolutional network pooling.

The second branch of methods tries to break up the softmax by a certain kernel factorization $A \approx \phi(Q)\phi(K)^T$, where ϕ_Q and $\phi_K \in \mathbb{R}^{N \times r'}$ with $r' < N$. Then by the association rule of multiplication, we can calculate $\phi(Q)\left[\phi(K)^T V\right]$ and avoid the quadratic cost. For example, Nyströmformer [15] uses a few landmarks as surrogates to construct $\phi(Q)$ and $\phi(K)$. Linear Transformer [16] directly chooses $\phi(x) = \text{elu}(x) + 1$. Performer [17] uses r' orthogonal random features to obtain $\phi(Q)$ and $\phi(K)$. Random features were also used in another work [18], with gating mechanism combined.

The third branch of methods chooses only a subset of (i, j) pairs in each attention layer. These include Sparse Transformers [10, 11] that use a set of neighboring tokens, Sinkhorn Transformer [12] that uses blockwise sparsity, Longformer [9] that uses dilated sparse connections, and BigBird [2] that uses both blockwise and dilated sparse connections.

3   Paramixer

Despite many variants of Transformers, there are still several drawbacks because they stick to dot-product + softmax or their approximation. Below we discuss the disadvantages of these two components and propose our solution without them.

3.1   Drawbacks of dot-products and softmax

The first drawback is the low-rank bottleneck: dot-product self-attention requires $D \ll N$; otherwise, the Q and K matrices are unaffordable for a large N. Bhojanapalli et al. have shown that the low-rank bottleneck restricts the

Figure 1: Illustration of (a & b) the CHORD and (c & d) the CDIL protocols for $N = 16$. Each node in the circular graph represents a sequence element. The links between nodes correspond to the non-zero entries in $W^{(m)}$ (here $m = 1$) output from $f^{(m)}$. Note that the sparse structure of all factors in CHORD is the same, while it varies at different m's in CDIL.

representation power and attention performance [13]. However, their workaround that uses $D = N$ does not apply for long sequences due to the quadratic cost.

Another limitation comes from the pairwise definition of dot-product. Although dot-products have a theoretical connection to Reproducing Kernel Hilbert Space (RKHS), self-attention cannot benefit from more than two factors in the matrix product because the kernels are defined in pairs.

Another key component, softmax, in Transformer is also problematic. First, such nonlinearity over dot-products leads to quadratic cost. Comprehensive approximation methods are required to approximate softmax for more economical matrix products.

Second, softmax limits the mixing capability. The softmax layer output probabilities and the mixing result are thus constrained in the convex hull of the existing elements. The constraint is more severe in sparse attention methods such as [2, 9, 10, 11, 12], where the mixing result is in a rather constrained convex hull of a few involved elements.

Moreover, softmax is incompatible with sparse attention. The latter was introduced to relieve the quadratic computing cost caused by softmax. However, after each sparse attention layer with softmax, the network can output probabilities for only a few sequence elements.

3.2 Parameterizing mixing links

Seeing the drawbacks of dot-products and softmax, we rethink self-attention as a transformation block and redesign the neural model. First, we drop the softmax because it is not compulsory for the mixing function but limits the mixing capability. Then the transformation becomes $V^{\text{new}} = AV$ for an unconstrained mixing matrix A.

Next, we consider parameterizing the mixing links or coefficients for each matrix row $A_{i:} = f(X_i; \theta)$, where $f : \mathbb{R}^d \mapsto \mathbb{R}^N$ is a neural network with weights θ. However, such simple parameterization does not solve the quadratic cost. We thus consider a sparse factorization of the mixing matrix A:

$$A = \prod_{m=1}^{M} W^{(m)}, \tag{3}$$

where each sparse factor $W^{(m)}$ is a full-rank sparse square matrix. Then we parameterize each sparse factor $W^{(m)}$ as

$$W_{i:}^{(m)} = f^{(m)}(X_i; \theta_m), \tag{4}$$

where $i = 1, \dots, N$ and $f : \mathbb{R}^d \mapsto \mathbb{R}^K$ is a Multilayer Perceptron (MLP) that outputs the K non-zero entries[4] of each row in $W^{(m)}$. If the total number of non-zeros in the factors is much smaller than N^2, we obtain a new economical self-attention method without dot-product and softmax. We call the new method Paramixer.

There are different ways or protocols to specify the sparse structure or the non-zero entries. We have studied two protocols and present them below. For short, we abbreviate $f_{ik}^{(m)} \stackrel{\text{def}}{=} \left[f^{(m)}(X_i; \theta_m) \right]_k$.

---

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

Figure 2: Illustration of an example Paramixer neural network ($M = 4$). After adding the positional embedding, it applies L Paramixer blocks and obtains the transformed tensor X_{new}.

The first protocol CHORD modifies from an algorithm with the same name in peer-to-peer lookup service [19], where the i-th row of each sparse factor $W^{(m)}$ is parameterized as

$$
W_{ij}^{(m)} = \begin{cases} f_{i1}^{(m)} & \text{if } j = i \\ f_{ik}^{(m)} & \text{if } j = i + 2^{k-2} \mod N \\ 0 & \text{otherwise.} \end{cases}
\tag{5}
$$

where $j = 1, \ldots, N$ and $k = 1, ..., K$. That is, each row has K non-zeros, and in total all $W^{(m)}$'s have MNK non-zeros. In this work, we follow the original CHORD algorithm to use $K = M = \log N$. Therefore the number of stored entries is $N \log^2 N$.

Each factor $W^{(m)}$ can be treated as the adjacency matrix of a directed graph, where the (i, j)-th non-zero entry can be seen as a link from i to j. The graph visualization and the parameterization are illustrated in Figure 1.

The product of the factorizing matrices corresponds to the connections in the circular graph after multiple sparse factors. Theorem 2 in [19] guarantees that the graph will become complete with high probability after $M = \log N$ sparse factors. That is, the resulting A matrix will become full after the matrix product.

The second protocol CDIL (Circular DILated) originates from Temporal Convolution Networks (TCN) [20], where we modify the dilated connections at both sides and along a circular graph to ensure the receptive fields are symmetric. The i-th row of $W^{(m)}$ is parameterized as

$$
W_{ij}^{(m)} = \begin{cases} f_{i1}^{(m)} & \text{if } j = i \\ f_{ik}^{(m)} & \text{if } j = i + p_{k-1} 2^m \mod N \\ 0 & \text{otherwise,} \end{cases}
\tag{6}
$$

where $p = \left[1, 2, ..., \frac{K-1}{2}, -1, -2, -\frac{K-1}{2} \right]$ and $k = 2, \ldots, K$. Similar to CHORD, the CDIL protocol includes self links but the other links appear on both sides, and the dilation 2^m varies at different m's. The CDIL protocol and parameterization are illustrated in Figure 1.

In CDIL, each factor contains KN non-zero entries, and in total, there are $KN \log N$ non-zeros if $M = \log N$, which is more economical than CHORD if $K < \log N$. It can be proven that the product $A = \prod_{m=1}^{M} W^{(m)}$ is a full matrix with the CDIL protocol.

The following proposition states that the factorizing matrices constructed by the two protocols are full-rank. The proof in given the supplemental document.

Proposition 1. $W^{(1)}, \ldots, W^{(M)}$ constructed in Eq. 5 or Eq. 6 are in general full-rank.

3.3 Paramixer neural networks

A Paramixer block transforms an $N \times d$ tensor X to another tensor of the same size. See Figure 2. Here we use an MLP $g : \mathbb{R}^d \mapsto \mathbb{R}^d$ with weights ζ to mix the columns of the input tensor, which is similar to the product with W_v in

Transformer. In this work, we used simple three-layer MLPs (Linear-GELU-Linear) for both $f^{(m)}$'s and g. Stacking L such blocks forms a neural network backbone, denoted by ParamixerNet, for representation learning.

We have tried two options for parameterization in multi-block setting: $W^{(l,m)} = f^{(l,m)}\left(X^{(l-1)}; \theta_m^{(l)}\right)$ and $W^{(l,m)} = f^{(l,m)}\left(X^{(0)}; \theta_m^{(l)}\right)$, where the superscript l indexes the l-th block and $X^{(l-1)}$ is the input to the l-th block, with $X^{(0)} = X + \text{PE}$. We find that the latter option makes the network easier to train and works better.

The ParamixerNet output can then be fed to a loss function $\mathcal{J}$, and overall, the learning task can be formulated as the following optimization problem:

$$\underset{\Theta}{\text{minimize}} \quad \mathcal{J}(\text{ParamixerNet}(X + \text{PE}; \Theta)), \tag{7}$$

where $\Theta = \left\{\theta_1^{(l)}, \dots, \theta_M^{(l)}, \zeta^{(l)}\right\}_{l=1}^{L}$. The optimization can be implemented with back-propagation, and a gradient-based algorithm such as Adam [21].

4 Experiments

We conducted four groups of experiments. In the first group, we demonstrate the scalability of Paramixer on long synthetic sequences with lengths up to tens of thousands of positions. Then, we tested the performance of Paramixer on the pubic Long Range Arena benchmark data sets. In the third group, we built a character-level document classification task to evaluate if Paramixer can handle real-world long text sequences with tens of thousands of tokens on average. Finally, we showcase if Paramixer performs well in modeling long genome sequences. We ran all experiments on a Linux machine with 3×NVIDIA Tesla V100 32GB, Intel Xeon Gold 6240 CPU @ 2.60GHz processors, with 754GB of system memory.

4.1 Synthetic Scalability Test

In this section, we examine the scalability of Paramixer and compare its performance with several competitors. We used two synthetic data sets composed of long sequences for supervised learning tasks. An experimental setup was inspired by [22], where similar synthetic sequences appeared for scalability tests. The details of both 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 $a_i \sim U(-1, 1)$, $b_i \in \{0, 1\}$, $i = 1, \dots, N$. 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 $y = 0.5 + \dfrac{a_{t_1} + a_{t_2}}{4}$. For example, an input sequence $[(0.5, 1), (-0.2, 0), (0.2, 1), (-0.8, 0), (0.6, 1)]$ will have the learning target $y = 0.825$. Unlike [22], we did not restrict the t_1 and t_2 choice and made 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. Each 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 $[a, d, Y, c, b, a, Y, c, d]$ should be classified as Class 3.

We generated data of different sequence lengths for each problem: from $N = 128$ to $N = 2^{15}$, progressively increasing the length by the factor of two. For each sequence length, a model can access $100\,000$ training sequences and $5\,000$ testing instances for evaluation.

We compared Paramixer with a group of popular methods based on scaled dot-product attention (referred as X-formers), including Linformer [7], Performer [17], Reformer [23] and Nyströmformer, which all [15] have claimed to be scalable. For completeness, we also included the original Transformer [1]. We used the open-source PyTorch implementations[5] of these models.

We fine-tuned the main hyperparameters in a standard cross-validation manner for Paramixer and X-formers, including 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

---
[5]available at `https://github.com/lucidrains`

Figure 3: Error percentage of Paramixer and the X-formers for both the Adding problem and the Temporal Order problem with increasing sequence lengths.

Image, automobile   **Image, horse**   **Pathfinder, Negative**   **Pathfinder, Positive**

Figure 4: Example data from the Image Classification (left two) and Pathfinder tasks (right two).

data was only two-dimensional, and one of them was real-valued. Directly using such a low-dimensional embedding space would limit the expressive power. So we added a linear layer to augment the dimensionality to allow sufficient freedom for the scaled dot-products in the X-former architectures. All the models were optimized using the Adam optimizer [21] with the learning rate of 0.001 using a batch size of 40.

The results are shown in Figure 3. For the Adding problem, we see that all models obtain 100% accuracy for $N \leq 256$. The X-former models become weak or ineffective when scaling to longer sequences, while Paramixer still performs well. Linformer and Reformer get an error rate of 33.38% and 30.20% for $N = 1024$, and become invalid on $N = 2048$. Transformer starts to lose some performance (99.94%) on $N = 1024$, and becomes not working when $N = 2048$. Performer survives when $N \leq 2048$, however, gets an error rate of 29.24% for $N = 4096$ and becomes as bad as random guessing for $N \geq 8192$. Nyströmformer achieves 100% accuracy for $N = 4096$ and fails on $N \geq 8192$. Instead, Paramixer reaches 100% accuracy for all the tested lengths.

For the Temporal Order problem, the performance of all the models is 100% or nearly 100% when $N \leq 256$, and starts to drop for $N \geq 512$. When $N = 2048$, the results of Performer (99.52%) and Nyströmformer (100%) are still solid while Transformer, Linformer, and Reformer have an error rate around 1.4%. When increasing N to 4,096 and 8,192, Linformer and Transformer fail in succession. When $N = 16384$, Reformer does not work while Performer and Nyströmformer give a weaker performance with error rates, 7.6% and 7.9%, respectively. On the contrary, Paramixer achieves 100% accuracy for $N \leq 8192$, and 98.84% accuracy for $N = 16384$.

In summary, Paramixer has better scalability than the attention neural networks based on scaled dot-products. When scaled to tens of thousands, Paramixer still gives very high prediction accuracy. The results suggest we can evaluate our model on more real-world tasks with very long sequences.

4.2 Long Range Arena Public Bechmark

Next, we evaluate the performance of Paramixer on Long Range Arena (LRA), a publicly available benchmark for modeling long sequential data [24]. The details of the tasks are the following:

- *ListOps*. ListOps is a classification task designed for measuring the ability of models to parse hierarchically constructed data [26]. Each sequence is composed of operators, digits, and left or right brackets. The brackets define lists of items. Each operator in a sequence takes the items in a list as input and returns a digit.

- *Text Classification*. We use the IMDb Review dataset [27] which requires the model to classify each review as positive or negative. The task uses a character-level representation for each sequence, which makes the

Table 1: Classification accuracy by Paramixer and X-formers on the four LRA tasks. Methods that are absent in the corresponding paper are signed by a dash ("-"). For Paramixer, 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 [24]	36.37	64.27	42.44	71.40
Transformer [25]	37.13	65.35	-	-
Transformer [15]	37.10	65.02	38.20	74.16
Sparse Transformer [24]	17.07	63.58	44.24	71.71
Longformer [24]	35.63	62.58	42.22	69.71
Linformer [24]	37.70	53.94	38.56	76.34
Linformer [25]	37.38	56.12	-	-
Linformer [15]	37.25	55.91	37.84	67.60
Reformer [24]	37.27	56.10	38.07	68.50
Reformer [25]	36.44	64.88	-	-
Reformer [15]	19.05	64.88	43.29	69.36
Performer [24]	18.01	65.40	42.77	<u>77.05</u>
Performer [25]	32.78	65.21	-	-
Performer [15]	18.80	63.81	37.07	69.87
BigBird [24]	36.06	64.02	40.83	74.87
Linear Transformer [24]	16.13	65.90	42.34	75.30
Transformer-LS [25]	<u>38.36</u>	68.40	-	-
RFA-Gaussian [18]	36.80	66.00	-	-
Nyströmformer [25]	37.34	65.75	-	-
Nyströmformer [15]	37.15	65.52	41.58	70.94
Paramixer (CHORD)	**39.57**±0.32	83.12±0.33	<u>45.01</u>±0.21	**80.49**±0.13
Paramixer (CDIL)	37.78±0.28	**83.32**±0.19	**46.58**±0.05	67.13±0.42

> tasks more challenging than the word-level version. We truncated or padded every sequence to a fixed length ($N = 4k$).

- *Image Classification.* This task is to classify images into one of ten classes. 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 matrices are shown in Figure 4.

- *Pathfinder.* This task is motivated by cognitive psychology [28], and constructed using synthetic images. Each image (size 32×32) contains two highlighted endpoints and some path-like patterns. The models need to classify whether there is a path consisting of dashes between two highlighted points. Similar to the Image Classification task, the predictors must flatten the image to a sequence of symbols with length 1024. Two example matrices are shown in Figure 4.

For a fair comparison, we followed the experiment settings in the original paper [24] and evaluated multi-block ParamixerNet in the above tasks. We constructed Paramixer in variable blocks and used cross-validation to report the model with the best hyperparameters. We ran the model four times with a different random seed for each task.

We compared Paramixer with many X-former architectures in prediction accuracies. If a method has different implementations, we quote all the alternatives and their results. The results show that Paramixer beats all the other self-attention-based transformers on all the tasks, having the best classification accuracy. Such stable cross-task wins suggest that forming an attention matrix with parameterizing mixing links works better than those based on scaled dot-products.

Remarkably, Paramixer has achieved state-of-the-art performance on Text and Pathfinder tasks. For Text Classification, ParamixerNet achieves 83.32%, which is 14.92 p.p. higher than the best Transformer variant, Transformer-LS (68.4%). Our method also wins with the accuracy of 80.49% on Pathfinder, where it gains about 5 percentage points higher than the runner-up model. The compelling improvement brought by ParamixerNet is a probable cause of the high-rank nature of Natural Language [29].

Table 2: Test accuracies on long document classification.

Model	$N = 16k$	$N = 32k$
Transformer	25.62	25.62
Linformer	64.69	65.36
Performer	25.62	25.20
Reformer	25.04	25.04
Transformer-LS	70.25	60.93
Nyströmformer	71.70	67.69
Paramixer	**83.89**	**84.55**

For Paramixer, we reported two variants for this task, using the CHORD and CDIL protocols independently. As shown in Table 1, CHORD wins on ListOps and Pathfinder, while CDIL gains the best results on Text and Image. However, CHORD still has a competitive accuracy on Text (83.12%) and Image (45.01%), which demonstrates that Paramixer has more stable performance using the CHORD protocol. Consequently, we used CHORD as a default protocol for the following experiments.

4.3 Long Document Classification

The goal of this subsection is to study the benefits of modeling long sequences in NLP tasks. The character-level text classification task from the LRA benchmark is not long enough to conclude this pattern, so we seek for Longer Document Data with tens of thousands of tokens.

Long-document-dataset is a publicly available dataset. It contains a collection of academic papers, which are parsed using the arXiv sanity preserver program and published on Github [30]. There are eleven diverse research areas a paper may belong to. Similar to the source article, we used only four classes of documents: cs.AI, cs.NE, math.AC, math.GR, having a final set of 11 956 documents in total. We used 70% of the data for training, 20% for validation, and 10% for the test. Unlike the original study [30], we transformed each article into a sequence of characters and finally got a challenging task with an average training sequence length of 52 112.

We truncate every sequence to a fixed length. Since NLP task is known to benefit from using longer sequence length, we tested Paramixer on sequence lengths of 16 384 and 32 768 to see if Paramixer can be in favor of using longer context. We studied other transformer-like variants as well.

As shown in Table 2, Paramixer outperforms the follow-up transformer-based competitors by 12.19 and 16.86 percentage points on two sequence lengths, respectively. Paramixer achieves higher accuracy when using 32k tokens, which signals a better generalization from using more extended context. Among all the competitors, only Linformer, Transformer-LS, and Nyströmformer can handle the sequences of this length to a certain degree. Nevertheless, the gap in performance between sequence lengths 32k and 16k is severe, suggesting that the low-rank factorization weakens the prediction power on high-rank NLP tasks.

4.4 Genome Classification

Inspired by a recent rise in using deep learning models in biological applications, such as Chromatin-profile prediction [2] and genome analysis [8]. However, directly processing genome and protein sequences with tens of thousands of positions are problematic. The standard preprocessing pipeline includes chunking a long sequence into many fragments, which are modeled independently. This approach is associated with inevitable information loss caused by the long-distant nature of interaction processes in the DNA sequences [31]. We tested our design on this type of data. By design, Paramixer can treat a full sequence as an input without the need for its preliminary segmentation, allowing it to capture highly non-local effects.

We built two data sets for this task. The first group of DNA sequences were downloaded from NONCODEv6 [32][6]. We used human and fruitfly DNA sequences to construct a binary-classification task. We filtered out the sequences shorter than 5k and thus got 9 536 human DNA sequences and 4401 fruitfly DNA sequences with mean sequence lengths 10 586 and 9 793, respectively. We name the data set HFDNA. As shown in Figure 5, the two species in HFDNA have similar sequence length distributions, which means they can not be directly distinguished based on a sequence length

---

[6]http://www.noncode.org/download.php

Figure 5: Sequence length distributions for HFDNA (top) and MTcDNA (bottom). X-axis is the range of genome sequence length, and y-axis is the percentage of total instance numbers for each bin.

Table 3: The performance of Paramixer in comparison to Xformers on the Genomic classification task. The metric is Area Under the Receiver Operating Characteristic Curve (ROC AUC) $\times 100\%$

Model	HFDNA	MTcDNA
Transformer	94.32	79.38
Linformer	91.65	77.06
Performer	93.46	78.92
Reformer	92.32	74.94
Transformer-LS	93.19	75.81
Nyströmformer	92.26	71.98
Paramixer	**100.00**	**84.86**

threshold. We split the data set to 60% training, 20% validation, and 20% test. Each sequence is either padded or truncated to a fixed length of 16 384.

The other data set, namely MTcDNA (mouse and turtles' cDNA), was built following the same preprocessing/split strategy. The original cDNA sequences were downloaded from Ensembl genome browser[7] [33, 34]. The task is to predict a binary class, given either mouse or turtles cDNA sequence. The mouse class presented in 12 300 cDNA sequences with a mean sequence length of 7 235, including two sub-species: *Mus musculus* and *Mus spretus*. The turtle class contains 4 193 cDNA sequences of *Chelonoidis abingdonii* and *Gopherus agassizii* with a mean sequence length of 7 072. The percentage histograms of sequence lengths are shown in Figure 5. It is clearly seen that the two species have a big overlap in terms of sequence length, which makes it hard to discriminate between them only by length.

We compared ParamixerNet with several transformer-based models on this task. Because both data sets are imbalanced, so we report ROC AUC values for this experiment. As shown in Table 3, Paramixer achieves 100% ROC AUC on HFDNA, outperforming the runner-up result from Transformer (94.32%). Notably, our design holds the best score at 84.86% ROC AUC on MTcDNA, which is higher than Transformer by 5.48 percentage points. The experimental results show that Paramixer can successfully handle long DNA sequences and has the potential to assist in the biological applications that require modeling long-distant gene interactions.

5 Conclusion

We have proposed a scalable and effective building block called Paramixer for attention neural networks. Our method replaced the dot-products and softmax with parameterization of the mixing links in full-rank sparse factors of the attention matrix and thus got rid of the low-rank bottleneck in most existing attention models. Our method has complexity as low as $O(N \log N)$ and can efficiently deal with sequences up to tens of thousands. Besides scalability, Paramixer has also demonstrated strong performance in terms of accuracy. Neural networks, by stacking the proposed Paramixer blocks, have defeated Transformer and many of its variants in a variety of tasks, including synthetic

---

[7]http://www.ensembl.org/info/data/ftp/index.html

data inference, the public Long Rang Arena benchmark, and classification of very long text documents and genome sequences.

In the future, we could study other applications beyond classification, for example, gene expression prediction from sequence and pretraining with unsupervised data. The basic Paramixer block could be extended using other existing attention techniques such as multiple heads and relative positional encoding. Later, in addition to the CHORD and CDIL protocols, we could consider the other predefined protocols or even adaptively learned protocols for the sparse structure.

References

[1] 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*, pages 5998–6008, 2017.

[2] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[6] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.

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

[8] Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, 18:1196–1203, 2021.

[9] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[10] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers. *arXiv preprint arXiv:2004.08483*, 2020.

[11] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[12] Yi Tay, Aston Zhang, Luu Anh Tuan, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and efficient neural natural language processing with quaternion networks. *arXiv preprint arXiv:1906.04393*, 2019.

[13] Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Low-rank bottleneck in multi-head attention models. In *International Conference on Machine Learning*, 2020.

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

[15] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nystr\" omformer: A nystr\" om-based algorithm for approximating self-attention. *arXiv preprint arXiv:2102.03902*, 2021.

[16] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[17] 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, 2020.

[18] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

[19] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[20] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. In *ECCV 2016 Workshops*, pages 47–54, 2016.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[24] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[25] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. *arXiv preprint arXiv:2107.02192*, 2021.

[26] Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018.

[27] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[28] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charlie Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 152–164, 2018.

[29] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

[30] Jun He, Liqun Wang, Liu Liu, Jiao Feng, and Hao Wu. Long document classification from local word glimpses via recurrent attention learning. *IEEE Access*, 7:40707–40718, 2019.

[31] Molly Gasperini, Jacob M Tome, and Jay Shendure. Towards a comprehensive catalogue of validated and target-linked human enhancers. *Nature Reviews Genetics*, 21(5):292–310, 2020.

[32] Lianhe Zhao, Jiajia Wang, Yanyan Li, Tingrui Song, Yang Wu, Shuangsang Fang, Dechao Bu, Hui Li, Liang Sun, Dong Pei, et al. Noncodev6: an updated database dedicated to long non-coding rna annotation in both animals and plants. *Nucleic Acids Research*, 49(D1):D165–D171, 2021.

[33] Kevin L Howe, Premanand Achuthan, James Allen, Jamie Allen, Jorge Alvarez-Jarreta, M Ridwan Amode, Irina M Armean, Andrey G Azov, Ruth Bennett, Jyothish Bhai, et al. Ensembl 2021. *Nucleic acids research*, 49(D1):D884–D891, 2021.

[34] Ensembl Release 104 (May 2021). http://www.ensembl.org/index.html, 2021. Accessed: 2021-5.

[35] A. W. Ingleton. The rank of circulant matrices. *Journal of the London Mathematical Society*, s1-31(4):445–460, 1956.

Appendix. Sparse Factorization of Large Square Matrices

1   Synthetic Data Experiments

For both problems in the scalability test, we generated sequences using the setup described in the main paper. We ran the experiments on sequences with variable lengths: from 128 to 32k. The longer sequences, the more complex the retrieval process. There is a slight difference in the pre-processing part. For the Adding problem, the input data was only two-dimensional. To avoid using such a low-dimensional embedding space, we augmented the dimensionality with an additional linear layer to assure sufficient freedom for dot-product attention architectures. The training configuration and hyperparameters are the same for both the Adding problem and the Temporal Order problem. Their summary is in Table A1

2   Long Range Arena

The data set for the LRA benchmark is publicly available. The information about data and the download link can be found in the official GitHub repository: `https://github.com/google-research/long-range-arena`.

- **ListOps** The raw data for this problem is organized as three separate files `basic_train.tsv`, `basic_test.tsv`, `basic_val.tsv` for training, testing, and validation data, respectively. The split is fixed. In addition to the tokens described in the main paper, each sequence has "(" and ")" symbols, which should be removed. To equalize the lengths of the sequences, we used the built-in PyTorch padding functional. After the sequences are prepared, the embedding layer processes each unique value, thus mapping elements to the embedding space. The rest of the training process is straightforward.
- **Text Classification** We downloaded IMDB data set using the `tensorflow-dataset` package, and got 25000 instances for training and another 25000 for testing. We went through the whole corpus and extracted the character vocabulary. Then we mapped each sequence to a vector of indices using this vocabulary. Finally, we truncated or padded each sequence to a fixed length of 4096. For every review, we add [”CLS”] token to each sequence and use the embedding of ["CLS"] token for final classification. We used three blocks Paramixer for this task.
- **Image Classification** CIFAR10 is a well-known dataset, which can be downloaded from the `torchvision` package. The train/test splitting is fixed. To make images grayscaled, we used standard transformation `transforms-grayscale` from the same package. An image is flattened to a sequence of length 1024. Then each element is mapped to a dictionary of size 256 (all possible intensity values) and given to the embedding layer.
- **Pathfinder** The problem data consists of two types of files: images and metafiles. Metafiles store information about all the images and their corresponding labels (positive or negative). There are three classes of images: `curv_baseline` (easy), `curv_contour_length_9` (medium), `curv_contour_length_14` (hard). An image class corresponds to the distance between its endpoints (curve length), thus positively correlates with the difficulty level. The exact data split is not provided. To separate the data into three parts, we iterated over all metafiles from the catalogs and constructed the training/val/test (90%/5%/5%) sets such that all three types of images are present equally. The rest of the processing is similar to the Image Classification task.

3   Long Document Classification

The task is a four-class classification problem. The class of a paper is defined by its arxiv categorization, namely, cs.AI, cs.NE, math.AC, and math.GR. Each class in the data set is almost equally presented, with a slight class imbalance: 2995, 3012, 2885, and 3065 documents, respectively. To transform the raw articles into sequences, we first went through the whole corpus and extracted the character vocabulary. Then we mapped each character sequence to a vector of indices using this vocabulary. We fine-tuned Paramixer and X-formers to get the best results. The hyperparameters of Paramixer were selected using a similar process. Final configurations are shown in Table A1. For every document, we add [”CLS”] token to each sequence and use the result embedding of ["CLS"] token for the final classification.

4   Genome Classification

When building MTcDNA we downloaded cDNA sequences of Chelonoidis abingdonii and Gopherus agassizii, and merged them as a turtle data set. Following the same strategy, we built the mouse data set using Mus musculus and

Table A1: Hyperparameters details for every task. N, B, V, E, H, lr refer to max sequence length, batch size, vocabulary size, embedding size, hidden states size, and learning rate, respectively. The vocabulary size includes padding index and ["CLS"].

Task	N	Protocol	n_links	lr	B	V	E	H	pos_embed	Pooling Type
Adding	32768	CHORD	15	0.001	40	-	32	32	True	FLAT
Temporal Order	16384	CHORD	14	0.001	40	6	32	32	True	FLAT
ListOps	2000	CHORD	12	0.001	48	16	32	32	True	FLAT
CIFAR10	1024	CDIL	3	0.001	64	256	32	32	True	FLAT
Text	4096	CDIL	9	0.0001	32	97	32	128	False	CLS
Pathfinder	1024	CHORD	11	0.001	64	256	32	32	True	FLAT
Long Document	16384	CHORD	15	0.0001	16	4290	100	128	False	CLS
Long Document	32768	CHORD	16	0.0001	16	4290	100	128	False	CLS
Genome Classification	16384	CHORD	15	0.0001	16	5	32	128	True	FLAT

Mus spretus. More details can be found in the main paper. For the HFDNA classification task, one Paramixer block is enough to get 100% accuracy. However, ParamixerNn with two blocks result in the best test accuracy for MTcDNA. The selected hyperparameters are listed in Table A1.

5 Proof of Proposition 3.1 in the main paper

Definition 1. *An $N \times N$ circulant matrix C takes the form*

$$C = \begin{bmatrix} c_0 & c_{N-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{N-1} & \cdots & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{N-2} & \cdots & \ddots & \ddots & c_{N-1} \\ c_{N-1} & c_{N-2} & \ddots & c_1 & c_0 \end{bmatrix}$$

Definition 2. *The polynomial*

$$f(x) = c_0 + c_1 x + \cdots + c_{N-1} x^{N-1}$$

is called the associated polynomial of circulant matrix C.

We have the following theorem in the literature [35]:

Theorem 2. *The rank of a circulant matrix C is equal to $N - d$, where d is the degree of the polynomial $GCD(f(x), x^{N-1})$.*

Now we can prove Preposition 3.1 for the CHORD protocol. The proof for CDIL follows similarly.

Proof. The associated polynomial of $W^{(m)}$ is

$$f(x) = \sum_{k=0}^{\log_2 N - 1} x^k$$

Because $GCD(f(x), x^N - 1) = 1 = x^0$, the rank of $W^{(m)}$ is N. □