S³Attention: Improving Long Sequences Attention with Smoothed Skeleton Sketching

Anonymous Author(s)

ABSTRACT

Attention-based models have achieved many remarkable break-throughs in numerous applications. However, the quadratic complexity of Attention makes the vanilla Attention-based models hard to apply to long sequences tasks. Various improved Attention structures are proposed to reduce the computation cost by inducing low rankness and approximating the whole sequences by sub-sequences. The most challenging part of those approaches is maintaining the proper balance between information preservation and computation reduction: the longer sub-sequences used, the better information is preserved, but at the price of introducing more noise and computational costs. In this paper, we propose a smoothed skeleton sketching-based Attention structure, coined S³Attention, which significantly improves upon the previous attempts to negotiate this trade-off. S³Attention has two mechanisms to effectively minimize the impact of noise while keeping the linear complexity to the sequence length: a smoothing block to mix information over long sequences and a matrix sketching method that simultaneously selects columns and rows from the input matrix. We verify the effectiveness of S³Attention both theoretically and empirically. Extensive studies over Long Range Arena (LRA) datasets and six time-series forecasting show that S³Attention significantly outperforms both vanilla Attention and other state-of-the-art variants of Attention structures. The code is available at https://anonymous.4open.science/r/S3Attention-B7E4.

KEYWORDS

Deep learning, Attention, Skeleton Approximation, Dimensionality Reduction

1 INTRODUCTION

Modern Attention-based models, first introduced in [61], have made significant contributions in several areas of artificial intelligence, including natural language processing (NLP) [5, 14, 18, 40], computer vision (CV) [19, 41, 60, 70, 75], and time series forecasting [69, 76]. These models have also been applied in a variety of long sequences

mining tasks, such as traffic [17, 21], health care [38, 72], retail [4, 8], security industry [53], and web applications [22, 31, 73]. The Attention scheme efficiently captures long-term global and short-term local correlations when the length of the token sequences is relatively small. However, due to the quadratic complexity of standard Attention, many approaches have been developed to reduce the computational complexity for long sequences (e.g., [77]). Most of them try to exploit the special patterns of the Attention matrix, such as low rankness, locality, sparsity, or graph structures. One group of approaches is to build a linear approximation for the softmax operator (e.g., [9, 12, 13, 48]). Despite the efficiency of the linear approximation, these approximation methods often perform worse than the original softmax based Attention. More discussion of efficient Attention-based models for long sequences can be found in the section of related work.

In this work, we focus on approaches that assume a low-rank structure of the input matrix. They approximate the global information in long sequences by sub-sequences (i.e., short sequences) of landmarks, and only compute Attention between queries and selected landmarks (e.g., [42, 45, 77, 78]). Although those models enjoy linear computational cost and often better performance than vanilla Attention, they face one major challenge, i.e., how to balance between information preservation and noise reduction. By choosing a larger number of landmarks, we are able to preserve more global information but at the price of introducing more noise into the sequential model and more computational cost.

In this work, we propose an efficient Attention architecture, termed S³Attention, that introduces two mechanisms to address the balance explicitly. First, we introduce a smoothing block into the Attention architecture. It effectively mixes global information over the long sequences by the Fourier transformation and local information over the sequences by a convolution kernel. Through the information mixture, we are able to reduce the noise for individual tokens over the sequences, and at the same time, improve their representativeness for the entire sequences. Second, we introduce a matrix sketching technique to approximate the input matrix by a smaller number of rows and columns. Standard Attention can be seen as reweighing the columns of the value matrix. Important columns are assigned high attention weights and remain in the output matrix, while small attention weights eliminate insignificant columns. The Attention mechanism is equivalent to column selection if we replace the softmax operator with the corresponding argmax operator. However, sampling only columns may not generate a good summary of the matrix, and could be subjected to the noises in individual columns. We address this problem by exploiting CUR [20] or Skeleton Sketching technique [11] in the matrix approximation community. Theoretically, for a rank-r matrix $X \in \mathbb{R}^{n \times d}$, we can take $O(r \log d)$ column samples and $O(r \log n)$ row samples to construct a so-called Skeleton approximation $X \approx CUR$, where C and R are matrices consisting of sampled columns and rows of X

respectively, and U is the pseudo-inverse of C and R's intersection. By combing these mechanisms, we found, both theoretically and empirically, that S^3Attention is able to preserve global information over long sequences and reduce the impact of noise simultaneously, thus leading to better performance than state-of-the-art variants of Attention based models for long sequences, without having to sacrifice the linear complexity w.r.t. sequence length.

In short, we summarize our main contributions as follows:

1. We propose the S^3Attention, an efficient model that integrates smoother column Attention and row Attention components to unfold a randomized linear matrix sketching algorithm.
2. By randomly selecting a fixed number of rows and columns, the proposed model achieves near-linear computational complexity and memory cost. The effectiveness of this selection method is verified both theoretically and empirically.
3. We conduct extensive experiments over Long-term sequences, long-term time series forecasting and GLUE tasks. In particular, the Long Range Arena benchmark [57], achieves an average accuracy of 64% and 66% with fixed parameters (suggested setting in [44, 57]) and fine-tuned parameters respectively. It improves from 62% of the best Attention-type model. Moreover, it also has a comparable performance with the recent state-of-art long-term time series forecasting models for long-term time series forecasting and GLUE tasks.

Organization.   We structure the rest of this paper as follows: In Section 2, we briefly review the relevant literature related to efficient Attention based models. Section 3 introduces the model structure and performs a theoretical analysis to justify the proposed model. We empirically verify the efficiency and accuracy of S^3Attention in Section 4. we discuss limitations and future directions in Section 5. The experimental details are provided in the appendix.

2   RELATED WORK

This section provides an overview of the literature mainly focusing on efficient Attention based models. The techniques include sparse or local Attention, low rankness, and kernel approximation. We refer the reader interested in their details to the survey [58].

Sparse Attention.   The general idea of these methods is restricting the query token to perform Attention only within a specific small region, such as its local region or some global tokens. In this setting, the Attention matrix becomes sparse compared to the original one. [49] proposes BlockBert, which introduces sparse block structures into the Attention matrix by multiplying a masking matrix. [46] applies Attention within blocks for the image generation task. [39] divides the whole sequences into blocks and uses a stride convolution to reduce the model complexity. However, these block-type Transformers ignore the connections among blocks. To address this issue, Transformer-XL [16] and Compressive Transformer [50] propose a recurrence mechanism to connect multiple blocks. Transformer-LS [77] combines local Attention with a dynamic projection to capture long-term dependence. [56] uses a meta-sorting network to permute over sequences and quasi-global Attention with local windows to improve memory efficiency.

Another approach in this category is based on stride Attention. Longformer [3] uses dilated sliding windows to obtain a sparse Attention matrix. Sparse Transformers [10] consider approximating a dense Attention matrix by several sparse factorization methods. In addition, some methods reduce the complexity by clustering tokens. For example, Reformer [34] uses a hash similarity measure to cluster tokens, and Routing Transformer [51] uses k-means to cluster tokens. BigBird [71] proposes a generalized Attention mechanism described by a directed graph to reduce Attention complexity. [36] considers using 2D Fourier Transformation to mix the token matrix directly. [54] uses max pooling scheme to reduce the computation costs.

Low-rank and Kernel Methods.   Inducing low rankness into the Attention matrix can quickly reduce the complexity and the kernel approximation is widely applied in efficient low-rank approximation. Linformer [65] and Luna [42] approximate softmax with linear functions, which yield a linear time and space complexity. [12] and [47] use random features tricks and reach promising numerical performance. [66] proposes Low-Rank Transformer based on matrix factorization. FMMformer [45] combines the fast multipole method with the kernel method. Synthesizer [55] uses a random low-rank matrix to replace the Attention matrix. Nyströmformer [68] adopts the Nyström method to approximate standard Attention. Linear Transformer [33] expresses Attention as a linear dot-product of kernel feature maps. [78] applies the Multigrid method to efficiently compute the Attention matrix recursively. Cosformer [48] develops a cosine-based re-weighting mechanism to linearize the softmax function. [9] proposes the Scatterbrain, which unifies locality-sensitive hashing and the kernel method into Attention for accurate and efficient approximation.

Sequence Length Reduction.   Reducing sequence length is an efficient means to reduce computational costs. Perceiver IO [32] encodes inputs to a latent space whose size is typically smaller than the inputs and outputs, making the process computationally scalable to even very large inputs and outputs. Funnel-Transformer [15] uses pooling tricks to reduce sequence length, significantly saving both FLOPs and memory. Swin Transformer [41] proposes a shifted windows method in vision Transformer, and the resulting complexity is linear in the image size. XCiT [2] considers an Attention over the hidden dimension for vision tasks. Charformer [59] downsamples the sequences of words to construct latent subwords.

Improved Recurrent Neural Network.   Another research track to solve the long sequences tasks is to improve the Recurrent Neural Network (RNN). [23] propose to view RNN as the state space model and use optimal polynomial projections to improve its memorization ability. The follow-up works [24–26, 28, 28, 30, 52] consider more sophisticated designs and obtain better performance and higher efficiency. The concurrent work [43] proposes coupling the moving average type of RNN structure with single-headed Attention and reaches quite promising results. Different from those works, in this paper, we consider a different research angle and focus on making improvements inside the Attention structure.

3   S³ATTENTION

We start by going over the vanilla Attention. For a sequences of length n, the vanilla Attention is dot-product type [61]. Following standard notation, the Attention matrix $A \in \mathbb{R}^{n \times n}$ is defined as:

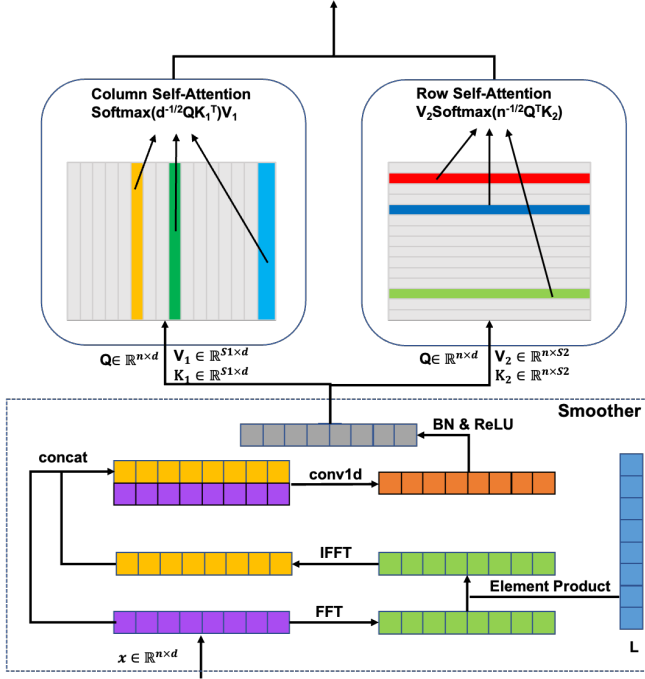$$A = \text{softmax}\left(\frac{1}{\sqrt{d}}QK^{\top}\right),$$

Figure 1: Illustration of the architecture of S³ Attention.

where $Q \in \mathbb{R}^{n \times d}$ denotes the queries while $K \in \mathbb{R}^{n \times d}$ denotes the keys, and d represents the hidden dimension. By multiplying the attention weights A with the values $V \in \mathbb{R}^{n \times d}$, we can calculate the new values as $\hat{V} = AV$.

Intuitively, the Attention is the weighted average over the old ones, where the weights are defined by the Attention matrix A. In this paper, we consider generating Q, K and V via the linear projection of the input token matrix X:

$$Q = XW_Q, \ K = XW_K, \ V = XW_V,$$

where $X \in \mathbb{R}^{n \times d}$ and $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$.

The vanilla procedure has two drawbacks in concentrating the information from V. First, when computing the $QK^\top$ part, full dense matrix multiplication is involved at a cost of $O(n^2)$ vector multiplications. It can be prohibitive for long sequences problems. On the other hand, if we view the softmax operator as an approximation of the argmax counterpart, $\hat{V}$ becomes a row selection from V. This column-wise information concentration is ignored.

3.1 Skeleton Sketching Based Attention

We propose a Skeleton Sketching induced Attention structure to address those issues. First, we modify the original Attention to build the column Attention as follows:

$$\hat{V}_1 = \text{softmax}\left(\frac{1}{\sqrt{d}} QK^\top P_1^\top\right) P_1 V,$$

where $P_1 \in \mathbb{R}^{s_1 \times n}$ denotes the sampling matrix and s_1 is the number of columns sampled. Let $i_1 < i_2 < \ldots < i_{s_1}$ be the indices of the randomly sampled columns. Let $P_{1,ab}$ denote the element located at the a-th column and b-th row and we have $P_{1,ab} = 1$ if $i_a = b$ and 0

otherwise. By these constructions, we can reduce the computational cost to $O(ns_1 d^2 + ns_1^2 d)$.

Similarly, we build the row sampling matrix $P_2 \in \mathbb{R}^{d \times s_2}$ indicating the locations of the s_2 sample rows. Compute the row Attention as:

$$\hat{V}_2 = VP_2 \, \text{softmax}\left(\frac{1}{\sqrt{n}} P_2^\top K^\top Q\right).$$

Finally, we apply the layer-norm on $\hat{V}_1$ and $\hat{V}_2$ and then them together to generate the final output:

$$\hat{V} = \text{layernorm}_1(\hat{V}_1) + \text{layernorm}_2(\hat{V}_2). \tag{1}$$

The usage of layer norm is to balance the output scales of column and row Attentions. A similar trick has been used in [77], where the layer norm is applied to resolve scale mismatches between the different Attention mechanisms.

Before going into the detailed analysis, we first introduce the incoherence parameter of a matrix, which is commonly used in many low-rank matrix applications.

DEFINITION 1 (μ-INCOHERENCE). *Given a rank-r matrix $X \in \mathbb{R}^{n \times d}$. Let $X = W\Sigma V^\top$ be its compact singular value decomposition. X is μ-incoherent if there exists a constant μ such that*

$$\max_i \|e_i^\top W\| \le \sqrt{\frac{\mu r}{n}} \qquad \text{and} \qquad \max_i \|e_i^\top V\| \le \sqrt{\frac{\mu r}{d}},$$

where e_i denotes the i-th canonical basis vector.

The μ-incoherence describes the correlation between the column/row spaces and the canonical basis vectors. The larger μ value implies a higher *overlapping*, which leads to a better chance of successful reconstruction from sparse row/column samples. We next use the following proposition to characterize the efficiency of sampling in both columns and rows.

PROPOSITION 1. *Let $X \in \mathbb{R}^{n \times d}$ be a rank-r, μ-incoherent matrix. Without loss of generality, we assume $n \ge d$. Let $E \in \mathbb{R}^{n \times d}$ be a noise matrix. By uniformly sampling $O(\mu r \log n)$ columns and rows from the noisy $X + E$, Skeleton approximation can construct a matrix $\hat{X}$ such that, with probability at least $1 - O(n^{-2})$,*

$$\|X - \hat{X}\| \le O\left(\frac{\|E\| \sqrt{nd}}{\mu r \log n}\right).$$

A similar result, under a slightly different setting, can be found in [6]. For the completeness of the paper, we provide the proof here.

PROOF. We resolve the sampling strategy. We consider a clear rank-r matrix $X \in \mathbb{R}^{n \times d}$, i.e., no additive noise and the rank is exact. Without loss of generality, we assume $n \ge d$. Provided X is μ-incoherent, by [11, Theorem 1.1], Skeleton approximation recovers X exactly, i.e.,

$$X = CUR,$$

with probability at least $1 - O(n^{-2})$ if we sample $O(\mu r \log n)$ rows and columns uniformly to form the submatrices C and R.

Thirdly, we resolve the error bound estimation. For the noisy matrix $X + E$, we directly apply [29, Corollary 4.3]. Thus, we have

$$\|X - \hat{C}\hat{U}\hat{R}\| \le O\left(\sqrt{\frac{nd}{l_C l_R}}\right)\|E\|,$$

where $\hat{C}$ and $\hat{R}$ are sampled from the noisy matrix, $\hat{U}$ is the pseudo-inverse of $\hat{C}$ and $\hat{R}$'s intersection, and l_C (resp. l_R) is the number of columns (resp. rows) being sampled in $\hat{C}$ (resp. $\hat{R}$).

Note that this error bound assumes good column/row sampling, i.e., the clear submatrices corresponding to $\hat{C}$ and $\hat{R}$ can recover X exactly. Therefore, by combining the above two results, we show the claim in Proposition 1. □

Several works (e.g., [11, 20]) have proposed explicit methods to construct $\hat{X}$. Those methods require computing the pseudo-inverse, generally inefficient in deep learning settings. [68] uses an approximation of the pseudo-inverse in the symmetric matrix setting. It is still an open question whether the approximated pseudo-inverse also works for the general matrix in deep learning settings. On the other hand, in the transformer model, a good matrix approximation is not our primary goal, and we thus pursue a different way that only maintains sufficient information to pass through the network via (1).

3.2 Smoother Component

Based on the analysis of Skeleton Sketching, the matrix incoherence parameter μ plays a crucial role in determining the number of rows and columns to sample. Decreasing in μ leads to a smaller sampling size. Furthermore, the μ-incoherence condition implies that the "energy" of the matrix is evenly distributed over its entries, i.e., the matrix is "smooth" [7]. In this subsection, we propose a novel smoother component to reduce the incoherence parameter without introducing excessive information loss.

Fourier Convolution. The incoherence parameter μ can be viewed as a measure of the smoothness of a matrix. A "smoother" matrix tends to have a smaller incoherence parameter. Intuitively, the adjacent columns or rows have similar values for a smooth matrix. Thus a few landmark columns or rows can represent the matrix with little error. On the other hand, if the matrix is harsh (e.g., containing spiky columns or rows), more landmarks are required. A common way to smooth a matrix is to convolute it with a smoothing kernel, such as a Gaussian kernel. However, directly using a fixed smoothing kernel can potentially remove too many details and harm the final performance. In the recent literature (e.g., Guo et al. 27), large convolution kernel-based Attentions show a supreme performance in vision Transformers. In this paper, we propose to use a data-driven convolution layer along the sequence dimension with a kernel size equal to the sequence length. In this setting, the information of a given row could be decentralized among the rows. As the input token matrix is computed through a FeedForward layer, the information among different rows is already adaptive allocated. Hence, we do not perform the convolution along the hidden dimension.

We use the Fast Fourier Transformation (FFT) to implement the convolution. Let $L_0 \in \mathbb{R}^{n \times d}$ be the convolution kernel matrix. Via the convolution theorem, the circular convolutions in the spatial domain are equivalent to pointwise products in the Fourier domain, and we then have:

$$X^{\text{smooth}} = X * L_0 = \mathcal{F}^{-1} \left[\mathcal{F}(X) \cdot \mathcal{F}(L_0) \right], \tag{2}$$

where $\mathcal{F}$, $*$, and $\cdot$ denote FFT operator, convolution operator, and point-wise product, respectively.

Equation (2) requires $3d$ times faster Fourier operations which could be prohibited when facing large d. In order to save the computational cost, we use the learnable matrix $L \in \mathbb{C}^{n \times d}$[1] in the frequency domain instead and apply segment-average (averaging segments of hidden dimension) to X. To simplify the notation, we assume there are integers s and r with $d = sr$. Instead of using (2), we apply the following (3) to smooth the token matrix.

$$X^{\text{smooth}} = \mathcal{F}^{-1} \left[\mathcal{F}(XS) \cdot L \right], \tag{3}$$

where

$$S = \begin{bmatrix} \frac{1}{s}\mathbf{1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \frac{1}{s}\mathbf{1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \frac{1}{s}\mathbf{1} \end{bmatrix} \in \mathbb{R}^{d \times d} \tag{4}$$

and $\mathbf{1}$ denotes the $s \times s$ matrix with all elements equal 1. As XS contains repeated rows, in (3), we can reduce the usage of faster Fourier operations to $r + d$ times. Moreover, to further control the computation cost, we only smooth the token matrix X instead of the query, key and value matrices Q, K and V separately.

In the following proposition, we show the smooth ability of the Fourier convolution.

PROPOSITION 2. *Let $\{x_1, ..., x_n\}$ be sequences with $\max_t |x_t| \leq a_{\max}$ and $\max_t |x_t - x_{t-1}| \leq b_{\max}$. Let $\{l_1, ..., l_n\}$ be sequences of i.i.d. $\frac{1}{n^2}\sigma^2$-subgaussian variables. Let $f(t)$ be the convolution of $\{x_t\}$ and $\{l_t\}$, i.e., $f(t) = \sum_{i=1}^{t} l_{t+1-i}x_i$. With probability at least $1 - \delta$, we have:*

$$|f(t) - f(t-1)| \leq b_{\max}\sigma\sqrt{\frac{1}{2n}\log\left(\frac{2n}{\delta}\right)} + a_{\max}\sigma\sqrt{\frac{1}{2n^2}\log\left(\frac{2}{\delta}\right)}.$$

PROOF. As $f(t)$ is the convolution function of $\{x_t\}$ and $\{l_t\}$, from the definition of convolution for $t = 1, 2, ..., n$ we have

$$f(t) = \sum_{i=1}^{t} l_{t+1-i}x_i$$

and

$$f(t) - f(t-1) = \underbrace{\sum_{i=1}^{t-1}(l_{i+1} - l_i)x_i}_{:=(a_t)} + l_1 x_t.$$

By Hoeffelding inequality, term (a_t) satisfies the following inequality with $\varepsilon > 0$.

$$\mathbb{P}(|(a_t)| \geq \varepsilon) = \mathbb{P}\left(\left|\sum_{i=1}^{t-1}(l_{i+1} - l_i)x_i\right| \geq \varepsilon\right)$$
$$\leq \exp\left(-\frac{2\varepsilon^2}{(t-1)b_{\max}^2 \cdot \frac{1}{n^2}\sigma^2}\right). \tag{5}$$

---

[1]In practice, we use the rFFT/irFFT, the fast (inverse) Fourier Transformation of real input instead of the general FFT/IFFT, and the size of the matrix L is reduced to $L \in \mathbb{C}^{\lfloor n/2 \rfloor + 1) \times d}$.

Combine (5) with the union bound over $t = 1, 2, ..., n$ and the following inequality (6) holds with probability at least $1 - \delta/2$:

$$\max_t |(a_t)| \le b_{\max}\sigma\sqrt{\frac{1}{2n}\log\left(\frac{2n}{\delta}\right)}. \tag{6}$$

Similarly, with probability $1 - \delta/2$, we have

$$\max_t |l_1 x_t| \le a_{\max}\sigma\sqrt{\frac{1}{2n^2}\log\left(\frac{2}{\delta}\right)}. \tag{7}$$

Therefore, via (6) and (7), with probability at least $1 - \delta$, we have

$$\max_t |f(t) - f(t-1)|$$

$$\le b_{\max}\sigma\sqrt{\frac{1}{2n}\log\left(\frac{2n}{\delta}\right)} + a_{\max}\sigma\sqrt{\frac{1}{2n^2}\log\left(\frac{2}{\delta}\right)}.$$

This finishes the proof. □

The Proposition 2 can be used to describe the Fourier convolution layer's behavior in the early training stage. Via some standard initialization methods (e.g., Kaiming initialization or Xavier initialization), the variance of elements in learnable matrix L is $O(n^{-1})$ and the scale of elements is $O(n^{-1/2})$.[2] To simplify our discussion, let us assume we use Kaiming normal initialization and L becomes a random complex Gaussian matrix with zero mean and variance $n^{-1}\sigma^2$. Using the fact that the FFT of Gaussian s remains Gaussian with $2n$ times larger variance, the $n^{-1}\sigma^2$ variance Gaussian sequences through inverse FFT (IFFT) would result in Gaussian sequences with $\frac{1}{2n^2}\sigma^2$ variance. By Proposition 2, the maximum difference between adjacent elements after the convolution is scaled on $b_{\max}\sigma n^{-1/2} + a_{\max}\sigma n^{-1} \approx b_{\max}\sigma n^{-1/2}$ when sequence length n is large enough. Thus as long as $\sigma < O(\sqrt{n})$, the sequences are smoothed by the Fourier convolution. This smoothness ability eventually helps stabilize the early training stage. The related experiments are summarized in Appendix I.

When the training procedure progresses, the learnable matrix L will be apart from the Sub-Gaussian initialization and converge to some data-driven matrix. In this regime, it is possible that the convoluted sequences may not necessarily preserve a very low variance as it is in the early stage. However, a good memorization ability can happen. In particular, in Proposition 3, we show it is possible to have a learned matrix L such that the historical information in XS up to time t can be compressed to some function parameterized with t-th token in the convoluted sequences.

PROPOSITION 3. *Let $X \in \mathbb{R}^{n \times d}$ be a bounded matrix and $S \in \mathbb{R}^{d \times d}$ constructed by (4). There exist matrices $G, L \in \mathbb{R}^{n \times d}$ such that*

$$\left\|(XS)_{1:t} - X_t^{\text{smooth}} G_{1:t}\right\| \le O\left(r^{3/2} t \log(n) d^{-1/2}\right),$$

where $(\cdot)_{1:t}$ is the submatrix of the first t rows of a given matrix, X_t^{smooth} is the t-th row of $X^{\text{smooth}} = \mathcal{F}^{-1}\left[\mathcal{F}(XS) \cdot L\right]$, and G satisfies $G_{i,j} = G_{i+s,j} = ... = G_{i+r(s-1),j} = g_i(j)$. Here $\{g_1(\cdot), ..., g_s(\cdot)\}$ is an orthogonal polynomial basis.

PROOF. The proof contains two parts. In the first part, we view the data sequences as a function of index t and construct the coefficients and orthogonal polynomials for function approximation. In

---

the second part, we show such coefficients can be computed with Fourier convolution i.e., (3).

Function Approximation. We reformulate the matrix XS as follow:

$$XS = \begin{bmatrix} \bar{x}_1 e & \bar{x}_2 e & \cdots & \bar{x}_r e \end{bmatrix},$$

where $e \in \mathbb{R}^{1 \times s}$ is the one vector and $\bar{x}_i \in \mathbb{R}^{n \times 1}$ is the average from $(s(i-1)+1)$-th column to (si)-th column of X.

Next, we focus on vector $\bar{x}_j$ and view its t-th element as the output of a function $h^j(t) = \bar{x}_{jt}$. Via analysis in [23, Appendixes C and D], we can form an approximation on $h^j(t)$ as follow:

$$h_{[x \le t]}^j(x) \approx \sum_i c_i^j(t) g_i(x),$$

where $\{g_i\}$ are sequences of orthogonal polynomial and $c(t)^j := [c_1^j(t), c_2^j(t), ..., c_s^j(t)] \in \mathbb{R}^{1 \times s}$ satisfy

$$\frac{d}{dt}c(t)^j = \frac{1}{t}c(t)^j A_0 + \frac{1}{ts\log n}h(t)b_0, \tag{8}$$

where $A_0 \in \mathbb{R}^{s \times s}$ and $b_0 \in \mathbb{R}^{1 \times s}$ are predefined matrix and vector respectively. Equation (8) is corresponding to the case with $\lambda_n = s\log n$ in [23].

We then use Forward Euler approach to discretize it:

$$\hat{c}(t)^j = \hat{c}(t-1)^j\left(\frac{1}{t}I + \frac{1}{t}A_0\right) + \frac{1}{ts\log n}h(t)b_0. \tag{9}$$

Via standard error analysis of Forward Euler approach, we have

$$c(t+1)^j = c(t)^j + \frac{1}{t}c(t)^j A_0 + \frac{1}{ts\log n}h(t)b_0 + \frac{d^2}{dt^2}c(t)^j|_{t=\xi}$$

$$= c(t)^j + \frac{1}{t}c(t)^j A_0 + \frac{1}{ts\log n}h(t)b_0 + \frac{1}{\xi s\log n}h(\xi)'b_0$$

$$= c(t)^j + \frac{1}{t}c(t)^j A_0 + \frac{1}{ts\log n}h(t)b_0 + O\left(\frac{1}{ts\log n}\right),$$

where $\xi \in [t, t+1]$.

It implies that for $t = 1, 2, ..., n$,

$$\|\hat{c}(t)^j - c(t)^j\| \le O\left(\frac{\log t}{s\log n}\right). \tag{10}$$

Combine (10) with the similar proof procedure in [23, Proposition 6], if $h^j(x)$ is quadratic spline interpolation on $\{\bar{x}_{jt}\}$, we obtain

$$\left\|\bar{x}_{jt} - \sum_{i=1}^s \hat{c}_i(t)g_i(x)\right\| \le O\left(t\log n/\sqrt{s}\right) = O\left(t\log n\sqrt{\frac{r}{d}}\right). \tag{11}$$

The desirable result in Proposition 3 is obtained by repeatedly using (11) with $j = 1, 2, ..., r$.

Coefficients via Fourier Convolution. The remaining task is to show that $\{\hat{c}(t)^j\}$ can be generated via Fourier convolution. To simplify the notation, we denote $A = \frac{1}{t}I + \frac{1}{t}A_0$ and $b = \frac{1}{t\log n}b_0$ and (9) becomes

$$\hat{c}(t)^j = \hat{c}(t-1)^j A + h(t)b. \tag{12}$$

We then repeatedly use (12) from $t = 1, 2, ...$ and one may verify

$$\hat{c}_t^j = \sum_{i=1}^{t-1} b A^{t-i} h(i) = \sum_{i=1}^{t-1} b A^{t-i} \bar{x}_{ji}$$

$$\Rightarrow \quad C^j = \bar{A}_j * (\bar{x}_j e), \tag{13}$$

---

where

$$C^j = \begin{bmatrix} \hat{c}_1^j \\ \hat{c}_2^j \\ \vdots \\ \hat{c}_n^j \end{bmatrix} \in \mathbb{R}^{n \times s}, \text{ and } \bar{A}_j = \begin{bmatrix} b \\ bA \\ \vdots \\ bA^{n-1} \end{bmatrix} \in \mathbb{R}^{n \times s}.$$

Next we repeatedly use (13) from $j = 1, 2, .., r$, and one has

$$\underbrace{\begin{bmatrix} C^1 & C^2 & \cdots & C^r \end{bmatrix}}_{:=X^{\text{smooth}}}$$

$$= \underbrace{\begin{bmatrix} \bar{A}_1 & \bar{A}_2 & \cdots & \bar{A}_r \end{bmatrix}}_{:=L_0} * \underbrace{\begin{bmatrix} \bar{x}_1 e & \bar{x}_2 e & \cdots & \bar{x}_r e \end{bmatrix}}_{=XS}$$

$$\Rightarrow \quad X^{\text{smooth}} = L_0 * XS$$

$$\Rightarrow \quad X^{\text{smooth}} = \mathcal{F}^{-1}\left(\mathcal{F}(L_0) \cdot \mathcal{F}(XS)\right)$$

$$\Rightarrow \quad X^{\text{smooth}} = \mathcal{F}^{-1}\left(L \cdot \mathcal{F}(XS)\right),$$

where we use the fact that L is constructed in frequency domain in Fourier convolution in (3). □

The Proposition 3 states that if we properly train the matrix L, the information in XS up to row t can be compressed into t-th row of X^{smooth} with a moderate tolerance. Therefore, when we sample in rows X^{smooth}, they will contain more information than the same number of rows in the original XS. Similar results are also discussed in FNet [36] and several RNN literature, such as [23] and [62]. In [23], several specific types of polynomials (e.g., Legendre or Chebyshev) are explored, and the corresponding matrix L is predefined instead of data-driven. Recently, [24] propose a sophisticated method that can be used to compute X^{smooth}. We leave it for future work.

Convolution Stem. X^{smooth} may encounter an over-smoothing situation that local details can be wiped out. We use a convolution stem (CNNs + BN + ReLU) to tackle this problem. We first concatenate X^{smooth} with the original token matrix X into a $n \times 2d$ matrix and then apply a 1D convolution with kernel size 3 to transform it back to $n \times d$ dimensions. At last, the output is normalized with the Batchnorm layer and truncated by the ReLU activation function to stabilize the training procedure. [64] report the ReLU activation coupled with the normalization layer plays an important role in various vision transformers and analyzes this phenomenon theoretically.

4 EXPERIMENTS

In this section, we test our S^{3}Attention on Long Range Arena (LRA) datasets [57] and six real-world time series benchmark dataset for long term forecasting. We also evaluate the transfer learning ability of S^{3}Attention on GLUE tasks. In recent literature (e.g., [23, 24, 28]), the RNN type model is also widely discussed for long sequences tasks. We don't include them as benchmark models since in this paper we focus on improving the Attention structure. The testing environment contains 12 Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz CPUs, 1 TESLA V100 SXM2 32G, and 90 GB memory. We implement the S^{3}Attention based on the official codes of [77] and [76] for LRA and time-series forecasting tasks respectively. The implementation detail (source code) for S^{3}Attention is provided in Appendix A.

4.1 Long-Range Arena

The open-source Long-Range Arena (LRA) benchmark [57] is originally proposed as a standard way to test the capabilities of Attention variants architectures on long sequences tasks.

We benchmark our model with several recent state-of-art efficient Attention architectures, including Sparse Transformer [10], Longformer [3], Linformer [65], Reformer [34], Sinkhorn Transformer [56], Synthesizer [55], BigBird [71], Linear Transformers [33], Performer [12], H-Transformer-1D [78], Nyströmformer [68], Transformer-LS [77], FNet [36], Luna [42], FMMformer [45], Cosformer [48] and Scatterbrain [9]. S^{3}Attention achieves the highest 66.08% average accuracy with tuned parameters and the second best 64.11% result with fixed parameters as shown in Table 1.

In particular, S^{3}Attention significantly outperforms the benchmarks on Image tasks by relatively large margins (12.6% and 20.6%, respectively), which support S^{3}Attention's smoothness effect on the low-level features and will benefit the high-level image classification tasks.

Moreover, we highlight the sampling efficiency of S^{3}Attention. The sequence length of LRA tasks is over one thousand. The efficient Transformers in literature usually can not project the token matrix to a very small size while maintaining comparable numerical performance, by only sampling 8 rows and columns from the token matrix, S^{3}Attention has already obtained 64.11% average score improving the previous best 62.03% score of Transformer-LS.

4.2 Long-Term Forecasting Tasks for Time Series

To further evaluate the proposed S^{3}Attention, we also conduct extensive experiments on six popular real-world benchmark datasets for long-term time series forecasting, including traffic, energy, economics, weather, and disease as shown in table 2

To highlight the relevant comparison, we mainly include five state-of-the-art (SOTA) Attention-based models, i.e., FEDformer[76], Autoformer [67], Informer [74], LogTrans [37], and Reformer [34] for comparison. FEDformer is selected as the main baseline as it achieves SOTA results in most settings. More details about baseline models, datasets, and implementations are described in Appendix.

Compared with SOTA work (i.e., FEDformer), the proposed S^{3}Attention yields a comparable performance in those tasks, with 3/6 datasets having larger winning counts in MSE/MAE. It is worth noting that the improvement is even more significant on certain datasets, e.g., Exchange (> 30% reduction in MSE and MAE). Although Exchange does not exhibit an apparent periodicity pattern, S^{3}Attention still achieves superior performance.

4.3 Transfer Learning in GLUE Tasks

We evaluate the transfer learning ability of the proposed model in the pretraining-finetuning paradigm in NLP tasks. We pre-train vanilla BERT [18], FNet [36], PoNet [54] and our S^{3}Attention with the same MLM loss in [18] on English Wikitext-103 and BooksCorpus datasets. All models are uncased and pre-trained with the same configuration with 1 million steps at most. We report the best GLUE

Table 1: Experimental results on Long-Range Arena benchmark. The best model is in boldface and the second best is underlined. The standard deviations of the S³Attention are reported in parentheses.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Average
Transformer	36.37	64.27	57.46	42.44	71.40	54.39
Local Attention	15.82	52.98	53.39	41.46	66.63	46.06
Sparse Transformer	17.07	63.58	59.59	44.24	71.71	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	53.46
Linformer	35.70	53.94	52.27	38.56	76.34	51.36
Reformer	37.27	56.10	53.40	38.07	68.50	50.67
Sinkhorn Transformer	33.67	61.20	53.83	41.23	67.45	51.39
Synthesizer	36.99	61.68	54.67	41.61	69.45	52.88
BigBird	36.05	64.02	59.29	40.83	74.87	55.01
Linear Transformer	16.13	65.90	53.09	42.34	75.30	50.55
Performer	18.01	65.40	53.82	42.77	77.05	51.41
Nystromformer	37.34	65.75	81.29	41.58	70.94	59.38
H-Transformer-1D	**49.53**	**78.69**	63.99	46.05	68.78	61.41
Transformer-LS	38.36	68.40	81.85	45.05	76.48	62.03
FNet	35.33	65.11	59.61	38.67	77.08	54.42
Luna	38.01	65.78	79.56	47.86	**78.89**	62.02
FMMformer	36.74	67.84	81.88	45.10	72.12	60.74
PoNet	38.80	69.82	80.35	46.88	70.39	61.05
Cosformer	37.9	63.41	61.36	43.17	70.33	55.23
Scatterbrain	38.6	64.55	80.22	43.65	69.91	59.38
S³Attention ($r, s_1, s_2 = 8$)	38.30(0.40)	69.27(0.83)	83.26(0.45)	53.90(1.54)	75.82(0.97)	64.11(2.07)
S³Attention (best)	39.15(0.48)	71.58(0.95)	**83.73(0.61)**	**57.73(1.83)**	78.20(1.32)	**66.08(2.56)**

results for each model from multiple hyper-parameters configurations in Table 3, and the detailed training configurations in Table 15 in Appendix G. Our S³Attention reaches 77.01 average scores (**96.0%** of the accuracy of vanilla BERT), which also outperform FNet by **4.6%** and PoNet by 0.3% relatively.

4.4 Training Speed and Peak Memory Usage

We compared the training speed (in terms of steps per second) and peak memory usage with several baseline models in LRA text classification task with various input lengths. The results are reported in Table 4. S³Attention achieves a 4x time speed advantage and 87% memory reduction compared to vanilla transformer models with 3k input setting and has a neck-to-neck performance compared to the most efficient baseline models.

4.5 Robustness Analysis

We conduct a noise-resistant experiment for S³Attention and 5 other Attention based models as shown in Table 5. We use the Image experiment setting in LRA datasets. During generating sample sequences, we randomly add noise with uniform distribution $\mathcal{U}(-a, a)$ to each position in the sequences. We consider $a \in [0, 2, 4, 8]$ and train every model with 5k steps and 5 replicates. Our model's performance remains robust with a high level of noise injection. This supports our theoretical robustness analysis and shows S³Attention indeed makes an appropriate tradeoff between information preservation and noise reduction.

4.6 Model Parameters Impact

S³Attention introduces three extra hyperparameters, r, s_1 and s_2. We test the influence when varying them and report results in

Table 6. We use S³Attention ($r, s_1, s_2 = 8$) as the baseline model and other parameters are reported in Table 10 in Appendix B.

Influence of r in Fourier Convolution. The r parameter is used to determine the number of segment-averages to compute in (3). The smaller r leads the matrix with more duplicate columns, and more details information is lost. On the other hand, according to Proposition 3, the larger r would potentially decrease the memorization ability and yield a high approximation error. In Table 6a, the best performance is observed when $r = 8$ or $r = 16$. For the case with $r = 1$, the token matrix is smoothed to rank one matrix, and the average accuracy drops 3.55 from the best setting. When the r value goes larger than 16, the accuracy in all experiments slightly decreases. We believe it is due to the over-fitting since the smoothed token matrix contains more flexibility and more irrelevant information training dataset is learned.

Influence of Sample Number s_1 in Row Attention. In Row Attention part, we randomly sample s_1 from key and value tokens. Table 6b reports that the optimal sampling amounts are different among tasks. In Pathfinder task, the optimal result is associated with $s_1 = 256$, while the best performance of other tasks the reached with $s_1 = 32$. Pathfinder task requires learning extreme long-range dependence (the connectivity between two circles far away from each other). The lack of enough tokens leads to inaccurate long-range dependence estimation and damages the final results. For the tasks like Image or Retrieval, the modest range dependence may already be enough to get promising performance, and we thus could use fewer token samples.

Influence of Sample Number s_2 in Column Attention. In Column Attention, s_2 columns are selected. The experiment results are shown in Table 6c. When setting $s_2 = 1$, average performance

Table 2: multivariate long-term series forecasting results on six datasets with input length of 96 **and prediction length** $O \in \{96, 192, 336, 720\}$ **(For ILI dataset, we set prediction length** $O \in \{24, 36, 48, 60\}$**) with input length** 36**. A lower MSE indicates better performance. All experiments are repeated 5 times.**

Methods		S^3Attention		FEDformer		Autoformer		Informer		LogTrans		Reformer	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.192**	**0.283**	0.203	0.287	0.255	0.339	0.705	0.690	0.768	0.642	0.658	0.619
	192	**0.255**	**0.324**	0.269	0.328	0.281	0.340	0.924	0.692	0.989	0.757	1.078	0.827
	336	**0.324**	**0.364**	0.325	0.366	0.339	0.372	1.364	0.877	1.334	0.872	1.549	0.972
	720	0.431	0.433	**0.421**	**0.415**	0.422	0.419	0.877	1.074	3.048	1.328	2.631	1.242
Electricity	96	0.218	0.332	**0.183**	**0.297**	0.201	0.317	0.304	0.405	0.258	0.357	0.312	0.402
	192	0.259	0.361	**0.195**	**0.308**	0.222	0.334	0.313	0.413	0.266	0.368	0.348	0.433
	336	0.267	0.367	**0.212**	**0.313**	0.231	0.338	0.290	0.381	0.280	0.380	0.350	0.433
	720	0.293	0.385	**0.231**	**0.343**	0.254	0.361	0.262	0.344	0.283	0.376	0.340	0.420
Exchange	96	**0.086**	**0.204**	0.139	0.276	0.197	0.323	1.292	0.849	0.968	0.812	1.065	0.829
	192	**0.188**	**0.292**	0.256	0.369	0.300	0.369	1.631	0.968	1.040	0.851	1.188	0.906
	336	**0.356**	**0.433**	0.426	0.464	0.509	0.524	2.225	1.145	1.659	1.081	1.357	0.976
	720	**0.727**	**0.669**	1.090	0.800	1.447	0.941	2.521	1.245	1.941	1.127	1.510	1.016
Traffic	96	0.592	0.352	**0.562**	**0.349**	0.613	0.388	0.824	0.514	0.684	0.384	0.732	0.423
	192	0.583	0.343	**0.562**	**0.346**	0.616	0.382	1.106	0.672	0.685	0.390	0.733	0.420
	336	0.598	0.346	**0.570**	**0.323**	0.622	0.337	1.084	0.627	0.733	0.408	0.742	0.420
	720	0.641	0.397	**0.596**	**0.368**	0.660	0.408	1.536	0.845	0.717	0.396	0.755	0.423
Weather	96	**0.182**	**0.262**	0.217	0.296	0.266	0.336	0.406	0.444	0.458	0.490	0.689	0.596
	192	**0.228**	**0.306**	0.276	0.336	0.307	0.367	0.525	0.527	0.658	0.589	0.752	0.638
	336	**0.295**	**0.355**	0.339	0.380	0.359	0.395	0.531	0.539	0.797	0.652	0.639	0.596
	720	**0.383**	**0.418**	0.403	0.428	0.578	0.578	0.419	0.428	0.869	0.675	1.130	0.792
ILI	24	2.431	0.997	**2.203**	**0.963**	3.483	1.287	4.631	1.484	4.480	1.444	4.400	1.382
	36	2.287	**0.972**	**2.272**	0.976	3.103	1.148	4.123	1.348	4.799	1.467	4.783	1.448
	48	2.418	1.002	**2.209**	**0.981**	2.669	1.085	4.066	1.36	4.800	1.468	4.832	1.465
	60	**2.425**	**1.043**	2.545	1.061	2.770	1.125	4.278	1.41	5.278	1.560	4.882	1.483
1st Count		12	13	12	11	0	0	0	0	0	0	0	0

Table 3: GLUE validation results. We report the mean of accuracy and F1 for QQP and MRPC, matthew correlations for CoLA, spearman correlations for STS-B, and accuracy for other tasks. For MNLI task, we consider the matched test set.

Model	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
BERT-Base	81.98	89.25	88.22	91.07	48.08	87.98	86.43	69.98	80.37
FNet-Base	73.20	85.83	80.57	88.66	40.67	80.64	80.88	57.41	73.48
PoNet-Base	77.02	87.59	84.37	89.29	45.38	84.66	81.82	64.27	76.80
S^3Attention (Ours)	76.86	87.67	84.12	90.14	46.72	84.87	81.84	63.87	77.01

Table 4: Benchmark results of all Xformer models with a consistent batch size of 32 across all models with various input lengths on the LRA text classification task The speed-up and memory-saving multipliers relative to Transformer shown in parentheses.

Model	Training Speed (Steps per second)				Peak Memory Usage (GB)			
	1 K	2 K	3 K	4 K	1 K	2 K	3 K	4 K
Transformer	23.8	7.8	3.9	*OOM*	3.7	11.1	22.1	*OOM*
Linformer	37.0(1.5x)	20.8(2.6x)	14.9(3.7x)	11.9	2.3	3.3	4.3	5.2
Reformer	28.5(1.2x)	15.1(1.9x)	11.9(3.0x)	9.1	2.2	3.2	4.2	4.9
Nystroformer	33.3(1.4x)	22.7(2.9x)	17.2(4.3x)	14.7	1.6	2.2	2.4	2.9
Performer	29.4(1.2x)	16.9(1.9x)	8.7(11.7x)	9.3	2.3	3.1	4.0	4.8
S^3Attention	32.2(1.4x)	20.4(2.6x)	15.9(4.0x)	12.3	1.8	2.3	2.8	3.2

decreases by 13.24%. Similar behavior is also observed in the first row of Table 6a with $r = 1$. The information loss due to lack of rank limits the final performance. In an average sense, $s_2 = 16$ gives the

Table 5: Average Accuracy on Image task (CIFAR-10 dataset) in Long Range Arena with noise injections. The relative performance changes are reported in parentheses.

Noise level	0	2	4	8
Transformer	41.39	40.29 (-2.82%)	28.56 (-31.12%)	28.12 (-32.18%)
Linformer	38.43	37.99 (-1.49%)	37.04 (-3.95%)	36.65 (-4.97%)
Reformer	38.04	37.64 (-1.12%)	35.26 (-7.37%)	34.88 (-8.37%)
Nystroformer	41.52	40.89 (-1.66%)	38.39 (-7.67%)	37.84 (-8.99%)
Performer	42.66	41.95 (-1.93%)	39.61 (-7.40%)	38.86 (-9.15%)
S³Attention	57.47	57.06 (-0.82%)	55.32 (-3.84%)	54.70 (-4.92%)

Table 6: Experimental results on varying r, s_1 and s_2. The best result is in boldface and the second best is underlined. And Ablation experiments for each component.

(a) Experimental results on varying r parameter in smoothing component.

r	LisOps	Text	Retrieval	Image	Pathfinder	Average
1	37.30	65.25	78.65	51.36	71.23	60.76
8	38.30	69.27	**83.26**	53.90	75.82	64.11
16	**38.62**	**70.02**	83.21	54.20	**76.15**	**64.44**
32	38.19	69.27	82.05	53.73	75.58	63.76
64	37.89	69.73	81.79	51.28	75.52	63.24

(b) Experimental results on varying s_1 parameter in Row Attention.

s_1	LisOps	Text	Retrieval	Image	Pathfinder	Average
8	38.30	69.27	83.26	53.90	75.82	64.11
32	**38.44**	**70.85**	**83.41**	**54.92**	77.97	**65.12**
64	37.88	70.53	83.02	51.22	78.02	64.33
128	37.33	69.24	81.58	49.08	78.12	63.07
256	37.02	65.72	79.30	46.24	**78.14**	61.29

(c) Experimental results on varying s_2 parameter in Column Attention.

s_2	LisOps	Text	Retrieval	Image	Pathfinder	Average
1	37.32	55.28	57.37	40.97	66.25	51.44
4	37.82	52.05	72.58	46.74	73.17	57.47
8	**38.30**	69.27	83.26	53.90	75.82	64.11
16	37.77	70.24	83.42	54.11	77.92	**64.73**
32	37.62	68.32	80.11	51.66	**78.18**	62.98

best result, and further increasing in s_2 slightly harms the accuracy in all tasks except Pathfinder.

4.7 Ablation Study

This subsection provides an ablation test on four components: Fourier Convolution, Convolution Stem, Column Attention, and Row Attention. We use S³Attention with (r, s_1, s_2 = 8) as the baseline, and the detailed settings are in Table 11 in Appendix B. In Table 7 we present the accuracy changes when removing each component. The performance-decreasing results in Table 7 indicate the four components used in S³Attention are necessary to reach promising results. The most significant component is Column Attention which leads 8.28 average accuracy difference. It reflects that a good summary of the whole sequences is important. Similar observations

Table 7: Ablation experiments. The S³Attention (r, s_1, s_2 = 8) is used as baseline. The differences by removing each component from the baseline model are reported.

Model	LisOps	Text	Retrieval	Image	Pathfinder	Average
Baseline	38.30	69.27	83.26	53.90	75.82	64.11
Fouri. Conv.	-0.47	-4.04	-13.98	-5.64	-6.59	-6.14
Conv. Stem	-0.13	-0.55	-1.51	-1.76	-0.47	-0.88
Col. Attn.	-1.16	-8.00	-9.16	-10.63	-12.45	-8.28
Row Attn.	-0.38	-1.92	-1.97	-2.64	-2.56	-1.89

are also reported in Transformer-LS [77] and XCiT [2], where the spirit of Attention over columns is used in the dynamic project and Cross-Covariance Attention, respectively. The second most effective part is Fourier Convolution. It reaches a 13.89% accuracy difference in the Retrieval task involving two 4k sequences. Fourier Convolution also works well on shorter sequences tasks (e.g., Image and Pathfinder) and brings a 6.12% accuracy difference.

5 CONCLUDING REMARKS

We propose S³Attention, a robust and efficient Attention architecture for modeling long sequences with a good balance between feature preserving and noise resistance. It aggregates a Fourier convolutional stem smoothing information among tokens and a Skeleton-Sketching-inspired efficient Attention. In particular, our proposed Skeleton Attention directly samples the columns and rows of the token matrix. Such a design increases the model's robustness and gives us a positive near-linear complexity side effect. We conduct a thorough theoretical and experimental analysis of the proposed model and show its effectiveness. Lastly, extensive experiments show that the proposed model achieves the best performance on Long Range Arena and a state-of-art performance in long-term time series forecasting tasks compared to various Attention-based baselines.

One limitation of the current S³Attention is that we need to use both FFT and IFFT in a sequential manner, which is potentially slower than the existing Fourier-based Attention architectures (e.g., [36]) that only involve the FFT. As our primary goal using Fourier convolution is to smooth the token matrix and reduce the incoherent parameter, we can use the Random Fourier Transformation [1] to modify S³Attention with only FFT. Another limitation is that the size of L matrix in the Fourier Convolution part is the same as the input sequences. On longer sequences, L will contain more learnable parameters that make the model easier to overfit. We may introduce low rankness or use a more sophisticated design, such as [24], to tackle this issue in the future.

Another future research direction is to incorporate the breakthroughs in state-space model fields (e.g., [23, 24, 28, 43]). Those works obtain very competitive performance and even beat various Attention variant architectures by a large margin in several long sequences tasks. In recent work [63], the state space model even reaches comparable performance in large-scale NLP pretraining tasks to the BERT-type models. It is possible to further boost S³Attention's performance by combining with those breakthroughs.

REFERENCES

[1] Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 557–563.

[2] Alaaeldin Ali, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. 2021. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems* 34 (2021).

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

[4] Joos-Hendrik Böse, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Dustin Lange, David Salinas, Sebastian Schelter, Matthias Seeger, and Yuyang Wang. 2017. Probabilistic demand forecasting at scale. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1694–1705.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[6] HanQin Cai, Keaton Hamm, Longxiu Huang, and Deanna Needell. 2021. Robust CUR decomposition: Theory and imaging applications. *SIAM Journal on Imaging Sciences* 14, 4 (2021), 1472–1503.

[7] Emmanuel J Candès and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational mathematics* 9, 6 (2009), 717–772.

[8] Ayush Chauhan, Archiki Prasad, Parth Gupta, Amiredddy Prashanth Reddy, and Shiv Kumar Saini. 2020. Time Series Forecasting for Cold-Start Items by Learning from Related Items using Memory Networks. In *Companion Proceedings of the Web Conference 2020 (WWW'20)*. 120–121.

[9] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. 2021. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems* 34 (2021).

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

[11] Jiawei Chiu and Laurent Demanet. 2013. Sublinear randomized algorithms for skeleton decompositions. *SIAM J. Matrix Anal. Appl.* 34, 3 (2013), 1361–1383.

[12] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking Attention with Performers. In *International Conference on Learning Representations*.

[13] Sankalan Pal Chowdhury, Adamos Solomou, Avinava Dubey, and Mrinmaya Sachan. 2021. On Learning the Transformer Kernel. *arXiv preprint arXiv:2110.08323* (2021).

[14] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).

[15] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *Advances in neural information processing systems* 33 (2020), 4271–4282.

[16] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019).

[17] Prathamesh Deshpande, Kamlesh Marathe, Abir De, and Sunita Sarawagi. 2021. Long Horizon Forecasting With Temporal Point Processes. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM'21)*. 571–579.

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

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

[20] Petros Drineas, Michael W Mahoney, and Shan Muthukrishnan. 2008. Relative-error CUR matrix decompositions. *SIAM J. Matrix Anal. Appl.* 30, 2 (2008), 844–881.

[21] Sayda Elmi and Kian-Lee Tan. 2021. DeepFEC: energy consumption prediction under real-world driving conditions for smart cities. In *Proceedings of the Web Conference 2021 (WWW'21)*. 1880–1890.

[22] Christos Faloutsos, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, and Yuyang Wang. 2020. Forecasting Big Time Series: Theory and Practice. In *Proceedings of the ACM Web Conference 2020 (WWW'20)*. 320–321.

[23] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. 2020. Hippo: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems* 33 (2020), 1474–1487.

[24] Albert Gu, Karan Goel, and Christopher Re. 2022. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*. https://openreview.net/forum?id=uYLFoz1vlAC

[25] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. 2021. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems* 34 (2021), 572–585.

[26] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Re. 2023. How to Train your HIPPO: State Space Models with Generalized Orthogonal Basis Projections. In *International Conference on Learning Representations*. https://openreview.net/forum?id=klK17OQ3KB

[27] Meng-Hao Guo, Cheng-Ze Lu, Zheng-Ning Liu, Ming-Ming Cheng, and Shi-Min Hu. 2022. Visual attention network. *arXiv preprint arXiv:2202.09741* (2022).

[28] Ankit Gupta, Albert Gu, and Jonathan Berant. 2022. Diagonal State Spaces are as Effective as Structured State Spaces. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=RjS0j6tsSrf

[29] Keaton Hamm and Longxiu Huang. 2021. Perturbations of CUR decompositions. *SIAM J. Matrix Anal. Appl.* 42, 1 (2021), 351–375.

[30] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. 2023. Liquid Structural State-Space Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=g4OTKRKfS7R

[31] Min Hou, Chang Xu, Zhi Li, Yang Liu, Weiqing Liu, Enhong Chen, and Jiang Bian. 2022. Multi-Granularity Residual Learning with Confidence Estimation for Time Series Prediction. In *Proceedings of the ACM Web Conference 2022 (WWW'22)*. 112–121.

[32] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. 2021. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795* (2021).

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

[34] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The Efficient Transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

[35] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 95–104.

[36] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2021. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824* (2021).

[37] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In *Advances in Neural Information Processing Systems*, Vol. 32.

[38] Bryan Lim, Ahmed Alaa, and Mihaela van der Schaar. 2018. Forecasting Treatment Responses Over Time Using Recurrent Marginal Structural Networks. *NeurIPS* 18 (2018), 7483–7493.

[39] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198* (2018).

[40] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[41] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10012–10022.

[42] Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. 2021. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems* 34 (2021).

[43] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. 2023. Mega: Moving Average Equipped Gated Attention. In *International Conference on Learning Representations*. https://openreview.net/forum?id=qNLe3iq2El

[44] Michael Mathieu, Mikael Henaff, and Yann LeCun. 2014. Fast training of convolutional networks through ffts: International conference on learning representations (iclr2014), cbls, april 2014. In *2nd International Conference on Learning Representations, ICLR 2014*.

[45] Tan Nguyen, Vai Suliafu, Stanley Osher, Long Chen, and Bao Wang. 2021. Fmmformer: Efficient and flexible transformer via decomposed near-field and far-field attention. *Advances in Neural Information Processing Systems* 34 (2021).

[46] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *International Conference on Machine Learning*. PMLR, 4055–4064.

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

[48] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. 2021. cosFormer: Rethinking Softmax In Attention. In *International Conference on Learning Representations*.

[49] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. 2019. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972* (2019).

[50] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507* (2019).

[51] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics* 9 (2021), 53–68.

[52] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. 2023. Simplified State Space Layers for Sequence Modeling. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Ai8Hw3AXqks

[53] G. Sreenu and M A Durai. 2019. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data* 6 (06 2019), 48. https://doi.org/10.1186/s40537-019-0212-5

[54] Chao-Hong Tan, Qian Chen, Wen Wang, Qinglin Zhang, Siqi Zheng, and Zhen-Hua Ling. 2021. PoNet: Pooling Network for Efficient Token Mixing in Long Sequences. *arXiv preprint arXiv:2110.02442* (2021).

[55] Y Tay, D Bahri, D Metzler, D Juan, Z Zhao, and C Zheng. 2020. Synthesizer: Rethinking self-attention in transformer models. arXiv 2020. *arXiv preprint arXiv:2005.00743* (2020).

[56] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020. Sparse sinkhorn attention. In *International Conference on Machine Learning*. PMLR, 9438–9447.

[57] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long Range Arena : A Benchmark for Efficient Transformers. In *International Conference on Learning Representations*.

[58] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732* (2020).

[59] Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast Character Transformers via Gradient-based Subword Tokenization. In *International Conference on Learning Representations*.

[60] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*. PMLR, 10347–10357.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[62] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. 2019. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems* 32 (2019).

[63] Junxiong Wang, Jing Nathan Yan, Albert Gu, and Alexander M Rush. 2022. Pretraining Without Attention. *arXiv preprint arXiv:2212.10544* (2022).

[64] Pichao Wang, Xue Wang, Hao Luo, Jingkai Zhou, Zhipeng Zhou, Fan Wang, Hao Li, and Rong Jin. 2021. Scaled relu matters for training vision transformers. *arXiv preprint arXiv:2109.03810* (2021).

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

[66] Genta Indra Winata, Samuel Cahyawijaya, Zhaojiang Lin, Zihan Liu, and Pascale Fung. 2020. Lightweight and efficient end-to-end speech recognition using low-rank transformer. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 6144–6148.

[67] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. 101–112.

[68] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. 2021. Nyströmformer: A Nyström-based Algorithm for Approximating Self-Attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 14138–14148.

[69] Jiehui Xu, Jianmin Wang, Mingsheng Long, et al. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems* 34 (2021).

[70] Li Yuan, Qibin Hou, Zihang Jiang, Jiashi Feng, and Shuicheng Yan. 2021. Volo: Vision outlooker for visual recognition. *arXiv preprint arXiv:2106.13112* (2021).

[71] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems* 33 (2020), 17283–17297.

[72] J Zhang and K Nawata. 2018. Multi-step prediction for influenza outbreak by an adjusted long short-term memory. *Epidemiology & Infection* 146, 7 (2018), 809–816.

[73] Dawei Zhou, Lecheng Zheng, Yada Zhu, Jianbo Li, and Jingrui He. 2020. Domain adaptive multi-modality neural attention network for financial forecasting. In *Proceedings of The Web Conference 2020 (WWW'20)*. 2230–2240.

[74] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*, Vol. 35. 11106–11115.

[75] Jingkai Zhou, Pichao Wang, Fan Wang, Qiong Liu, Hao Li, and Rong Jin. 2021. ELSA: Enhanced Local Self-Attention for Vision Transformer. *arXiv preprint arXiv:2112.12786* (2021).

[76] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. 2022. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *39th International Conference on Machine Learning (ICML)*.

[77] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. 2021. Long-short transformer: Efficient transformers for language and vision. *Advances in Neural Information Processing Systems* 34 (2021).

[78] Zhenhai Zhu and Radu Soricut. 2021. H-Transformer-1D: Fast One-Dimensional Hierarchical Attention for Sequences. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 3801–3815.

Supplementary Material for
S^3Attention: Improving Long Sequence Attention with Smoothed Skeleton Sketching

A ALGORITHMS

Algorithm 1 Skeleton Attention

```python
class Skeleton_Attention(nn.Module):
    def __init__(self, num_head = 2, head_dim = 32,seq_len, left_rank = 8,right_rank = 8, dropout = 0.1):
        super(Skeleton_Attention, self).__init__()
        self.num_head = num_head
        self.head_dim = head_dim
        self.seq_len = seq_len
        self.left_rank = left_rank
        self.right_rank = right_rank

        self.ln_1 = nn.LayerNorm(self.num_head * self.head_dim)
        self.ln_2 = nn.LayerNorm(self.num_head * self.head_dim)

        self.drop_attn = torch.nn.Dropout(p=dropout)

        self.index_set_right = torch.randperm(self.head_dim)
        self.index_set_right = self.index_set_right[:self.right_rank]

        self.index_set_left = torch.randperm(self.seq_len)
        self.index_set_left = self.index_set_left[:self.left_rank]

    def combine_heads(self, X):
        X = X.transpose(1, 2)
        X = X.reshape(X.size(0), X.size(1), self.num_head * self.head_dim)
        return X

    def split_heads(self, X):
        X = X.reshape(X.size(0), X.size(1), self.num_head, self.head_dim)
        X = X.transpose(1, 2)
        return X

    def forward(self,Q, K, V):
        #### Row Attention ####
        if self.left_rank <= self.seq_len:
            K1 = K[:,:,self.index_set_left,:]
            V1 = V[:,:,self.index_set_left,:]
        else:
            K1 = K
            V1 = V

        dots = Q @ K1.transpose(-1,-2)
        dots = dots / math.sqrt(self.head_dim)
        attn = nn.functional.softmax(dots,dim=-1)
        attn = self.drop_attn(attn)

        #### Column Attention ####
        Q2 = Q.transpose(-1,-2)
        if self.right_rank <= self.head_dim:

            K2 = K[:,:,:,self.index_set_right]
            V2 = V[:,:,:,self.index_set_right]
        else:
            K2 = K
            V2 = V

        dots_r = Q2 @ K2
        dots_r = dots_r / math.sqrt(self.seq_len)
        attn_r = nn.functional.softmax(dots_r,dim=-1).transpose(-1,-2)
        attn_r = self.drop_attn(attn_r)

        X = self.split_heads(self.ln_1(self.combine_heads(torch.matmul(attn,V1))))/2 + self.split_heads(self.ln_2(self.combine_heads(torch.
            matmul(V2,attn_r))))/2

        return X
```

Algorithm 2 Smoother component

```
class Smoother(nn.Module):

    def __init__(self, hidden_size, seq_len, dropout = 0.5, num_head = 2,transformer_dim = 64, fold = 1):

        super(Smoother, self).__init__()

        self.hidden_size = hidden_size
        self.seq_len = seq_len
        self.dropout = dropout
        self.num_head = num_head
        self.dim = transformer_dim
        self.fold = fold

        self.weights_fft = nn.Parameter(torch.empty(self.seq_len//2+1, self.hidden_size,2))
        nn.init.kaiming_normal_(self.weights_fft, mode='fan_in', nonlinearity='relu')

        self.tiny_conv_linear = torch.nn.Conv1d(in_channels = self.hidden_size*2 , out_channels = self.hidden_size, kernel_size = 3, padding=
            1, groups = 1)
        self.dropout = torch.nn.Dropout(p=self.dropout)
        self.bn_1 = nn.BatchNorm1d(self.seq_len)

    def forward(self, x):

        #### Compute Segment Average ####
        B,S,H = x.shape
        u = x.reshape(B,S,self.fold,H//self.fold)
        u = torch.mean(u,dim = -1)

        #### Fourier Convolution ####
        fft_u = fft.rfft(u, n = self.seq_len, axis = -2)
        fft_u = torch.view_as_real(fft_u)
        fft_u = fft_u.repeat(1,1,H//self.fold,1)
        self.weight_used = self.weights_fft.unsqueeze(0)
        temp_real = fft_u[...,0]*self.weight_used[...,0] - fft_u[...,1]*self.weight_used[...,1]
        temp_imag = fft_u[...,0]*self.weight_used[...,1] + fft_u[...,1]*self.weight_used[...,0]
        out_ft = torch.cat([temp_real.unsqueeze(-1),temp_imag.unsqueeze(-1)],dim = -1)
        out_ft = torch.view_as_complex(out_ft)
        m = fft.irfft(out_ft, n = self.seq_len, axis = -2)

        #### Convolution Stem ####
        input_h = torch.cat((m, x), dim = -1)
        h = self.tiny_conv_linear(input_h.permute(0,2,1)).permute(0,2,1)
        h = self.dropout(F.relu(self.bn_1(h)))

        return h
```

Algorithm 3 pseudo code for Time-Series Forecasting

```
    def forward(self, x_in):
        B1,H1,C1 = x_in.shape
        for i in range(len(self.encoder)):
            attn_layer = self.encoder[i]
            #standardize the input data
            if i == 0:
                tmp_mean = torch.mean(x_in[:,:,:],dim = 1,keepdim = True)
                tmp_std = torch.sqrt(torch.var(x_in[:,:,:],dim = 1,keepdim = True)+1e0)
                x_in = (x_in - tmp_mean)/(tmp_std)

                enc_out1 = self.enc_embedding(x_in)

            enc_out1= attn_layer(enc_out1) + enc_out1

        #decoder via Fourier Extrapolation
        dec_out = self.fourierExtrapolation(post(enc_out1))
        output = (dec_out.reshape(B1,-1,C1))*(tmp_std)+tmp_mean
        return output
```

B EXPERIMENT CONFIGURATIONS

In this section, we report the configurations in Tables 8-11 for the experiments in Section 4.

Algorithm 4 Fourier Extrapolation

```python
class fourierExtrapolation(nn.Module):
    def __init__(self,inputSize,n_harm = 8,n_predict = 96):
        super().__init__()
        self.n = inputSize
        self.n_harm = n_harm
        self.f = torch.fft.fftfreq(self.n)
        self.indexes = list(range(self.n))


        # sort indexes by frequency, lower -> higher
        self.indexes.sort(key = lambda i: torch.absolute(self.f[i]))
        self.indexes = self.indexes[:1 + self.n_harm * 2]

        self.n_predict = n_predict


        # compute init phase
        self.t = torch.arange(0, self.n + self.n_predict)
        self.t1 = self.t.unsqueeze(0).unsqueeze(-1).float().to('cuda')
        self.f = self.f.unsqueeze(0).unsqueeze(-1).to('cuda')
        self.t = self.t.unsqueeze(0).unsqueeze(-1).unsqueeze(-1)to('cuda')
        self.g = self.f[:,self.indexes,:].permute(0,2,1).unsqueeze(1)
        self.phase_init = 2 * 3.1415 * self.g * self.t


    def fourierExtrapolation(self,x):

        # x in frequency domain
        x_freqdom = torch.fft.fft(x,dim = -2)
        x_freqdom = torch.view_as_real(x_freqdom)
        # select importance frequencies
        x_freqdom = x_freqdom[:,self.indexes ,:,:]
        x_freqdom = torch.view_as_complex(x_freqdom)
        ampli = torch.absolute(x_freqdom) / self.n # amplitude
        phase = torch.angle(x_freqdom) # phase

        ampli = ampli.permute(0,2,1).unsqueeze(1)
        phase = phase.permute(0,2,1).unsqueeze(1)

        self.restored_sig = ampli * torch.cos(self.phase_init + phase)

        return torch.sum(self.restored_sig,dim = -1)
```

Table 8: Experiment Configuration of S^3Attention ($r, s_1, s_2 = 8$).

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Epoch	5	30	15	60	100
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	0	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
r, s_1, s_2	8,8,8	8,8,8	8,8,8	8,8,8	8,8,8
dropout in embedding	0	0.5	0.1	0.1	0
dropout in attention	0	0.1	0.1	0.1	0
dropout in smoother	0	0.5	0.1	0.5	0.5

C ADDITIONAL RESULTS ON LRA

We have already provided the average of 5 runs with different random seeds in Table 1. Here we also provide the standard deviations for these experiments in Table 12.

D DATASET AND IMPLEMENTATION DETAILS

In this subsection, we summarize the details of the datasets used in this paper as follows:

LRA datasets: **ListOps**(2K length mathematical expression task which investigates the parsing ability); **Text** (up to 4K byte/character-level document classification task that tests capacity in character compositionality); **Retrieval** (byte/character-level document matching task, which exams the information compression ability with two 4K length sequence); **Image** (pixel-wise sequence image classification based on

Table 9: Experiment Configuration of S^{3}Attention (best).

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Epoch	10	30	15	60	100
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	1e-2	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
r, s_1, s_2	8,8,8	8,8,8	8,32,32	8,16,16	8,128,32
dropout in embedding	0	0.5	0.1	0.5	0.1
dropout in attention	0	0.1	0.1	0.1	0.1
dropout in smoother	0	0.5	0.1	0.5	0.5

Table 10: Experiment Configuration for Model Parameters Impact.

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Epoch	5	30	15	60	100
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	0	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
dropout in embedding	0	0.5	0.1	0.1	0
dropout in attention	0	0.1	0.1	0.1	0
dropout in smoother	0	0.5	0.1	0.5	0.5

Table 11: Experiment Configuration for Ablation.

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	0	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
r, s_1, s_2	8,8,8	8,8,8	8,8,8	8,8,8	8,8,8
dropout in embedding	0	0.5	0.1	0.1	0
dropout in attention	0	0.1	0.1	0.1	0
dropout in smoother	0	0.5	0.1	0.5	0.5

Table 12: Accuracy on Long Range Arena (LRA) with standard errors shown in parenthesis. All results are averages of 5 runs with different random seeds.

Model	LisOps	Text	Retrieval	Image	Pathfinder
S^{3}Attention ($r, s_1, s_2 = 8$)	38.30 (0.40)	69.27 (0.83)	83.26 (0.45)	53.90 (1.54)	75.82 (0.97)
S^{3}Attention (best)	39.15 (0.48)	71.58 (0.95)	83.73 (0.61)	57.73 (1.83)	78.20 (1.32)

the CIFAR-10 dataset); **Pathfinder** (long-range spatial dependency identification task. The input images contain two small points/circles and dash-line paths. The model needs to identify whether two points/circles are connected);The LRA has several desirable advantages that made us focus on it as the evaluation benchmark: **generality** (only requires the encoder part); **simplicity** (data augmentation and pretraining are out of scope); **challenging long inputs** (difficulty enough and room to improve); **diversity aspects** (tasks covering math, language, image, and spatial modeling); and **lightweight** (run with low resource requirement).

Time series datasets:1) ETT [74] dataset contains two sub-dataset: ETT1 and ETT2, collected from two separated counties. Each of them has two versions of sampling resolutions (15min & 1h). ETT dataset contains multiple time series of electrical loads and one time sequence of oil temperature. 2) Electricity[3] dataset contains the electricity consumption for more than three hundred clients with each column corresponding to one client. 3) Exchange [35] dataset contains the current exchange of eight countries. 4) Traffic[4] dataset contains the occupation rate of freeway systems in California, USA. 5) Weather[5] dataset contains 21 meteorological indicators for a range of one year

---

[3]https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014.
[4]http://pems.dot.ca.gov.
[5]https://www.bgc-jena.mpg.de/wetter.

Table 13: Details of time series benchmark datasets.

DATASET	LENGTH	DIMENSION	FREQUENCY
ETTM2	69680	8	15 MIN
EXCHANGE	7588	9	1 DAY
WEATHER	52696	22	10 MIN
ELECTRICITY	26304	322	1H
ILI	966	8	7 DAYS
TRAFFIC	17544	863	1H

in Germany. 6) Illness[6] dataset contains the influenza-like illness patients in the United States. Table 13 summarizes all the features for the six benchmark datasets. They are all split into the training set, validation set and test set by the ratio of 7:1:2 during modeling.

GLUE datasets: The GLUE benchmark covers various natural language understanding tasks and is widely used in evaluating transfering ability. The tasks can be devided in to two types, single-sentence tasks (SST-2 and CoLA), and sentence-pair tasks (MNLI, QQP,QNLI,STS-B,MRPC,RTE). Following the same settings in [18], we exclude WNLI task.

E   EXPERIMENTS ON THE SMOOTHNESS EFFECT OF FOURIER CONVOLUTION

In this section, we verify Fourier convolution component in the Smoother block can reduce the incoherence value in the early training stage. We use S^3Attention with $(r, s_1, s_2 = 8)$ as the test model and test on an NLP dataset: Text, and a vision dataset: Pathfinder. We compute the μ-incoherence value. of the token matrix before and after the Fourier convolution (denoted as μ_X and $\mu_{X^{smooth}}$, respectively) for each samples in the validation dataset. Since we do not explicitly force the token matrix to be low-rank required by Definition 1, we report the incoherence value for different rankness settings (rank = 16 and rank = 32) approximately, and the mean and standard deviation of incoherence value can be found in Table 14. The average incoherence value reduced 30% after the Fourier convolution in both datasets. Moreover, We observe that the standard deviation significantly decreases, which suggests the Fourier convolution may also potentially stabilize the training procedure.

Table 14: The average incoherence parameters after 100 training steps with standard errors shown in the parenthesis.

Dataset	μ_X (rank = 32)	$\mu_{X^{smooth}}$ (rank = 32)	μ_X (rank = 16)	$\mu_{X^{smooth}}$ (rank = 16)
Text	2.75 (0.027)	2.05 (0.007)	3.98 (0.046)	3.23 (0.038)
Pathfinder	3.83 (0.221)	1.99 (0.001)	4.88 (0.264)	3.48 (0.001)

F   ILLUSTRATION ON EFFECT OF THE SMOOTHER AND SKELETON ATTENTION IN TOKEN MATRIX

In this section, an illustration of the Smoother and Skeleton Attention part is shown in Figure 2. We smooth the input token matrix to ensure the sampling in rows and columns containing more local and/or global information. Thus, sampling several rows and columns from the smoothed token matrix can be more effective than the samples from the original token matrix.
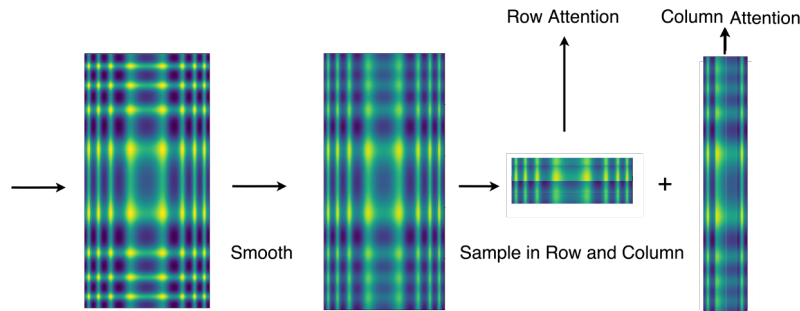
Figure 2: Illustration on effect of the Smoother and Skeleton Attention on Token Matrix.

---

[6]https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html.

G   TRANSFER LEARNING

Table 15: The training configurations for Pretraining and GLUE tasks

	Pre-training	GLUE
Max Steps	1000K	-
Max Epochs	-	[4,20]
Learning Rate	1e-4	[5e-5,1e-4]
Batch Size	256	[16,32]
Warm-up Steps	5000	-
Sequence Length	512	128
Learning Rate Decay	-	Linear
Clip	-	1
Dropout	-	0.1

H   EXTRA ALGORITHMS FOR LONGTERM TIMESERIES FORECASTING

Table 16: multivariate long-term series forecasting results on six datasets with input length of 96 **and prediction length** $O \in \{96, 192, 336, 720\}$ **(For ILI dataset, we set prediction length** $O \in \{24, 36, 48, 60\}$**) with input length** 60**. A lower MSE indicates better performance. All experiments are repeated 5 times.**

Methods		S3Attention		FEDformer		Autoformer		Informer		LogTrans		Reformer		Performer		Nystroformer	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm2	96	**0.192**	**0.283**	0.203	0.287	0.255	0.339	0.365	0.453	0.768	0.642	0.658	0.619	2.520	1.220	0.617	0.600
	192	**0.255**	**0.324**	0.269	0.328	0.281	0.340	0.533	0.563	0.989	0.757	1.078	0.827	0.949	0.745	0.856	0.706
	336	**0.324**	**0.364**	0.325	0.366	0.339	0.372	1.363	0.887	1.334	0.872	1.549	0.972	1.701	1.001	1.394	0.887
	720	0.431	0.433	**0.421**	**0.415**	0.422	0.419	3.379	1.338	3.048	1.328	2.631	1.242	2.531	1.228	2.466	1.185
Electricity	96	0.218	0.332	**0.183**	**0.297**	0.201	0.317	0.274	0.368	0.258	0.357	0.312	0.402	0.281	0.375	0.273	0.364
	192	0.259	0.361	**0.195**	**0.308**	0.222	0.334	0.296	0.386	0.266	0.368	0.348	0.433	0.283	0.387	0.282	0.370
	336	0.267	0.367	**0.212**	**0.313**	0.231	0.338	0.300	0.394	0.280	0.380	0.350	0.433	0.301	0.387	0.302	0.387
	720	0.293	0.385	**0.231**	**0.343**	0.254	0.361	0.373	0.439	0.283	0.376	0.340	0.420	0.301	0.387	0.292	0.373
Exchange	96	**0.086**	**0.204**	0.139	0.276	0.197	0.323	0.847	0.752	0.968	0.812	1.065	0.829	0.801	0.729	0.801	0.729
	192	**0.188**	**0.292**	0.256	0.369	0.300	0.369	1.204	0.895	1.040	0.851	1.188	0.906	1.284	0.925	1.284	0.925
	336	**0.356**	**0.433**	0.426	0.464	0.509	0.524	1.672	1.036	1.659	1.081	1.357	0.976	1.408	0.964	1.408	0.964
	720	**0.727**	**0.669**	1.090	0.800	1.447	0.941	2.478	1.310	1.941	1.127	1.510	1.016	1.654	1.017	1.654	1.017
Traffic	96	0.592	0.352	**0.562**	**0.349**	0.613	0.388	0.824	0.391	0.684	0.384	0.732	0.423	0.709	0.391	0.709	0.400
	192	0.583	0.343	**0.562**	**0.346**	0.616	0.382	0.696	0.379	0.685	0.390	0.733	0.420	0.681	0.369	1.127	0.611
	336	0.598	0.346	**0.570**	**0.323**	0.622	0.337	0.777	0.420	0.733	0.408	0.742	0.420	0.682	0.366	0.867	0.477
	720	0.641	0.397	**0.596**	**0.368**	0.660	0.408	0.864	0.472	0.717	0.396	0.755	0.423	0.675	0.360	0.686	0.369
Weather	96	**0.182**	**0.262**	0.217	0.296	0.266	0.336	0.300	0.384	0.458	0.490	0.689	0.596	0.597	0.598	0.701	0.612
	192	**0.228**	**0.306**	0.276	0.336	0.307	0.367	0.598	0.544	0.658	0.589	0.752	0.638	0.606	0.587	0.655	0.604
	336	**0.295**	**0.355**	0.339	0.380	0.359	0.395	0.578	0.523	0.797	0.652	0.639	0.596	0.731	0.646	0.746	0.642
	720	**0.383**	**0.418**	0.403	0.428	0.578	0.578	1.059	0.741	0.869	0.675	1.130	0.792	0.837	0.682	0.961	0.751
ILI	24	**2.185**	**0.926**	2.203	0.963	3.483	1.287	4.631	1.677	4.480	1.444	4.400	1.382	3.937	1.298	4.378	1.364
	36	**2.155**	**0.937**	2.272	0.976	3.103	1.148	4.755	1.467	4.799	1.467	4.783	1.448	4.007	1.329	5.332	1.554
	48	2.333	0.954	**2.209**	**0.981**	2.669	1.085	4.763	1.469	4.800	1.468	4.832	1.465	4.575	1.451	5.575	1.614
	60	**2.018**	**0.958**	2.545	1.061	2.770	1.125	5.264	1.564	5.278	1.560	4.882	1.483	4.020	1.366	4.742	1.469

I   LEARNING CURVE TABLES FOR LRA EXPERIMENTS

n this section, we present the training and testing performance for the first 5000 training steps on five LRA datasets. All hyperparameters are kept the same as the baseline model in ablation study. The average training accuracy/loss and test accuracy/loss are reported from the Table 17 to Table 21. The results indicate the smoothed tokens lead to better learning behavior and faster convergence, which supports its stabilization ability.

Table 17: Results for Listops dataset.

step	with smoother				without smoother			
	train acc	train loss	test acc	test loss	train acc	train loss	test acc	test loss
500	0.147	2.28	0.171	2.26	0.138	2.28	0.092	2.31
1000	0.16	2.27	0.173	2.26	0.154	2.27	0.172	2.26
1500	0.162	2.26	0.174	2.26	0.159	2.26	0.171	2.26
2000	0.163	2.26	0.175	2.26	0.161	2.26	0.177	2.26
2500	0.165	2.26	0.175	2.26	0.164	2.26	0.172	2.26
3000	0.191	2.23	0.273	2.12	0.167	2.26	0.165	2.26
3500	0.262	2.08	0.273	2.12	0.176	2.25	0.175	2.26
4000	0.295	1.99	0.357	1.83	0.182	2.25	0.178	2.25
4500	0.327	1.87	0.361	1.74	0.194	2.24	0.235	2.2
5000	0.361	1.74	0.364	1.73	0.206	2.22	0.252	2.16

Table 18: Results for Text dataset.

step	with smoother				without smoother			
	train acc	train loss	test acc	test loss	train acc	train loss	test acc	test loss
500	0.507	0.694	0.506	0.693	0.507	0.693	0.505	0.693
1000	0.516	0.692	0.516	0.692	0.516	0.692	0.507	0.693
1500	0.531	0.69	0.528	0.692	0.528	0.692	0.521	0.692
2000	0.573	0.681	0.542	0.69	0.534	0.69	0.524	0.693
2500	0.603	0.666	0.632	0.648	0.544	0.684	0.547	0.686
3000	0.658	0.626	0.649	0.631	0.57	0.679	0.584	0.674
3500	0.685	0.603	0.666	0.619	0.626	0.649	0.616	0.664
4000	0.708	0.58	0.68	0.607	0.639	0.634	0.616	0.656
4500	0.716	0.56	0.692	0.595	0.666	0.614	0.621	0.656
5000	0.737	0.549	0.696	0.585	0.685	0.598	0.625	0.658

Table 19: Results for Retrieval dataset.

step	with smoother				without smoother			
	train acc	train loss	test acc	test loss	train acc	train loss	test acc	test loss
500	0.514	0.693	0.517	0.692	0.523	0.693	0.512	0.692
1000	0.516	0.692	0.535	0.688	0.523	0.692	0.534	0.69
1500	0.53	0.688	0.576	0.676	0.527	0.69	0.545	0.685
2000	0.541	0.684	0.63	0.635	0.531	0.688	0.573	0.675
2500	0.558	0.673	0.675	0.591	0.54	0.685	0.608	0.649
3000	0.575	0.661	0.697	0.564	0.557	0.676	0.627	0.629
3500	0.589	0.65	0.703	0.555	0.568	0.668	0.639	0.618
4000	0.603	0.638	0.672	0.586	0.579	0.659	0.66	0.604
4500	0.657	0.593	0.696	0.562	0.592	0.649	0.653	0.608
5000	0.711	0.549	0.72	0.538	0.679	0.575	0.702	0.558

Table 20: Results for Pathfinder dataset.

step	with smoother				without smoother			
	train acc	train loss	test acc	test loss	train acc	train loss	test acc	test loss
500	0.502	0.693	0.501	0.693	0.501	0.693	0.501	0.693
1000	0.504	0.693	0.501	0.693	0.502	0.693	0.501	0.693
1500	0.505	0.693	0.502	0.693	0.502	0.693	0.501	0.693
2000	0.506	0.693	0.503	0.693	0.506	0.693	0.502	0.693
2500	0.508	0.692	0.505	0.692	0.506	0.692	0.502	0.693
3000	0.51	0.691	0.509	0.692	0.507	0.692	0.502	0.693
3500	0.513	0.691	0.507	0.692	0.507	0.692	0.504	0.693
4000	0.517	0.69	0.512	0.691	0.514	0.692	0.511	0.692
4500	0.527	0.688	0.515	0.691	0.517	0.692	0.515	0.692
5000	0.535	0.682	0.529	0.689	0.526	0.689	0.52	0.691

Table 21: Results for Image dataset.

step	with smoother				without smoother			
	train acc	train loss	test acc	test loss	train acc	train loss	test acc	test loss
500	0.372	1.72	0.407	1.65	0.33	1.84	0.346	1.81
1000	0.454	1.49	0.447	1.51	0.41	1.64	0.38	1.73
1500	0.499	1.38	0.47	1.47	0.449	1.54	0.401	1.7
2000	0.535	1.29	0.496	1.42	0.488	1.44	0.401	1.7
2500	0.558	1.22	0.505	1.4	0.516	1.36	0.381	1.83
3000	0.569	1.18	0.514	1.39	0.528	1.32	0.379	1.84
3500	0.594	1.14	0.528	1.34	0.545	1.29	0.374	1.96
4000	0.605	1.11	0.541	1.33	0.549	1.27	0.365	2.02
4500	0.616	1.08	0.536	1.34	0.595	1.17	0.357	2.03
5000	0.625	1.04	0.523	1.38	0.583	1.14	0.349	2.05