On the Parameterization and Initialization of Diagonal State Space Models

Albert Gu[†], **Ankit Gupta**[‡], **Karan Goel**[†], **Christopher Ré**[†]
[†] Department of Computer Science, Stanford University
[‡] IBM Research
{albertgu,knrg}@stanford.edu, chrismre@cs.stanford.edu
ankitgupta.iitkanpur@gmail.com

Abstract

State space models (SSM) have recently been shown to be very effective as a deep learning layer as a promising alternative to sequence models such as RNNs, CNNs, or Transformers. The first version to show this potential was the S4 model, which is particularly effective on tasks involving long-range dependencies by using a prescribed state matrix called the HiPPO matrix. While this has an interpretable mathematical mechanism for modeling long dependencies, it introduces a custom representation and algorithm that can be difficult to implement. On the other hand, a recent variant of S4 called DSS showed that restricting the state matrix to be fully diagonal can still preserve the performance of the original model when using a specific initialization based on approximating S4's matrix. This work seeks to systematically understand how to parameterize and initialize such diagonal state space models. While it follows from classical results that almost all SSMs have an equivalent diagonal form, we show that the initialization is critical for performance. We explain why DSS works mathematically, by showing that the diagonal restriction of S4's matrix surprisingly recovers the same kernel in the limit of infinite state dimension. We also systematically describe various design choices in parameterizing and computing diagonal SSMs, and perform a controlled empirical study ablating the effects of these choices. Our final model S4D is a simple diagonal version of S4 whose kernel computation requires just 2 lines of code and performs comparably to S4 in almost all settings, with state-of-the-art results for image, audio, and medical time-series domains, and averaging 85% on the Long Range Arena benchmark.

1 Introduction

A core class of models in modern deep learning are sequence models, which are parameterized mappings operating on arbitrary sequences of inputs. Recent approaches based on state space models (SSMs) have outperformed traditional deep sequence models such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and Transformers, in both computational efficiency and modeling ability. In particular, the S4 model displayed strong results on a range of sequence modeling tasks, especially on long sequences [9]. Its ability to model long-range dependencies arises from being defined with a particular state matrix called the "HiPPO matrix" [6], which allows S4 to be viewed as a convolutional model that decomposes an input onto an orthogonal system of smooth basis functions[10].

However, beyond its theoretical interpretation, actually computing S4 as a deep learning model requires a sophisticated algorithm with many linear algebraic techniques that are difficult to understand and implement. These techniques were necessitated by parameterizing its state matrix as a **diagonal plus low-rank** (DPLR) matrix, which is necessary to capture HiPPO matrices. A natural question is

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

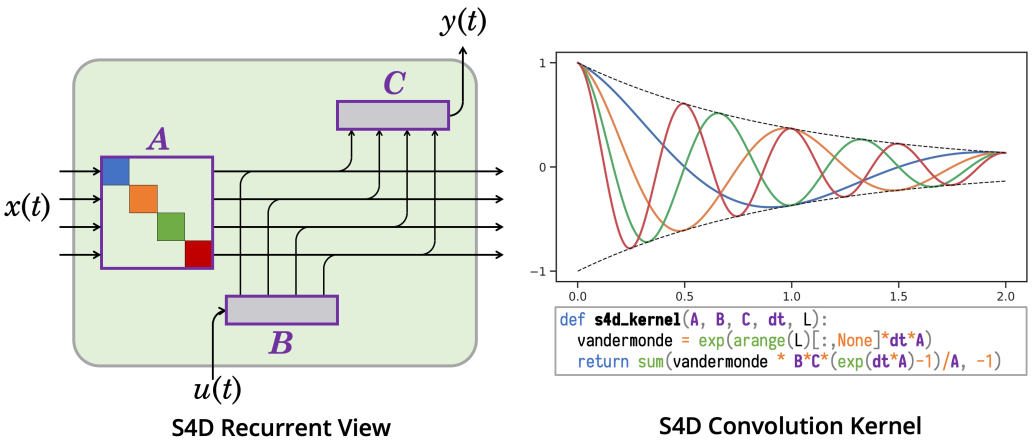


Figure 1: S4D is a diagonal SSM which inherits the strengths of S4 while being much simpler. (*Left*) The diagonal structure allows it to be viewed as a collection of 1-dimensional SSMs. (*Right*) As a convolutional model, S4D has a simple interpretable convolution kernel which can be implemented in two lines of code. Colors denote independent 1-D SSMs; purple denotes trainable parameters.

whether simplifications of this parameterization and algorithm are possible. In particular, removing the low-rank term would result in a **diagonal state space model** (diagonal SSM) that is dramatically simpler to implement and understand.

Although it is known that almost all SSMs have an equivalent diagonal form—and therefore (complex) diagonal SSMs are fully expressive algebraically—they may not represent all SSMs numerically, and finding a good initialization is critical. Gu et al. [9] showed that it is difficult to find a performant diagonal SSM, and that many alternative parameterizations of the state matrix – including by random diagonal matrices – are much less effective empirically, which motivated the necessity of the more complicated HiPPO matrix. However, recently Gupta [11] made the empirical observation that a variant of S4 using a *particular diagonal matrix* is nearly as effective as the original S4 method. This matrix is based on the original HiPPO matrix and is defined by simply chopping off the low-rank term in the DPLR representation.

The discovery of performant diagonal state matrices opens up new possibilities for simplifying deep state space models, and consolidating models such as S4 and DSS to understand and improve them. First, the strongest version of DSS *computes* the SSM with a complex-valued softmax that complicates the algorithm, and is actually less efficient than S4. Additionally, DSS and S4 differ in several auxiliary aspects of *parameterizing* SSMs that can conflate performance effects, making it more difficult to isolate the core effects of diagonal versus DPLR state matrices. Most importantly, DSS relies on *initializing* the state matrix to a particular approximation of S4's HiPPO matrix. While S4's matrix has a mathematical interpretation for addressing long-range dependencies, the efficacy of the diagonal approximation to it remains theoretically unexplained.

In this work, we seek to systematically understand how to train diagonal SSMs. We introduce the **S4D** method, a diagonal SSM which combines the best of S4's *computation* and *parameterization* and DSS's *initialization*, resulting in a method that is extremely simple, theoretically princpled, and empirically effective.

- First, we describe S4D, a simple method outlined by S4 for computing diagonal instead of DPLR matrices, which is based on **Vandermonde matrix multiplication** and is even simpler and more efficient than the DSS. Outside of the core state matrix, we categorize different representations of the other components of SSMs, introducing flexible design choices that capture both S4 and DSS and allow different SSM parameterizations to be systematically compared (Section 3).
- We provide a new mathematical analysis of DSS's initialization, showing that the diagonal approximation of the original HiPPO matrix surprisingly produces the same dynamics as S4 when the state size goes to infinity. We propose even simpler variants of diagonal SSMs using different initializations of the state matrix (Section 4).
- We perform a controlled study of these various design choices across many domains, tasks, and sequence lengths, and additionally compare diagonal (S4D) versus DPLR (S4) variants. Our best S4D methods are competitive with S4 on almost all settings, with near state-of-the-art results on

image, audio, and medical time series benchmarks, and achieving **85%** on the Long Range Arena benchmark (Section 5).

2 Background

Continuous State Spaces Models S4 investigated state space models (1) that are parameterized maps on signals $u(t) \mapsto y(t)$. These SSMs are linear time-invariant systems that can be represented either as a linear ODE (equation (1)) or convolution (equation (2)).

$$\begin{aligned} x'(t) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ y(t) &= \boldsymbol{C}x(t) \end{aligned} \tag{1} \qquad \begin{aligned} K(t) &= \boldsymbol{C}e^{t\boldsymbol{A}}\boldsymbol{B} \\ y(t) &= (K * u)(t) \end{aligned} \tag{2}$$

Here the parameters are the state matrix $\boldsymbol{A} \in \mathbb{C}^{N\times N}$ and other matrices $\boldsymbol{B} \in \mathbb{C}^{N\times 1}, \boldsymbol{C} \in \mathbb{C}^{1\times N}$. In the case of diagonal SSMs, $\boldsymbol{A}$ is diagonal and we will overload notation so that $\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{C}_n$ denotes the entries of the parameters.

An intuitive way to view the convolution kernel (2) is to interpret it as a linear combination (controlled by $\boldsymbol{C}$) of **basis kernels** $K_n(t)$ (controlled by $\boldsymbol{A}, \boldsymbol{B}$)

$$K(t) = \sum_{n=0}^{N-1} \boldsymbol{C}_n K_n(t) \qquad K_n(t) := \boldsymbol{e}_n^\top e^{t\boldsymbol{A}}\boldsymbol{B} \tag{3}$$

We denote this basis as $K(t) = K_{\boldsymbol{A},\boldsymbol{B}}(t) = e^{t\boldsymbol{A}}\boldsymbol{B}$ if necessary to disambiguate; note that it is a vector of N functions. In the case of diagonal SSMs, each function $K_n(t)$ is just $e^{t\boldsymbol{A}_n}\boldsymbol{B}_n$.

S4: Structured State Spaces As a deep learning model, SSMs have many elegant properties with concrete empirical and computational benefits [8]. For example, the convolutional form (2) can be converted into a temporal recurrence that is substantially faster for autoregressive applications [5].

However, making SSMs effective required overcoming two key challenges: choosing appropriate values for the matrices, and computing the kernel (2) efficiently.

First, Gu et al. [8] showed that naive instantiations of the SSM do not perform well, and instead relied on a particular (real-valued) matrix $\boldsymbol{A}$ called the HiPPO-LegS matrix (4).[1] These matrices were derived so that the basis kernels $K_n(t)$ have closed-form formulas $L_n(e^{-t})$, where $L_n(t)$ are normalized Legendre polynomials. Consequently, the SSM has a mathematical interpretation of decomposing the input signal $u(t)$ onto a set of infinitely-long basis functions that are orthogonal respect to an exponentially-decaying measure, giving it long-range modeling abilities [10].

Second, S4 introduced a particular parameterization that decomposed this $\boldsymbol{A}$ matrix into the sum of a normal and rank-1 matrix (5), which can be unitarily conjugated into a (complex) diagonal plus rank-1 matrix. Leveraging this structured form, they then introduced a sophisticated algorithm for efficiently computing the convolution kernel (2) for state matrices that are **diagonal plus low-rank (DPLR)**.

$$\boldsymbol{A}_{nk} = -\begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & n > k \\ n+1 & n = k \\ 0 & n < k \end{cases} \qquad \boldsymbol{B}_n = (2n+1)^{\frac{1}{2}} \quad \boldsymbol{P}_n = (n+1/2)^{\frac{1}{2}}$$

(**HiPPO-LegS matrix used in S4**) (4)

$$\boldsymbol{A}^{(N)}_{nk} = -\begin{cases} (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n > k \\ \frac{1}{2} & n = k \\ (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n < k \end{cases} \qquad \boldsymbol{A} = \boldsymbol{A}^{(N)} - \boldsymbol{P}\boldsymbol{P}^\top, \qquad \boldsymbol{A}^{(D)} := \mathrm{eig}(\boldsymbol{A}^{(N)})$$

(**Normal / diagonal plus low-rank form**) (5)

DSS: Diagonal State Spaces S4 was originally motivated by searching for a *diagonal state matrix*, which would be even more structured and result in very simple computation of the SSM. However, the HiPPO-LegS matrix cannot be stably transformed into diagonal form [9, Lemma 3.2], and they were unable to find any diagonal matrices that performed well, resulting in the DPLR formulation.

Gupta [11] made the surprising empirical observation that simply removing the low-rank portion of the DPLR form of the HiPPO-LegS matrix results in a diagonal matrix that performs comparably

[1] HiPPO also specifies formulas for $\boldsymbol{B}$, but the state matrix $\boldsymbol{A}$ is more important. There are many other HiPPO instantiations besides LegS, but HiPPO-LegS is the main one that S4 uses and the term "HiPPO matrix" without the suffix refers to this one.

to the original S4 method. More precisely, their initialization is the diagonal matrix $\boldsymbol{A}^{(D)}$, or the diagonalization of $\boldsymbol{A}^{(N)}$ in (5). They termed $\boldsymbol{A}^{(N)}$ the *skew-HiPPO* matrix, which we will also call the *normal-HiPPO* matrix. To be more specific and disambiguate these variants, we may also call $\boldsymbol{A}^{(N)}$ the HiPPO-LegS-N or HiPPO-N matrix and $\boldsymbol{A}^{(D)}$ the HiPPO-LegS-D or HiPPO-D matrix.

In addition to this initialization, they proposed a method for computing a diagonal SSM kernel. Beyond these two core differences, several other aspects of their parameterization differ from S4's.

In Sections 3 and 4, we systematically study the components of DSS: we categorize different ways to parameterize and compute the diagonal state space, and explain the theoretical interpretion of this particular diagonal $\boldsymbol{A}$ matrix.

3 Parameterizing Diagonal State Spaces

We describe various choices for the computation and parameterization of diagonal state spaces. Our categorization of these choices leads to simple variants of the core method. Both DSS and our proposed S4D can be described using a combination of these factors (Section 3.4).

3.1 Discretization

The true continuous-time SSM can be represented as a continuous convolution $y(t) = (K * u)(t) = \int_0^\infty \boldsymbol{C} e^{s\boldsymbol{A}} \boldsymbol{B} u(t-s)\, ds$.

In discrete time, we view an input sequence $u_0, u_1, \ldots$ as uniformly-spaced samples from an underlying function $u(t)$ and must approximate this integral. Standard methods for doing so that preserve the convolutional structure of the model exist. The first step is to discretize the parameters. Two simple choices that have been used in prior work include

$$(\textbf{Bilinear})\quad \overline{\boldsymbol{A}} = (\boldsymbol{I} - \Delta/2\boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2\boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\boldsymbol{I} - \Delta/2\boldsymbol{A})^{-1} \cdot \Delta\boldsymbol{B}$$

$$(\textbf{ZOH})\quad \overline{\boldsymbol{A}} = \exp(\Delta\boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta\boldsymbol{A})^{-1}(\exp(\Delta\cdot\boldsymbol{A}) - \boldsymbol{I})\cdot\Delta\boldsymbol{B}.$$

With these methods, the discrete-time SSM output is just

$$y = u * \overline{\boldsymbol{K}} \qquad \text{where } \overline{\boldsymbol{K}} = (\boldsymbol{C}\overline{\boldsymbol{B}}, \boldsymbol{C}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \boldsymbol{C}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}). \tag{6}$$

These integration rules have both been used in prior works (e.g. LMU and DSS use ZOH [26, 11] while S4 and its predecessors use bilinear [6, 8, 9]).

In Section 5, we show that there is little empirical difference between them. However, we note that there is a curious phenomenon where the bilinear transform actually perfectly smooths out the kernel used in DSS to match the S4 kernel (Section 4 Fig. 2d). We additionally note that numerical integration is a rich and well-studied topic and more stable methods of approximating the convolutional integral may exist. For example, it is well-known that simple rules like the Trapezoid rule [18] can dramatically reduce numerical integration error when the function has bounded second derivative.

3.2 Convolution Kernel

The main computational difficulty of the original S4 model is computing the convolution kernel $\overline{\boldsymbol{K}}$. This is extremely slow for general state matrices $\boldsymbol{A}$, and S4 introduced a complicated algorithm for DPLR state matrices. When $\boldsymbol{A}$ is diagonal, the computation is nearly trivial. By (6),

$$\overline{\boldsymbol{K}}_\ell = \sum_{n=0}^{N-1} \boldsymbol{C}_n \overline{\boldsymbol{A}}_n^\ell \overline{\boldsymbol{B}}_n \implies \overline{\boldsymbol{K}} = (\overline{\boldsymbol{B}}^\top \circ \boldsymbol{C}) \cdot \mathcal{V}_L(\overline{\boldsymbol{A}}) \qquad \text{where } \mathcal{V}_L(\overline{\boldsymbol{A}})_{n,\ell} = \overline{\boldsymbol{A}}_n^\ell \tag{7}$$

where $\circ$ is Hadamard product, $\cdot$ is matrix multiplication, and $\mathcal{V}$ is known as a **Vandermonde matrix**.

Time and Space Complexity The naive way to compute (7) is by materializing the Vandermonde matrix $\mathcal{V}_L(\overline{\boldsymbol{A}})$ and performing a matrix multiplication, which requires $O(NL)$ time and space.

However, Vandermonde matrices are well-studied and theoretically the multiplication can be computed in $\widetilde{O}(N+L)$ operations and $O(N+L)$ space. In fact, Vandermonde matrices are closely related to Cauchy matrices, which are the computational core of S4's DPLR algorithm, and have identical complexity [17].

Proposition 1. *The time and space complexity of computing the kernel of diagonal SSMs is equal to that of computing DPLR SSMs.*

We note that on modern parallelizable hardware such as GPUs, a simple fast algorithm is to compute (7) with naive summation (using $O(NL)$ operations), but without materializing the Vandermonde matrix (using $O(N+L)$ space). Just as with S4, this may require implementing a custom kernel in some modern deep learning frameworks such as PyTorch to achieve the space savings.

3.3 Parameterization

Parameterization of $\boldsymbol{A}$. Note that the kernel $K(t) = \boldsymbol{C}e^{t\boldsymbol{A}}\boldsymbol{B}$ blows up to ∞ as $t \to \infty$ if $\boldsymbol{A}$ has any eigenvalues with positive real part. Goel et al. [5] found that this is a serious constraint that affects the stability of the model, especially when using the SSM autoregressively. They propose to force the real part of $\boldsymbol{A}$ to be negative, also known as the left-half plane condition in classical controls, by parameterizing the real part inside an exponential function $\boldsymbol{A} = -\exp(\boldsymbol{A}_{Re}) + i \cdot \boldsymbol{A}_{Im}$.

We note that instead of $\exp$, any activation function can be used as long as its range is bounded on one side, such as ReLU, softplus, etc. The original DSS does not constrain the real part of $\boldsymbol{A}$, which is sufficient for simple tasks involving fixed-length sequences, but could become unstable in other settings.

Parameterization of $\boldsymbol{B}, \boldsymbol{C}$. Another choice in the parameterization is how to represent $\boldsymbol{B}$ and $\boldsymbol{C}$. Note that the computation of the final discrete convolution kernel $\overline{\boldsymbol{K}}$ depends only on the elementwise product $\boldsymbol{B} \circ \boldsymbol{C}$ (equation (7)). Therefore DSS chose to parameterize this product directly, which they call $\boldsymbol{W}$, instead of $\boldsymbol{B}$ and $\boldsymbol{C}$ individually.

However, we observe that this is equivalent to keeping independent $\boldsymbol{B}$ and $\boldsymbol{C}$, and simply freezing $\boldsymbol{B} = \mathbf{1}$ while training $\boldsymbol{C}$. Therefore, just as S4 has separate parameters $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ and uses a fixed initialization for $\boldsymbol{A}$ and $\boldsymbol{B}$, S4D also proposes separate $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ and uses fixed initializations for $\boldsymbol{A}$ (discussed in Section 4) and $\boldsymbol{B}$ (set to $\mathbf{1}$). Then the difference between S4D and DSS is simply that DSS does not train $\boldsymbol{B}$. In our ablations, we show that training $\boldsymbol{B}$ gives a minor but consistent improvement in performance.

3.4 S4D: the Diagonal Version of S4

A key component of our exposition is disentangling the various choices possible in representing and computing state space models. With this categorization, different choices can be mixed and matched to define variants of the core method. Table 1 compares S4, DSS, and S4D, which have a core structure and kernel computation, but have various choices of other aspects of the parameterization.

Table 1: (**Parameterization choices for Structured SSMs**.) Aside from the core structure of $\boldsymbol{A}$ and the computation of its convolution kernel, SSMs have several design choices which are consolidated in S4D.

Method	Structure	Kernel Computation	Discretization	Constraint $\Re(\boldsymbol{A})$	Trainable $\boldsymbol{B}$	Initialization of $\boldsymbol{A}$
S4	DPLR	Cauchy	Bilinear	exp	Yes	HiPPO
DSS	diagonal	softmax	ZOH	id (none)	No	HiPPO-D
S4D	diagonal	Vandermonde	either	exp / ReLU	optional	various

4 Initialization of Diagonal State Matrices

The critical question remains: which diagonal state matrices $\boldsymbol{A}$ are actually effective? We comment on the limitations of diagonal SSMs, and then provide three instantiations of S4D that perform well empirically.

Expressivity and Limitations of Diagonal SSMs. We first present a simplified view on the expressivity of diagonal SSMs mentioned by [11]. First, it is well-known that almost all matrices diagonalize over the complex plane. Therefore it is critical to use complex-valued matrices in order to use diagonal SSMs.

Proposition 2. *The set $\mathcal{D} \subset \mathbb{C}^{N\times N}$ of diagonalizable matrices is dense in $\mathbb{C}^{N\times N}$, and has full measure (i.e. its complement has measure 0).*

It is also well known that the state space $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ is exactly equivalent to (i.e. expresses the same map $u \mapsto y$) the state space $(\boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}, \boldsymbol{V}^{-1}\boldsymbol{B}, \boldsymbol{C}\boldsymbol{V})$, known in the SSM literature as a state space transformation. Therefore Proposition 2 says that *(almost) all SSMs are equivalent to a diagonal SSM*.

However, we emphasize that Proposition 2 is about *expressivity* which does not guarantee strong performance of a trained model after optimization. For example, Gu et al. [9] and Gupta [11] show that parameterizing $\boldsymbol{A}$ as a dense real matrix or diagonal complex matrix, which are both fully expressive classes, performs poorly if randomly initialized.

Second, Proposition 2 does not take into account numerical representations of data, which was the original reason S4 required a low-rank correction term instead of a pure diagonalization [9, Lemma 3.2]. In Section 5.2, we also show that two different initializations with the *same spectrum* (i.e., are equivalent to the same diagonal $\boldsymbol{A}$) can have very different performance.

S4D-LegS. The HiPPO-LegS matrix has DPLR representation $\boldsymbol{A}^{(D)} - \boldsymbol{P}\boldsymbol{P}^\top$, and Gupta [11] showed that simply approximating it with $\boldsymbol{A}^{(D)}$ works quite well (5). Our first result is providing a clean mathematical interpretation of this method. Theorem 3 shows a surprising fact that does not hold in general for DPLR matrices (Appendix A.1), and arises out of the special structure of this particular matrix.

Theorem 3. *Let $\boldsymbol{A} = \boldsymbol{A}^{(N)} - \boldsymbol{P}\boldsymbol{P}^\top$ and $\boldsymbol{B}$ be the HiPPO-LegS matrices, and $K_{\boldsymbol{A},\boldsymbol{B}}(t)$ be its basis. As the state size $N \to \infty$, the SSM basis $K_{\boldsymbol{A}^{(N)},\boldsymbol{B}/2}(t)$ limits to $K_{\boldsymbol{A},\boldsymbol{B}}(t)$ (Fig. 2).*

Note that $\boldsymbol{A}^{(N)}$ is then *unitarily* equivalent to $\boldsymbol{A}^{(D)}$, which preserves the stability and timescale [10] of the system.

We define **S4D-LegS** to be the S4D method for this choice of diagonal $\boldsymbol{A} = \boldsymbol{A}^{(D)}$. Theorem 3 explains the empirical results in [11] whereby this system performed quite close to S4, but was usually slightly worse. This is because DSS is a variant of S4D-LegS, which by Theorem 3 is a noisy approximation to S4-LegS. Fig. 2 illustrates this result, and also shows a curious phenomenon involving different discretization rules that is open for future work.

S4D-Inv. To further simplify S4D-LegS, we analyze the structure of $\boldsymbol{A}^{(D)} = \operatorname{diag}\langle \boldsymbol{A} \rangle$ in more detail. The real part is easy to understand, which follows from the analysis in [9]: $\Re(\boldsymbol{A}) = -\frac{1}{2}\mathbf{1}$. Let the imaginary part be sorted, i.e. $\Im(\boldsymbol{A})_n$ is the n-th largest (positive) imaginary component. We empirically deduced the following conjecture for the asymptotics of the imaginary part.

Conjecture 4. *As $N \to \infty$, $\Im(\boldsymbol{A})_0 \to \frac{1}{\pi}N^2 + c$ where $c \approx 0.5236$ is a constant. For a fixed N, the other eigenvalues satisfy an inverse scaling in n: $\Im(\boldsymbol{A})_n = \Theta(n^{-1})$.*

Fig. 3 empirically supports this conjecture. Based on Conjecture 4, we propose the initialization S4D-Inv to use the following inverse-law diagonal matrix which closely approximates S4D-LegS.

$$(\textbf{S4D-Inv}) \quad \boldsymbol{A}_n = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{2n+1} - 1\right) \qquad (8)$$

$$(\textbf{S4D-Lin}) \quad \boldsymbol{A}_n = -\frac{1}{2} + i\pi n \qquad (9)$$

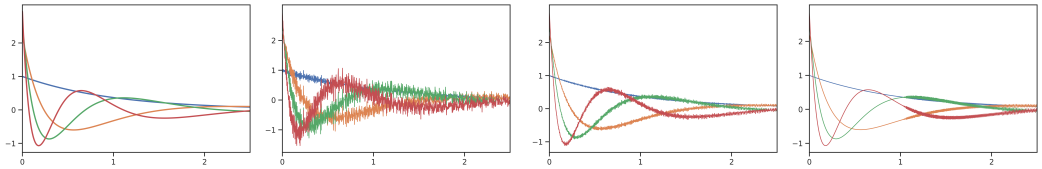
S4D-Lin. While S4D-Inv can be seen as an approximation to the original S4-LegS, we propose an even simpler scaling law for the imaginary parts that can be seen as an approximation of S4-FouT ([10]), where the imaginary parts are simply the Fourier series frequencies (i.e. matches the diagonal part of the DPLR form of S4-FouT). Fig. 1 (*Right*) illustrates the S4D-Lin basis $e^{t\boldsymbol{A}}\boldsymbol{B}$, which are simply damped Fourier basis functions.

5 Experiments

Our experimental study shows that S4D has strong performance in a wide variety of domains and tasks, including the well-studied Long Range Arena (LRA) benchmark where the best S4D variant is competitive with S4 on all tasks and significantly outperforms all non-SSM baselines.

We begin with controlled ablations of the various representations of diagonal state space models. Sections 5.1 and 5.2 ablate the proposed methods for parameterizing, computing, and initializing diagonal SSMs from Sections 3 and 4. Section 5.3 show full results of larger models on standard benchmarks,

Methodology and Datasets. In order to study the effects of different S4 and S4D variants in a controlled setting, we propose the following protocol. We focus on three datasets covering a varied



(a) S4-LegS (original S4 kernel) (b) S4D-LegS ($N = 256$, ZOH) (c) S4D-LegS ($N = 1024$, ZOH) (d) S4D-LegS ($N = 1024$, Bilinear)

Figure 2: **(Visualization of Theorem 3).** (a) The particular $(\boldsymbol{A}, \boldsymbol{B})$ matrix chosen in S4 results in smooth basis functions $e^{t\boldsymbol{A}}\boldsymbol{B}$ with a closed form formula in terms of Legendre polynomials. By the HiPPO theory, convolving against these functions has a mathematical interpretation as orthogonalizing against an exponentially-decaying measure. (b, c) By special properties of this state matrix, removing the low-rank term of its NPLR representation produces the same basis functions as $N \to \infty$, explaining the empirical effectiveness of DSS. (c) Curiously, the bilinear transform instead of ZOH smooths out the kernel to exactly match S4-LegS as N grows.

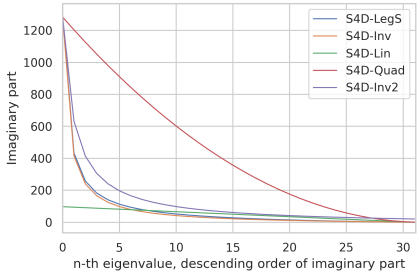


Figure 3: (**S4D eigenvalues.**) All S4D methods have eigenvalues $-\frac{1}{2} + \lambda_n i$. S4D-LegS theoretically approximates dynamics of the original (non-diagonal) S4 (Blue), and has eigenvalues following an inverse law $\lambda_n \propto n^{-1}$ (Orange). The precise law is important: other scaling laws with the same range, including an inverse law with different constant (Purple) and a quadratic law (Red), perform empirically worse (Section 5.2). A very different linear law based on Fourier frequencies also performs well (Green).

range of data modalities (image pixels, biosignal time series, audio waveforms), sequence lengths (1K, 4K, 16K), and tasks (classification and regression with bidirectional and causal models).

- **Sequential CIFAR (sCIFAR).** CIFAR-10 images are flattened into a sequence of length 1024, and a bidirectional sequence model is used to perform 10-way classification.
- **BIDMC Vital Signs.** EKG and PPG signals of length 4000 are used to predict respiratory rate (RR), heart rate (HR), and blood oxygen saturation (SpO2). We focus on SpO2 in this study.
- **Speech Commands (SC).**[2] A 1-second raw audio waveform comprising 16000 samples is used for 35-way spoken word classification. We use an autoregressive (AR) model to vary the setting; this causal setting more closely imitates autoregressive speech generation, where SSMs have shown recent promise [5].

We fix a simple architecture and training protocol that works generically. The architecture has 4 layers and hidden dimension $H = 128$, resulting in $\sim 100K$ parameters. All results are averaged over multiple seeds (full protocol and results including std. reported in Appendix B).

5.1 Parameterization, Computation, Discretization

Given the same diagonal SSM matrices $\boldsymbol{A}, \boldsymbol{B}$, there are many variants of how to parameterize the matrices and compute the SSM kernel described in Section 3. We ablate the different choices described in Table 1. Results are in Table 2, and show that:

(i) Computing the model with a softmax instead of Vandermonde product does not make much difference

(ii) Training $\boldsymbol{B}$ is consistently slightly better

(iii) Different discretizations (Section 3.1) do not make a noticeable difference

(iv) Unrestricting the real part of $\boldsymbol{A}$ (Section 3.3) may be slightly better

These ablations show that for a fixed initialization $(\boldsymbol{A}, \boldsymbol{B})$, different aspects of parameterizing SSMs make little difference overall. This justifies the parameterization and algorithm S4D uses (Section 3.4), which preserves the choices of the original S4 model and is simpler than DSS. For the remaining

[2] We note that a line of prior work including S4 [14, 19, 9] all used a smaller 10-class subset of SC, so our results on the full dataset are not directly comparable.

Trainable B	Method	sCIFAR	SC (AR)	BIDMC (SpO2)
No	Softmax	85.04	89.80	0.1299
No	Vandermonde	84.78	89.62	0.1355
Yes	Softmax	85.37	90.06	0.1170
Yes	Vandermonde	85.37	90.34	0.1274

Discretization	Real part of A	sCIFAR	SC (AR)	BIDMC (SpO2)
Bilinear	exp	85.20	89.52	0.1193
	ReLU	85.06	90.22	0.1172
	-	85.35	90.58	0.1102
ZOH	exp	85.02	89.93	0.1303
	ReLU	84.98	90.03	0.1232
	-	85.15	90.19	0.1289

Table 2: Ablations of different parameterizations of diagonal SSMs using S4D-Inv. (*Left*) trainability and computation; (*Right*) discretization and parameterization.

of the experiments in Section 5.2 and Section 5.3, we fix the S4D parameterization and algorithm described in Section 3. Note that this computes exactly the same kernel as the original S4 algorithm when the low-rank portion is set to 0, allowing controlled comparisons of the critical state matrix $\boldsymbol{A}$ for the remainder of this section.

5.2 S4D Initialization Ablations

The original S4 model proposed a specific formula for the $\boldsymbol{A}$ matrix, and the first diagonal version [11] used a specific matrix based on it. Our new proposed variants S4D-Inv and S4D-Lin also define precise formulas for the initialization of the $\boldsymbol{A}$ matrix (8). This raises the question of whether the initialization of the $\boldsymbol{A}$ still needs to be so precise, despite the large simplifications from the original version. We perform several natural ablations on these initializations, showing that even simple variations of the precise formula can degrade performance.

Imaginary part scaling factor. The scaling rules for the imaginary parts of S4D-Inv and S4D-Lin are simple polynomial laws, but how is the constant factor chosen and how important is it? These constants are based on approximations to HiPPO methods (e.g. Conjecture 4). Note that the range of imaginary components for S4D-Inv and S4D-Lin are quite different (Fig. 3); the largest imaginary part is $\frac{N^2}{\pi}$ for S4D-Inv and πN for S4D-Lin.

We consider scaling all imaginary parts by a constant factor of 0.01 or 100.0 to investigate whether the constant matters. Note that this preserves the overall shape of the basis functions (Fig. 1, dashed lines) and simply changes the frequencies, and it is not obvious that this should degrade performance. However, both changes substantially reduce the performance of S4D in all settings.

Randomly initialized imaginary part. Next, we consider choosing the imaginary parts randomly. For S4D-Inv, we keep the real parts equal to $-\frac{1}{2}$ and set each imaginary component to $\boldsymbol{A}_n = -\frac{1}{2} + i\frac{N}{\pi}(\frac{N}{2u+1} - 1)$ for $u \sim N \cdot \mathcal{U}[0,1]$. Note that when u is equally spaced in $[0,1]$ instead of uniformly random, this exactly recovers S4D-Inv (8), so this is a sensible random approximation to it.

Similarly, we consider a variant of S4D-Lin that replaces the n in (9) with $N \cdot \mathcal{U}[0,1]$.

Table 3a (*Random Imag*) shows that this small change causes minor degradation in performance. We additionally note that the randomly initialized imaginary ablation can be interpreted as follows. Fig. 3 shows the asymptotics of the imaginary parts of SSM matrices, where the imaginary parts of the eigenvalues correspond to y-values corresponding to uniformly spaced nodes on the x-axis. This ablation then replaces the uniform spacing on the x-axis with uniformly random x values.

Randomly initialized real part.

We considering initializing the real part of each eigenvalue as $-\mathcal{U}[0,1]$ instead of fixing them to $-\frac{1}{2}$. Table 3a(Left, *Random Real*) shows that this also causes minor but consistent degradation in performance on the ablation datasets. Finally, we also consider randomizing both real and imaginary parts, which degrades performance even further.

Ablation: Other S4D matrices.

Other simple variants of initializations show that it is not just the range of the eigenvalues but the actual distribution that is important (Fig. 3). Both S4D-Inv2 and S4D-Quad have real part $-\frac{1}{2}$ and imaginary part satisfying the same maximum value as Conjecture 4. The S4D-Inv2 initialization uses the same formula as S4D-Inv, but replaces a $2n+1$ in the denominator with $n+1$. The S4D-Quad initialization uses a polynomial law with power 2 instead of -1 (S4D-Inv) or 1 (S4D-Lin).

We include two additional methods here that are not based on the proposed S4D-Inv or S4D-Lin methods. First, S4D-Rand uses a randomly initialized diagonal $\boldsymbol{A}$, and validates that it performs

Table 3: (*Initialization and Trainability ablations*)

Ablation	sCIFAR	SC (AR)	BIDMC
S4D-Lin	**85.12**	**90.66**	**0.128**
Scale 0.01	-7.27	-1.92	+0.040
Scale 100	-7.91	-4.04	+0.077
Random Imag	-0.42	-3.08	-0.001
Random Real	-0.73	-0.87	+0.011
Random Both	-1.28	-5.88	+0.007
S4D-Inv	**84.79**	**90.27**	**0.114**
Scale 0.01	-5.03	-0.08	+0.028
Scale 100	-7.77	-52.31	+0.034
Random Imag	-0.29	-0.52	+0.010
Random Real	0.12	-2.18	+0.032
Random Both	-1.55	-0.55	+0.024
S4D-Inv2	-2.62	-39.84	+0.005
S4D-Quad	-1.83	-0.62	+0.024
S4D-Random	-6.32	-1.95	+0.034
S4D-Real	-5.45	-10.17	+0.066

(a) Ablations of the initialization of the diagonal $\boldsymbol{A}$ matrix in S4D. Very simple changes that largely preserve the structure of the diagonal eigenvalues all degrade performance.

	sCIFAR		SC (AR)		BIDMC
Frozen (A, B)	Acc (first)	Acc (best)	Acc (first)	Acc (best)	RMSE (best)
S4-LegS	53.63	86.19	33.87	85.33	0.1049
S4-LegT	54.76	86.30	8.77	57.35	0.1106
S4-FouT	55.28	86.05	9.27	69.57	0.1072
S4-LegS+FouT	54.38	86.53	34.06	83.37	0.0887
S4D-LegS	50.87	84.81	22.76	77.18	0.0960
S4D-Inv	53.19	84.40	18.49	76.53	0.0995
S4D-Lin	51.75	84.96	19.09	75.58	0.0935
Trainable (A, B)					
S4-LegS	54.23	86.29	62.19	90.68	0.1033
S4-LegT	55.16	86.12	55.86	90.42	0.1146
S4-FouT	55.89	85.93	60.56	90.83	0.1136
S4-LegS+FouT	55.00	86.18	61.76	91.01	0.0970
S4D-LegS	50.41	85.64	47.54	88.47	0.1148
S4D-Inv	53.42	84.59	45.73	89.69	0.1132
S4D-Lin	52.23	85.75	47.68	89.56	0.1032

(b) Results for all S4 and S4D methods on the ablation datasets, when the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices are either frozen (*Top*) or trained (*Bottom*). Diagonal state matrices are highly competitive with full DPLR versions, achieving strong results on all datasets.

poorly, in line with earlier findings [9, 11]. Second, S4D-Real uses a particular real initialization with $\boldsymbol{A}_n = -(n+1)$. This is the exact same spectrum as the original S4(-LegS) method, which validates that it is not just the diagonalization that matters, highlighting the limitations of Proposition 2.

5.3 Full Comparisons of S4D and S4 Methods

Trainable $\boldsymbol{A}, \boldsymbol{B}$ matrices.

Table 3b shows the performance of all S4D and S4 variants [10] on the ablations datasets. We observe several interesting phenomena:

(i) Freezing the matrices performs comparably to training them on sCIFAR and BIDMC, but is substantially worse on SC. We hypothesize that this results from Δ being poorly initialized for SC, so that at initialization models do not have context over the entire sequence, and training $\boldsymbol{A}$ and $\boldsymbol{B}$ helps adjust for this. As further evidence, the *finite window methods* S4-LegT and S4-FouT (defined in [10]) have the most limited context and suffer the most when $\boldsymbol{A}$ is frozen.

(ii) The full DPLR versions are often slightly better than the diagonal version throughout the entire training curve. We report the validation accuracy after 1 epoch of training on sCIFAR and SC to illustrate this phenomenon. Note that this is not a consequence of having more parameters (Appendix B).

Large models on ablation datasets.

Finally, we relax the strict requirements on model size and regularization for the ablation datasets, and show the performance of S4 and S4D variants on the test sets with a larger model (architecture and training details in Appendix B) when the model size and regularization is simply increased (Table 4). We note that results for each dataset are better than the original S4 model, which was already state-of-the-art on these datasets [8, 9].

Long Range Arena.

We use the same hyperparameter setting for the state-of-the-art S4 model in [10] on the Long Range Arena benchmark for testing long dependencies in sequence models. S4D variants are highly competitive on all datasets except Path-X, and outperform the S4 variants on several of them. On Path-X using this hyperparameter setting with bidirectional models, only S4D-Inv, our simpler approximation to the original S4-LegS model, achieves above random chance, and has an average of 85% on the full LRA suite, more than 30 points better than the original Transformer [24].

Table 4: (**Ablation datasets:** Full results with larger models.) For Speech Commands, we show both an autoregressive model as in the ablations, and an unconstrained bidirectional model.

Model	sCIFAR	SC		BIDMC		
	Test	AR	Bi.	HR	RR	SpO2
S4-LegS	**91.80** (0.43)	**93.60** (0.13)	96.08 (0.15)	**0.332** (0.013)	0.247 (0.062)	0.090 (0.006)
S4-FouT	91.22 (0.25)	91.78 (0.10)	95.27 (0.20)	0.339 (0.020)	0.301 (0.030)	**0.068** (0.003)
S4D-LegS	89.92 (1.69)	93.57 (0.09)	95.83 (0.14)	0.367 (0.001)	0.248 (0.036)	0.102 (0.001)
S4D-Inv	90.69 (0.06)	93.40 (0.67)	96.18 (0.27)	0.373 (0.024)	0.254 (0.022)	0.110 (0.001)
S4D-Lin	90.42 (0.03)	93.37 (0.05)	**96.25** (0.03)	0.379 (0.006)	**0.226** (0.008)	0.114 (0.003)

Table 5: (**Long Range Arena**) Accuracy on full suite of LRA tasks. Hyperparameters in Appendix B.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
S4-LegS	59.60 (0.07)	86.82 (0.13)	90.90 (0.15)	88.65 (0.23)	94.20 (0.25)	**96.35**	**86.09**
S4-FouT	57.88 (1.90)	86.34 (0.31)	89.66 (0.88)	**89.07** (0.19)	**94.46** (0.24)	✗	77.90
S4D-LegS	60.47 (0.34)	86.18 (0.43)	89.46 (0.14)	88.19 (0.26)	93.06 (1.24)	91.95	84.89
S4D-Inv	60.18 (0.35)	**87.34** (0.20)	**91.09** (0.01)	87.83 (0.37)	93.78 (0.25)	92.80	85.50
S4D-Lin	**60.52** (0.51)	86.97 (0.23)	90.96 (0.09)	87.93 (0.34)	93.96 (0.60)	✗	78.39
S4 (original)	58.35	76.02	87.09	87.26	86.05	88.10	80.48
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66

6 Conclusion

State space models based on S4 are a promising family of models for modeling many types of sequential data, with particular strengths for continuous signals and long-range interactions. These models are a large departure from conventional sequence models such as RNNs, CNNs, and Transformers, with many new ideas and moving parts. This work provides a more in-depth exposition for all aspects of working with S4-style models, from their core structures and kernel computation algorithms, to miscellaneous choices in their parameterizations, to new theory and methods for their initialization. We systematically analyzed and ablated each of these components, and provide recommendations for building a state space model that is as simple as possible, while as theoretically principled and empirically effective as S4. We believe that S4D can be a strong generic sequence model for a variety of domains, that opens new directions for state space models theoretically, and is much more practical to understand and implement for practitioners.

Acknowledgments

We gratefully acknowledge the support of DARPA under Nos. FA86501827865 (SDH) and FA86501827882 (ASED); NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), and 1937301 (RTML); ONR under No. N000141712266 (Unifying Weak Supervision); the Moore Foundation, NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, the Okawa Foundation, American Family Insurance, Google Cloud, Swiss Re, Brown Institute for Media Innovation, Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program, Fannie and John Hertz Foundation, National Science Foundation Graduate Research Fellowship Program, Texas Instruments, and members of the Stanford DAWN project: Teradata, Facebook, Google, Ant Financial, NEC, VMWare, and Infosys. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA, NIH, ONR, or the U.S. Government.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *The International Conference on Learning Representations (ICLR)*, 2019.

[3] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[4] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2021.

[5] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state-space models. In *The International Conference on Machine Learning (ICML)*, 2022.

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

[7] Albert Gu, Caglar Gulcehre, Tom Le Paine, Matt Hoffman, and Razvan Pascanu. Improving the gating mechanism of recurrent neural networks. In *The International Conference on Machine Learning (ICML)*, 2020.

[8] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with the structured learnable linear state space layer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[9] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022.

[10] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré. How to train your hippo: State space models with generalized basis projections. *arXiv preprint arXiv:2206.12037*, 2022.

[11] Ankit Gupta. Diagonal state spaces are as effective as structured state spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[14] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.

[15] James Morrill, Cristopher Salvi, Patrick Kidger, James Foster, and Terry Lyons. Neural rough differential equations for long time series. *The International Conference on Machine Learning (ICML)*, 2021.

[16] Naoki Nonaka and Jun Seita. In-depth benchmarking of deep neural network architectures for ecg diagnosis. In *Machine Learning for Healthcare Conference*, pages 414–439. PMLR, 2021.

[17] Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

[18] Anthony Ralston and Philip Rabinowitz. *A first course in numerical analysis*. Courier Corporation, 2001.

[19] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021.

[20] David W Romero, Robert-Jan Bruintjes, Jakub M Tomczak, Erik J Bekkers, Mark Hoogendoorn, and Jan C van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. In *The International Conference on Learning Representations (ICLR)*, 2022.

[21] T Konstantin Rusch and Siddhartha Mishra. Unicornn: A recurrent model for learning very long time dependencies. *The International Conference on Machine Learning (ICML)*, 2021.

[22] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[23] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I Webb. Time series extrinsic regression. *Data Mining and Knowledge Discovery*, pages 1–29, 2021. doi: https://doi.org/10.1007/s10618-021-00745-9.

[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. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

[25] Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in RNNs with auxiliary losses. In *The International Conference on Machine Learning (ICML)*, 2018.

[26] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 15544–15553, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5, e.g. not matching the baseline S4 on Path-X.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code is a simple modification from the original S4 [9] repository and is publicly available.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See the Appendix.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The experiment code and configs are publically available with resource and timing information reported.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We used standard benchmarks and synthetic data.

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Method Details

Listing 1 Full Numpy example of the parameterization and computation of a 1D S4D-Lin model

```
def parameters(N, dt_min=1e-3, dt_max=1e-1):
  # Initialization
  log_dt = np.random.rand() * (np.log(dt_max)-np.log(dt_min)) + np.log(dt_min)
  A = -0.5 + 1j * np.pi * np.arange(N//2) # S4D-Lin initialization
  B = np.ones(N//2) + 0j
  C = np.random.randn(N//2) + 1j * np.random.randn(N) # Variance preserving initialization
  return log_dt, np.log(-A.real), A.imag, B, C

def kernel(L, log_dt, log_A_real, A_imag, B, C):
  # Discretization (e.g. bilinear transform)
  dt, A = np.exp(log_dt), -np.exp(log_A_real) + 1j * A_imag
  dA, dB = (1+dt*A/2) / (1-dt*A/2), dt*B / (1-dt*A/2)

  # Computation (Vandermonde matrix multiplication - can be optimized)
  # Return twice the real part - same as adding conjugate pairs
  return 2 * ((B*C) @ (dA[:, None] ** np.arange(L))).real

def forward(u, parameters):
    L = u.shape[-1]
    K = kernel(L, *parameters)
    # Convolve y = u * K using FFT
    K_f, u_f = np.fft.fft(K, n=2*L), np.fft.fft(u, n=2*L)
    return np.fft.ifft(K_f*u_f, n=2*L)[..., :L]
```

A.1 Proofs

We prove Theorem 3, and then show why this it is a surprising result that is not true in general to low-rank perturbations of SSMs.

We start with the interpretation of the S4-LegS matrix shown in [10], which corresponds to Fig. 1 (Left).

Theorem 5. *Let $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{P}$ be the matrices defined in equation* (4)*. The SSM kernels $K_n(t) = \boldsymbol{e}_n^\top e^{t\boldsymbol{A}}\boldsymbol{B}$ have the closed form formula*

$$K_n(t) = L_n(e^{-t})e^{-t}$$

where L_n are the Legendre polynomials shifted and scaled to be orthonormal on the interval $[0, 1]$.

Lemma A.1. *The functions $L_n(e^{-t})$ are a complete orthonormal basis with respect to the measure $\omega(t) = e^{-t}$.*

Proof. The polynomials are defined to be orthonormal on $[0, 1]$, i.e.

$$\int_0^1 L_n(t)L_m(t)\,dt = \delta_{n,m}.$$

By the change of variables $t = e^{-s}$ with $\frac{dt}{ds} = -e^{-s}$,

$$-\int_{-\infty}^0 L_n(e^{-s})L_m(e^{-s})e^{-s}\,ds = \delta_{n,m} = \int_0^\infty L_n(e^{-s})L_m(e^{-s})e^{-s}\,ds$$

which shows the orthonormality.

Completeness follows from the fact that polynomials are complete. □

Proof of Theorem 3. We start with the standard interpretation of SSMs as convolutional systems. The SSM $x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$ is equivalent to the convolution

$$x_n(t) = (u * K_n)(t) = \int_{-\infty}^t u(s)K_n(t-s)\,ds = \int_0^\infty u(t-s)K_n(s)\,ds$$

for the SSM kernels (equation (3)).

Defining $u^{(t)}(s) = u(t-s)$, we can write this as

$$x_n(t) = \langle u^{(t)}, K_n \rangle_\omega$$

where $\omega(s) = e^{-s}$ and $\langle p(s), q(s) \rangle_\omega = \int_0^\infty p(s)q(s)\omega(s)\, ds$ is the inner product in the Hilbert space of L2 functions with respect to measure ω.

By Theorem 5, the K_n are a complete orthonormal basis in this Hilbert space. There $x_n(t)$ represents a decomposition of the function $u^{(t)}$ with respect to this basis, and can be recovered as a linear combination of these projections

$$u^{(t)} = \sum_{n=0}^{\infty} x_n(t) K_n.$$

Pointwise over the inner times s,

$$u^{(t)}(s) = \sum_{n=0}^{\infty} x_n(t) K_n(s).$$

This implies that

$$\begin{aligned} u(t) = u^{(t)}(0) &= \sum_{n=0}^{\infty} x_n(t) K_n(0) \\ &= \sum_{n=0}^{\infty} x_n(t) L_n(0) = \sum_{n=0}^{\infty} x_n(t)(2n+1)^{\frac{1}{2}} \\ &= \boldsymbol{B}^\top x(t) \end{aligned}$$

Intuitively, due to the function reconstruction interpretation of HiPPO [10], we can approximate $u(t)$ using knowledge in the current state $x(t)$. There in the limit $N \to \infty$, the original SSM is equivalent to

$$\begin{aligned} x'(t) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ &= \boldsymbol{A}x(t) + \frac{1}{2}\boldsymbol{B}u(t) + \frac{1}{2}\boldsymbol{B}u(t) \\ &= \boldsymbol{A}x(t) + \frac{1}{2}\boldsymbol{B}\boldsymbol{B}^\top x(t) + \frac{1}{2}\boldsymbol{B}u(t) \\ &= \boldsymbol{A}x(t) + \boldsymbol{P}\boldsymbol{P}^\top x(t) + \frac{1}{2}\boldsymbol{B}u(t) \\ &= \boldsymbol{A}^N x(t) + \frac{1}{2}\boldsymbol{B}u(t) \end{aligned}$$

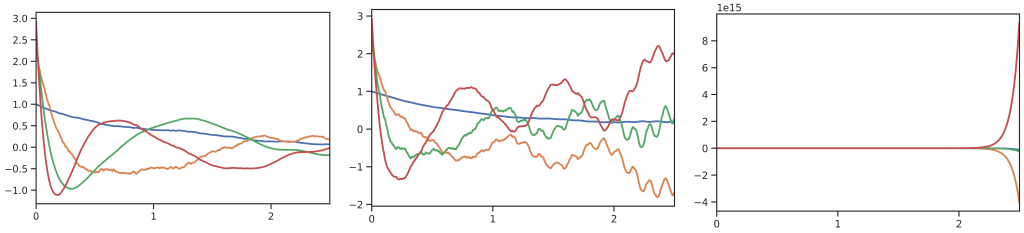
□

General low-rank perturbations. Finally, we remark that this phenomenon where removing the low-rank correction to a DPLR matrix approximates the original dynamics, is unique to this HiPPO-LegS matrix. We note that if instead of $\boldsymbol{P}\boldsymbol{P}^\top$, a *random* rank-1 correction is added to the HiPPO-LegS matrix in Theorem 3, the resulting SSM kernels look completely different and in fact diverge rapidly as the magnitude of $\boldsymbol{P}$ increases (Fig. 4).

Similarly, Fig. 5a shows a new S4 variant called S4-FouT that is also DPLR [10], but removing the low-rank component dramatically changes the SSM kernels.

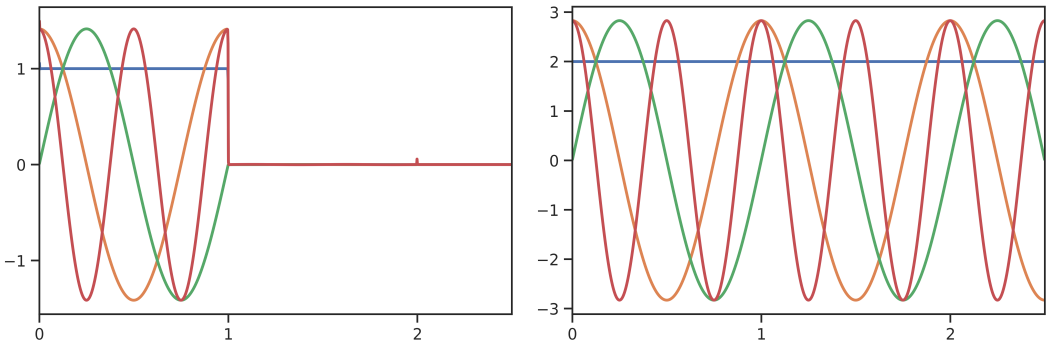
B Experiment Details

Ablation datasets training protocol. The architecture has 4 layers and hidden dimension $H = 128$, resulting in around 100K trainable parameters. The $\boldsymbol{A}$ and $\boldsymbol{B}$ parameters were tied across the H SSM copies; therefore the S4 models have only $H \times \{\text{num. layers}\}$ more parameters than S4D models, arising from the $\boldsymbol{P}$ tensor in the DPLR representation $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{P}^\top$. This choice was made



(a) $\sigma = 0.3$ (b) $\sigma = 0.4$ (c) $\sigma = 0.5$

Figure 4: Basis kernels for $(\boldsymbol{A}+\boldsymbol{P}\boldsymbol{P}^\top, \boldsymbol{B})$ for HiPPO-LegS $(\boldsymbol{A}, \boldsymbol{B})$ and random i.i.d. Gaussian $\boldsymbol{P}$ with varying std σ, illustrating that the SSM basis is very sensitive to low-rank perturbations. Note that the normal-HiPPO matrix $\boldsymbol{A}^{(N)} = \boldsymbol{A}+\boldsymbol{P}\boldsymbol{P}^\top$ for $\boldsymbol{P}$ with entries of magnitude $N^{\frac{1}{2}}$ which is far larger, highlighting how unexpected the theoretical result Theorem 3 is.



(a) S4-FouT (b) Diagonal approximation to S4-FouT

Figure 5: (a) S4-FouT is a version of S4 that produces *truncated Fourier* basis functions choosing a particular $(\boldsymbol{A}, \boldsymbol{B})$. This captures sliding Fourier transforms as a state space model. (b) Removing the low-rank term from the FouT matrix does *not* approximate S4-FouT. This diagonal state matrix has real part 0 that produces infinite oscillations and does not perform well empirically.

because it generally does not affect performance much, while reducing parameter count and ensuring that S4 vs. S4D models have very similar numbers of parameters.

All results are averaged over 2 or 3 seeds.

All models use learning rate 0.004, 0.01 weight decay, and no other regularization or data augmentation. For the classification tasks (sCIFAR and SC). we use a cosine scheduler with 1 epoch warmup and decaying to 0. For the regression task (BIDMC), we use a multistep scheduler following [21, 8].

Reported results are all best validation accuracy, except for the large models in Table 4.

Full results for parameterization ablations. Table 6 and Table 7 contain the raw results for Table 2 including standard deviations.

Full results for large models on ablations datasets. Tables 8 to 10 show full results comparing our proposed methods against the best models from the literature; citations indicate numbers from prior work.

Table 6: Full results for Table 2 (Left) including standard deviations.

Trainable $\boldsymbol{B}$	Method	sCIFAR	SC (AR)	BIDMC (SpO2)
No	Softmax	85.04 (0.22)	89.80 (0.21)	0.1299 (0.0048)
No	Vandermonde	84.78 (0.16)	89.62 (0.03)	0.1355 (0.0039)
Yes	Softmax	85.37 (0.43)	90.06 (0.11)	0.1170 (0.0039)
Yes	Vandermonde	85.37 (0.43)	90.34 (0.18)	0.1274 (0.0020)

Table 7: Full results for Table 2 (Right) including standard deviations.

Discretization	Real part	sCIFAR	SC (AR)	BIDMC (SpO2)
Bilinear	Exp	85.20 (0.18)	89.52 (0.01)	0.1193 (0.0069)
Bilinear	-	85.35 (0.27)	90.58 (0.37)	0.1102 (0.0075)
Bilinear	ReLU	85.06 (0.06)	90.22 (0.25)	0.1172 (0.0063)
ZOH	Exp	85.02 (0.24)	89.93 (0.07)	0.1303 (0.0014)
ZOH	-	85.15 (0.13)	90.19 (0.58)	0.1289 (0.0035)
ZOH	ReLU	84.98 (0.72)	90.03 (0.13)	0.1232 (0.0065)

Table 8: (**Sequential CIFAR image classification.** Test accuracy (Std. dev.)

Model	sCIFAR
S4-LegS	**91.80** (0.43)
S4-FouT	91.22 (0.25)
S4-(LegS/FouT)	<u>91.58</u> (0.17)
S4D-LegS	89.92 (1.69)
S4D-Inv	90.69 (0.06)
S4D-Lin	90.42 (0.03)
Transformer [25]	62.2
FlexConv [20]	80.82
TrellisNet [2]	73.42
LSTM [12, 7]	63.01
r-LSTM [25]	72.2
UR-GRU [7]	<u>74.4</u>
HiPPO-RNN [6]	61.1
LipschitzRNN [4]	64.2

Table 9: (**BIDMC Vital signs prediction.**) RMSE for predicting respiratory rate (RR), heart rate (HR), and blood oxygen (SpO2).

Model	HR	RR	SpO2
S4-LegS	**0.332** (0.013)	0.247 (0.062)	0.090 (0.006)
S4-FouT	<u>0.339</u> (0.020)	0.301 (0.030)	**0.068** (0.003)
S4-(LegS/FouT)	0.344 (0.032)	**0.163** (0.008)	<u>0.080</u> (0.007)
S4D-LegS	0.367 (0.001)	0.248 (0.036)	0.102 (0.001)
S4D-Inv	0.373 (0.024)	0.254 (0.022)	0.110 (0.001)
S4D-Lin	0.379 (0.006)	<u>0.226</u> (0.008)	0.114 (0.003)
UnICORNN [21]	1.39	1.06	0.869
coRNN [21]	1.81	1.45	-
CKConv	2.05	1.214	1.051
NRDE [15]	2.97	1.49	1.29
LSTM	10.7	2.28	-
Transformer	12.2	2.61	3.02
XGBoost [23]	4.72	1.67	1.52
Random Forest [23]	5.69	1.85	1.74
Ridge Regress. [23]	17.3	3.86	4.16

Note that earlier works on the Speech Commands dataset typically use pre-processing such as MFCC features, or a 10-class subset of the full 35-class dataset [14, 19, 9]. As we are not aware of a collection of strong baselines for raw waveform classification using the full dataset, we trained several baselines from scratch for Table 10. The InceptionNet, ResNet-18, and XResNet-50 models are 1D adaptations from Nonaka and Seita [16] of popular CNN architectures for vision. The ConvNet architecture is a generic convolutional neural network that we tuned for strong performance, comprising:

- Four stages, each composed of three identical residual blocks.
- The first stage has model dimension (i.e. channels, in CNN nomenclature) $H = 64$. Each stage doubles the dimension of the previous stage (with a position-wise linear layer) and ends in an average pooling layer of width 4. Thus, the first stage operates on inputs of length 16384, dimension 64 (the input is zero-padded from 16000 to 16384) and the last on length 256, dimension 512.
- Each residual block has a (pre-norm) BatchNorm layer followed by a convolution layer and GeLU activation.
- Convolution layers have a kernel size of 25.

Long Range Arena. Our Long Range Arena experiments follow the same setup as the original S4 paper with some differences in model architecture and hyperparameters. The main global differences are as follows:

Bidirectional The original S4 layer is unidirectional or causal, which is an unnecessary constraint for the classification tasks appearing in LRA. Goel et al. [5] propose a bidirectional version of S4 that simply concatenates two S4 convolution kernels back-to-back. We use this for all tasks.

GLU feedforward S4 consists of H independent 1-dimensional SSMs, each of which are processed by an independent S4 SSM mapping $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$. These outputs are then mixed with a

Table 10: (**Speech Commands classification.**) Test accuracy on 35-way keyword spotting. Training examples are 1-second audio waveforms sampled at 16000Hz, or a 1-D sequence of length 16000. Last column indicates 0-shot testing at 8000Hz where examples are constructed by naive decimation.

Model	Parameters	16000Hz	8000Hz
S4-LegS	307K	96.08 (0.15)	91.32 (0.17)
S4-FouT	307K	95.27 (0.20)	91.59 (0.23)
S4-(LegS/FouT)	307K	95.32 (0.10)	90.72 (0.68)
S4D-LegS	306K	95.83 (0.14)	91.08 (0.16)
S4D-Inv	306K	96.18 (0.27)	**91.80** (0.24)
S4D-Lin	306K	**96.25** (0.03)	91.58 (0.33)
InceptionNet	481K	61.24 (0.69)	05.18 (0.07)
ResNet-18	216K	77.86 (0.24)	08.74 (0.57)
XResNet-50	904K	83.01 (0.48)	07.72 (0.39)
ConvNet	26.2M	95.51 (0.18)	07.26 (0.79)

Table 11: The values of the best hyperparameters found for all datasets; full models on ablation datasets (Top) and LRA (Bottom). LR is learning rate and WD is weight decay. BN and LN refer to Batch Normalization and Layer Normalization.

	Depth	**Features** H	**State Size** N	**Norm**	**Pre-norm**	**Dropout**	**LR**	**Batch Size**	**Epochs**	**WD**	$(\Delta_{min}, \Delta_{max})$
sCIFAR	6	512	64	LN	False	0.1	0.01	50	200	0.05	$(0.001, 0.1)$
SC	6	128	64	BN	True	0	0.01	16	40	0.05	$(0.001, 0.1)$
BIDMC	6	128	256	LN	True	0	0.01	32	500	0.05	$(0.001, 0.1)$
ListOps	8	128	64	BN	False	0	0.01	50	40	0.05	$(0.001, 0.1)$
Text	6	256	64	BN	True	0	0.01	16	32	0.05	$(0.001, 0.1)$
Retrieval	6	256	64	BN	True	0	0.01	64	20	0.05	$(0.001, 0.1)$
Image	6	512	64	LN	False	0.1	0.01	50	200	0.05	$(0.001, 0.1)$
Pathfinder	6	256	64	BN	True	0	0.004	64	200	0.03	$(0.001, 0.1)$
Path-X	6	256	64	BN	True	0	0.0005	32	50	0.05	$(0.0001, 0.01)$

position-wise linear layer, i.e. $\boldsymbol{W}y$ for a learned matrix $\boldsymbol{W} \in \mathbb{R}^{H \times H}$. Instead of this linear mapping, we use a GLU activation $(\boldsymbol{W}_1 y) \circ \sigma(\boldsymbol{W}_2 y)$ for $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{H \times H}$ [3]. These have been empirically found to improve linear layers of DNNs in general [22].

Cosine scheduler Instead of the plateau scheduler used in [9], we use a cosine annealing learning rate scheduler for all tasks.

Regularization Almost all tasks used no dropout and 0.05 weight decay.

Architecture Almost all tasks used an architecture with 6 layers, $H = 256$ hidden units, BatchNorm, pre-norm placement of the normalization layer.

Exceptions to the above rules are described below. Full hyperparameters are in Table 11.

sCIFAR / LRA Image. This dataset is grayscale sequential CIFAR-10, and the settings for this task were taken from S4's hyperparameters on the normal sequential CIFAR-10 task. In particular, this used LayerNorm [1] instead of BatchNorm [13], a larger number of hidden features H, post-norm instead of pre-norm, and minor dropout. We note that the choice of normalization and increased H do not make a significant difference on final performance, still attaining classification accuracy in the high 80's. Dropout does seem to make a difference.

BIDMC. We used a larger state size of $N = 256$, since we hypothesized that picking up higher frequency features on this dataset would help. We also used a step scheduler that decayed the LR by 0.5 every 100 epochs, following prior work [21, 8].

ListOps. We hypothesized that this task benefits from deeper models, because of the explicit hierarchical nature of the task, so the architecture used here had 8 layers and $H = 128$ hidden features. However, results are very close with much smaller models. We also found that post-norm generalized better than pre-norm, but results are again close (less than 1% difference).

PathX. As described in [10], the initialization range for PathX is decreased from $(\Delta_{min}, \Delta_{max}) = (0.001, 0.1)$ to $(\Delta_{min}, \Delta_{max}) = (0.0001, 0.01)$.