Diagonal State Spaces are as Effective as Structured State Spaces

Ankit Gupta*
IBM Research
ankitgupta.iitkanpur@gmail.com

Albert Gu
Stanford University
albertgu@stanford.edu

Jonathan Berant
Tel Aviv University
joberant@cs.tau.ac.il

Abstract

Modeling long range dependencies in sequential data is a fundamental step towards attaining human-level performance in many modalities such as text, vision, audio and video. While attention-based models are a popular and effective choice in modeling short-range interactions, their performance on tasks requiring long range reasoning has been largely inadequate. In an exciting result, Gu et al. (2022) proposed the *Structured State Space* (S4) architecture delivering large gains over state-of-the-art models on several long-range tasks across various modalities. The core proposition of S4 is the parameterization of state matrices via a diagonal plus low rank structure, allowing efficient computation. In this work, we show that one can match the performance of S4 even without the low rank correction and thus assuming the state matrices to be diagonal. Our *Diagonal State Space (DSS)* model matches the performance of S4 on Long Range Arena tasks, speech classification on Speech Commands dataset, while being conceptually simpler and straightforward to implement.

1 Introduction

The Transformer architecture [VSP+17] has been successful across many areas of machine learning. Transformers pre-trained on large amounts of unlabelled text via a denoising objective have become the standard in natural language processing, exhibiting impressive amounts of linguistic and world knowledge [BMR+20, BMH+22]. This recipe has also led to remarkable developments in the areas of vision [RPG+21, RKH+21] and speech [DXX18, BZMA20].

The contextualizing component of the Transformer is the multi-head attention layer which, for inputs of length L, has an expensive $\Omega(L^2)$ complexity. This becomes prohibitive on tasks where the model is required to capture long-range interactions of various parts of a long input. To alleviate this issue, several Transformer variants have been proposed with reduced compute and memory requirements [QML+20, KKL20, RSVG21, BPC20, GB20, KVPF20, WLK+20, CLD+21, ZS21, GDG+21] (see [TDBM20] for a survey). Despite this effort, all these models have reported inadequate performance on benchmarks created to formally evaluate and quantify a model's ability to perform long-range reasoning (such as Long Range Arena [TDA+21] and SCROLLS [SSI+22]).

In a recent breakthrough result, Gu et al. [GGR22] proposed S4, a sequence-to-sequence model that uses linear state spaces for contextualization instead of attention. It has shown remarkable performance on tasks requiring long-range reasoning in domains such as text, images and audio. For instance, on Long Range Arena it advances the state-of-the-art by 19 accuracy points over the best performing Transformer variant. Its remarkable abilities are not limited to text and images and carry over to tasks such as time-series forecasting, speech recognition and audio generation [GGDR22].

Despite S4's achievements, its design is complex and is centered around the HiPPO theory, which is a mathematical framework for long-range modeling [VKE19, GDE+20, GJG+21]. [GGR22]

*work done while author was part of IBM AI Residency program.

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

showed that state space models with various alternative initializations perform poorly in comparison to initializing the state-space parameters with a particular HiPPO matrix. In order to leverage this matrix, they parameterize the learned state spaces using a Diagonal Plus Low Rank (DPLR) structure and, as a result, need to employ several reduction steps and linear algebraic techniques to be able to compute the state space output efficiently, making S4 difficult to understand, implement and analyze.

In this work, we show that it is possible to match S4's performance while using a much simpler, fully-diagonal parameterization of state spaces. While we confirm that random diagonal state spaces are less effective, we observe that there do in fact exist effective diagonal state matrices: simply removing the low-rank component of the DPLR HiPPO matrix still preserves its performance. Leveraging this idea, our proposed Diagonal State Space (DSS) model enforces state matrices to be diagonal, making it significantly simpler to formulate, implement and analyze, while being provably as expressive as general state spaces. In contrast to S4, DSS does not assume any specialized background beyond basic linear algebra and can be implemented in just a few lines of code. Our implementation fits in a single page and is provided in §A.7 (Figure 6).

We evaluate the performance of DSS on Long Range Arena (LRA) which is a suite of sequence-level classification tasks with diverse input lengths ($1K$-$16K$) requiring similarity, structural, and visual-spatial reasoning over a wide range of modalities such as text, natural/synthetic images, and mathematical expressions. Despite its simplicity, DSS delivers an average accuracy of 81.88 across the 6 tasks of LRA, comparable to the state-of-the-art performance of S4 (80.21). In addition, DSS maintains a comfortable 20 point lead over the best Transformer variant (81.88 vs 61.41).

In the audio domain, we evaluate the performance of DSS on raw speech classification. On the Speech Commands dataset [War18], which consists of raw audio samples of length $16K$, we again found the performance of DSS to be comparable to that of S4 (98.2 vs 98.1).

To summarize, our results demonstrate that DSS is a simple and effective method for modeling long-range interactions in modalities such as text, images and audio. We believe that the effectiveness, efficiency and transparency of DSS can significantly contribute to the adoption of state space models over their attention-based peers. Our code is available at `https://github.com/ag1988/dss`.

2 Background

We start by reviewing the basics of time-invariant linear state spaces.

State Spaces A continuous-time state space model (SSM) parameterized by the state matrix $A \in \mathbb{R}^{N\times N}$ and vectors $B \in \mathbb{R}^{N\times 1}$, $C \in \mathbb{R}^{1\times N}$ is given by the differential equation:

$$\frac{dx}{dt}(t) = Ax(t) + Bu(t) \ \ , \ \ y(t) = Cx(t) \tag{1}$$

which defines a function-to-function map $u(t) \mapsto y(t)$. For a given value of time $t \in \mathbb{R}$, $u(t) \in \mathbb{R}$ denotes the value of the input signal u, $x(t) \in \mathbb{R}^{N\times 1}$ denotes the state vector and $y(t) \in \mathbb{R}$ denotes the value of the output signal y. We call a state space *diagonal* if it has a diagonal state matrix.

Discretization For a given sample time $\Delta \in \mathbb{R}_{>0}$, the discretization of a continuous state space (Equation 1) assuming *zero-order hold*[2] on u is defined as a sequence-to-sequence map from $(u_0, \dots, u_{L-1}) = u \in \mathbb{R}^L$ to $(y_0, \dots, y_{L-1}) = y \in \mathbb{R}^L$ via the recurrence,

$$\begin{aligned} x_k &= \overline{A}x_{k-1} + \overline{B}u_k \ \ , \ \ y_k = \overline{C}x_k \\ \overline{A} &= e^{A\Delta} \ , \ \overline{B} = (\overline{A} - I)A^{-1}B \ , \ \overline{C} = C \ . \end{aligned} \tag{2}$$

Assuming $x_{-1} = 0$ for simplicity, this recurrence can be explicitly unrolled as

$$y_k \ = \ \sum_{j=0}^{k} \overline{CA}^j \overline{B} \cdot u_{k-j} \ . \tag{3}$$

For convenience, the scalars $\overline{CA}^k\overline{B}$ are gathered to define the SSM kernel $\overline{K} \in \mathbb{R}^L$ as

$$\overline{K} \ = \ (\overline{CB}, \overline{CAB}, \dots, \overline{CA}^{L-1}\overline{B}) \ = \ (\ Ce^{A\cdot k\Delta}(e^{A\Delta} - I)A^{-1}B\ )_{0\leqslant k<L}, \tag{4}$$

[2] assumes the value of a sample of u is held constant for a duration of one sample interval Δ [Gaj03].

where the last equality follows by substituting the values of $\overline{A}$, $\overline{B}$, $\overline{C}$ from Equation 2. Hence,

$$y_k \;=\; \sum_{j=0}^{k} \overline{K}_j \cdot u_{k-j} \;. \tag{5}$$

Given an input sequence $u \in \mathbb{R}^L$, it is possible to compute the output $y \in \mathbb{R}^L$ sequentially via the recurrence in Equation 2. Unfortunately, sequential computation is prohibitively slow on long inputs and, instead, Equation 5 can be used to compute all elements of y in parallel, provided we have already computed $\overline{K}$.

Computing y from u and $\overline{K}$ is easy. Given an input sequence $u \in \mathbb{R}^L$ and the SSM kernel $\overline{K} \in \mathbb{R}^L$, naively using Equation 5 for computing y would require $O(L^2)$ multiplications. Fortunately, this can be done much more efficiently by observing that for the univariate polynomials

$$\overline{K}(z) = \sum_{i=0}^{L-1} \overline{K}_i z^i \;\text{ and }\; u(z) = \sum_{i=0}^{L-1} u_i z^i,$$

y_k is the coefficient of z^k in the polynomial $\overline{K}(z) \cdot u(z)$, i.e. all y_k's can be computed simultaneously by multiplying two degree $L-1$ polynomials. It is well-known that this can be done in $O(L \log(L))$ time via Fast Fourier Transform (FFT) [CLRS09]. We denote this fast computation of Equation 5 via the discrete convolution as

$$y = \overline{K} * u \;. \tag{6}$$

Hence, given the SSM kernel $\overline{K} \in \mathbb{R}^L$, the output of a discretized state space can be computed efficiently from the input. The challenging part is computing $\overline{K}$ itself as it involves computing L distinct matrix powers (Equation 4). Similar to S4, instead of directly using A, B, C, our idea is to use an alternate parameterization of state spaces for which it would be easier to compute $\overline{K}$.

3 Method

Having stated the necessary background, we now turn to the main contribution of our work.

3.1 Diagonal State Spaces

For a general state matrix A it is expensive to compute the matrix exponentials $e^{A \cdot k\Delta}$ for all $0 \leqslant k < L$ (Equation 4). However, if A was diagonalizable over $\mathbb{C}$ and we were given its eigendecomposition $A = V \Lambda V^{-1}$, where Λ is diagonal, then we could easily compute $e^{A \cdot k\Delta} = V e^{\Lambda \cdot k\Delta} V^{-1}$ as it is trivial to exponentiate diagonal matrices.[3] In this case Equation 4 simplifies to

$$\overline{C}\overline{A}^k\overline{B} \;=\; C e^{V\Lambda V^{-1} \cdot k\Delta}(e^{V\Lambda V^{-1}\Delta} - I)(V\Lambda V^{-1})^{-1} B \;=\; (CV) e^{\Lambda k\Delta}(e^{\Lambda\Delta} - I)\Lambda^{-1}(V^{-1}B) \;.$$

Leveraging the above observation, we base our model on the following proposition which asserts that, under mild technical assumptions, diagonal state spaces are as expressive as general state spaces. We use the operator $\overline{K}_{\Delta,L}(A, B, C) \in \mathbb{C}^{1\times L}$ to denote the kernel (Equation 4) of length L for state space (A, B, C) over $\mathbb{C}$ and sample time Δ.

Proposition 1. *Let $K \in \mathbb{C}^{1\times L}$ be the kernel of length L of a given state space (A, B, C) and sample time $\Delta > 0$, where $A \in \mathbb{C}^{N\times N}$ is diagonalizable over $\mathbb{C}$ with eigenvalues $\lambda_1, \ldots, \lambda_N$ and $\forall i$, $\lambda_i \neq 0$ and $e^{L\lambda_i\Delta} \neq 1$. Let $P \in \mathbb{C}^{N\times L}$ be $P_{i,k} = \lambda_i k\Delta$ and Λ be the diagonal matrix with $\lambda_1, \ldots, \lambda_N$. Then there exist $\widetilde{w}, w \in \mathbb{C}^{1\times N}$ such that*

$$\text{(a)}\;\; K \;=\; \overline{K}_{\Delta,L}(\Lambda,\; (1)_{1\leqslant i\leqslant N},\; \widetilde{w}) \;=\; \widetilde{w} \cdot \Lambda^{-1}(e^{\Lambda\Delta} - I) \cdot \text{elementwise-exp}(P),$$

$$\text{(b)}\;\; K \;=\; \overline{K}_{\Delta,L}(\Lambda,\; ((e^{L\lambda_i\Delta} - 1)^{-1})_{1\leqslant i\leqslant N},\; w) \;=\; w \cdot \Lambda^{-1} \cdot \text{row-softmax}(P).$$

[3] $e^A = \sum_{j=0}^{\infty} \frac{A^j}{j!} = \sum_{j=0}^{\infty} \frac{(V\Lambda V^{-1})^j}{j!} = V\left(\sum_{j=0}^{\infty} \frac{\Lambda^j}{j!}\right)V^{-1}$. For $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_N)$ we have $\Lambda^j = \text{diag}(\lambda_1^j, \ldots, \lambda_N^j)$ and hence $e^A = V \cdot \text{diag}(\exp(\lambda_1), \ldots, \exp(\lambda_N)) \cdot V^{-1}$.

Input: parameters $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$, $w \in \mathbb{C}^N$, sample time $\Delta_{\log} \in \mathbb{R}$
Output: SSM kernel $\overline{K} \in \mathbb{R}^L$ (Proposition 1(b))

1: $\Lambda \leftarrow \Lambda_{\text{re}} + i \cdot \Lambda_{\text{im}}$, $\Delta \leftarrow \exp(\Delta_{\log})$ ▷ Δ is positive real
2: $P_{N\times L} \leftarrow (\Delta * \Lambda)_{N\times 1} * [0, 1, \ldots L-1]_{1\times L}$ ▷ outer product
3: $S \leftarrow \text{row-softmax}_\epsilon(P)$ ▷ numerically stable (§A.2)
4: $\overline{K} \leftarrow \text{Re}(\ (w/\Lambda)_{1\times N} \cdot S\ )$ ▷ real part[5]

Algorithm 1: $\text{DSS}_{\text{SOFTMAX}}$ KERNEL (SKETCH)

The proof of Proposition 1 is provided in §A.1. In the above equations, the last equality follows by using Equation 4 to explicitly compute the expression for the kernel of the corresponding diagonal state space. Hence, for any given state space with a well-behaved state matrix there exists a diagonal state space computing the same kernel.[4] More importantly, the expressions for the kernels of the said diagonal state spaces no longer involve matrix powers but only a structured matrix-vector product.

Proposition 1(a) suggests that we can parameterize state spaces via $\Lambda, \widetilde{w} \in \mathbb{C}^N$ and simply compute the kernel as shown in Proposition 1(a). Unfortunately, in practice, the real part of the elements of Λ can become positive during training making the training unstable on long inputs. This is because the matrix $\text{elementwise-exp}(P)$ contains terms as large as $\exp(\lambda_i \Delta(L-1))$ which even for a modest value of L can be very large. To address this, we propose two methods to model diagonal state spaces.

DSS_{EXP} In this variant, we use Proposition 1(a) to model our state space but restrict the real part of elements of Λ to be negative. DSS_{EXP} has parameters $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$, $\widetilde{w} \in \mathbb{C}^N$ and $\Delta_{\log} \in \mathbb{R}$. Λ is computed as $-\text{elementwise-exp}(\Lambda_{\text{re}}) + i \cdot \Lambda_{\text{im}}$ where $i = \sqrt{-1}$ [GGDR22]. Δ is computed as $\exp(\Delta_{\log}) \in \mathbb{R}_{>0}$ and the kernel is then computed via Proposition 1(a).

DSS_{EXP} provides a remarkably simple computation of state space kernels but restricts the space of the learned Λ (the real part must be negative). It is not clear if such a restriction could be detrimental for some tasks, and we now present an alternate method that provides the simplicity of Proposition 1(a) while being provably as expressive as general state spaces.

$\text{DSS}_{\text{SOFTMAX}}$ Instead of restricting the elements of Λ, another option for bounding the elements of $\text{elementwise-exp}(P)$ is to normalize each row by the sum of its elements, which leads us to Proposition 1(b). $\text{DSS}_{\text{SOFTMAX}}$ has parameters $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$, $w \in \mathbb{C}^N$ and $\Delta_{\log} \in \mathbb{R}$. Λ is computed as $\Lambda_{\text{re}} + i \cdot \Lambda_{\text{im}}$, Δ is computed as $\exp(\Delta_{\log}) \in \mathbb{R}_{>0}$, and the kernel is then computed via Proposition 1(b). A sketch of this computation is presented in Algorithm 1. We note that, unlike over $\mathbb{R}$, softmax can have singularities over $\mathbb{C}$ and slight care must be taken while computing it (e.g., $\text{softmax}(0, i\pi)$ is not defined). We instead use a corrected version softmax_ϵ and detail it in §A.2.

3.2 Diagonal State Space (DSS) Layer

We are now ready to describe the DSS layer. We retain the skeletal structure of S4 and simply replace the parameterization and computation of the SSM kernel by one of the methods described in §3.1.

Each DSS layer receives a length-L sequence u of H-dimensional vectors as input, i.e., $u \in \mathbb{R}^{H\times L}$, and produces an output $y \in \mathbb{R}^{H\times L}$. The parameters of the layer are $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$, $\Delta_{\log} \in \mathbb{R}^H$ and $W \in \mathbb{C}^{H\times N}$. For each coordinate $h = 1, \ldots, H$, a state space kernel $\overline{K}_h \in \mathbb{R}^L$ is computed using one of methods described in §3.1. For example, in case of $\text{DSS}_{\text{SOFTMAX}}$, Algorithm 1 is used with parameters $\Lambda_{\text{re}}, \Lambda_{\text{im}} \in \mathbb{R}^N$, $W_h \in \mathbb{C}^N$ and $(\Delta_{\log})_h \in \mathbb{R}$. The output $y_h \in \mathbb{R}^L$ for coordinate h is computed from $u_h \in \mathbb{R}^L$ and $\overline{K}_h$ using Equation 6.

Following S4, we also add a residual connection from u to y followed by a GELU non-linearity [HG16]. This is followed by a position-wise linear projection $W_{\text{out}} \in \mathbb{R}^{H\times H}$ to enable information exchange among the H coordinates.

[4] It is possible for norms of the parameters of the resulting diagonal state spaces to be much larger than that of the original state space. For example, this occurs for the HiPPO matrix [GGR22].

[5] As the inputs u in Equation 5 are over $\mathbb{R}$ and we require the outputs to also be over $\mathbb{R}$ we explicitly cast the complex-valued kernel produced by Proposition 1 to $\mathbb{R}$ without affecting the soundness (details in §A.5).

The DSS layer can be implemented in just a few lines of code and our PyTorch implementation of $\text{DSS}_{\text{SOFTMAX}}$ layer is provided in §A.7 (Figure 6). The implementation of DSS_{EXP} layer is even simpler and is omitted.

Complexity For batch size B, sequence length L and hidden size H, the DSS layer requires $O(NHL)$ time and space to compute the kernels, $O(BHL\log(L))$ time for the discrete convolution and $O(BH^2L)$ time for the output projection. For small batch size B, the time taken to compute the kernels becomes important whereas for large batches more compute is spent on the convolution and the linear projection. The kernel part of DSS layer has $2N + H + 2HN$ real-valued parameters.

3.3 Initialization of DSS layer

The performance of state spaces models is known to be highly sensitive to initialization [GGR22]. In line with the past work, we found that carefully initializing the parameters of the DSS layer is crucial to obtain state-of-the-art performance (§4).

The real and imaginary parts of each element of W are initialized from $\mathcal{N}(0,1)$. Each element of $\Delta_{\log}$ is initialized as e^r where $r \sim \mathcal{U}(\log(.001), \log(.1))$. $\Lambda \in \mathbb{C}^N$ is initialized using eigenvalues of the normal part of normal plus low-rank form of HiPPO matrix [GGR22]. Concretely, $\Lambda_{\text{re}}, \Lambda_{\text{im}}$ are initialized such that the resulting Λ is the vector of those N eigenvalues of the following $2N \times 2N$ matrix which have a positive imaginary part.

$$\begin{cases} (2i+1)^{1/2}(2j+1)^{1/2}/2 & i < j \\ -1/2 & i = j \\ -(2i+1)^{1/2}(2j+1)^{1/2}/2 & i > j \end{cases}$$

Henceforth, we would refer to the above initialization of Λ as *Skew-Hippo* initialization.[6] In all our experiments, we used the above initialization with $N = 64$. The initial learning rate of all DSS parameters was 10^{-3} and weight decay was not applied to them. Exceptions to these settings are noted in §A.3.

3.4 States of DSS via the Recurrent View

In §3, we argued that it can be slow to compute the output of state spaces via Equation 2 and instead leveraged Equation 5 for fast computation. But in situations such as autoregressive decoding during inference, it is more efficient to explicitly compute the states of a state space model similar to a linear RNN (Equation 2). We now show how to compute the states of the DSS models described in §3.1. Henceforth, we assume that we have already computed Λ and Δ.

DSS_{EXP} As stated in Proposition 1(a), DSS_{EXP} computes $\overline{K}_{\Delta,L}(\Lambda,\ (1)_{1\leqslant i\leqslant N},\ \widetilde{w})$. For this state space and sample time Δ, we use Equation 2 to obtain its discretization

$$\overline{A} = \text{diag}(e^{\lambda_1\Delta}, \ldots, e^{\lambda_N\Delta}) \quad , \quad \overline{B} = \left(\lambda_i^{-1}(e^{\lambda_i\Delta} - 1)\right)_{1\leqslant i\leqslant N}$$

where diag creates a diagonal matrix of the scalars. We can now compute the states using the SSM recurrence $x_k = \overline{A}x_{k-1} + \overline{B}u_k$ (Equation 2). As $\overline{A}$ is diagonal, the N coordinates do not interact and hence can be computed independently. Let us assume $x_{-1} = 0$ and say we have already computed x_{k-1}. Then, for the i'th coordinate independently compute

$$x_{i,k} = e^{\lambda_i\Delta}x_{i,k-1} + \lambda_i^{-1}(e^{\lambda_i\Delta} - 1)u_k \ .$$

Note that in DSS_{EXP} , $\text{Re}(\lambda_i) < 0$ and hence $|e^{\lambda_i\Delta}| = |e^{\text{Re}(\lambda_i)\Delta}| < 1$. Intuitively, if $|\lambda_i|\Delta \approx 0$, we would have $x_{i,k} \approx x_{i,k-1}$ and be able to copy history over many timesteps. On the other hand if $\text{Re}(\lambda_i)\Delta \ll 0$ then $x_{i,k} \approx -\lambda_i^{-1}u_k$ and hence the information from the previous timesteps would be forgotten similar to a "forget" gate in LSTMs.

[6] Following our work, [GGGR22] formalized the connection between Skew-Hippo initialization and the original HiPPO matrices and provided a theoretical justification for its empirical properties.

DSS$_{\text{SOFTMAX}}$ As stated in Proposition 1(b), DSS$_{\text{SOFTMAX}}$ computes $\overline{K}_{\Delta,L}(\Lambda, ((e^{L\lambda_i\Delta}-1)^{-1})_{1\leqslant i\leqslant N}, w)$. For this state space and sample time Δ, we obtain the discretization

$$\overline{A} = \text{diag}(e^{\lambda_1\Delta}, \ldots, e^{\lambda_N\Delta}) \quad , \quad \overline{B} = \left(\frac{e^{\lambda_i\Delta}-1}{\lambda_i(e^{\lambda_i\Delta L}-1)}\right)_{1\leqslant i\leqslant N} .$$

For the i'th coordinate we can independently compute

$$x_{i,k} = e^{\lambda_i\Delta}x_{i,k-1} + \frac{u_k(e^{\lambda_i\Delta}-1)}{\lambda_i(e^{\lambda_i\Delta L}-1)} .$$

Let us drop the coordinate index i for clarity to obtain

$$x_k = e^{\lambda\Delta}x_{k-1} + \frac{u_k(e^{\lambda\Delta}-1)}{\lambda(e^{\lambda\Delta L}-1)}$$

where x_{k-1} is now a scalar. As the expression involves the term $e^{\lambda\Delta L}$, where L can be large, directly computing such terms can result in numerical instability and we must avoid exponentiating scalars with a positive real part. We make two cases depending on the sign of $\text{Re}(\lambda)$ and compute x_k via an intermediate state $\widetilde{x}_k$ as follows. Let $p = I[\text{Re}(\lambda) > 0] \in \{0, 1\}$. Then,

$$\widetilde{x}_k = e^{\lambda\Delta(1-p)} \cdot \widetilde{x}_{k-1} + e^{-k\lambda\Delta p} \cdot u_k \quad , \quad x_k = \widetilde{x}_k \cdot \frac{e^{\lambda\Delta p(k-(L-1))}}{\lambda} \cdot \frac{e^{\lambda\Delta(1-2p)}-1}{e^{\lambda\Delta(1-2p)L}-1} .$$

The above equation can be parsed as follows. If $\text{Re}(\lambda) \leqslant 0$ then we have

$$\widetilde{x}_k = e^{\lambda\Delta} \cdot \widetilde{x}_{k-1} + u_k \quad , \quad x_k = \widetilde{x}_k \cdot \frac{(e^{\lambda\Delta}-1)}{\lambda(e^{\lambda\Delta L}-1)}$$

and hence if $\text{Re}(\lambda)\Delta \ll 0$ then $\widetilde{x}_k \approx u_k$ and $x_k \approx u_k/\lambda$. In this case, information from the previous timesteps would be ignored and the information used would be local. On the other hand if $\text{Re}(\lambda) > 0$ then we would have

$$\widetilde{x}_k = \widetilde{x}_{k-1} + e^{-k\lambda\Delta} \cdot u_k \quad , \quad x_k = \widetilde{x}_k \cdot \frac{e^{\lambda\Delta(k-(L-1))}}{\lambda} \cdot \frac{e^{-\lambda\Delta}-1}{e^{-\lambda\Delta L}-1}$$

and hence if $\text{Re}(\lambda)\Delta \gg 0$ then $\widetilde{x}_0 \approx u_0$ and $\widetilde{x}_k \approx \widetilde{x}_{k-1} \approx u_0$, $x_{k<L-1} \approx 0$ and $x_{L-1} \approx u_0/\lambda$. Hence, the model would be able to copy information even from extremely distant positions, allowing it to capture long-range dependencies.

4 Experiments

We evaluate the performance of DSS on sequence-level classification tasks over text, images and audio. Overall, we find its performance is comparable to S4.

Long Range Arena (LRA) LRA [TDA+21] is a standard benchmark for assessing the ability of models to process long sequences. LRA contains 6 tasks with diverse input lengths $1K$-$16K$, encompassing modalities such as text and images. Several Transformer variants have been benchmarked on LRA but all have underperformed due to factors such as their high compute-memory requirements, implicit locality bias and inability to capture long-range dependencies.

Table 1 compares DSS against S4, the Transformer variants reported in [TDA+21], as well as follow-up work. State space models (S4, DSS) shown in Table 1 are left-to-right *unidirectional* whereas other models could be bidirectional.

Despite its simplicity, DSS delivers state-of-the-art performance on LRA. Its performance is comparable to that of S4, with a modest improvement in test accuracy averaged across the 6 tasks (81.88 vs 80.21).[7] In addition, DSS maintains a comfortable 20 point lead over the best performing Transformer variant (81.88 vs 61.41).

[7] The large gap between S4 and DSS on TEXT is due to the use of a larger learning rate for $\Delta_{\log}$ in DSS. For our S4 runs, we decided to use the official hyperparameters as provided by [GGR22].

MODEL	LISTOPS	TEXT	RETRIEVAL	IMAGE	PATHFINDER	PATH-X	AVG
(input length)	2000	2048	4000	1024	1024	16384	
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
H-Transformer-1D	49.53	78.69	63.99	46.05	68.78	✗	61.41
S4 (as in [GGR22])	58.35	76.02	87.09	**87.26**	86.05	**88.10**	80.48
S4 (**our run**)	57.6	75.4	87.6	86.5	**86.2**	88.0	80.21
$\text{DSS}_{\text{SOFTMAX}}$ (ours)	**60.6**	**84.8**	**87.8**	85.7	84.6	87.8	**81.88**
DSS_{EXP} (ours)	59.7	84.6	87.6	84.9	84.7	85.6	81.18
$\text{DSS}_{\text{EXP-NO-SCALE}}$ (ours)	59.3	82.4	86.0	81.2	81.3	✗	65.03

Table 1: (**Long Range Arena**) Accuracy on the full suite of LRA tasks. (*Top*) Transformer variants reported in LRA, (*Middle*) other long-range models reported in the literature, (*Bottom*) state space models.

MODEL	MFCC	RAW
Transformer	90.75	✗
Performer	80.85	30.77
ODE-RNN	65.9	✗
NRDE	89.8	16.49
ExpRNN	82.13	11.6
LipschitzRNN	88.38	✗
CKConv	**95.3**	71.66

MODEL	MFCC	RAW
WaveGAN-D	✗	96.25
S4 (as in [GGR22])	93.96	**98.32**
S4 (**our run**)		98.1
$\text{DSS}_{\text{SOFTMAX}}$ (ours)		97.7
DSS_{EXP} (ours)		98.2
$\text{DSS}_{\text{EXP-NO-SCALE}}$ (ours)		97.7

Table 2: (**Speech Commands (SC)**) Transformer, CTM, RNN, CNN, and SSM models. (*MFCC*) Standard pre-processed MFCC features (length-161). (*Raw*) Unprocessed signals (length-16000). ✗ denotes not applicable or computationally infeasible on a single GPU. All results except ours are reproduced from [GGR22].

Interestingly, despite being less expressive than $\text{DSS}_{\text{SOFTMAX}}$, DSS_{EXP} also reports an impressive performance which makes it specially appealing given its simplicity during training and inference. We investigate an even simpler version of DSS_{EXP} , denoted as $\text{DSS}_{\text{EXP-NO-SCALE}}$, which is same as DSS_{EXP} except that we omit the term $\Lambda^{-1}(e^{\Lambda\Delta} - I)$ in the expression of the kernel shown in Proposition 1(a) and instead compute it as $\widetilde{w} \cdot \text{elementwise-exp}(P)$. As shown in Table 1, the performance of $\text{DSS}_{\text{EXP-NO-SCALE}}$ is generally inferior to that of DSS_{EXP} with the model failing on the challenging PATH-X task which requires the model to capture extremely long-range interactions.

Raw Speech Classification Audio is typically digitized using a high sampling rate resulting in very long sequences. This provides an interesting domain for investigating the abilities of long-range models. We evaluate the performance of DSS on the Speech Commands (SC) dataset [War18], consisting of raw audio samples of length 16000, modeled as a 10-way classification task. As shown in Table 2, the performance of all DSS variants is comparable to that of S4 (98.2 vs 98.1).

In all experiments and ablations, S4 and DSS use identical model hyperparameters such as hidden size, number of layers, etc. Our experimental setup was built on top of the training framework provided by the S4 authors and for our S4 runs we followed their official instructions.[8] Details about model initialization, and hyperparameters are provided in §A.3.

4.1 Analyzing the Performance of DSS

While the experimental results presented above are encouraging, and clearly demonstrate the effectiveness of DSS at modeling long-range dependencies, it is not clear what exactly are the main factors

[8] https://github.com/HazyResearch/state-spaces

MODEL	LISTOPS	TEXT	IMAGE	PATHFINDER	PATH-X	SC
Non-State-Space best	49.5	78.7	47.4	77.8	✗	96.5
$\text{DSS}_{\text{SOFTMAX}}$	**60.6**	**84.8**	**85.7**	**84.6**	**87.8**	**97.7**
$\text{DSS}_{\text{SOFTMAX}}$ (**random init**)	57.6	80.54	72.13	77.5	✗	96.8
$\text{DSS}_{\text{SOFTMAX}}$ (**kernel length** 128)	51.6	75.41	83.8	65.1	✗	96.7
$\text{DSS}_{\text{SOFTMAX}}$ ($\text{argmax}_{k<L}\lvert K_k\rvert$-95-percentile)	89	124	66	87	1269	81

Table 3: *(Top)* Ablation analysis of $\text{DSS}_{\text{SOFTMAX}}$. "Non-State-Space best" is the best performance among models in top and middle sections of Table 1. *(Bottom)* "$\text{argmax}_{k<L}|K_k|$-95-percentile" is the 95th percentile of $\{\text{argmax}_{0\leqslant k<L}|K_k| : K$ is a kernel in $\text{DSS}_{\text{SOFTMAX}}$ $\}$ and is explained in §4.2.

contributing to its performance. To investigate this further, we performed an ablation analysis aimed at answering the following questions:

- How significant is the Skew-Hippo initialization (§3.3) to the model performance? Would initializing Λ randomly work just as well?
- Is the main source of superior performance of state space models (S4, DSS), compared to previous models, their ability to model long-range dependencies? Would restricting DSS to only model local interactions hurt its performance on the above tasks?

Random Initialization To answer the first question, we repeated the experiments after initializing each element of Λ_{re} and Λ_{im} in $\text{DSS}_{\text{SOFTMAX}}$ by randomly sampling from $\mathcal{N}(0, 1)$.

Truncated Kernels To answer the second question, instead of constructing a kernel of length equal to the length L of the input, we restricted the length of the kernel constructed in $\text{DSS}_{\text{SOFTMAX}}$ (Algorithm 1) to 128, significantly shorter than the length of the input. To understand the implication of this restriction recall Equation 5 which states that $y_k = \sum_{j=0}^{k} \overline{K}_j \cdot u_{k-j}$.

For a given context size $c = 128$, restricting $\overline{K}_{\geqslant c} = 0$ would imply

$$y_k = \sum_{j=0}^{\min(k,c-1)} \overline{K}_j \cdot u_{k-j}$$

and hence the output y_k at position k would only depend on $u_k, \ldots, u_{k-c+1}$. This would restrict each $\text{DSS}_{\text{SOFTMAX}}$ layer to only model local interactions and the model would require several layers to have a broader receptive field and capture long-range interactions.

As shown in Table 3, randomly initializing the Λ parameters of DSS leads to significant performance degradation on the majority of tasks, with the model failing to perform on PATH-X. This is inline with the findings of [GGR22] who also reported the initialization of S4 to be critical to its performance. Interestingly, despite this performance reduction, DSS manages to outperform all non state-space-based models on every task.

Truncating the length of the kernel also leads to a significant reduction in performance across most tasks (Table 3), suggesting that the superior performance of state-space models on these tasks can indeed be partly attributed to their ability to capture long-range dependencies. Moreover, on some tasks such as LISTOPS and IMAGE, using a truncated kernel still manages to outperform all Transformer variants, which is surprising as Transformer layers are known to be effective at capturing interactions at such short ranges.

4.2 Analysis of Learned DSS Parameters

To further explore the inner workings of DSS, we visually inspected the trained parameters and kernels of $\text{DSS}_{\text{SOFTMAX}}$.

The kernels of all layers of the trained $\text{DSS}_{\text{SOFTMAX}}$ are shown in Figure 2 and reveal a stark contrast between the tasks. On the tasks IMAGE and SC, for almost all kernels, the absolute values of the first 128 positions are significantly higher than the later positions indicating that these kernels are mostly local. On the other hand for PATH-X, for a significant proportion of kernels the opposite is true, indicating that these kernels are modeling long-range interactions.

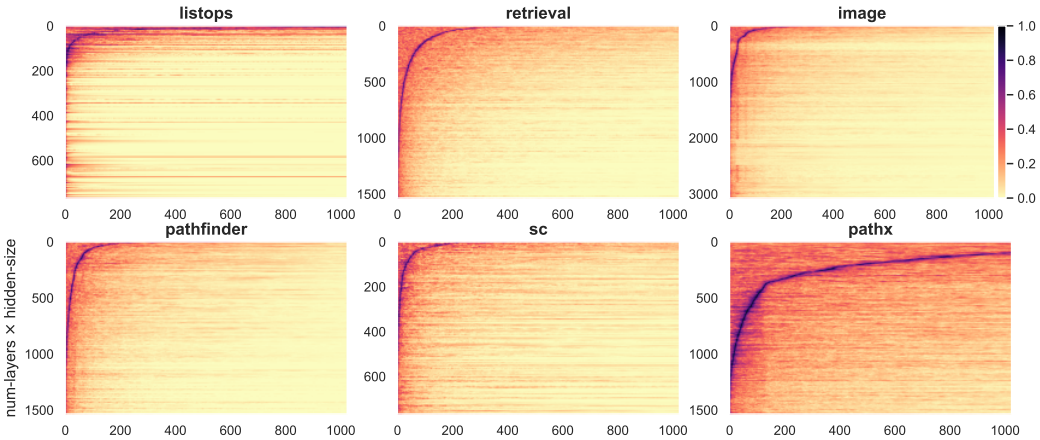

Figure 2: Kernels of trained $\text{DSS}_{\text{SOFTMAX}}$. Each row of pixels corresponds to one of the (number-of-layers $\times$ H) kernels. For a row (kernel) $K \in \mathbb{R}^L$ the element $K_k \in \mathbb{R}$ is shown as $|K_k|/||K||_\infty$ to enable visualization. To enable comparison across tasks, only the first 1024 positions are shown.

This partially explains why the performance of $\text{DSS}_{\text{SOFTMAX}}$ on IMAGE and SC does not decrease significantly even after limiting the kernel length to 128, whereas PATHFINDER and PATH-X report significant degradation (Table 3). The last row of Table 3 shows the 95th percentile of the set $\{\text{argmax}_{0 \leqslant k < L} |K_k| : K \text{ is a kernel in } \text{DSS}_{\text{SOFTMAX}}\}$, i.e., the set of positions over all kernels, at which for a given kernel its absolute value is maximized. As expected, most kernels of IMAGE and SC are local whereas for PATH-X they are mostly long range.

The values of the real and imaginary parts of Λ are shown in Figure 3 (and §A.6, Figure 5). We observe that, although all elements of Λ_{re} are initialized as -0.5, their values can change significantly during training and can become positive (see plots for LISTOPS, SC). It can also be observed that a λ with a more negative $\text{Re}(\lambda)$ generally tends to have a larger $\text{Im}(\lambda)$, which interestingly is a property not satisfied by Skew-Hippo initialization.

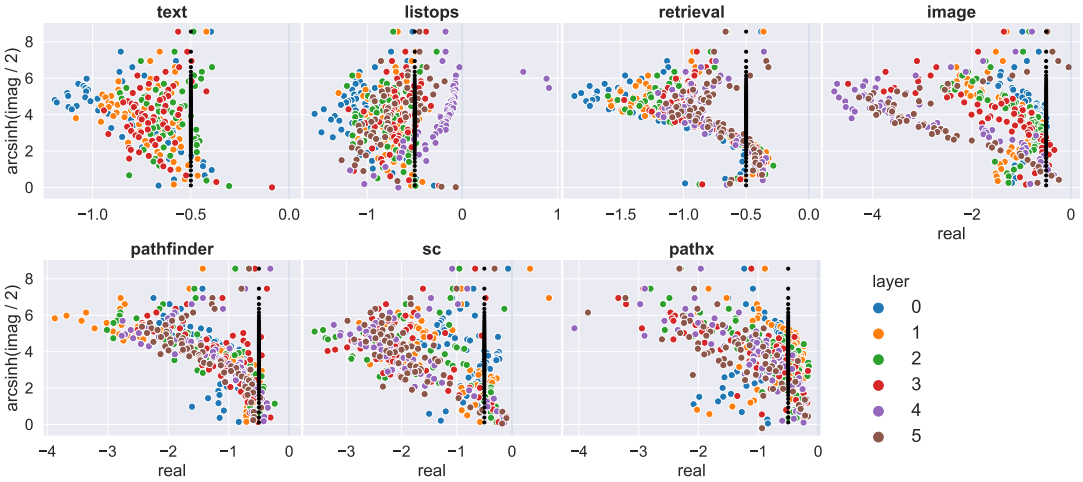

Figure 3: Values of Λ in trained $\text{DSS}_{\text{SOFTMAX}}$ for tasks described in §4. For $\lambda = x + iy$, we show y on log-scale for better visualization by plotting λ as $(x, \text{arcsinh}(y/2)) = (x, \log(y/2 + \sqrt{(y/2)^2 + 1}))$. Black dots correspond to Skew-Hippo initialization (§3.3).

The values of the trained $\Delta_{\log}$ corresponding to the real and imaginary parts of Λ are shown in Figure 4 and, in this case as well, change significantly during training. The values of $\Delta_{\log}$ on short-range tasks such as IMAGE and SC generally tend to be larger compared to long-range tasks such as PATHFINDER and PATH-X, inline with the intuition that, when $\text{Re}(\lambda) < 0$, smaller values of Δ correspond to modeling long-range dependence (§3.4).

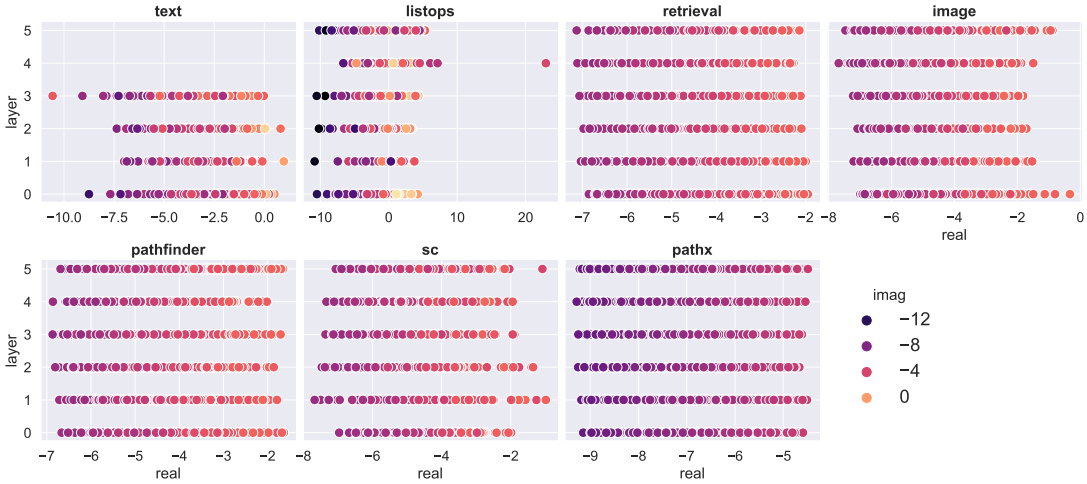

Figure 4: Trained $\Delta_{\log}$ in $\text{DSS}_{\text{SOFTMAX}}$ for tasks described in §4. As noted in §A.3 we used separate $\Delta_{\log,\text{re}}$, $\Delta_{\log,\text{im}} \in \mathbb{R}$ parameters to respectively scale the real and imaginary parts of Λ.

We note that the plot for LISTOPS reveals an outlier with a value of 22 which after exponentiation in Algorithm 1 would result in an extreme large Δ. This can potentially lead to training instabilities and we plan to address this issue in future work.

Limitations and future work In this work, we evaluated DSS on sequence-level classification tasks. In future work, we plan to include token-level generation tasks such as language modeling, forecasting, etc. Another important future direction is to pretrain DSS-based models on large amounts of raw data. Lastly, we found that the initialization and learning rates of DSS parameters Λ_{re}, Λ_{im}, $\Delta_{\log}$ (§3.3) play an important role in the performance and convergence of the model. Informally, for tasks such as PATH-X that require very long-range interactions, a smaller initialization of $\Delta_{\log}$ was beneficial. An in-depth analysis of this phenomenon could be helpful and remains for future work.

Acknowledgments and Disclosure of Funding

We thank Ramon Fernandez Astudillo for carefully reviewing the preliminary draft and suggesting several helpful edits. We thank Omer Levy, Achille Fokoue and Luis Lastras for their support. Our experiments were conducted on IBM's Cognitive Computing Cluster, with additional resources from Tel Aviv University. This research was supported by (1) IBM AI Residency program and (2) Defense Advanced Research Projects Agency (DARPA) through Cooperative Agreement D20AC00004 awarded by the U.S. Department of the Interior (DOI), Interior Business Center, and (3) the European Research Council (ERC) under the European Union Horizons 2020 research and innovation programme (grant ERC DELPHI 802800).

References

[BMH+22] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR, 17–23 Jul 2022.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott

Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 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.

[BPC20] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *ArXiv preprint*, abs/2004.05150, 2020.

[BZMA20] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. 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.

[CFG+21] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the gpu, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6, 2021.

[CLD+21] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[DXX18] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pages 5884–5888. IEEE, 2018.

[Gaj03] Zoran Gajic. *Linear dynamic systems and signals*. Prentice Hall/Pearson Education Upper Saddle River, 2003.

[GB20] Ankit Gupta and Jonathan Berant. GMAT: Global memory augmentation for transformers. *ArXiv preprint*, abs/2006.03274, 2020.

[GDE+20] 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.

[GDG+21] Ankit Gupta, Guy Dar, Shaya Goodman, David Ciprut, and Jonathan Berant. Memory-efficient transformers via top-k attention. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, pages 39–52, Virtual, November 2021. Association for Computational Linguistics.

[GGDR22] Karan Goel, Albert Gu, Chris Donahue, and Christopher Re. It's raw! Audio generation with state-space models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7616–7633. PMLR, 17–23 Jul 2022.

[GGGR22] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *ArXiv*, abs/2206.11893, 2022.

[GGR22] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022.

[GJG+21] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state-space layers. *Advances in neural information processing systems*, 34, 2021.

[HG16] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *ArXiv preprint*, abs/1606.08415, 2016.

[KKL20] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[KVPF20] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 2020.

[QML+20] Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2555–2565, Online, 2020. Association for Computational Linguistics.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[RPG+21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 2021.

[RSVG21] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

[SSI+22] Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, and Omer Levy. Scrolls: Standardized comparison over long language sequences. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2022.

[TDA+21] 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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[TDBM20] Yi Tay, M. Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ArXiv*, abs/2009.06732, 2020.

[VKE19] Aaron Voelker, Ivana Kajic, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15544–15553, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[War18] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *ArXiv*, abs/1804.03209, 2018.

[WLK+20] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768, 2020.

[ZS21] Zhenhai Zhu and Radu Soricut. H-transformer-1D: Fast one-dimensional hierarchical attention for sequences. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3801–3815, Online, 2021. Association for Computational Linguistics.

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 end of §4.2.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (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 §A.1.
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] See §1, §A.3.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See §A.3.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (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] See §A.3.
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? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (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]
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 Supplemental Material

A.1 Diagonal State Spaces

We restate Proposition 1 for convenience.

Proposition. *Let $K \in \mathbb{C}^{1\times L}$ be the kernel of length L of a given state space (A, B, C) and sample time $\Delta > 0$, where $A \in \mathbb{C}^{N\times N}$ is diagonalizable over $\mathbb{C}$ with eigenvalues $\lambda_1, \ldots, \lambda_N$ and $\forall i$, $\lambda_i \neq 0$ and $e^{L\lambda_i\Delta} \neq 1$. Let $P \in \mathbb{C}^{N\times L}$ be $P_{i,k} = \lambda_i k\Delta$ and Λ be the diagonal matrix with $\lambda_1, \ldots, \lambda_N$. Then there exist $\widetilde{w}, w \in \mathbb{C}^{1\times N}$ such that*

$$\text{(a)}\quad K = \overline{K}_{\Delta,L}(\Lambda,\ (1)_{1\leqslant i\leqslant N},\ \widetilde{w}) = \widetilde{w}\cdot\Lambda^{-1}(e^{\Lambda\Delta}-I)\cdot\text{elementwise-exp}(P),$$

$$\text{(b)}\quad K = \overline{K}_{\Delta,L}(\Lambda,\ ((e^{L\lambda_i\Delta}-1)^{-1})_{1\leqslant i\leqslant N},\ w) = w\cdot\Lambda^{-1}\cdot\text{row-softmax}(P).$$

Proof. Let A be diagonalizable over $\mathbb{C}$ as $A = V\Lambda V^{-1}$ with eigenvalues $\lambda_1, \ldots, \lambda_N \in \mathbb{C}$. From Equation 4 we have

$$K = (\ Ce^{A\cdot k\Delta}(e^{A\Delta}-I)A^{-1}B\ )_{0\leqslant k<L},$$

where

$$K_k = Ce^{A\cdot k\Delta}(e^{A\Delta}-I)A^{-1}B = (CV)e^{\Lambda k\Delta}(e^{\Lambda\Delta}-I)\Lambda^{-1}(V^{-1}B)\ .$$

For $CV \in \mathbb{C}^{1\times N}$ and $V^{-1}B \in \mathbb{C}^{N\times 1}$ let $(CV)^\top * (V^{-1}B) = \widetilde{w} \in \mathbb{C}^N$ be the element-wise product of CV and $V^{-1}B$. Then,

$$K_k = \sum_{i=1}^{N} \frac{e^{\lambda_i k\Delta}(e^{\lambda_i\Delta}-1)}{\lambda_i}\cdot\widetilde{w}_i \tag{7}$$

$$= \sum_{i=1}^{N} \frac{e^{\lambda_i k\Delta}(e^{\lambda_i\Delta}-1)}{\lambda_i(e^{L\lambda_i\Delta}-1)}\cdot((e^{L\lambda_i\Delta}-1)\widetilde{w}_i) \tag{8}$$

$$= \sum_{i=1}^{N}(\widetilde{w}_i\cdot(e^{L\lambda_i\Delta}-1))\cdot\frac{1}{\lambda_i}\cdot\frac{e^{\lambda_i k\Delta}}{(\sum_{r=0}^{L-1}e^{r\lambda_i\Delta})} \tag{9}$$

where the last equality follows from $(z^L-1) = (z-1)(z^0+\ldots+z^{L-1})$ and using $z^L \neq 1$.

Let $P \in \mathbb{C}^{N\times L}$ be the matrix $P_{i,k} = \lambda_i\cdot k\Delta$ and let $E = \text{elementwise-exp}(P)$. It is easy to verify that Equation 7 can be re-written as a vector-matrix product as

$$K = \widetilde{w}\cdot\Lambda^{-1}(e^{\Lambda\Delta}-I)\cdot E\ .$$

Similarly, for the state space $(\Lambda,\ (1)_{1\leqslant i\leqslant N},\ \widetilde{w}))$ and sample time Δ its kernel $\widetilde{K}$ can be obtained from Equation 4 as

$$\widetilde{K}_k = \widetilde{w}\cdot e^{\Lambda\cdot k\Delta}(e^{\Lambda\Delta}-I)\Lambda^{-1}\cdot[1,\ldots,1]_{N\times 1}$$
$$= \sum_{i=1}^{N}\widetilde{w}_i\cdot\frac{e^{\lambda_i k\Delta}(e^{\lambda_i\Delta}-1)}{\lambda_i}$$

which is also the expression for K_k (Equation 7). This proves part (a) and we now consider part (b). Let $w \in \mathbb{C}^N$ be defined as

$$w_i = \widetilde{w}_i\cdot(e^{L\lambda_i\Delta}-1).$$

Then from Equation 9,

$$K_k = \sum_{i=1}^{N} w\cdot\frac{1}{\lambda_i}\cdot\frac{e^{\lambda_i k\Delta}}{(\sum_{r=0}^{L-1}e^{r\lambda_i\Delta})}\ . \tag{10}$$

Let $S = \text{row-softmax}(P)$ denote the matrix obtained after applying softmax on the rows of P, i.e.

$$S_{i,k} = \frac{e^{\lambda_i k\Delta}}{\sum_{r=0}^{L-1}e^{r\lambda_i\Delta}}.$$

It is easy to verify that Equation 10 can be expressed as a vector-matrix product

$$K = w \cdot \Lambda^{-1} \cdot S \,.$$

Similarly, for the state space $(\Lambda,\ ((e^{L\lambda_i\Delta}-1)^{-1})_{1\leqslant i\leqslant N},\ w))$ and sample time Δ its kernel $\widehat{K}$ can be obtained from Equation 4 as

$$\begin{aligned}\widehat{K}_k &= w \cdot e^{\Lambda\cdot k\Delta}(e^{\Lambda\Delta}-I)\Lambda^{-1}\cdot[\ldots,(e^{L\lambda_i\Delta}-1)^{-1},\ldots]_{N\times 1}\\ &= \sum_{i=1}^{N} w_i \cdot \frac{e^{\lambda_i k\Delta}(e^{\lambda_i\Delta}-1)}{\lambda_i(e^{L\lambda_i\Delta}-1)}\\ &= \sum_{i=1}^{N} \widetilde{w}_i \cdot \frac{e^{\lambda_i k\Delta}(e^{\lambda_i\Delta}-1)}{\lambda_i}\end{aligned}$$

which is also the expression for K_k (Equation 7). □

A.2 Numerically Stable softmax

As noted in §3.1, softmax can have singularities over $\mathbb{C}$. To address this issue, we use a simple correction to make it well-defined over the entire domain:

- softmax : Given $(x_0,\ldots,x_{L-1}) = x \in \mathbb{C}^L$, let $\mathrm{softmax}(x) \in \mathbb{C}^L$ be defined as $(\mathrm{softmax}(x))_k = e^{x_k}(e^{x_0}+\ldots+e^{x_{L-1}})^{-1}$. Note that for any $c \in \mathbb{C}$, $\mathrm{softmax}(x_0,\ldots,x_{L-1}) = \mathrm{softmax}(x_0-c,\ldots,x_{L-1}-c)$. Unlike over $\mathbb{R}$, softmax can have singularities over $\mathbb{C}$ as sum of exponentials can vanish. E.g. $e^0+e^{i\pi}=0$ and hence $\mathrm{softmax}(0,i\pi)$ is not defined.
- max : Given $(x_0,\ldots,x_{L-1}) = x \in \mathbb{C}^L$, let $\max(x)$ be the x_i with the maximum real part, i.e. $x_{\mathrm{argmax}_i \mathrm{Re}(x_i)}$.
- $\mathrm{reciprocal}_\epsilon$: Given $x \in \mathbb{C}$ and $\epsilon \in \mathbb{R}_{>0}$, let $\mathrm{reciprocal}_\epsilon(x) = \frac{\overline{x}}{x\cdot\overline{x}+\epsilon}$ where $\overline{x}$ is the complex conjugate of x. The denominator is always in $\mathbb{R}_{\geqslant\epsilon}$ and $|\mathrm{reciprocal}_\epsilon| \leqslant (2\sqrt{\epsilon})^{-1}$.
- $\mathrm{softmax}_\epsilon$: Given $(x_0,\ldots,x_{L-1}) = x \in \mathbb{C}^L$ let $m = \max(x)$ and $\widetilde{x}_i = x_i - m$. Note that $|e^{\widetilde{x}_i}| \leqslant 1$. Given $\epsilon \in \mathbb{R}_{>0}$, let $\mathrm{softmax}_\epsilon(x) \in \mathbb{C}^L$ be

  $$(\mathrm{softmax}_\epsilon(x))_k = e^{\widetilde{x}_k}\cdot \mathrm{reciprocal}_\epsilon\left(\sum_{r=0}^{L-1} e^{\widetilde{x}_r}\right).$$

  $\mathrm{softmax}_\epsilon$ is always bounded and differentiable.

In our implementation, we use $\mathrm{softmax}_\epsilon$ with $\epsilon = 10^{-7}$.

SSM Softmax In our current implementation of $\mathrm{softmax}(x)$ (Figure 6), we exploit the specific structure of x that arises in Algorithm 1. We now describe an alternate method based on FFT which also uses this specific structure and might lead to a faster implementation in the future.

Claim 1 (SSM Softmax)**.** *Given $c \in \mathbb{C}$, let $p = I[\mathrm{Re}(c) > 0]$, $n = 1-p$, $e = \exp(c\cdot(n-p))$ and $r = (n-pe)/(p-ne)$. Let $\omega = \exp(-2\pi i/L)$ where $i = \sqrt{-1}$. Then,*

$$\begin{aligned}\mathrm{softmax}(c\cdot 0,\ldots,c\cdot(L-1)) &= \mathrm{inverseFFT}\left(\frac{1-e}{n-pe+(p-ne)\omega^k}\right)_{0\leqslant k<L}\\ &= \mathrm{inverseFFT}\left(\frac{r+1}{r+\omega^k}\right)_{0\leqslant k<L}.\end{aligned}$$

Proof. There are 2 cases depending on sign of $\mathrm{Re}(c)$.

Case 1 ($p=0, n=1$): In this case we have $e = \exp(c)$. For the map

$$F(z) = \frac{1-e}{1-e^L}\sum_{k=0}^{L-1}(ez)^k = \frac{(1-e)(1-(ez)^L)}{(1-e^L)(1-ez)}$$

we get the coefficients of $F(z)$ as

$$\text{invFFT}(F(\omega^k)_{0\leqslant k<L}) \;=\; \left(\frac{(1-e)}{(1-e^L)}e^k\right)_{0\leqslant k<L} \;=\; \text{softmax}(c\cdot 0,\ldots,c\cdot(L-1)).$$

We have,

$$F(\omega^k) \;=\; \frac{(1-e)(1-(e\omega^k)^L)}{(1-e^L)(1-e\cdot\omega^k)} \;=\; \frac{1-e}{1-e\cdot\omega^k}$$

where last equality follows from $\omega^L = 1$.

Case 2 ($p = 1, n = 0$): In this case we have $e = \exp(-c)$. For the map

$$\begin{aligned} F(z) \;&=\; \frac{1-e}{1-e^L}\sum_{k=0}^{L-1} e^k z^{L-1-k} \;=\; \frac{(1-e)z^{L-1}}{1-e^L}\sum_{k=0}^{L-1}\left(\frac{e}{z}\right)^k \\ &=\; \frac{(1-e)z^{L-1}}{1-e^L}\frac{1-\left(\frac{e}{z}\right)^L}{1-\frac{e}{z}} \;=\; \frac{(1-e)}{(1-e^L)}\frac{(z^L-e^L)}{(z-e)} \end{aligned}$$

we get the coefficients of $F(z)$ as

$$\begin{aligned} \text{invFFT}(F(\omega^k)_{0\leqslant k<L}) \;&=\; \left(\frac{(1-e)}{1-e^L}e^{L-1-k}\right)_k \\ =\; \left(\frac{(e^{-1}-1)}{(e^{-1})^L-1}(e^{-1})^k\right)_{0\leqslant k<L} \;&=\; \text{softmax}(c\cdot 0,\ldots,c\cdot(L-1)) \end{aligned}$$

as $e^{-1} = \exp(c)$. Moreover, we have

$$F(\omega^k) \;=\; \frac{(1-e)(\omega^{k\cdot L}-e^L)}{(1-e^L)(\omega^k-e)} \;=\; \frac{1-e}{-e+\omega^k}$$

where last equality follows from $\omega^L = 1$.

Finally, the second equality of the main Claim follows from $1 - e = n - pe + p - ne$. □

The computation of softmax in Claim 1 is numerically stable and we always exponentiate scalars with a negative real part. The computed function has singularities at $c \in \{-2\pi ik/L, 0 \leqslant k < L\}$.

A.3 Experimental Setup

We now describe the training details for DSS and S4 on LRA and Speech Commands (§4).

Sequence Classification Head: Both LRA and Speech Commands are sequence classification tasks. The final layer of the DSS stack outputs a sequence which is aggregated into a single vector via mean pooling along the length dimension. Exceptions to this were TEXT and PATHFINDER tasks where the rightmost token was used as the aggregate.

We used a separate $\Delta_{\text{log,re}}, \Delta_{\text{log,im}} \in \mathbb{R}$ parameters to respectively scale the real and imaginary parts of Λ. I.e. for a given $\Lambda \in \mathbb{C}^N$, we computed $\Delta * \Lambda$ as $\exp(\Delta_{\text{log,re}})\Lambda_{\text{re}} + i\cdot\exp(\Delta_{\text{log,im}})\Lambda_{\text{im}}$.

For all datasets, we used AdamW optimizer with a constant learning rate schedule with decay on validation plateau. However, for the DSS parameters (§3.2) initial learning rate was 10^{-3} and weight decay was not used, with a few exceptions noted below.

We used hyperparameters such as model sizes, number of update steps, etc as recommended by the S4 authors on their official repository and are listed in Table 4. We made the following exceptions for DSS trainings:

- LISTOPS: learning rate of Δ_{log} was 0.02 instead of 10^{-3}.
- TEXT: learning rate of Δ_{log} 0.02 instead of 10^{-3}.
- IMAGE: we used seed 0 and trained for 200 epochs instead of 100.

	Depth	Features H	Norm	Pre-norm	Dropout	LR	Batch Size	Epochs	WD	Patience
ListOps	6	128	BN	False	0	0.01	50	50	0.01	5
Text	4	128	BN	True	0	0.01	50	40	0	10
Retrieval	6	256	BN	True	0	0.002	64	25	0	20
Image	6	512	LN	False	0.2	0.004	50	200	0.01	10
Pathfinder	6	256	BN	True	0.1	0.004	100	200	0	10
Path-X	6	256	BN	True	0.0	0.0005	32	100	0	40
Speech Commands (Raw)	6	128	BN	True	0.1	0.01	20	200	0	20

Table 4: Hyperparameters for the S4 and DSS models. Exceptions for DSS are detailed in §A.3. (Top) LRA and (Bottom) Speech Commands. LR is initial learning rate and WD is weight decay. BN and LN refer to Batch Normalization and Layer Normalization.

- PATHFINDER: we used Patience $= 13$.
- PATH-X: we used batch size 16 and trained for 35 epochs. $\Delta_{\log}$ was initialized as e^r where $r \sim \mathcal{U}(\log(.0001), \log(.01))$ and its learning rate was 10^{-4}. This was beneficial in early convergence of the model.

For our experiments, the test accuracy that we report in §4 was measured at the checkpoint with the highest validation accuracy.

All our experiments were conducted on a single A100 GPU (40GiB).

A.4 Benchmarking Running Times of DSS and S4

We compared the running time of DSS with that of the official S4 implementation on a single NVIDIA 3090 GPU. Implementations of both S4 and DSS versions below utilize the PyKeOps library for memory efficiency [CFG+21].

- Running times on the PATH-X task with input length $L = 16384$ are summarized in Table 5. For large batches the time taken to perform the FFT-based convolution and the feedforward part dominates and hence the speedups of DSS over S4 are less pronounced. For smaller B, the time taken to compute the kernel becomes significant and DSS provides upto $1.8\times$ speedup over S4.
- We also isolated and benchmarked the time taken to solely compute the S4 kernel vs the DSS kernel (including the time to perform a backward pass on the sum of all kernel entries). As summarized in Table 6, for large L, DSS kernel is more than $2\times$ faster than S4 kernel.

B	1	16
S4	133	470
DSS_{EXP}	72	400
$\text{DSS}_{\text{SOFTMAX}}$	75	410

Table 5: Time (msec) taken for a single gradient update on PATH-X using hyperparameters in Table 4 and batch size B.

L	4096	16384	65536
S4	4.3	10.8	39.7
DSS_{EXP}	4.2	7.0	18.1
$\text{DSS}_{\text{SOFTMAX}}$	4.3	6.9	17.8

Table 6: Time (msec) taken to compute the kernel of length L for $N = 64$, $H = 256$.

A.5 Additional Remarks

Casting the kernel to $\mathbb{R}$ In this work, we require the inputs u and outputs y to always be over $\mathbb{R}$, i.e., we assume Equation 5 to be with an explicit casting operation $y_k = \mathrm{Re}(\sum_{j=0}^{k} \overline{K}_j \cdot u_{k-j})$. For u over $\mathbb{R}$, this further implies $y_k = \sum_{j=0}^{k} \mathrm{Re}(\overline{K}_j) \cdot u_{k-j}$. Hence, we can explicitly cast the complex-valued kernel produced by Proposition 1 to $\mathbb{R}$ during training (Algorithm 1, final step) and cast the outputs over $\mathbb{C}$ to $\mathbb{R}$ during inference in §3.4.

A.6 Learned Parameters of $\text{DSS}_{\text{SOFTMAX}}$

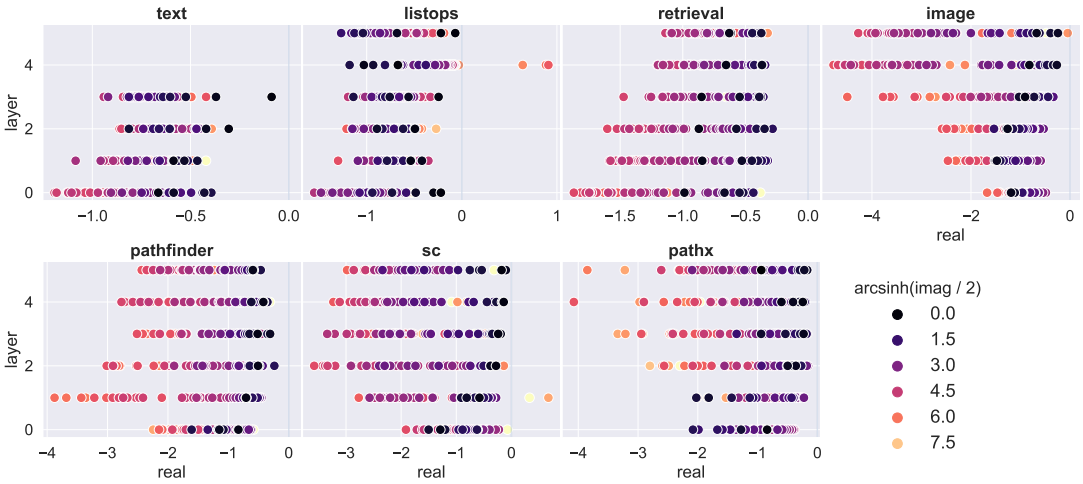

Figure 5: Trained Λ in $\text{DSS}_{\text{SOFTMAX}}$ for tasks described in §4.

A.7 Implementation of $\text{DSS}_{\text{SOFTMAX}}$

```
def reciprocal(x, epsilon=1e-7):
    x_conj = x.conj()       # conjugate
    return x_conj / (x*x_conj + epsilon)

def dss_kernel(L):
    # L: kernel length
    # Lambda: [N 2],   log_dt: [H],   W: [H N 2]  (floats)
    Lambda, log_dt, W = get_layer_parameters()
    # complex parameter stored as 2 floats denoting real,
    # imaginary parts as ADAM moments are non-linear

    # convert reals to complex
    Lambda, W = map(torch.view_as_complex, (Lambda, W))      # [N], [H N]
    dt_Lambda = log_dt.exp().unsqueeze(-1) * Lambda          # [H L]
    pos = torch.arange(L, device=W.device)                   # [L]
    P = dt_Lambda.unsqueeze(-1) * pos                        # [H N L]

    # fast softmax using structure of P
    Lambda_gt_0 = Lambda.real > 0                            # [N]
    if Lambda_gt_0.any():
        with torch.no_grad():
            P_max = dt_Lambda * (Lambda_gt_0 * (L-1))        # [H N]
        P = P - P_max.unsqueeze(-1)
    S = P.exp()                                              # [H N L]
    dt_Lambda_neg = dt_Lambda * (1 - 2*Lambda_gt_0)          # [H N]
    # 1 / S.sum(-1) == num / den
    num = dt_Lambda_neg.exp() - 1                            # [H N]
    den = (dt_Lambda_neg * L).exp() - 1                      # [H N]
    W = W * num * reciprocal(den * Lambda)                   # [H N]

    # mixture of softmaxes
    return torch.einsum('hn,hnl->hl', W, S).real             # [H L]

def state_space(u):
    # u: batch of input sequences
    # B: batch size, H: hidden size, L: sequence length
    B, H, L = u.shape
    # compute state space kernel for each of H coordinates
    K = dss_kernel(L)                                        # [H L]
    # multiply two degree L-1 polynomials
    # (u0 + u1*z ... uL-1*z^L-1)(K0 + K1*z ... KL-1*z^L-1)
    # zero-pad them to degree 2L-1 to avoid wrap-around
    K_f = torch.fft.rfft(K, n=2*L)                           # [H L+1]
    u_f = torch.fft.rfft(u, n=2*L)                           # [B H L+1]
    y_f = K_f * u_f                                          # [B H L+1]
    y = torch.fft.irfft(y_f, n=2*L)[..., :L]                 # [B H L]
    # yi = ui*K0 + ... u0*Ki
    # residual connection, non-linearity, output projection not shown
    return y
```

Figure 6: Core implementation of $\text{DSS}_{\text{SOFTMAX}}$ layer (§3.2) in PyTorch.