On the Parameterization and Initialization of Diagonal State Space Models

Anonymous Author(s)
Affiliation
Address
email

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 also requires a custom representation and algorithm that makes the model difficult to understand and 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 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. First, we explain why DSS works mathematically, as the diagonal approximation to S4 surprisingly recovers the same dynamics in the limit of infinite state dimension. We then 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 3 lines of code and performs comparably to S4 in almost all settings, with state-of-the-art results in image, audio, and medical time-series domains, and 85% average on the Long Range Arena benchmark.

1 Introduction

A core problem in deep learning is sequence modeling, which is typically characterized by defining a parameterized sequence-to-sequence function that can be composed. Recent approaches based on state space models (SSMs) have shown promise to outperform 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 [7]. Its ability to model long-range dependencies arises from being defined with a particular state matrix called the "HiPPO matrix" [4], which allows S4 to be viewed as a convolutional model that decomposes an input onto an orthogonal system of smooth basis functions (Fig. 1).

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 the state matrix as a **diagonal plus low-rank** (DPLR) matrix, which is necessary to capture HiPPO matrices. A natural question is whether simplifications of this parameterization and algorithm are possible. In particular, removing

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

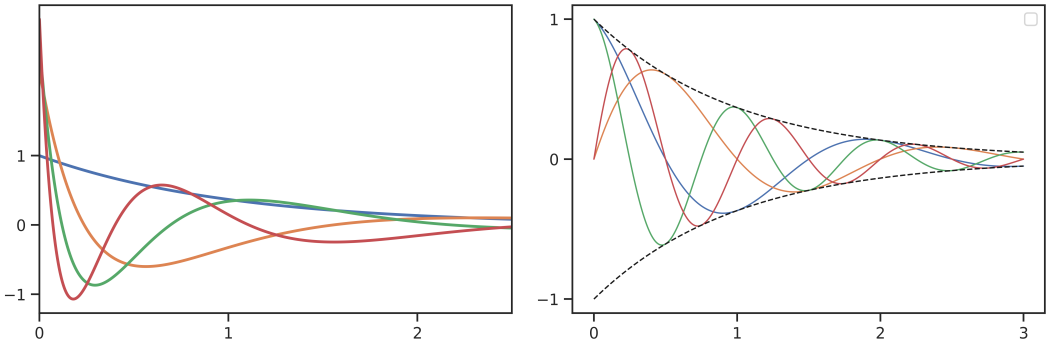


Figure 1: State space models can be interpreted as convolutional models where the convolution kernel is a linear combination of basis functions. (*Left*) The particular $\boldsymbol{A}$ matrix chosen in S4 results in smooth basis functions with a closed form formula in terms of Legendre polynomials, resulting in a simple mathematical interpretation of the method as orthogonalizing against an exponentially-decaying measure. This basis grants long-range modeling properties, with the tradeoff of being difficult to compute the system with this matrix. (*Right*) We analyze much simpler state spaces with diagonal matrices. We show that it is critical to carefully initialize both the real parts which control the decay (dashed line), and the imaginary parts which control the frequencies (solid lines), allowing our diagonal SSM S4D to match S4 in nearly all settings.

the low-rank term would result in a **diagonal state space model** (DSSM) that is dramatically simpler to implement and understand.

Although it is known that almost all SSMs have an equivalent diagonal form—and therefore (complex) DSSMs are fully expressive algebraically—they may not represent all SSMs numerically, and finding a good initialization is critical. Gu et al. [7] 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 [8] made the surprising 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 efficacy of this particular matrix compared to other diagonal matrices remains theoretically unexplained.

In this work, we seek to systematically understand how to train DSSMs.

- First, we categorize and consolidate various possible representations of diagonal SSMs, allowing the various design choices to be systematically compared – in particular the discretization, parameterization, and normalization of SSM models (Section 3).
- We theoretically provide a new mathematical analysis of DSS to understand why it works , showing that the diagonal matrix they chose based on the original HiPPO matrix is surprisingly an exact approximation of the dynamics of S4 as the state size goes to infinity. We propose even simpler variants of diagonal SSMs using different initializations of the state matrix. Our class of **S4D** methods compute the same function as S4 in the case of diagonal instead of DPLR state matrices, allowing dramatically simpler implementation (Section 4).
- We perform a controlled study of these various design choices across various settings, tasks, and sequence lengths, with both diagonal (S4D) and DPLR (S4) variants. Our best S4D methods are competitive with S4 on most 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 of the form that map signals $u(t) \mapsto y(t)$, which are linear time-invariant systems that can be represented either as a linear ODE (equation (1)) or convolution (equation (2)).

$$\begin{aligned} x'(t) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ y(t) &= \boldsymbol{C}x(t) \end{aligned} \tag{1}$$

$$\begin{aligned} K(t) &= \boldsymbol{C}e^{t\boldsymbol{A}}\boldsymbol{B} \\ y(t) &= (K * u)(t) \end{aligned} \tag{2}$$

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 $\boldsymbol{A}, \boldsymbol{B}$)

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

We denote this basis as $K_{\boldsymbol{A},\boldsymbol{B}}(t) = e^{t\boldsymbol{A}}\boldsymbol{B}$ if necessary to disambiguate. In the case of DSSMs, we will denote $\boldsymbol{A} = \mathrm{diag}\langle \boldsymbol{a} \rangle$ for $\boldsymbol{a} \in \mathbb{C}^N$. Note that in this simple case $K_n(t)$ 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 [6]. For example, the convolutional form (2) can be converted into a temporal recurrence that is substantially faster for autoregressive applications [3].

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

First, Gu et al. [6] 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). This matrix has a mathematical interpretation of decomposing the input signal $u(t)$ onto a set of orthogonal basis functions $L_n(e^{-t})$ (where $L_n(t)$ are normalized Legendre polynomials) with respect to an exponentially-decaying measure, giving it long-range modeling abilities [0].[1]

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)**.

$$\boldsymbol{A}_{nk} = -\begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & n > k \\ n+1 & n = k \\ 0 & n < k \end{cases} \qquad \boldsymbol{B}_n = \boldsymbol{P}_n = (2n+1)^{\frac{1}{2}} \tag{4}$$

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

$$\boldsymbol{A}^N_{nk} = -\begin{cases} (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n > k \\ \frac{1}{2} & n = k \\ (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n < k \end{cases} \qquad \boldsymbol{A} = \boldsymbol{A}^N - \boldsymbol{P}\boldsymbol{P}^\top, \qquad \boldsymbol{A}^D := \mathrm{diag}(\boldsymbol{A}^N) \tag{5}$$

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

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 [7, Lemma 3.2], and they were unable to find any diagonal matrices that worked well, resulting in the DPLR formulation.

Gupta [8] 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 to the original S4 method. More precisely, their initialization is the diagonal matrix $\boldsymbol{A}^D$ (5).

In addition to this initialization of a diagonal state matrix $\boldsymbol{A}$, they proposed a method for computing the resulting state space. In Sections 3 and 4, we systematically study the components of DSS. We categorize different ways to parameterize and compute the diagonal state space. Most importantly, we show how to theoretically interpret 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 3.5).

3.1 Discretization

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

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

[1] The citation [0] refers to a concurrent submission that is included in the supplemental.

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

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

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

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

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

These integration rules have both been used in prior works (e.g. LMU and DSS use ZOH [18, 8]) while S4 and its predecessors use bilinear [4, 6, 7]).

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. 2c). 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 [12] 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. In the diagonal case, the computation is nearly trivial. Note that

$$\overline{\boldsymbol{K}}_\ell = \sum_{n \in [N]} \overline{\boldsymbol{B}}_n \boldsymbol{a}_n^\ell \boldsymbol{C}_n = (\overline{\boldsymbol{B}} \circ \boldsymbol{C}) \cdot \mathcal{V}_L(\boldsymbol{a}) \qquad \mathcal{V}_L(\boldsymbol{a})_{n,\ell} = \boldsymbol{a}_n^\ell \tag{7}$$

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

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. [3] 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 classic left-half plane condition, by parameterizing the real part inside an exponential function $\boldsymbol{A} = -\exp(\boldsymbol{A}_r) + i \cdot \boldsymbol{A}_i$.

We note that instead of $\exp$, other choices can be made such as a simple clamp (i.e. ReLU). Alternatively, the original DSS uses no constraint on the real part of $\boldsymbol{A}$, which is sufficient for simple tasks involving fixed-length sequences, but would become unstable in other settings.

Parameterization of $\boldsymbol{B}, \boldsymbol{C}$. 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 we can choose to store this product as the parameter, instead of $\boldsymbol{B}$ and $\boldsymbol{C}$ individually.

3.4 Normalization

An issue that requires consideration for SSMs is variance preservation. A simple proxy to ensure that the SSM is well behaved is [0]:

For constant input $u(t) = c$, the output $y(t)$ should have mean 0, var. c^2 (w.r.t. the initialization)

While this is satisfied automatically for HiPPO SSMs such as S4 [0], it is difficult to ensure for DSSMs. To address this, DSS used a **softmax** in their kernel which explicitly normalized every basis function $K_n(t)$ of the SSM (3) during every computation of the kernel. This has several drawbacks, including making the kernel more complicated, and additionally has the drawback of being calibrated only to a specific target length – in other words, changing the length L of the target kernel requires changing the underlying parameters $\boldsymbol{A}$ and $\boldsymbol{B}$.

We would like to avoid the softmax, but address the normalization issue. While an analog to the HiPPO normalization cannot be analytically derived in the DSSM case, we simply enforce the condition at initialization by scaling $\boldsymbol{B}$ by a constant. Given an arbitrary diagonal $\boldsymbol{A} = \text{diag}(\boldsymbol{a})$

and vector $\boldsymbol{B}$, The n-th SSM basis function is $e^{t\boldsymbol{a}_n}\boldsymbol{B}_n$. With the constant input $u(t) = 1$, the n-th component of the state (as $t \to \infty$) is given by (8), and the full state vector $x(t)$ has norm (9)

$$x_n(t) = \int_0^\infty e^{s\boldsymbol{a}_n}\boldsymbol{B}_n \, ds = -\frac{\boldsymbol{B}_n}{\boldsymbol{a}_n} \qquad (8)$$

$$\|x(t)\|^2 \left(\sum_{n=0}^{N-1} \left| \frac{\boldsymbol{B}_n}{\boldsymbol{a}_n} \right|^2 \right)^{\frac{1}{2}}. \qquad (9)$$

Dividing $\boldsymbol{B}$ by this scalar yields a proper initialization that passes the above test. We call this the **scalar init** normalization, since it simply normalizes by a scalar at initialization.

3.5 S4D: the Diagonal Version of S4

A key component of our exposition is disentangling the various choices possible in representing and computing a diagonal state space. With this categorization, different choices can be mixed and matched to define variants of the core method. We define S4D so that the model $u \mapsto y$ *exactly matches the output of S4 when the low-rank component is set to* 0. This allows us to compare the performance of diagonal vs. DPLR state spaces while controlling all other factors. (We also mention how the components can be swapped out with alternatives, e.g. more sophisticated discretizations, to define promising alternate methods, and leave this as a direction for future work.) In our empirical study in Section 5, we systematically ablate the effects of each of these components.

Method	Discretization	Parameterization of $\boldsymbol{A}$	... of $\boldsymbol{B}, \boldsymbol{C}$	Normalization	Initialization of $\boldsymbol{A}$
DSS	ZOH	no restriction	combine	softmax	HiPPO-LegS-D
S4D	Bilinear	exp real part	independent	scalar init	various
(S4D-Trap)	Trapezoid	exp real part	independent	scalar init	various

Compared to DSS, the parameterization of S4D is guaranteed to be stable even in unbounded autoregressive settings, and its computation is much simpler due to not requiring the softmax normalization; our proposed scalar init normalization (Section 3.4) attains most of the benefits. The entire S4D method is almost trivial to implement, requiring just a few lines of code each for the initialization, kernel computation, and forward pass (Listing 1).

4 Initialization of Diagonal State Matrices

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

Expressivity and Limitations of Diagonal SSMs. We first present a simplified view on the expressivity of DSSMs mentioned by [8]. 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 DSSMs.

Proposition 1. *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).*

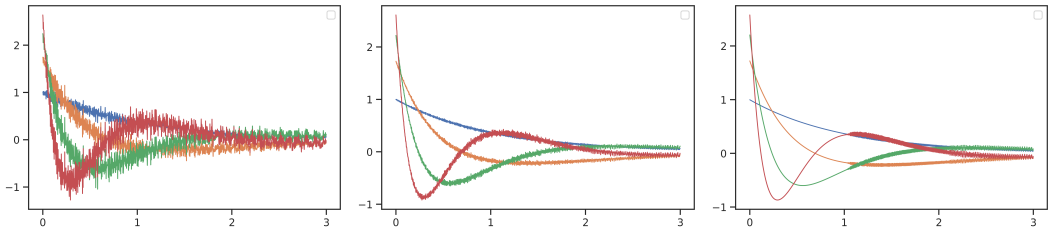
It is also well known that the state space $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ is exactly equivalent to (i.e. expresses the same map $u \mapsto y$) the state space $(\boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}, \boldsymbol{V}^{-1}\boldsymbol{B}, \boldsymbol{C}\boldsymbol{V})$, known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

However, we emphasize that Proposition 1 is about *expressivity* which does not guarantee strong performance of a trained model after optimization. For example, Gu et al. [7] and Gupta [8] show that randomly initialized dense real matrices or diagonal complex matrices, which are both fully expressive, perform much worse than S4.

Second, Proposition 1 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. In Section 5.2, we also show that two different representations of matrices initialized with the *same spectrum* (i.e., are equivalent to the same diagonal $\boldsymbol{A}$) can have vastly different performance.

S4D-LegS. Gupta [8] showed that taking the DPLR form of the HiPPO-LegS matrix $\boldsymbol{A}^D - \boldsymbol{P}\boldsymbol{P}^\top$ and simply approximating it with $\boldsymbol{A}^D$ works quite well (5). Our first result is providing a clean mathematical interpretation of this method. Theorem 2 shows a surprising fact that does not hold in general for DPLR matrices (Appendix A.2), and arises out of the special structure of this particular matrix.

Theorem 2. *Let $\boldsymbol{A} = \boldsymbol{A}^D - \boldsymbol{P}\boldsymbol{P}^\top$ and $\boldsymbol{B}$ be the HiPPO-LegS matrices, and $K_{\boldsymbol{A},\boldsymbol{B}}$ be its basis (Fig. 1). As the state size $N \to \infty$, the SSM basis $K_{\boldsymbol{A}^D,\boldsymbol{B}}$ is unitarily equivalent to $K_{\boldsymbol{A},\boldsymbol{B}}$ (Fig. 2).*



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

Figure 2: **(Visualization of Theorem 2).** (a, b) By special properties of the original S4 matrix, removing its DPLR low-rank term produces the same basis functions as $N \to \infty$, explaining the empirical effectiveness of DSS. S4D-Inv is a simplified variant of S4D-LegS with approximate eigenvalues. (c) Curiously, the bilinear transform instead of ZOH smooths out the kernel to exactly match S4-LegS (Fig. 1 (*Left*)) as N grows.

(Note that *unitary* equivalence is important as it preserves the stability and timescale [0] of the system.)

We define **S4D-LegS** to be the S4D method for this choice of diagonal $\boldsymbol{A} = \boldsymbol{A}^D$ and $\boldsymbol{B}$. Theorem 2 explains the empirical results in [8] whereby this system performed quite close to S4, but was generally consistently slightly worse. This is because DSS is a variant of S4D-LegS, which by Theorem 2 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 [7]: $\Re(\boldsymbol{a}) = -\frac{1}{2}\mathbf{1}$. Let the imaginary part be sorted, i.e. $\Im(\boldsymbol{\Lambda})_n$ is the n-th largest (positive) imaginary component. We empirically deduced the following conjecture for the asymptotics of the imaginary part.

Conjecture 3. *As $N \to \infty$, $\Im(\boldsymbol{\Lambda})_0 \to \frac{1}{\pi}N^2 + c$ where $c \approx 0.5236$ is a constant. For a fixed N, the other eigenvalues satisfy an inverse scaling in n: $\Im(\boldsymbol{\Lambda})_n = \Theta(n^{-1})$.*

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

$$(\textbf{S4D-Inv}) \quad \boldsymbol{\Lambda}_n = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{2n+1} - 1\right) \qquad (10) \qquad (\textbf{S4D-Lin}) \quad \boldsymbol{\Lambda}_n = -\frac{1}{2} + i\pi n \qquad (11)$$

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 ([0]), where the imaginary parts are simply the Fourier series frequencies (.e. matches the diagonal part of the DPLR form of S4-FouT). Fig. 1 (*Right*) illustrates an idealized basis that S4D-Lin can produce, which are simply damped Fourier basis functions.

General DSSM Basis Functions. The empirical study in Section 5 performs many ablations of different diagonal initializations, showing that many natural variants of the proposed methods do not perform as well. The overall guiding principles for the diagonal state matrix $\boldsymbol{\Lambda}$ are twofold, which can be seen from the closed form of the basis functions $K_n(t) = e^{t\boldsymbol{a}_n}\boldsymbol{B}_n$ (Eq. (3)).

First, the real part of $\boldsymbol{a}_n$ controls the decay rate of the function. $\boldsymbol{a}_n = -\frac{1}{2}$ is a good default that bounds the basis functions by the envelope $e^{-\frac{t}{2}}$, giving a constant timescale (Fig. 1 (*Right*)).

Second, the imaginary part of $\boldsymbol{a}_n$ controls the oscillating frequencies of the basis function. Critically, these need to be spread out, which explains why random initializations of $\boldsymbol{\Lambda}$ do not perform well. The S4D-Inv and S4D-Lin use simple asymptotics for these imaginary components that provide interpretable methods. We believe that alternative initializations that have different mathematical interpretations may exist, which is an interesting question for future work.

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 performs all non-SSM baselines.

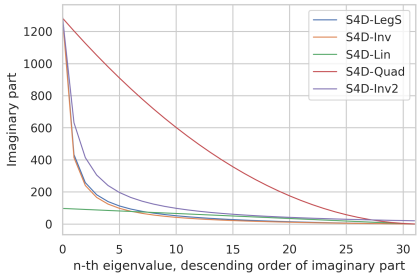


Figure 3: (**S4D eigenvalues.**) All S4D methods have eigenvalues $-\frac{1}{2} + \lambda_n i$. S4D-LegS theoretically approximates dynamics of the original (non-diagonal) S4 (Blue), and has eigenvalues following an inverse law $\lambda_n \propto n^{-1}$ (Orange). The precise law is important: other 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).

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 DSSMs 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 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 autoregresive (AR) or *causal* model to examine different settings; the causal setting more closely imitates autoregressive speech generation, where SSMs have shown recent promise [3].

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, Normalization, 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 several aspects of the parametrization, normalization, and discretization. Results are in Table 1, and show that:

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

(ii) Unrestricting the real part Section 3.3 may be slightly better at the expense of generation stability [3], but overall makes little difference.

(iii) Normalizing the model with either softmax or the simple scalar rescaling Section 3.4 provides a consistent improvement, validating the importance of proper normalization.

Discretization	Normalization	sCIFAR	SC (uni)	BIDMC (SpO2)
Bilinear	none	84.53	89.52	0.1204
ZOH	none	84.50	89.53	0.1341
Bilinear	scalar init	85.14	89.90	0.1090
ZOH	scalar init	84.84	90.19	0.1299
ZOH	softmax	85.14	90.36	0.1299

Discretization	Real part	sCIFAR	SC (uni)	BIDMC (SpO2)
Bilinear	Exp	85.20	89.52	0.1193
	None	85.35	90.58	0.1102
	Relu	85.06	90.22	0.1172
ZOH	Exp	85.02	89.93	0.1303
	None	85.15	90.19	0.1289
	Relu	84.98	90.03	0.1232

Table 1: (*Left*) Ablations on Discretization and Normalization of DSSMs. (*Right*) Ablations on Discretization and Parameterization of DSSMs.

These ablations justify the simpler parameterization and algorithm that we use for S4D (Section 3.5) compared to DSS. For the remaining 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

[2] We note that a line of prior work including S4 [10, 13, 7] all used a smaller 10-class subset of SC, so our numbers are not directly comparable.

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 [8] 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 (10). This raises the question of whether the initialization of the $\boldsymbol{A}$ still needs to be so precise, despite the drastic simplifications from the original version. We perform several natural ablations on these initializations, showing that even simple variations of the precise formula will degrade performance.

Imaginary part scaling factor. The scaling rules for S4D-Inv and S4D-Lin are simple polynomial laws, but how is the constant factor chosen and how important is it? These constants for the imaginary parts of S4D initializations were chosen based on connections to S4 methods based on HiPPO. 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) and simply changes the frequencies, and it is not obvious that this should degrade performance. However, both changes substantially reduce the performance of both S4D methods 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 $u \sim N \cdot Unif(0, 1)$. Note that when u is equally spaced in $[0, 1]$ instead of uniformly random, this exactly recovers S4D-Inv (10), so this is a sensible random approximation to it.

Similarly, we consider a variant of S4D-Lin that replaces the n in (11) with $N \cdot Unif(0, 1)$.

Table 2a (*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 $-Unif(0, 1)$ instead of fixing them to $-\frac{1}{2}$. Table 2a(Left, *Random Real*) shows that this also causes minor but consistent degradation in performance on the diagnostic 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 (S4D-Inv2, Quad) (Fig. 3) show that it is not just the range of the eigenvalues but the actual distribution that is important.

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 poorly, in line with earlier findings [7, 8]. Second, S4D-Real uses a particular real diagonal initialization with $\boldsymbol{\Lambda}_n = -(n + 1)$. 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 limitation of Proposition 1.

5.3 Full Comparisons of S4D and S4 Methods

Trainable $\boldsymbol{A}, \boldsymbol{B}$ matrices. Table 2b shows the performance of all S4D and S4 variants [0] on the diagnostics 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 HTTYH I) have the most limited context and suffer the most when $\boldsymbol{A}$ is frozen. (ii) The full DPLR versions are often slightly better throughout training. We report the validation accuration after 1 epoch of training on sCIFAR and SC to illustrate this phenomenon.

Table 2: (*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 diagnostic 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.

Table 3: (**Diagnostic 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)	95.42 (0.20)	0.339 (0.020)	0.301 (0.030)	**0.068** (0.003)
S4D-LegS	89.92 (1.69)	93.57 (0.09)	95.97 (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)

Large models on diagnostic datasets. Finally, we relax the strict requirements on model size and regularization for the diagnostic 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 3). We note that results for each dataset are better than the original S4 model, which was already state-of-the-art on these datasets.

Long Range Arena We use the same hyperparameter setting for the state-of-the-art S4 model in [0] 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 [16].

Table 4: (**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)	**96.76** (-)	**86.16**
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)	✗	77.89
S4D-Inv	60.18 (0.35)	**87.34** (0.20)	**91.09** (0.01)	87.83 (0.37)	93.78 (0.25)	91.22 (-)	85.24
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

References

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

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

[3] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state-space models. *arXiv preprint arXiv:2202.09729*, 2022.

[4] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/102f0bb6efb3a6128a3c750dd16729be-Abstract.html`.

[5] 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.

[6] 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.

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

[8] Ankit Gupta. Diagonal state spaces are as effective as structured state spaces. *arXiv preprint arXiv:2203.14343*, 2022.

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

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

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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`.

[17] 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.

[18] 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 [7] repository, which is publicly available. All experimental details to reproduce results are in the Appendix.
   (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)? [No] Our datasets were small scale, and the Long Range Arena details match those of S4 and are publicly available.
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]