Resurrecting Recurrent Neural Networks for Long Sequences

Antonio Orvieto[1,+], **Samuel L Smith**[2], **Albert Gu**[2], **Anushan Fernando**[2], **Caglar Gulcehre**[2], **Razvan Pascanu**[2] **and Soham De**[2]
[1]ETH Zurich, [2]DeepMind, [+]Work done at DeepMind.

Recurrent Neural Networks (RNNs) offer fast inference on long sequences but are hard to optimize and slow to train. Deep state-space models (SSMs) have recently been shown to perform remarkably well on long sequence modeling tasks, and have the added benefits of fast parallelizable training and RNN-like fast inference. However, while SSMs are superficially similar to RNNs, there are important differences that make it unclear where their performance boost over RNNs comes from. In this paper, we show that careful design of deep RNNs using standard signal propagation arguments can recover the impressive performance of deep SSMs on long-range reasoning tasks, while also matching their training speed. To achieve this, we analyze and ablate a series of changes to standard RNNs including linearizing and diagonalizing the recurrence, using better parameterizations and initializations, and ensuring proper normalization of the forward pass. Our results provide new insights on the origins of the impressive performance of deep SSMs, while also introducing an RNN block called the Linear Recurrent Unit that matches both their performance on the *Long Range Arena* benchmark and their computational efficiency.

1. Introduction

Recurrent neural networks (RNNs) have played a central role since the early days of deep learning, and are a natural choice when modelling sequential data (Elman, 1990; Hopfield, 1982; McCulloch and Pitts, 1943; Rumelhart et al., 1985). However, while these networks have strong theoretical properties, such as Turing completeness (Chung and Siegelmann, 2021; Kilian and Siegelmann, 1996), it is well-known that they can be hard to train in practice. In particular, RNNs suffer from the vanishing and exploding gradient problem (Bengio et al., 1994; Hochreiter, 1991; Pascanu et al., 2013), which makes it difficult for these models to learn about the long-range dependencies in the data. Several techniques were developed that attempt to mitigate this issue, including orthogonal/unitary RNNs (Arjovsky et al., 2016; Helfrich et al., 2018), and gating mechanisms such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs) (Cho et al., 2014a). Nonetheless, these models are still slow to optimize due to the inherently sequential nature of their computation (Kalchbrenner et al., 2016), and are therefore hard to scale.

In recent years, Transformers (Vaswani et al., 2017) have gained increasing prominence for sequence modelling tasks, achieving remarkable success in a wide range of applications (Brown et al., 2020; Dosovitskiy et al., 2020; Jumper et al., 2021). Compared to RNNs, attention layers are easier to scale and parallelize during training, and crucially they do not suffer from the vanishing gradient problem, since the interaction between any two tokens in the sequence is modeled by direct edges in the network. A key issue with attention layers however is that their computational and memory costs scale quadratically as $O(L^2)$ with the sequence length L. Transformers can therefore be especially expensive to deploy on long sequences. RNNs, which scale linearly with the sequence length, are therefore typically faster than transformers at inference time even for modest sequence lengths (Liu et al., 2019).

Motivated by these problems, Gu et al. (2021a) recently introduced the S4 model, a carefully designed deep state-space model (SSM) achieving remarkable performance on tasks from the Long Range Arena (LRA) (Tay et al., 2020), a benchmark explicitly designed to require very long-ranged reasoning. S4 is theoretically principled and inspired by continuous-time linear SSMs; well-established components of modern control systems. More importantly, the S4 layer and its variants (DSS, S4D, S5, etc) (Gu et al., 2022a; Gupta et al., 2022a; Smith et al., 2022) overcome the $O(L^2)$ bottleneck of attention layers by modeling interactions between

Corresponding author(s): antonio.orvieto@inf.ethz.ch, sohamde@deepmind.com

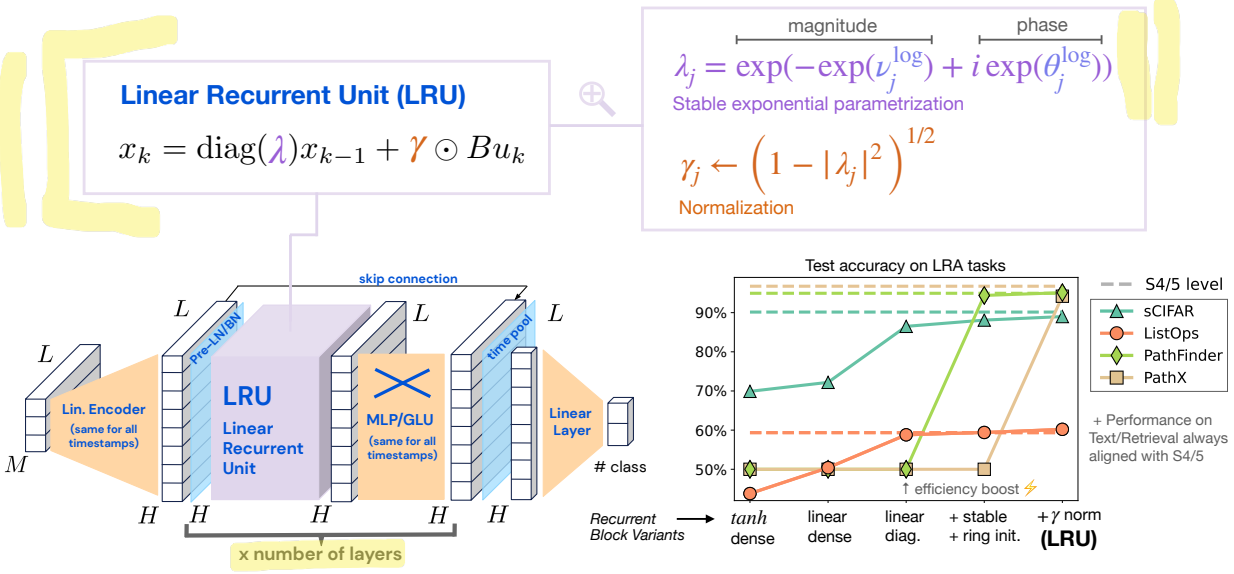

Figure 1 | ***(Left)*** *Deep Linear Recurrent Unit (LRU) architecture introduced in this paper, inspired by S4 (Gu et al., 2021a). The model is a stack of LRU blocks, with nonlinear projections in between, and also uses skip connections and normalization methods like batch/layer normalization. We expand on the details in §D and provide pseudocode in §A. We also use the same architecture structure (Norm-Recurrence-GLU-Skip) for every variant of the recurrent module in our study (*tanh *dense, linear dense, etc..).* ***(Right)*** *Summary of effects for the main steps outlined in the introduction towards designing LRUs starting from* tanh *RNNs. Shown is the average performance (3 seeds) of the recurrent module at each step on the Long Range Arena (LRA), compared to average performance of deep SSMs. For all LRA tasks, we match the performance of deep SSMs like S4/S4D/S5 with LRUs. Detailed results in §3.*

tokens using a hidden state (like RNNs) under proper discretization techniques. These models can be made very efficient at inference time by simply unrolling the layer like an RNN. Futhermore, since SSMs are linear in the temporal dimension, they are easily parallelizable during training, in contrast to the slow sequential nature of training a typical RNN. This makes them very computationally efficient on long sequences.

While the S4 model is equivalent to an RNN during inference, it has a number of unique characteristics during training. For example, S4 is parameterized as a discretization of a latent continuous-time system of differential equations. S4 also uses specific initializations of the state matrices motivated from the theory of polynomial projections (Gu et al., 2020). While these characteristics might seem to motivate the impressive performance of these models, later works (Gu et al., 2022a; Gupta et al., 2022a,b; Smith et al., 2022) have suggested that the specific initialization used by S4 is often not crucial for performance, and that the discretization rules which achieve best performance may deviate from theory (Smith et al., 2022). It is therefore unclear what these unique characteristics of the deep SSMs are doing mechanistically, and how they can be simplified.

Motivated by the striking similarities between RNNs and deep SSMs, and in an attempt to better understand the underlying mechanism driving the performance of these models, we study the power and limitations of RNNs when used as core components of deep architectures for long-range reasoning. Our main goal is to answer the question:

> *"Can we match the performance and efficiency of deep continuous-time SSMs using deep RNNs?"*

We give a *positive answer* to this question. We show that the performance boost provided by deep SSMs like S4 can also be achieved via a series of small changes to a vanilla deep RNN. With these changes, we can recover the performance and efficiency of these deep SSMs on the Long Range Arena (LRA) benchmark (Tay et al., 2020). We call this new RNN model the Linear Recurrent Unit (or LRU for short).

Main Steps. We outline here the main steps needed towards crafting performant and efficient RNN models. Note while some of these observations have been made in prior works (see §B), we provide novel perspectives and careful ablations leading to new insights. Each step presented in this paper unveils a specific property of

recurrent networks, and showcases the challenges and best practices in training and initializing deep RNNs.

- **Linear Recurrences.** When replacing SSM layers in a deep architecture with vanilla RNN layers using tanh or ReLU activations, the performance on Long Range Arena (LRA) drops significantly. Surprisingly, in §3.1 we find that simply *removing the nonlinearities in the recurrence of the RNN* (i.e., using linear recurrences) gives a substantial boost in test accuracy. We motivate this effect in §E.1 by showing that stacking linear RNN layers and nonlinear MLP blocks (Fig.1) can indeed model complex nonlinear sequence-to-sequence maps without the need for nonlinearities in the recurrence. While dropping the nonlinearity does not seem to harm expressivity, it leads to several advantages, from the ability to directly control how quickly the gradients might vanish or explode, to allowing us to parallelize training. Our findings also partially motivate the success of deep SSMs, where the recurrence is also linear.
- **Complex Diagonal Recurrent Matrices.** Dense linear RNN layers can be reparameterized to a complex diagonal form without affecting the expressivity of the network or the features at initialization (§3.2). Diagonal linear RNN layers additionally allow for a *highly parallelizable unrolling* of the recurrence using parallel scans to substantially improve training speeds (Martin and Cundy, 2017). We validate that these observations, which have been leveraged by prior SSMs (Gupta et al., 2022a; Smith et al., 2022), also provide important efficiency improvements for linear RNN layers.
- **Stable Exponential Parameterization.** In §3.3 we show that using an exponential parameterization for the diagonal recurrent matrix has important benefits. Crucially, this enables us to easily enforce stability during training, which in turn allows us to modify the initialization distribution to facilitate long-range reasoning and improve performance. Our results indicate that rather than the specific deterministic initializations used by several recent SSMs, it is the eigenvalue distribution of the recurrent layer at initialization that determines if the model can capture long-range reasoning.
- **Normalization.** In §3.4 we show that normalizing the hidden activations on the forward pass is important when learning tasks with very long-range dependencies. With this final modification, our RNNs can match the performance of deep SSMs on all tasks in the LRA benchmark. Connecting back to state-space models, we show in §4 how our normalization can be linked to the discretization structure in S4.

We summarize the deep Linear Recurrent Unit (LRU) architecture used in this paper, and the effect of each of the above steps on performance in Fig.1. We emphasize that the main purpose of our work is not to surpass the performance of S4-based models, but rather to demonstrate that simple RNNs can also achieve strong performance on long range reasoning tasks when properly initialized and parameterized. We believe the insights derived in this paper can be useful to design future architectures, and to simplify existing ones.

2. Preliminaries

In this section, we compare the key architectural components (RNNs and SSMs) studied in this work, and also describe our methodology and experimental setup. For a more thorough discussion or related architectures, the reader can check our related work section §B.

2.1. Recap of recurrent block structures

We give an overview of the main architectural components considered in this paper, focusing on the major difference between Vanilla RNNs and recent S4-like deep SSMs (Gu et al., 2021a, 2022a; Gupta et al., 2022a; Smith et al., 2022).

RNN Layer. Let $(u_1, u_2, \dots, u_L)$ be a sequence of H_{in}-dimensional inputs, which can be thought of as either the result of intermediate layer computations (which keep the sequential structure) or as the initial input. An RNN layer with N-dimensional hidden state computes a sequence of H_{out}-dimensional outputs $(y_1, y_2, \dots, y_L)$ through a recurrent computation[1] using learnable parameters $A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times H_{\text{in}}}, C \in \mathbb{R}^{H_{\text{out}}\times N}, D \in \mathbb{R}^{H_{\text{out}}\times H_{\text{in}}}$:

$$x_k = \sigma(Ax_{k-1} + Bu_k), \quad y_k = Cx_k + Du_k, \tag{1}$$

[1] We do not use bias parameters as they can be incorporated into the MLP blocks preceding and following the RNN block. Classical RNNs also included a nonlinearity on the output $y_k = \sigma_{\text{out}}(Cx_k + b)$ with $D = 0$. Having $D \neq 0$ basically introduces a skip connection (standard in modern architectures), and the σ_{out} can be thought of as part of the MLP following the RNN.

starting from $x_0 = 0 \in \mathbb{R}^N$. σ here denotes a nonlinearity, often chosen to be a tanh or sigmoid activation. If σ is the identity function, then we say the RNN layer is *linear*.

S4-like recurrent layer. We present a simplified[2] version of the S4 recurrence introduced in Gu et al. (2021a). The input $(u_0, u_1, \ldots, u_{L-1})$ is now seen as the result of sampling a latent continuous-time signal $u_{\text{ct}} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{H_{\text{in}}}$ at multiples of a stepsize $\Delta > 0$: i.e. $u_{\text{ct}}(\Delta k) := u_k$ for all $k \in 0, \ldots, L-1$. The output sequence $(y_0, y_1, \ldots, y_{L-1})$ is then sampled, again with stepsize Δ, from the signal $y_{\text{ct}} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{H_{\text{out}}}$ computed by the following continuous-time state-space model, initialized at $x_{\text{ct}}(0) = 0$:

$$\begin{aligned} \frac{d}{dt} x_{\text{ct}}(t) &= \tilde{A} x_{\text{ct}}(t) + \tilde{B} u_{\text{ct}}(t), \\ y_{\text{ct}}(t) &= \Re\left[\tilde{C} x_{\text{ct}}(t)\right] + \tilde{D} u_{\text{ct}}(t), \end{aligned} \tag{2}$$

where $\Re(p)$ denotes the real part of a complex-valued vector p, $\tilde{A} = \text{diag}(\tilde{a})$ with $\tilde{a} \in \mathbb{C}^N$, $\tilde{B} \in \mathbb{C}^{N \times H_{\text{in}}}$, $\tilde{C} \in \mathbb{C}^{H_{\text{out}} \times N}$ and $\tilde{D} \in \mathbb{R}^{H_{\text{out}} \times H_{\text{in}}}$. Ignoring the continuous-time nature of this model, the most striking differences compared to Eq.(1) are that (a) the computation on the right-hand-side is *linear* in the hidden state and in the input, and (b) most parameters are *complex* valued, with $\tilde{A}$ being diagonal. While $\tilde{B}, \tilde{C}, \tilde{D}$ follow complex random or uniform initialization, the transition matrix $\tilde{A}$ is *structured*, i.e., initialized *deterministically* through HiPPO theory (Gu et al., 2020) in diagonal form. Common choices (Gu et al., 2022a) are $\tilde{a}_n = -\frac{1}{2} + i\pi n$ (S4D-Lin) and $\tilde{a}_n = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{n+1} - 1\right)$ (S4D-Inv), for $n = 1, 2, \ldots, N$.

For training and inference, the continuous-time system in Eq.(2) is discretized at stepsize Δ through a high-accuracy Zero-Order-Hold (ZOH) or Bilinear method. The ZOH method gives

$$x_k = A x_{k-1} + B u_k, \quad y_k = C x_k + D u_k, \tag{3}$$

where $x_{-1} = 0$, $A = \exp(\Delta\tilde{A})$, $B = (A - I)\tilde{A}^{-1}\tilde{B}$, $C = \tilde{C}$ and $D = \tilde{D}$, and exp denotes the matrix exponential. Under the assumption that u_{ct} is constant in between timestamps (which can be thought of as a modeling assumption), this numerical integration is *exact* (Jacquot, 2019). Moreover, note that all these discretization operations can be quickly performed element-wise since $\tilde{A}$ is diagonal.

Some key differences. It is worth pointing out a few structural and computational properties, to highlight some crucial differences between RNNs and SSMs:

- Since Eq.(3) is linear, it can be efficiently parallelized until $k = L - 1$ using parallel scans (Martin and Cundy, 2017; Smith et al., 2022), unlike a nonlinear RNN where the computation has to be performed sequentially.
- While Eq.(3) is similar to the linear RNN computation, it is crucial to note that (a) A and B are parameterized in a peculiar way, prescribed by discretization, and (b) these matrices share parameters; in particular Δ affects both A and B. These differences are critical as in SSMs learning is performed on the continuous-time parameters $\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}, \Delta$; hence parameterization choices directly affect optimization.
- Unlike vanilla RNNs, most SSMs use complex-valued diagonal recurrent matrices that are initialized deterministically using HiPPO theory, and the literature attributes much of the success of SSMs to the specific initialized used (Gu et al., 2021a, 2022b; Gupta et al., 2022a).

The points above motivate our investigation: in this paper we consider the same architecture as Gu et al. (2021a, 2022a); Smith et al. (2022), but replace the SSM layer in the recurrent core by an RNN. We then study which steps need to be taken to gradually retrieve S4-like performance on LRA (Tay et al., 2020) tasks. The effectiveness of each of our steps is supported by empirical evidence and theoretical considerations, and leads to the architecture presented in Fig.1.

2.2. Experimental setup

In this paper, we consider the Long Range Arena benchmark (Tay et al., 2020), a set of tasks designed to test the ability of models to do long-range sequence modelling (except we use coloured images instead of grayscale images for the sequential CIFAR-10 classification task). Transformers fail to perform well on most of these tasks,

[2] This version is most similar to S5 (Smith et al., 2022), but is here presented for ease of reasoning for a single discretization parameter Δ, shared across input dimensions. For more details, see §B.

while deep SSMs have shown remarkable performance on these tasks (Dao et al., 2022a; Gu et al., 2021a). This makes it an appropriate benchmark to explore the long-range modelling capabilities of deep RNNs.

For all our experiments, we use a network of 6 layers with residual connections and layer/batch normalization (Ba et al., 2016; Ioffe and Szegedy, 2015) similar to Gu et al. (2021a) (Fig.1), and we replace the SSM layers with RNN layers, building up to our LRU recurrence in a sequence of steps (see §3). All experiments are repeated three times, and we report the mean and standard error. Networks are trained using the AdamW optimizer (Loshchilov and Hutter, 2017). We use a smaller learning rate and no weight decay on the recurrent parameters, as suggested by Gu et al. (2021a); Steil (2004). We tune hyperparameters such as learning rates for all models on a logarithmic grid for best accuracy. See §D for more details on our experimental setup.

3. Designing Performant Deep RNNs

In this section, we discuss the fundamental steps needed for designing RNNs to reach the impressive performance of deep SSMs on the LRA benchmark. We present these steps, already outlined in the introduction, in logical order, and support each claim with experimental evidence and theoretical considerations, expanded in §E.

We consider the architecture of Fig.1, where the recurrent computation is gradually modified starting from a vanilla RNN. We start by showcasing the advantage of using linear recurrences in §3.1; then, in §3.2, we show how to speed-up training and inference without affecting expressivity and initialization distribution. In §3.3, we discuss how (and why) changing the parameterization and initialization distribution enables us to make the RNN stable and improve long-range modeling. Finally, in §3.4, we finalize the LRU architecture by proposing a normalization strategy for the hidden activations that results in a close match in performance with deep SSMs.

3.1. Linear RNN layers are performant

One of the main findings of our work is that linear RNN layers can be surprisingly expressive when coupled with nonlinear MLP or GLU (Dauphin et al., 2017) blocks, outperforming tuned nonlinear RNN variants in the same architecture. In Tb.1, we show that simply removing[3] the nonlinearity, and therefore computing the next state as $x_k = Ax_{k-1} + Bu_k$, is able to improve test accuracy on most LRA tasks. While the boost provided by vanilla linear RNN blocks leads to performance which is still far behind S4 on some tasks (sCIFAR, PathFinder and PathX), this first finding motivates us to drop nonlinearities in the recurrence for the rest of this paper. In later sections, we leverage the linearity of the recurrence to significantly speed up training as well as derive principled initialization and normalization principles to learn long-range dependencies. We note that, on the Text and Retrieval tasks, performance using vanilla RNNs already matches performance of deep SSMs (see Tb.3 for the performance of S4D/S5 on these tasks).

Recurrence	sCIFAR	ListOps	Text	Retrieval
RNN-ReLU	69.7 (0.2)	37.6 (8.0)	88.0 (0.1)	88.5 (0.1)
RNN-Tanh	69.9 (0.3)	43.9 (0.1)	87.2 (0.1)	88.9 (0.2)
RNN-Lin	**72.2** (0.2)	**50.4** (0.2)	**89.1** (0.1)	**89.1** (0.1)

Table 1 | *The effect of removing the nonlinearity from the recurrent unit on test accuracy (§3.1). We show here results only for the sCIFAR, ListOps, Text and Retrieval tasks in LRA as these models did not exceed random guessing on PathFinder/PathX (further improvements in Tb.2 and 3). Performance of deep SSMs shown in Tb.3.*

The empirical result in Tb.1 is *surprising*, since recurrent nonlinearities are believed to be a key component for the success of RNNs — both in the theory and in practice (Erichson et al., 2021; Pascanu et al., 2013; Siegelmann, 2012). Indeed, a strong property of single-layer sigmoidal and tanh RNNs is Turing completeness, which cannot be achieved by the linear variant (Chung and Siegelmann, 2021). However, the architecture we use (Fig.1) is deeper than a standard RNN and includes nonlinearies, placed position-wise *after* each RNN block. In §E.1, we investigate how the expressivity and trainability of deep models is affected by recurrent

[3] All other settings in the recurrent block are kept the same as in the Vanilla RNN module of Haiku (Hennigan et al., 2020). That is, all matrices have Glorot (Glorot and Bengio, 2010) initialization. The rest of the architecture is kept as in Fig.1, where the LRU block is replaced by an RNN.

nonlinearities. Leveraging a spectral analysis and Koopman operator theory (Koopman and Neumann, 1932), we discuss how interleaving linear RNN layers with nonlinear feedforward blocks is sufficient to approximate highly nonlinear systems. A key observation in our analysis is that position-wise nonlinearities effectively transfer signal information to higher frequencies, enabling the system to go beyond linearity in the spectral domain and increasing the layer capacity. To further strengthen our claim on the advantage of linear recurrences, in §E.2 we show that, while linear and nonlinear RNNs share an important class of approximating functionals (linear operators, see Wang et al. (2022)), nonlinear activations can potentially slow down training.

3.2. Using complex diagonal recurrent matrices is efficient

We now show that we can significantly speed up training and inference for deep linear RNNs without losing performance by using complex-valued diagonal recurrent matrices. While the idea of diagonalizing linear systems for computational efficiency is a dominating feature of all deep SSMs since the introduction of DSS by Gupta et al. (2022a), in this section we construct our diagonalized version to exactly match the initialization spectrum (see §3.2.1) of the Glorot-initialized deep linear RNN in Tb.1. Our main purpose with this approach is to *disentangle the effects of initialization and diagonalization on performance* (cf. Tb.2 and Tb.3).

We start in §3.2.1 by recalling some useful linear algebra elements, and then proceed in §3.2.2 with a discussion on how to diagonalize the recurrence while preserving the eigenvalue spectrum at initialization.

3.2.1. Linear RNN eigendecomposition

The recurrence $x_k = Ax_{k-1} + Bu_k$ can be unrolled easily using the assumption that $x_{-1} = 0 \in \mathbb{R}^N$:

$$x_0 = Bu_0, \quad x_1 = ABu_0 + Bu_1, \quad x_2 = A^2Bu_0 + ABu_1 + Bu_2, \quad \dots \quad \Longrightarrow \quad x_k = \sum_{j=0}^{k-1} A^j Bu_{k-j}. \tag{4}$$

Exponentiations of the matrix A in the equation above are the source of the well-known *vanishing/exploding gradient* issue in RNNs (Bengio et al., 1994; Pascanu et al., 2013). While in nonlinear RNNs the state x_k is forced to live on the compact image of the activation function, the hidden-state of our linear variant can potentially explode or vanish exponentially as k increases. This phenomenon can be better understood by leveraging an eigenvalue (a.k.a. spectral) analysis: up to an arbitrarily small perturbation of the entries, every matrix $A \in \mathbb{R}^{N\times N}$ is diagonalizable[4] (Axler, 1997), i.e. one can write $A = P\Lambda P^{-1}$, where $P \in \mathbb{C}^{N\times N}$ is an invertible matrix and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_N) \in \mathbb{C}^{N\times N}$. It is essential to note that, unlike the symmetric setting where eigenvalues and eigenvectors are real, in the non-symmetric case[5] one has to allow for *complex* entries to achieve full equivalence. Plugging the decomposition $A = P\Lambda P^{-1}$ into Eq.(4) and multiplying both sides by P^{-1}, we get $\bar{x}_k = \sum_{j=0}^{k-1} \Lambda^j \bar{B} u_{k-j}$, where $\bar{x}_k := P^{-1}x_k$, $\bar{B} := P^{-1}B$. The output can then be computed as $y_k = \Re[\bar{C}\bar{x}_k] + Du_k \in \mathbb{R}^H$, where $\bar{C} = CP^{-1}$, and we take the real part of $\bar{C}\bar{x}_k$. Therefore, instead of learning (A, B, C, D), one can equivalently learn $(\Lambda, \bar{B}, \bar{C}, D)$, where $\Lambda, \bar{B}, \bar{C}$ are complex valued, and Λ is a diagonal matrix.

Are complex numbers really necessary? We adopt complex numbers since they provide a convenient and compact representation of non-symmetric matrices in diagonal form. However this is not the only option – one could work (almost) as efficiently using real numbers. We discuss how this can be achieved in §E.3.

Stability. Since $\bar{x}_k = \sum_{j=0}^{k-1} \Lambda^j \bar{B} u_{k-j}$, the norm of component j of $\bar{x}$ at timestamp k evolves such that $|x_{k,j}| = O(|\bar{x}_{k,j}|) = O(|\lambda_j|^k)$. Therefore, a sufficient condition to ensure stability (i.e. x_k does not explode) is therefore $|\lambda_j| < 1$ for all j (Gu et al., 2021a).

3.2.2. Learning in the diagonalized space

Learning recurrent linear systems in diagonal form provides substantial computational speedups both for training and inference. For example, in our implementation of sCIFAR, we found diagonal linear RNNs to be ~8 times faster to train than a dense RNN with ReLUs, matching the speed of our implementations of S4D and S5. The main reasons for this computational benefit are that (a) taking powers of diagonal matrices is

[4] In other words, the set of non-diagonalizable matrices has measure zero, see e.g. Zhinan (2002) for a proof idea.

[5] Take e.g. $A = ((0, 1)(-1, 0))$. The solution to the standard eigenvalue equation gives $\lambda = \pm i$, where i is the imaginary unit.

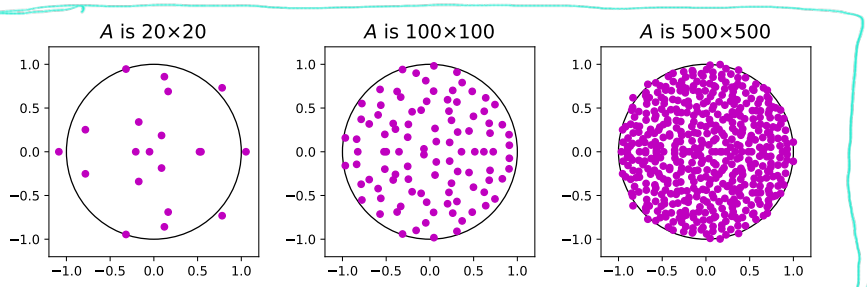

Figure 2 | *Eigenvalues of $A \in \mathbb{R}^{N\times N}$ following Glorot initialization: each entry of A is sampled independently from a Gaussian with mean 0 and variance $1/N$. The eigenvalues are complex (A is not symmetric) and are represented on the complex plane. The black circle is the unit disk $\{|z| = 1\} \subseteq \mathbb{C}$. The limit behavior (uniform initialization) is predicted by Thm. 3.1.*

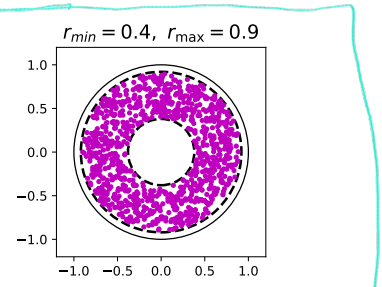

Figure 3 | *Eigenvalues of a diagonal matrix A with entries sampled using Lemma 3.2. For $r_{\min} = 0$, $r_{\max} = 1$, the distribution coincides with Glorot init. in the limit.*

trivial (speeding up both training and inference), while exponentiating dense matrices is computationally expensive, and (b) while nonlinear recurrences must be computed sequentially, unrolling a linear recurrence can be parallelized using associative scans resulting in faster training (Gupta et al., 2022a; Smith et al., 2022).

Equivalent initialization. To disentangle the benefits of diagonal linear systems from the role of initialization, we seek an initialization for the diagonal system which keeps the eigenvalue spectrum of the recurrence unchanged when comparing our diagonal system with the dense linear RNN in §3.1, where A followed Glorot initialization. Fortunately, we can use a classical result from random matrix theory (Ginibre, 1965).

Theorem 3.1 (Strong circular law)**.** *Let μ_N be the empirical spectral measure of A_N, where A_N is a real $N \times N$ matrix with i.i.d. Gaussian entries, each with zero mean and variance $1/N$. Then, μ_N converges weakly almost surely as $N \to \infty$ to the uniform probability measure on $\{|z| \leq 1\} \subseteq \mathbb{C}$.*

The theorem above, illustrated in Fig.2, shows that under Glorot initialization the spectrum of A is *de-facto* sampled from the unit disk in $\mathbb{C}$. This result motivates the strong performance of linear RNNs in §3.1, since it implies Glorot initialization provides an approximately stable initialization (see definition in §3.2.1).[6] Moreover, from Theorem 3.1, an *equivalent spectral initialization* follows for the diagonal system, which holds exactly for the large width limit: Λ should be diagonal with entries sampled uniformly on the unit disk. Using the definition of exponential of a complex number: $\exp(-\nu + i\theta) := e^{-\nu}(\cos(\theta) + i\sin(\theta))$, we adopt a simple scheme for sampling uniformly on a ring in between circles with radii $r_{\min}$ and $r_{\max}$in $\mathbb{C}$.

Lemma 3.2. *Let u_1, u_2 be independent uniform random variables on the interval $[0, 1]$. Let $0 \leq r_{\min} \leq r_{\max} \leq 1$. Compute $\nu = -\frac{1}{2}\log\left(u_1(r_{\max}^2 - r_{\min}^2) + r_{\min}^2\right)$ and $\theta = 2\pi u_2$. Then $\exp(-\nu + i\theta)$ is uniformly distributed on the ring in $\mathbb{C}$ between circles of radii $r_{\min}$ and $r_{\max}$.*

We recover the spectrum of Glorot-initialization (in the limit of infinite width) by setting $r_{min} = 0$ and $r_{max} = 1$ (we will explore tuning these hyper-parameters in §3.3). Tb.2 (first two rows) shows the results of learning deep linear RNNs in complex diagonal form,[7] where each diagonal entry of Λ is initialized uniformly on unit disk in $\mathbb{C}$ using Lemma 3.2 with $[r_{\min}, r_{\max}] = [0, 1]$. In our experiments, $\bar{B}, \bar{C}$ (which we rename for convenience back to B and C) follow Glorot initialization for both real and imaginary parts (parameterized separately), with halved variance in each component to preserve lengths on the input-output projections (Glorot and Bengio, 2010). Finally, after the SSM computation, the real part of the signal is kept and the imaginary discarded (as in Gu et al. (2022a); Gupta et al. (2022a)).

Our results in Tb.2 show that diagonalizing the recurrence surprisingly improves accuracy on tasks like ListOps and sCIFAR. More importantly, it drastically reduces training and inference time on all LRA tasks (see Tb.4 in §C.1 for training speed comparisons), and makes the RNN just as fast to train as deep SSMs like S4D and S5.

[6] Later in training, the system is less likely to become unstable if the learning rate is small enough.

[7] To avoid issues with backpropagation on complex variables, each complex parameter in the network is stored and learned as a pair of floats encoding real and imaginary parts.

	sCIFAR	ListOps	Pathfinder
Dense A	72.2 (0.2)	50.4 (0.2)	✗
Λ Real + Im	86.5 (0.1)	58.8 (0.3)	✗
Λ Exp	85.4 (0.7)	**60.5** (0.3)	65.4 (9.0)
Λ Stable Exp	87.2 (0.4)	59.4 (0.3)	93.5 (0.5)
+ Ring Init	**88.1** (0.0)	59.4 (0.3)	**94.4** (0.3)

Table 2 | *Test accuracy of a linear diagonal complex RNNs under different parametrizations of the transition matrix (see §3.2). Performance directly improves the results in Tb.1, and showcases the advantage of exponential (polar) representation of Λ. In bold font is the best parametrization option for linear RNN blocks. Ring Init denotes a changed initialization where $r_{\min}$ and $r_{\max}$ are tuned. Performance on the Text and Retrieval tasks is not shown as linear RNNs already align with S4 results (c.f. Tb.1 with Tb.3). These models cannot solve PathX yet, and requires normalizing the hidden activations and initializing the eigenvalues of Λ with small phase (see Tb.3).*

3.3. Benefits of stable exponential parameterization

In §3.2 we showed that moving to complex diagonal recurrences is computationally efficient. However we also observed that learning the diagonal model can be more unstable than learning the dense model in some experiments. To learn long-range dependencies and avoid quickly vanishing gradients, eigenvalues in the recurrence need to have magnitude close to 1 (Gu et al., 2022b; Gupta et al., 2022a); however, these eigenvalues are also likely to make the system unstable during training. In this section, we show the benefits of a stable parameterization of the RNN, and of tuning $r_{\min}$ and $r_{\max}$ (see Lemma 3.2).

Optimization under exponential parameterization. Lemma 3.2 suggests a natural parameterization of the diagonalized RNN as $\Lambda = \text{diag}(\exp(-\nu + i\theta))$ with $\nu \in \mathbb{R}^N$ and $\theta \in \mathbb{R}^N$ as the learnable parameters (instead of the real and imaginary parts of Λ). As we explain in §E.2 leveraging an easy-to-visualize 2-dimensional example (see Fig.8), this choice decouples magnitude and oscillation frequencies, making optimization with Adam easier. The positive effects of this exponential parametrization, which resembles some features of ZOH discretization (see §2 and §4) and notably takes the performance of PathFinder above random chance, can be observed in the third row of Tb.2.

Enforcing stability. An important benefit of the exponential parameterization is that it makes it simple to enforce stability on the eigenvalues. To see this, note that at initialization, $|\lambda_j| = |\exp(-\nu_j)| \leq 1$ since $\nu_j > 0$. Therefore, to preserve stability during training, we can use an exponential or another positive nonlinearity: $\lambda_j := \exp(-\exp(\nu_j^{\log}) + i\theta_j)$, where $\nu^{\log} \in \mathbb{R}^N$ is the parameter we optimize, and we set $\nu_j^{\log} := \log(\nu)$ at initialization. Note that a similar idea is used in deep SSMs (Gu et al., 2021a) in the context of discretization. We choose an exponential non-linearity over a simple ReLU nonlinearity to increase granularity around $|\lambda| = 1$, achieved at $\nu^{\log} = -\infty$ (while $|\lambda| = 0$ is achieved at $\nu^{\log} = \infty$). Stable parameterization helps on most LRA tasks. In the fourth row of Tb.2, we show its effects on sCIFAR, ListOps and Pathfinder. We observe the most drastic improvement on Pathfinder, one of the harder long-range dependency tasks in LRA, where performance now reaches above 93%.

The benefits of the stable parameterization becomes more apparent when we explore the idea of initializing the eigenvalues of Λ on a ring closer to the unit disk (increasing $r_{\min}$ closer to 1 in Lemma 3.2) to bias the network towards longer-range interactions and avoid vanishing gradients. Indeed, as discussed in detail in Gu et al. (2022b); Gupta et al. (2022a), for reasonings requiring consideration of interactions between distant tokens, eigenvalues in the recurrence need to have magnitude close to 1. Otherwise, as clear from the diagonal version of Eq.(4), when taking powers of eigenvalues close to the origin, the signal from past tokens quickly dies out (see §3.2.1). As we show in the last row of Tb.5 in §C, without enforcing stability, performance starts to degrade as we increase $r_{\max}$ past 0.9 in the sCIFAR task. With stability enforced, we can increase $r_{\max}$ up to 0.99 and improve performance. We see similar benefits on the other tasks where we sweep different values of $r_{\min}$ and $r_{\max}$ (Tbs.7 & 8 have more details). Finally, note that while here we explore changing the magnitude of the eigenvalues of Λ, in §3.4 we also show the benefits of initializing the eigenvalues to have a small phase to learn more global patterns, useful for particularly long-range reasoning tasks.

	sCIFAR	ListOps	Text	Retrieval	Pathfinder	PathX
LRU	89.0 (0.1)	60.2 (0.8)	89.4 (0.1)	89.9 (0.1)	95.1 (0.1)	94.2 (0.4)
S4D (our reprod.)	91.5 (0.2)	60.2 (0.3)	86.4 (0.0)	89.5 (0.0)	94.2 (0.3)	97.5 (0.0)
S5 (our reprod.)	88.8 (0.1)	58.5 (0.3)	86.2 (0.1)	88.9 (0.0)	95.7 (0.1)	96.0 (0.1)
S4 (paper results)	*91.1*	*59.6*	*86.8*	*90.9*	*94.2*	*96.4*
S4D-LegS (paper results)	89.9	60.5	86.2	89.5	93.1	91.9
S5 (paper results)	*90.1*	*62.2*	*89.3*	*91.4*	95.3	*98.6*

Table 3 | *Performance after adding the γ normalization to the diagonal RNN with stable exponential parameterization and initialization on the ring (see §3.4). For PathX, we additionally use a smaller eigenvalue phase at initialization. We name this architecture* ***LRU****. We sweep $r_{\min}$ and $r_{\max}$ for setting the initialization distribution and the learning rate. We also report results from S4/S4D/S5 (along with reproductions in our own pipeline with similar hyperparameter sweeps as our RNN models). LRU reaches similar performance as these deep SSMs on all LRA tasks.*

3.4. Additional considerations for long-range reasoning tasks

Up to this point, our model did not succeed in learning PathX — the hardest dataset in our benchmark, with a sequence length of $16k$ tokens. In this section, we discuss the additional modifications we need to make to improve our model's ability to learn very long-range dependencies and finalize our LRU model.

Normalization. In §3.3, we initialized the eigenvalues of Λ close to the unit disk for better performance on long-range tasks. However, we observed that as we moved $r_{\min}$ and $r_{\max}$ closer to 1, the training loss also started to blow up at initialization (see Fig.5). In this section, we first present a result explaining this phenomenon, before deriving a practical normalization scheme for the hidden activations to tackle this problem and further improve performance.

Proposition 3.3 (Forward-pass blow-up)**.** *Let Λ be diagonal with eigenvalues sampled uniformly on the ring in $\mathbb{C}$ between circles of radii $r_{\min} < r_{\max} < 1$. Then, under constant or white-noise input and Glorot input projection, we have that the squared norm of the state x_k converges as $k \to \infty$ to the following quantity.*

$$\mathbb{E}[\|x_\infty\|_2^2] = \frac{1}{r_{\max}^2 - r_{\min}^2} \log\left(\frac{1 - r_{\min}^2}{1 - r_{\max}^2}\right) \mathbb{E}[\|Bu\|_2^2].$$

This result has the following intuitive form if $r_{\min} = r_{\max} = r$: if we initialize ρ-close to the unit disk, the forward pass blows up by a factor $1/\rho$ (since the contributions from previous states take longer to decay): let $\epsilon = r_{\max}^2 - r_{\min}^2$ and $\rho = 1 - r_{\max}^2$, then:

$$\lim_{\epsilon \to 0} \frac{\mathbb{E}[\|x_\infty\|_2^2]}{\mathbb{E}[\|Bu\|_2^2]} = \lim_{\epsilon \to 0} \left[\frac{1}{\epsilon} \log\left(1 + \frac{\epsilon}{\rho}\right)\right] = \lim_{\epsilon \to 0} \left[\frac{1}{\epsilon}\left(\frac{\epsilon}{\rho} + O(\epsilon^2)\right)\right] = \frac{1}{\rho} = \frac{1}{1 - r^2}. \tag{5}$$

Towards the derivation of an effective normalization scheme for the forward pass, we present a simplified derivation of the $1/\rho$ gain formula for the one-dimensional setting under white-noise input[8]: let $\Lambda = \lambda \in \mathbb{C}$, and $B = 1$. Let p^* denote the conjugate of $p \in \mathbb{C}$, we have that $|p|^2 = p^* p$ and in expectation over the input, using Eq.(4) and the fact that $\mathbb{E}[u_{k-i} u_{k-j}] = 0$ for $i \neq j$:

$$\mathbb{E}|x_k|^2 = \left(\sum_{i=0}^{k-1} \lambda^i \mathbb{E}[u_{k-i}]\right)\left(\sum_{j=0}^{k-1} \lambda^j \mathbb{E}[u_{k-j}]\right)^* = \sum_{i,j=0}^{k-1} \lambda^i (\lambda^j)^* \mathbb{E}[u_{k-i} u_{k-j}] = \sum_{i=0}^{k-1} |\lambda|^{2i} \overset{\infty}{\to} \frac{1}{1 - |\lambda|^2}. \tag{6}$$

Since the formula above holds for every Euclidean direction in our recurrence (Λ is diagonal), we can add a normalization parameter that is initialized element-wise. Additionally, note that as λ approaches 1, $1 - |\lambda|^2$

[8] We use the random input assumption for our normalization scheme as we found it to work well in practice.

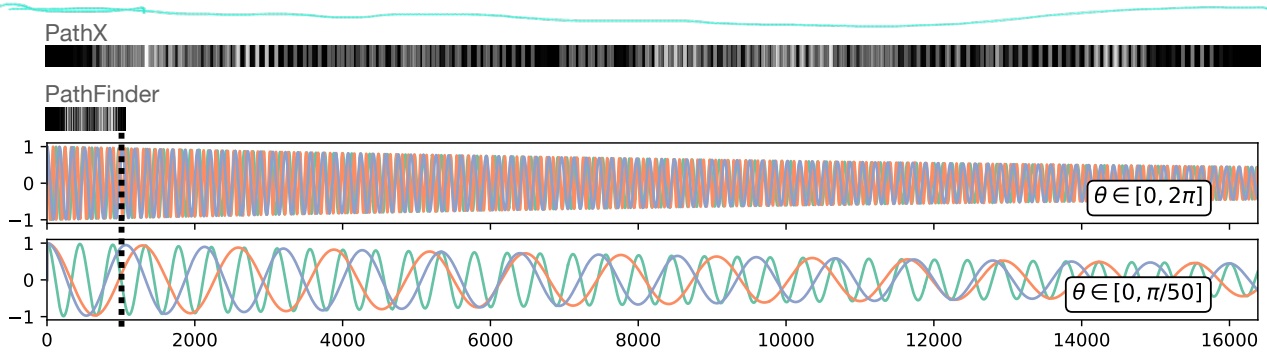

Figure 4 | *Evolution of $x \in \mathbb{R}^3$ under impulse input $u = (1, 0, 0, \ldots, 0) \in \mathbb{R}^{16k}$. Plotted in different colors are the 3 components of x. Λ has parameters $\nu_j = 0.00005$ and θ_j sampled uniformly in $[0, 2\pi]$ or with small phase $[0, \pi/50]$. For small sequences, such as $L = 1024$ (PathFinder, sCIFAR), $[0, 2\pi]$ produces kernels with acceptable overall number of oscillations: information about u_0 is recalled only a few times in the overall state history. Instead, for high L, the range of the imaginary part at initialization has to be smaller to obtain a similar effect.*

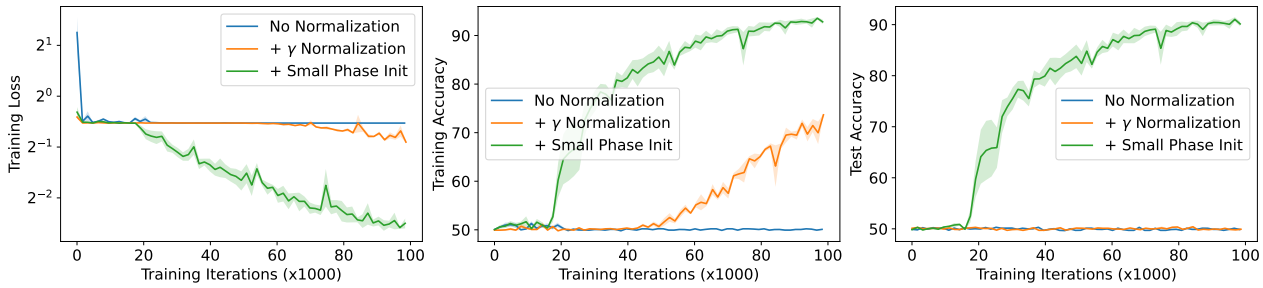

Figure 5 | *Effect of normalization and using a small phase at initialization on the PathX task. For each setting, we show mean and standard errors over three independent runs for 100k iterations. Without normalization, the model presents higher loss values at initialization and quickly converges to a suboptimal value, where train and test accuracy are both at random chance. Adding normalization helps: the train loss is lower at initialization, and the optimizer is able to escape the suboptimal region and train accuracy also increases. Interestingly, this model still fails to generalize at all. Finally, reducing initialization phase (i.e. tuning the range of θ) dramatically improves convergence on the training set, while also generalizing to the test set.*

approaches 0, making further adaptations with SGD of this parameter hard. Therefore, we use normalization parameter $\gamma^{\log} \in \mathbb{R}^N$, initialized element-wise as $\gamma_i^{\log} \leftarrow \log(\sqrt{1 - |\lambda_i|^2})$,[9] and modify the recurrence as:

$$x_k = \Lambda x_{k-1} + \exp(\gamma^{\log}) \odot (Bu_k), \tag{7}$$

where $\odot$ denotes the element-wise product. The γ parameter allows the RNN to adaptively scale the input fed into the corresponding eigendirection. We found the γ normalization to consistently improve performance on tasks that benefit from initializing close to the unit disk, such as sCIFAR and Pathfinder, as shown in Tb.3.

Reducing Eigenvalue Phase at Initialization. In the context of the diagonalized recurrence, we have $\Lambda = \mathrm{diag}(\exp(-\exp(\nu^{\log}) + \theta))$, where $\nu^{\log} \in \mathbb{R}^N$ is the vector of log eigenvalue magnitudes and $\theta \in \mathbb{R}^N$ the vector of eigenvalue *phases*. While $\nu^{\log}$ encodes the distance to the origin, θ is the angle from the vector $1 + 0i$. *For long sequences*, initializing uniformly $\theta \sim [0, 2\pi]$ implies that most state entries will exhibit an overall large number of oscillations at initialization, see upper panel in Fig.4. Equivalently, in this setting, most state dimensions are the result of *convolutions*[10] capturing an average of *local oscillation patterns*. This behavior is independent from the ability of capturing long-range dependencies (controlled by $\nu^{\log}$), but pertains to the nature of the information stored by the RNN. Therefore, we claim that initializing Λ with uniform phase on long sequence data inherently biases the network towards learning spurious features in the input sequence. The model cannot recover from this suboptimal initialization: we indeed observe that, for our best to far model on PathX, the

[9] We also tried setting γ_i to $\sqrt{1 - |\lambda_i|^2}$ in each training iteration, and found it to work similarly in practice to a trainable γ.
[10] See (Gu et al., 2022a) for a discussion of kernel perspectives.

training loss after a few iterations converges to a highly suboptimal minimizer which leads to random chance test performance (see Fig.5). To fix this issue, we found it sufficient to restrict the range of θ to a thin slice around 0, biasing the model towards learning more global features. Since the optimal values of θ are small, we parameterize the phase logarithmically: $\theta = \exp(\theta^{\log})$, where $\theta^{\log}$ is optimized, to aid optimization.

Restricting the range of the phase at initialization to be $[0, \pi/10]$, our LRU achieved 94.2% *on PathX*, aligning with state-of-the-art deep SSMs. We did not explore using a smaller phase at initialization for the other LRA tasks, although we believe this might further improve performance on other tasks as well. Note that using both γ normalization and restricting the eigenvalue phase at initialization were crucial to solving PathX. We were unable to learn when using restricted phase at initialization without also introducing γ normalization.

With all the components of §3 taken together, we name this new model the **Linear Recurrent Unit** (or **LRU** for short). It provides a flexible, interpretable, and principled framework for initializing and learning deep RNNs efficiently, and matches performance and efficiency of deep SSMs across all LRA tasks as shown in Tb.3.

4. Insights on S4 and Variants

We believe our ablations in §3 explain the underlying mechanisms driving the success of deep SSMs. Hence, to conclude the paper, in this section, we inspect in detail the main similarities and differences between our LRU model and diagonal SSMs, and elaborate a few insights. As in §2, to avoid technicalities, we provide a simplified discussion capturing the main features of models stemming from the original S4 paper. For a comparison of different models, we defer the reader to §B.

As detailed in §2, diagonal SSMs (DSS, S4D, S5) are instantiated and parameterized through *discretization* of a latent continuous-time model $\dot{x}_{\text{ct}}(t) = \tilde{A}x_{\text{ct}}(t) + \tilde{B}u_{\text{ct}}(t)$, where $A = \text{diag}(\tilde{a})$ is initialized with complex entries, often prescribed or inspired by HiPPO theory (Gu et al., 2020). Zero-Order-Hold (ZOH) discretization with stepsize Δ leads to the recurrence $x_k = \exp(\Delta\tilde{A})x_{k-1} + (\exp(\Delta\tilde{A}) - I)\tilde{A}^{-1}\tilde{B}u_k$. This formula, while arguably complex compared to our Eq.(7), relates to it as outlined in the next paragraphs.

Matrix exponentials make training easier. The exponential in the ZOH formula is due to exact integration of $\dot{x}_{\text{ct}}(t) = \tilde{A}x_{\text{ct}}(t)$, which leads to $x_{\text{ct}}(\Delta k) = \exp(\Delta\tilde{A})x_{\text{ct}}(\Delta(k-1))$. In addition, to enforce stability, in models inspired by S4 the real part of A is often fed into a positive nonlinearity, as we also do in §3.3. From our results §3.3 and our discussion on optimization advantages (see also §E.2), we claim that the power of exponential parameterization is not necessarily attributable to accurate integration (which is not present in our system), but is more fundamentally rooted in a magnitude-phase decoupling on the recurrence (this makes training with Adam easier, see Fig.8), as well as in the overall advantage of learning in diagonalized space (see Tb.2). We also note that stabilizing the recurrence by adding a nonlinearity was beneficial also in our experiments, although this is not prescribed by the theory underlying S4.

Structured initialization is not necessary. While Gu et al. (2022a); Gupta et al. (2022b); Smith et al. (2022) also discuss initializations for A deviating from the HiPPO structure (see §2 and §B), to the best of our knowledge we are the first to show that simple uniform initialization on a slice of the unit disk, combined with proper normalization, is able to also solve the hardest task in LRA: PathX.[11] We also show (Tb.2) that uniform initialization on the disk, which is simply the diagonalized version of Glorot initialization (Thm. 3.1), is sufficient to achieve performance close to more complex deep state-space models on the remaining LRA tasks. Our results ultimately suggest that HiPPO theory, while fundamental for the development of this field, should not be thought of as the main source of S4 success.

Discretization changes initialization spectrum. For simplicity, let us restrict our attention to S4D-Lin, for which $A = \text{diag}(\tilde{a})$ with $\tilde{a}_n = -\frac{1}{2} + i\pi n$, yielding a diagonal transition matrix with elements (i.e. eigenvalues) initialized at $\exp(-\Delta/2 + i\pi\Delta n)$. Under typical choices e.g. $\Delta = 1e{-}3, N = 128$, the SSM eigenvalues have magnitude $\exp(-\Delta/2) \approx 0.9995$, and phase $\theta = \pi\Delta n \in [0, \pi/8]$ — i.e. initialization is performed on a ring[12] close to the unit circle in $\mathbb{C}$, with restricted phase connected to the eigenvalues magnitude. As is clear from

[11] Among the models in (Gu et al., 2022a), only S4D-inv and S4D-LegS (options heavily inspired by the HiPPO theory) perform beyond random guessing on PathX. In S5, the skew-symmetric component of the HiPPO matrix is used for initialization.

[12] For all diagonal SSMs, Δ is actually a vector initialized in the range $[\Delta_{\min}, \Delta_{\max}]$. This interval can be directly mapped through the exponential map to a ring in complex space (see Lemma 3.2).

the results in §3.3 and §3.4, linking the eigenvalues phase and magnitude is not necessary to achieve good performance: indeed, as it can be seen in Tb.3, test accuracy on the Long Range Arena (except PathX) can be recovered by using a more natural magnitude-independent initialization on the complete ring. As we discussed in §3.4, changing the initialization phase to a small range around 0 can be motivated by first principles, yet is only needed for extremely long sequences: this modification is already hard-coded in S4, where choosing a small Δ also shrinks the phase.[13] However, our results clearly show that connecting real and imaginary parts during training through the Δ parameter is not necessary to achieve good performance, even on PathX.

Discretization performs normalization. The most striking visual difference between ours and ZOH-discretized S4 recurrence is in the matrix multiplier for u_k: $(\exp(\Delta\tilde{A}) - I)\tilde{A}^{-1}\tilde{B}$. After conducting experiments on S4D, we found that simply replacing this multiplier with its first-order expansion in Δ, i.e. $\Delta\tilde{B}$, yields a close match in performance. For input dimension $H = 1$ and unit $B \in \mathbb{R}^{N\times 1}$ (to keep reasoning simple), the corresponding recurrence is $x_k = \exp(\Delta\tilde{a}) + \Delta 1_N u_k$. Elementwise unrolling of this recurrence – without the Δ in front of u – yields $|x_{k,i}| \leq \sum_{j=0}^{k-1} |\exp(\Delta\tilde{a}_i)|^j u_{k-j,i}$, which in the limit $k \to \infty$ gives $O(\Delta^{-1})$. Therefore, the Δ multiplier in front of B effectively scales the recurrence to avoid blow-ups — similar to our γ normalization factor.

Parameter sharing is not necessary. As a result of discretization, the Δ parameter multiplying both $\tilde{A}$ and $\tilde{B}$ couples the recurrence formula with the input projection during training. In our S4 ablations, we found that decoupling these in two separate parameters — keeping the same initialization to guarantee no blow-ups (see last paragraph) — does not decrease performance, suggesting that the ODE discretization viewpoint (which induces parameter sharing) is not necessary to achieve S4 performance.

From this discussion, we conclude that the success of (diagonal) state-space models is attributable to the use of linear recurrences and complex diagonal exponential matrices, combined with the normalization and initialization induced by discretization. On the other hand, other artifacts of discretization such as parameter sharing or the continuous-time interpretation do not necessarily contribute to its performance.

5. Conclusion

In this paper, we introduce a new RNN layer called the Linear Recurrent Unit or LRU and show how it can be effectively and efficiently used as core layers of deep sequence models. We provide theoretical insights and extensive ablations on a series of step-by-step modifications of a vanilla RNN—linearization, diagonalization, stable exponential parameterization and normalization—that substantially improve performance, especially on tasks requiring long range reasoning. While our recurrence shares similarities with modern deep SSMs, our design does not rely on discretization of a latent continous-time system or on structured transition matrices. Instead our improvements directly follow from initialization and forward pass analysis arguments standard in the deep learning community, starting from a Glorot-initialized RNNs. Our final model matches the performance of modern deep state-space models (e.g. S4 or S5) on all LRA tasks.

Acknowledgements

The authors would like to thank Michalis Titsias, Aleksandar Botev, James Martens and Yee Whye Teh for the interesting discussions and perspectives on our work.

[13] This is a useful effect of having a latent continuous-time model: choosing eigenvalues close to the unit circle (i.e. small Δ) changes the oscillation frequencies in the discretized system.

References

M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*. PMLR, 2016.

S. Axler. *Linear algebra done right*. Springer Science & Business Media, 1997.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 1994.

N. Bordin, C. Dallago, M. Heinzinger, S. Kim, M. Littmann, C. Rauer, M. Steinegger, B. Rost, and C. Orengo. Novel machine learning approaches revolutionize protein knowledge. *Trends in Biochemical Sciences*, 2022.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, et al. JAX: composable transformations of python+ numpy programs, 2018.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, S. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014b.

S. Chung and H. Siegelmann. Turing completeness of bounded-precision recurrent neural networks. *Advances in Neural Information Processing Systems*, 2021.

T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022a.

T. Dao, D. Y. Fu, K. K. Saab, A. W. Thomas, A. Rudra, and C. Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022b.

Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*. PMLR, 2017.

S. De and S. Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 2020.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, N. Houlsby, S. Gelly, X. Zhang, and J. Uszkoreit. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

J. L. Elman. Finding structure in time. *Cognitive science*, 1990.

N. B. Erichson, O. Azencot, A. Queiruga, L. Hodgkinson, and M. W. Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2021.

J. Ginibre. Statistical ensembles of complex, quaternion, and real matrices. *Journal of Mathematical Physics*, 1965.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010.

K. Goel, A. Gu, C. Donahue, and C. Ré. It's raw! audio generation with state-space models. *arXiv preprint arXiv:2202.09729*, 2022.

A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 2020.

A. Gu, K. Goel, and C. Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021a.

A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 2021b.

A. Gu, A. Gupta, K. Goel, and C. Ré. On the parameterization and initialization of diagonal state space models. *arXiv preprint arXiv:2206.11893*, 2022a.

A. Gu, I. Johnson, A. Timalsina, A. Rudra, and C. Ré. How to train your hippo: State space models with generalized orthogonal basis projections. *arXiv preprint arXiv:2206.12037*, 2022b.

A. Gupta, A. Gu, and J. Berant. Diagonal state spaces are as effective as structured state spaces. In *Advances in Neural Information Processing Systems*, 2022a.

A. Gupta, H. Mehta, and J. Berant. Simplifying and understanding state space models with diagonal linear rnns. *arXiv preprint arXiv:2212.00768*, 2022b.

R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

R. Hasani, M. Lechner, T.-H. Wang, M. Chahine, A. Amini, and D. Rus. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.

K. Helfrich, D. Willmott, and Q. Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*. PMLR, 2018.

T. Hennigan, T. Cai, T. Norman, and I. Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

S. Hochreiter. Untersuchungen zu dynamischen neuronales netzen. *Diploma thesis, Institut f"ur Informatik, Technische Universit"at M"unchen*, 1991.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 1982.

S. L. Hyland and G. Rätsch. Learning unitary operators with help from u (n). In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

M. M. Islam and G. Bertasius. Long movie clip classification with state-space video models. In *ECCV 2022*. Springer, 2022.

R. G. Jacquot. *Modern digital control systems*. Routledge, 2019.

H. Jeffreys. *The theory of probability*. OUP Oxford, 1998.

L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*. PMLR, 2017.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

E. Kaiser, J. N. Kutz, and S. L. Brunton. Data-driven discovery of koopman eigenfunctions for control. *Machine Learning: Science and Technology*, 2021.

N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

J. Kilian and H. T. Siegelmann. The dynamic universality of sigmoidal neural networks. *Information and computation*, 1996.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

B. O. Koopman and J. v. Neumann. Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences*, 1932.

M. Korda and I. Mezić. On convergence of extended dynamic mode decomposition to the koopman operator. *Journal of Nonlinear Science*, 2018.

M. Korda and I. Mezić. Koopman model predictive control of nonlinear dynamical systems. In *The Koopman Operator in Systems and Control*. Springer, 2020.

V. R. Kostic, P. Novelli, A. Maurer, C. Ciliberto, L. Rosasco, and massimiliano pontil. Learning dynamical systems via koopman operator regression in reproducing kernel hilbert spaces. In *Advances in Neural Information Processing Systems*, 2022.

J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

M. Lezcano-Casado and D. Martınez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*. PMLR, 2019.

Y. Li, T. Cai, Y. Zhang, D. Chen, and D. Dey. What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*, 2022a.

Z. Li, J. Han, E. Weinan, and Q. Li. Approximation and optimization theory for linear continuous-time recurrent neural networks. *J. Mach. Learn. Res.*, 2022b.

L. Liu, H. Wang, J. Lin, R. Socher, and C. Xiong. Mkd: a multi-task knowledge distillation approach for pretrained language models. *arXiv preprint arXiv:1911.03588*, 2019.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

X. Ma, C. Zhou, X. Kong, J. He, L. Gui, G. Neubig, J. May, and L. Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

E. Martin and C. Cundy. Parallelizing linear recurrent neural nets over sequence length. *arXiv preprint arXiv:1709.04057*, 2017.

A. Mauroy and I. Mezić. Global stability analysis using the eigenfunctions of the koopman operator. *IEEE Transactions on Automatic Control*, 2016.

A. Mauroy, Y. Susuki, and I. Mezić. *Koopman operator in systems and control*. Springer, 2020.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 1943.

H. Mehta, A. Gupta, A. Cutkosky, and B. Neyshabur. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.

Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *International Conference on Machine Learning*. PMLR, 2017.

T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*. Makuhari, 2010.

R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

E. Nguyen, K. Goel, A. Gu, G. Downs, P. Shah, T. Dao, S. Baccus, and C. Ré. S4nd: Modeling images and videos as multidimensional signals with state spaces. In *Advances in Neural Information Processing Systems*, 2022.

A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 2013.

J. L. Proctor, S. L. Brunton, and J. N. Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 2018.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 2010.

H. T. Siegelmann. *Neural networks and analog computation: beyond the Turing limit*. Springer Science & Business Media, 2012.

J. T. Smith, A. Warrington, and S. W. Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

J. J. Steil. Backpropagation-decorrelation: online recurrent learning with o (n) complexity. In *2004 IEEE international joint conference on neural networks*. IEEE, 2004.

A. Surana. Koopman operator based observer synthesis for control-affine nonlinear systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016.

Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

A. Voelker, I. Kajić, and C. Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 2019.

C. R. Vogel. *Computational methods for inverse problems*. SIAM, 2002.

S. Wang, Z. Li, and Q. Li. The effects of nonlinearity on approximation capacity of recurrent neural networks, 2022.

S. H. Weintraub. Jordan canonical form: theory and practice. *Synthesis Lectures on Mathematics and Statistics*, 2009.

M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 2015.

S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. *Advances in neural information processing systems*, 2016.

Z. Zhinan. The jordan canonical form of a rational random matrix. *Science Direct Working Paper*, 2002.

T. Zhou, Z. Ma, Q. Wen, L. Sun, T. Yao, R. Jin, et al. Film: Frequency improved legendre memory model for long-term time series forecasting. *arXiv preprint arXiv:2205.08897*, 2022.

Supplementary Materials

A. Simplified Implementation of the Linear Recurrent Unit

We present here a simplified JAX implementation (Bradbury et al., 2018) of the Linear Recurrent Unit (LRU). The state of the LRU is driven by the input $(u_k)_{k=1}^L$ of sequence length L according to the following formula (and efficiently parallelized using an associative scan): $x_k = \Lambda x_{k-1} + \exp(\gamma^{\log}) \odot (Bu_k)$, and the output is computed at each timestamp k as follows: $y_k = Cx_k + Du_k$. In our code, B, C follow Glorot initialization, with B scaled additionally by a factor 2 to account for halving the state variance by taking the real part of the output projection. D is random H-dimensional and mutiplies elementwise each u_k, where k is the timestamp. Λ is initialized with the help of Lemma 3.2, with phase potentially restricted to a thin slice (see §3.4).

```python
import jax
import jax.numpy as jnp
import numpy as np
parallel_scan = jax.lax.associative_scan

def forward(lru_parameters, input_sequence):
    """Forward pass of the LRU layer. Output y and input_sequence are of shape (L, H)."""

    # All LRU parameters
    nu_log, theta_log, B_re, B_im, C_re, C_im, D, gamma_log = lru_parameters

    # Materializing the diagonal of Lambda and projections
    Lambda = jnp.exp(-jnp.exp(nu_log) + 1j*jnp.exp(theta_log))
    B_norm = (B_re + 1j*B_im) * jnp.expand_dims(jnp.exp(gamma_log), axis=-1)
    C = C_re + 1j*C_im

    # Running the LRU + output projection
    # For details on parallel scan, check discussion in Smith et al (2022).
    Lambda_elements = jnp.repeat(Lambda[None, ...], input_sequence.shape[0], axis=0)
    Bu_elements = jax.vmap(lambda u: B_norm @ u)(input_sequence)
    elements = (Lambda_elements, Bu_elements)
    _, inner_states = parallel_scan(binary_operator_diag, elements) # all x_k
    y = jax.vmap(lambda x, u: (C @ x).real + D * u)(inner_states, input_sequence)

    return y

def init_lru_parameters(N, H, r_min=0, r_max=1, max_phase=6.28):
    """Initialize parameters of the LRU layer."""

    # N: state dimension, H: model dimension
    # Initialization of Lambda is complex valued distributed uniformly on ring
    # between r_min and r_max, with phase in [0, max_phase].
    u1 = np.random.uniform(size = (N,))
    u2 = np.random.uniform(size = (N,))
    nu_log = np.log(-0.5*np.log(u1*(r_max**2-r_min**2) + r_min**2))
    theta_log = np.log(max_phase*u2)

    # Glorot initialized Input/Output projection matrices
    B_re = np.random.normal(size=(N,H))/np.sqrt(2*H)
    B_im = np.random.normal(size=(N,H))/np.sqrt(2*H)
    C_re = np.random.normal(size=(H,N))/np.sqrt(N)
    C_im = np.random.normal(size=(H,N))/np.sqrt(N)
    D = np.random.normal(size=(H,))

    # Normalization factor
    diag_lambda = np.exp(-np.exp(nu_log) + 1j*np.exp(theta_log))
    gamma_log = np.log(np.sqrt(1-np.abs(diag_lambda)**2))

    return nu_log, theta_log, B_re, B_im, C_re, C_im, D, gamma_log

def binary_operator_diag(element_i, element_j):
    # Binary operator for parallel scan of linear recurrence.
    a_i, bu_i = element_i
    a_j, bu_j = element_j
    return a_j * a_i, a_j * bu_i + bu_j
```

B. Related works

We first discuss standard RNN-based approaches for sequence-to-sequence modeling, and then provide a historical overview on the progress of the literature stemming from the S4 paper (Gu et al., 2021a).

Recurrent neural networks (RNNs). Before the rise of transformers (Vaswani et al., 2017), RNNs were widely used in various applications of natural language processing tasks such as language modeling (Mikolov et al., 2010), machine translation (Cho et al., 2014b) and text summarization (Nallapati et al., 2016). The modern RNN structure (see Eq.1) is mainly attributed to the works of Rumelhart et al. (1985). However, it is possible to see the Hopfield Networks as a particular form of RNN (Hopfield, 1982). Modern RNN formulations are also often related to the Elman Networks (Elman, 1990). The issue of vanishing or exploding gradients, as described by Bengio et al. (1994); Pascanu et al. (2013), is one barrier to training Recurrent Neural Networks (RNNs) with gradient descent. This problem limits the ability of RNNs to learn, especially on tasks with long input sequences. One of the critical contributions to the success of RNNs was the introduction of gating mechanisms such as the Long Short-Term Memory (LSTM) proposed by the Hochreiter and Schmidhuber (1997). LSTMs address the vanishing gradients problem by introducing input, output, and forget gates, which enable the network to selectively remember or forget information from previous time steps. Another popular variant of gated RNNs is the Gated Recurrent Unit (GRU) (Cho et al., 2014b) which simplifies the LSTM architecture by merging input and forget gates into a single update gate.

Mitigating the vanishing gradient problem with orthogonal and unitary RNNs. Recently, Arjovsky et al. (2016) introduced unitary evolution RNNs (uRNN), where eigenvalues in the RNN transition matrix (see Eq. (1)) are restricted to live on the unit circle. The induced map driving the hidden state evolution, therefore, mixes state components taking into account new inputs — but the signal from past timestamps is not exponentially vanishing/exploding as in the vanilla RNN case (see discussion on stability in §3.2.1). This idea is powerful but introduces two problems: (1) choosing unitary transitions restricts the function approximation class, and (2) training unitary matrices is expensive since a projection on the Stiefel manifold is required at each gradient step. To resolve the second issue, many works devoted attention to carefully designed reparameterization of the transition matrix as e.g., with the product of simpler matrices (Arjovsky et al., 2016), Givens rotations (Jing et al., 2017), Householder reflections (Mhammedi et al., 2017), or as exponentials of skew-symmetric matrices (Hyland and Rätsch, 2017; Lezcano-Casado and Martınez-Rubio, 2019). The approximation capacity of these models is discussed and improved in (Wisdom et al., 2016). A further step in designing efficient orthogonal RNNs is provided by Helfrich et al. (2018), who parametrized skew-symmetric matrix using the Cayley transforms, resulting in a fully real parameter space. Other works which proposed conceptually different solutions to mitigate the vanishing gradient problem include combinations with rectified linear units (Le et al., 2015), Lipschitz RNNs (Erichson et al., 2021), and approaches based on dilated convolutions to increase context size (Bai et al., 2018; Oord et al., 2016)

Deep state-space models (SSMs), a historical overview. Inspired by interesting approaches involving continuous-time representation for recurrent neural networks (Voelker et al., 2019), Gu et al. (2020) recently provided an alternative view on the vanishing gradient problem: one can design *linear* continuous-time state-space models (SSMs), of the form $\dot{x}(t) = Ax(t) + Bu(t)$ where the state $x(t) \in \mathbb{R}^N$ is guaranteed to compress all relevant (under a certain metric) information about previously observed (one-dimensional) inputs $u([0,t])$. For instance, by using specific pair of matrices ($A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1}$), one can discretize the continuous-time SSM above using a stable, accurate integrator (e.g., bilinear or zero-order-hold) and retrieve the hidden state $x(t)$, which contains the coefficients for the best N-th degree polynomial approximation to $u([0,t])$. The idea of Gu et al. (2020) was to then use the resulting discretized *structured* (i.e., using structured HiPPO matrices) state-space model as a starting for the design and initialization of a novel gated RNN.

Later, Gu et al. (2021a) scaled up this idea into a deep architecture, where a collection (one for each input dimension) of discretized continuous-time structured SSM was placed at each layer as a substitute[14] for the attention block, in an attempt to mitigate the $O(L^2)$ issue in transformers and provide a theoretically principled component for sequence-to-sequence modeling. The model reached state-of-the-art on the Long Range Arena benchmark (Tay et al., 2020), effectively showcasing the power of discretized linear recurrences using structured

[14]This idea is also leveraged in FNet (Lee-Thorp et al., 2021), where the attention mechanism is replaced with a simpler linear token-mixing strategy.

transition matrices. Notably, the resulting model, named **S4**, uses a convenient and stable representation of the HiPPO transition, which is initialized using a normal + low-rank matrix and then learned efficiently in diagonal + low-rank form using fast Fourier transforms (FFTs) and Cauchy kernels.

In the months following the publication of S4, Gupta et al. (2022a) noticed that most of S4 performance can be retrieved by only considering the diagonal component of the HiPPO matrix, and therefore showed the power of discretized diagonal structured continuous-time state space models. This architecture is known as **DSS**. As the interest of the community was rising, with first applications of DSS and S4 in language (Mehta et al., 2022), vision (Nguyen et al., 2022) and audio (Goel et al., 2022), Gu et al. (2022a) further simplified DSS providing a diagonal form (**S4D**) with theoretical guarantees in the infinite width setting. Notably Gu et al. (2022a) showed that, to retrieve most performance of S4, one can simply initialize the transition matrix A in diagonal form, with entries $a_n = -\frac{1}{2} + i\pi n$ (S4D-Lin) or $a_n = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{n+1} - 1\right)$ (S4D-Inv). Our interest in S4-like models spiked at this point since the findings of Gu et al. (2022a) suggest that, given the effectiveness of such simplified versions of A, the root of S4 success might be attributable to more fundamental effects are orthogonal to the HiPPO theory.

Shortly after, Smith et al. (2022) found that one can also depart from the formal one-dimensional discretization structure of S4, rooted in the HiPPO theory, and considered a simplified version where all input dimensions are efficiently and simultaneously processed using parallel scans (Martin and Cundy, 2017) — not separately like in S4, S4D, and DSS. This model (named **S5**) set a new state-of-the art on PathX, the hardest task in the Long Range Arena, and provides further evidence for a conceptually simpler motivation for the performance of deep state-space models. Indeed, as already mentioned, S5 is not precisely the discretization of a latent continuous-time SSM, yet still includes parameters like discretization stepsizes that have an ambiguous interpretation in this context[15], suggesting further investigations are needed.

At the same time, a few interesting works developed novel variants of the S4 architecture. **Liquid S4** used the original (non-diagonal) S4 formulation combined with liquid time-constant networks (Hasani et al., 2021, 2022). Similar to DSS, S4D, and S5, **Mega** also simplified S4 to a diagonal SSM (Ma et al., 2022) while showing additionally that restricting the diagonal A to real numbers – giving it an exponential moving average (EMA) interpretation – can still work well when combined with attention and a gated block design. Another intriguing view was provided by the **SGConv** model (Li et al., 2022a), which leverages the convolutional interpretation of SSMs (Gu et al., 2021b) to design a purely filter-based version of S4, with no latent continuous-time model or need for discretization.

The discretization viewpoint also attracted the interest of Gupta et al. (2022b), concurrent to this work, who pointed out that, after numerical integration, diagonal state-space models and linear RNNs share the same function approximation class. Gupta et al. (2022b) then introduced **DLR**, most closely related to DSS and S4D (each input is processed independently at each layer) but where the discretization stepsize Δ is absorbed into the continuous-time transition matrix A (see §2). Their focus was on a new set of synthetic long-range tasks with strong supervision (e.g. segmentation), while ours is on the established Long Range Arena benchmark.

To conclude, we point the reader to interesting recent applications of models inspired by the S4 architecture. In addition to earlier applications in NLP (Mehta et al., 2022), more sophisticated architectures based on S4 recently showed great promise in language modeling (Dao et al., 2022b; Ma et al., 2022). Specifically, Dao et al. (2022b) designed a new generative language model, **H3**, that outperforms GPT-Neo-2.7B with SSMs, augmented with two attention layers. Besides language, deep state-space models were also found successful for long video/audio understanding and generation tasks (Goel et al., 2022; Islam and Bertasius, 2022; Nguyen et al., 2022), and have attracted interest in biology (Bordin et al., 2022) and time series forecasting (Zhou et al., 2022).

[15] One can still view S5 as a discretized version of a continuous-time SSM. However, this requires adjusting the input projection matrix.

C. Additional experimental results

C.1. Training speedups

In Tb.4, we show training speed comparisons of the LRU with a regular RNN with tanh activations, as well as with the S4D and S5 models. As we elaborate in §2.2, for the LRU, we closely followed the optimal model sizes of the S5 model. Consequently, we also see similar training speeds as the S5 model on all tasks.

Model	sCIFAR	ListOps	Text	Retrieval	Pathfinder	PathX
Tanh RNN	2.0	1.1	0.5	0.5	2.1	0.14
LRU	15.9 (8x)	2.1 (1.9x)	14.7 (29x)	5.7 (11.4x)	15.5 (7.4x)	2.4 (17x)
S4D (our reproduction)	13.5	2.2	10.6	3.0	24.5	2.6
S5 (our reproduction)	15.9	2.2	14.4	5.7	15.6	2.3

Table 4 | *Speeds (steps/sec) during training on a A100 GPU. We also show the speedup of the LRU over the tanh RNN for each task. The batch size used for each task is specified in Tb.9.*

C.2. Effect of stability and normalization

In this section, we explore further the effect of introducing stability during training (§3.3), as well as introducing the γ normalization factor as shown in Eq.(7). To do this, we consider the sCIFAR experiment where we sweep over different settings of $r_{\max}$ and $r_{\min}$ to see the effect when initializing closer to the unit disk. We keep the learning rate fixed at 0.004 for these experiments, which we found to be optimal when initializing with $r_{\max} = 1.0$ and $r_{\min} = 0.0$ under a stable exponential parameterization.

We show our results in Tb.5. In the first table Tb.5(A), we show results with our baseline where we use the exponential parameterization described in §3.3. We see that under this setting, the optimal performance is achieved when $r_{\max} = r_{\min} - 0.9$, and performance degrades as $r_{\max}$ is increased beyond 0.9.

In Tb.5(B) we show results after enforcing stability. We now notice that for each $r_{\min}$, the optimal performance is achieved by a higher $r_{\max}$ than before, i.e., training is more when initializing closer to the unit disk. Our optimal performance in this setting is achieved using $r_{\min} = 0.0$ and $r_{\max} = 0.99$. Note that even in this setting, performance can sometimes degrade when moving to even higher $r_{\max}$.

Finally, in Tb.5(C) we also incorporate the γ normalization factor, and we now notice no degradation in performance even when $r_{\max} = 0.999$. We found training to be more stable in this setting, and our best result of 89.0% performance is also obtained in this setting, with $r_{\min} = 0.9$ and $r_{\max} = 0.999$.

These ablations further motivate the benefits of enforcing stability and using the normalization parameter for better performance and more stable training, particularly when required to learn very long-range dependencies.

C.3. Expanded tables

Below we show our full results on the Long Range Arena, expanding on Tables 1, 2, and 3 in the main paper. The tables are presented in logical order: in Table 6, we show that vanilla (dense) RNNs profit from dropping recurrent nonlinearities when used in the context of the architecture in Fig. 1. Next, in Table 7 we diagonalize our linear RNN model from §3.1 and show how different parametrization for the diagonal elements affect performance. For all the rows in Table 7, initialization of the diagonal RNN was performed uniform on the disk, to match the random Glorot initialization of our dense version (Thm. 3.1).

Further, the last row in Table 7 shows the positive effects of changing initialization distribution to a thin ring close to the circle boundary — effectively enabling long-range reasoning through mitigation of vanishing gradients. Our settings for the ring are reported on the first row of Table 8. Finally, the second row of this table shows the improvements that can be achieved by including model normalization (Eq. (7)), which closes the accuracy gap with deep SSMs.

r_{max} \ r_{min}	0	0.5	0.9
0.9	**87.6** (0.4)	**87.8** (0.1)	**87.9** (0.2)
0.99	83.8 (0.9)	85.8 (1.2)	81.9 (3.8)
0.999	83.9 (0.2)	84.8 (0.4)	84.8 (0.8)

(A) No stability.

r_{max} \ r_{min}	0	0.5	0.9
0.9	86.2 (0.2)	86.6 (0.3)	87.3 (0.1)
0.99	**87.8** (0.2)	**87.7** (0.1)	**88.1** (0.0)
0.999	87.4 (0.2)	87.4 (0.1)	87.5 (0.4)

(B) With stability.

r_{max} \ r_{min}	0	0.5	0.9
0.9	86.4 (0.1)	86.5 (0.1)	88.3 (0.1)
0.99	**88.1** (0.1)	88.4 (0.1)	**89.0** (0.2)
0.999	**88.1** (0.1)	**88.6** (0.0)	**89.0** (0.1)

(C) With γ normalization.

Table 5 | *Effect of stability and normalization and different r_{min} and r_{max} values on test accuracy for the sCIFAR10 task. Both stability and normalization allow for initializing eigenvalues closer to the unit disk, resulting in improved performance.*

Recurrence	sCIFAR	ListOps	Text	Retrieval	Pathfinder	PathX
RNN-Lin	72.2 (0.2)	50.4 (0.2)	89.1 (0.1)	89.1 (0.1)	✗	✗
RNN-ReLU	69.7 (0.2)	37.6 (8.0)	88.0 (0.1)	88.5 (0.1)	✗	✗
RNN-Tanh	69.9 (0.3)	43.9 (0.1)	87.2 (0.1)	88.9 (0.2)	✗	✗
S4D (our reproduction)	91.5 (0.2)	60.2 (0.3)	86.4 (0.0)	89.5 (0.0)	94.2 (0.3)	97.5 (0.0)
S5 (our reproduction)	88.8 (0.1)	58.5 (0.3)	86.2 (0.1)	88.9 (0.0)	95.7 (0.1)	96.0 (0.1)
S4 (paper results)	*91.1*	*59.6*	*86.8*	*90.9*	*94.2*	*96.4*
S4D-LegS (paper results)	89.9	60.5	86.2	89.5	93.1	91.9
S5 (paper results)	*90.1*	*62.2*	*89.3*	*91.4*	95.3	*98.6*

Table 6 | *Placing a Vanilla RNN as recurrent core in the architecture of Fig. 1. Shown is the effect of removing the RNN non-linearity on test accuracy (§3.1).*

D. Detailed experimental setup

In this section, we describe our experimental details.

D.1. Architecture

We consider the standard S4 architecture of Gu et al. (2021a) and replace the S4 layers with RNN layers or with S5 (Smith et al., 2022) or S4D (Gu et al., 2022a) layers for our baselines. We give an overview of the architecture used in Fig.1. The input is first encoded into H features, followed by a stack of residual blocks. For all our experiments, we use networks with a depth of 6 residual blocks. Each residual block consists of identity skip connection, and the residual path containing a normalization layer (in our case, we always use batch normalization in our experiments), followed by the RNN/SSM block. While using the "post-norm" option of adding the normalization layer after the skip and residual branches typically improves performance, we stick to this design due to this architecture being more scalable in general (De and Smith, 2020).

Each RNN/SSM block first contains the recurrent layer as described in Eqs.(1) and (3) in §2. This is followed by a mixing layer. For all experiments except PathX, we use the GLU activation function (Dauphin et al., 2017) with dropout as the mixing layer, similar to Gu et al. (2021a). For PathX, we instead use a GLU activation function without one additional linear transform; the same as used by Smith et al. (2022) for their experiments.

We use bidirectional models for our experiments on PathFinder and PathX, using a similar setup as Gu et al. (2021a), and use unidirectional models for the rest of our experiments.

	sCIFAR	ListOps	Text	Retrieval	Pathfinder	PathX
Dense A	72.2 (0.2)	50.4 (0.2)	89.1 (0.1)	89.1 (0.1)	✗	✗
Λ Real + Im	86.5 (0.1)	58.8 (0.3)	87.4 (0.3)	87.8 (0.5)	✗	✗
Λ Exp	85.4 (0.7)	60.5 (0.3)	86.5 (0.4)	89.4 (0.1)	65.4 (9.0)	✗
Λ Stable Exp	87.2 (0.4)	59.4 (0.3)	87.6 (0.3)	89.1 (0.2)	93.5 (0.5)	✗
+ Ring Init	88.1 (0.0)	59.4 (0.3)	89.4 (0.1)	90.1 (0.1)	94.4 (0.3)	✗
S4D (our reproduction)	91.5 (0.2)	60.2 (0.3)	86.4 (0.0)	89.5 (0.0)	94.2 (0.3)	97.5 (0.0)
S5 (our reproduction)	88.8 (0.1)	58.5 (0.3)	86.2 (0.1)	88.9 (0.0)	95.7 (0.1)	96.0 (0.1)
S4 (paper results)	*91.1*	*59.6*	*86.8*	*90.9*	*94.2*	*96.4*
S4D-LegS (paper results)	89.9	60.5	86.2	89.5	93.1	91.9
S5 (paper results)	*90.1*	*62.2*	*89.3*	*91.4*	95.3	*98.6*

Table 7 | *Test accuracy of a linear diagonal complex RNNs under different parameterizations of the transition matrix (see §3.2). Performance directly improves the results in Tb. 1, and showcases the advantage of exponential (polar) representation of Λ. In bold font is the best parameterization option for linear RNN blocks. Ring Init denotes a changed initialization where $r_{\min}$ and $r_{\max}$ are tuned. Performance and Text and Retrieval task already aligns with S4 results in the dense setting (c.f. Tb.1 with Tb. 3). No model with able to solve PathX, which requires normalization (see Tb.3).*

	sCIFAR	ListOps	Text	Retrieval	Pathfinder	PathX
Linear Dense RNN	72.2 (0.2)	50.4 (0.2)	89.1 (0.1)	89.1 (0.1)	✗	✗
Diagonal Complex RNN	86.5 (0.1)	58.8 (0.3)	87.4 (0.3)	87.8 (0.5)	✗	✗
Stable Exp Param w/ Ring Init	88.1 (0.0)	59.4 (0.3)	89.4 (0.1)	90.1 (0.1)	94.4 (0.3)	✗
$[r_{\min}, r_{\max}]$	[0.9, 0.99]	[0.0, 1.0]	[0.0, 0.9]	[0.5, 0.9]	[0.9, 0.999]	
$+\gamma$ Normalization **(LRU)**	89.0 (0.1)	60.2 (0.8)	89.4 (0.1)	89.9 (0.1)	95.1 (0.1)	94.2 (0.4)
$[r_{\min}, r_{\max}]$	[0.9, 0.999]	[0.0, 0.99]	[0.5, 0.9]	[0.5, 0.9]	[0.9, 0.999]	[0.999, 0.9999]
S4D (our reproduction)	91.5 (0.2)	60.2 (0.3)	86.4 (0.0)	89.5 (0.0)	94.2 (0.3)	97.5 (0.0)
S5 (our reproduction)	88.8 (0.1)	58.5 (0.3)	86.2 (0.1)	88.9 (0.0)	95.7 (0.1)	96.0 (0.1)
S4 (paper results)	*91.1*	*59.6*	*86.8*	*90.9*	*94.2*	*96.4*
S4D-LegS (paper results)	89.9	60.5	86.2	89.5	93.1	91.9
S5 (paper results)	*90.1*	*62.2*	*89.3*	*91.4*	95.3	*98.6*

Table 8 | *Effects of normalization on linear diagonal RNNs with stable exponential parameterization (see §3.4). In bold is our best performing model, and we report the closely matching deep SSM results below. Tunings for our rings are also reported. Results showcase the advantage of taking initialization close to the unit circle under proper γ normalization. For PathX, we initialize eigenvalues to have a phase range of $[0, \pi/10]$, for all other tasks we use a range of $[0, 2\pi]$ (see §3.4).*

D.2. General experimental details

We use AdamW as our optimizer (Loshchilov and Hutter, 2017). We use warmup for the learning rate, where we start from a value of 10^{-7} and increase the learning rate linearly up a specified value for the first 10% of training. This is followed by cosine annealing for the rest of training down to a value of 10^{-7}.

We used a smaller learning rate for the RNN/SSM parameters A and B. When using normalization in our RNNs, we also used a smaller learning rate on the normalization parameter γ. For our S5 and S4D baselines, we used a smaller learning rate for the discretization step size Δ. This smaller learning rate was determined by multiplying the base learning rate by a factor < 1 (See Tb.9 for the learning rate factor used for each task).

We use weight decay for all parameters except the RNN/SSM parameters A and B (and γ and Δ when applicable).

All experiments were carried out on accelerated hardware A100 GPUs.

D.3. Hyperparameters

We closely followed the hyperparameter settings of the S5 model Smith et al. (2022) for all our experiments, with minimal additional tuning. For our S5 baseline, we tuned the model dimension H and state dimension N,

Task	Depth	H	N	Iterations	Batch size	LR factor	Weight Decay	Dropout
sCIFAR	6	512	384	180k	50	0.25	0.05	0.1
ListOps	6	128	256	80k	32	0.5	0.05	0.0
Text	6	256	192	50k	32	0.1	0.05	0.1
Retrieval	6	128	256	100k	64	0.5	0.05	0.1
PathFinder	6	192	256	500k	64	0.25	0.05	0.0
PathX	6	128	256	250k	32	0.25	0.05	0.0

Table 9 | *List of all the hyper-parameters used for each task for the LRU model.*

and used the optimal values for the LRU model as well. For the S4D baseline, we also tuned H and N. For all our experiments, we tuned the base learning rate on a logarithmic grid of 2 to choose the optimal learning rate. We present the hyperparameters we used for each LRU experiment in Tb.9.

D.4. Tasks

We use the 6 tasks in the Long Range Arena benchmark for our experiments (Tay et al., 2020), with the only difference being we use colored sCIFAR images instead of the grayscale sCIFAR images used in LRA.

E. Theoretical insights

We provide here theoretical groundings for some observations made in §3. We start by showing in §E.1 that, when interleaved with MLP blocks, stacked linear RNNs can model highly nonlinear dynamical systems. We provide two separate views that justify our findings: in §E.1.1, we provide a spectral explanation, while in §E.1.2 we present a function-space prespective. Our results, combined with the observation that nonlinear RNNs are difficult to optimize (§E.2), provide a justification for the results in Tb. 1. Next, motivated by the results in Tb. 3 we in discuss in the same section optimization of linear RNN blocks, and show that exponential reparameterization can accelerate training.

E.1. Expressivity of linear RNN stacks

In our sequence-to-sequence setting, it is a natural to seek models which (at least in the width limit) are able to map inputs u to outputs y (last layer) using a flexible nonlinear transition map T learned from data. Mathematically, a fully-expressive *causal* model should be able to approximate $y_k = T(u_k, u_{k-1}, \dots, u_1)$, where T is an arbitrary nonlinear map.

E.1.1. Spectral perspective

We show in this section how interleaving linear RNNs with MLPs in a deep architecture provides a flexible and modular recipe for the approximation of nonlinear transition maps.

Spectral limitations of linear RNNs. It is a standard result (Li et al., 2022b) that *linear* RNNs can approximate any shift-invariant *linear* map T. In continuous-time, on the spectral domain, this property is easier to study: let $Y(\omega)$ and $U(\omega)$ be the Fourier transforms for two continuous-time signals $u, y : \mathbb{R} \to \mathbb{R}$. If there exists a function $H : \mathbb{R} \to \mathbb{R}$ such that $Y(\omega) = H(\omega)U(\omega)$, then this can be approximated by a continuous-time linear RNN $\dot{x} = Ax + Bu$ for some coefficients $A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1}$, and the approximation can be made arbitrarily accurate as $N \to \infty$. However, one thing a linear RNN *cannot do* is store information under frequencies which are not present in the input signal: if the input is a sine wave of a certain frequency, the output will be a scaled and shifted sine wave of the *same frequency*.

Spectral effects of interleaving with MLPs. In our architecture (Fig.1) an activation function, as well as a linear position-wise layer, is placed right after each RNN output. As can be seen in Fig. 6, this operation causes spectral leakage: information gets copied over different frequency components.

The behavior shown in Fig. 6 can be characterized exactly:

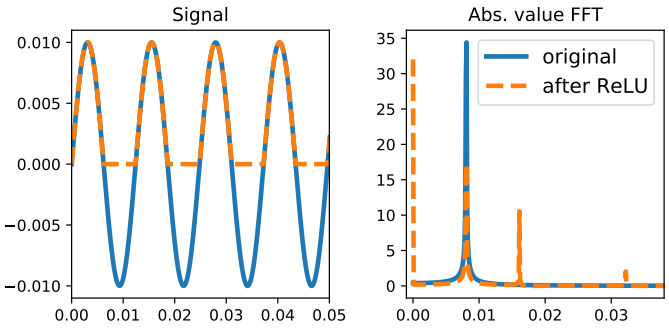

Figure 6 | *ReLU nonlinearity leaks information from the original signal to higher frequencies, as shown formally in Prop. E.1.*

Proposition E.1 (Spectral effect of ReLU)**.** *Let $u : \mathbb{R} \to \mathbb{R}$ be a continuous-time signal. Let P_i be the i-th region activated by the ReLU applied to u, and let us write $P_i = [p_i - L_i, p_i + L_i]$. Then*

$$\mathcal{F}_{ReLU(u)} = \mathcal{F}_u(\omega) \star \left[\sum_i 2L_i e^{-i\omega p_i} sinc(\omega L_i) \right]. \tag{8}$$

where $\mathcal{F}$ denotes the Fourier transform, $\star$ the convolution operation and $sinc(x) := \sin(x)/x$.

This result is simple to parse: the Fourier transform of a ReLU activated signal is equal to the Fourier transform before the ReLU, convolved with a kernel which transports information to higher frequencies — an operation which is *impossible* for linear RNNs, even as the width increases. As such, **introducing an MLP completes the list of requirements for approximations of a nonlinear transition map: frequencies can be scaled up and down arbitrarily by the RNN, and can then be translated in the space using the ReLU**. As depth increases, these operations can be combined in a modular fashion, leading to highly nonlinear dynamics using easy-to-learn linear blocks, interleaved with simple activations.

To conclude, we provide a proof for the proposition above.

Proof. Recall that multiplications in the time domain are convolutions in the frequency domain.

$$\begin{aligned}
u_1(t) \cdot u_2(t) &= \mathcal{F}_{U_1}^{-1}(t) \cdot \mathcal{F}_{U_2}^{-1}(t) && (9)\\
&= \left(\int_{-\infty}^{\infty} U_1(\nu) e^{i\nu t} d\nu \right) \cdot \left(\int_{-\infty}^{\infty} U_2(\xi) e^{i\xi t} d\xi \right) && (10)\\
&= \int_{-\infty}^{\infty} U_1(\nu) \left(\int_{-\infty}^{\infty} U_2(\xi) e^{i(\xi+\nu)t} d\xi \right) d\nu && (11)\\
&= \int_{-\infty}^{\infty} U_1(\nu) \left(\int_{-\infty}^{\infty} U_2(\omega - \nu) e^{i\omega t} d\omega \right) d\nu && (12)\\
&= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} U_1(\nu) U_2(\omega - \nu) d\nu \right) e^{i\omega t} d\omega && (13)\\
&= \mathcal{F}_{U_1 \star U_2}^{-1}(t). && (14)
\end{aligned}$$

Let now $u_1 = u$ and $u_2 = \chi(u_1 > 0)$, then $u_1 \cdot u_2 = \text{ReLU}(u)$. Next, let P_i be the i-th region activated by the ReLU, and let us write $P_i = [p_i - L_i, p_i + L_i]$. We can write $\chi(u_1 > 0) = \sum_i \chi_{[p_i - L_i, p_i + L_i]}$.

Recall now the following basic properties:

1. $\mathcal{F}_{x(t-t_0)}(\omega) = e^{-i\omega t_0} \mathcal{F}_{x(t)}(\omega)$.
2. The Fourier transform of a rectangular pulse between $-\tau$ and τ is $2\tau \cdot \text{sinc}(\omega\tau)$, where $\text{sinc}(x) = \sin(x)/x$.

Therefore, we have

$$\mathcal{F}_{\chi_{[p_i - L_i, p_i + L_i]}}(\omega) = e^{-i\omega p_i} \mathcal{F}_{\chi_{[-L_i, L_i]}}(\omega) = 2L_i e^{-i\omega p_i} \text{sinc}(\omega L_i). \tag{15}$$

This concludes the proof:

$$\mathcal{F}_{\mathrm{ReLU}(u)} = U \star \left[\sum_i 2L_i e^{-i\omega p_i} \mathrm{sinc}(\omega L_i) \right]. \tag{16}$$

□

E.1.2. Insights from Koopman operator theory

We show how Koopman operator theory (Koopman and Neumann, 1932), combined with recent advances in dynamic mode decomposition (Kutz et al., 2016; Schmid, 2010; Williams et al., 2015), can provide a solid theoretical foundation for understanding the class of functions that can be approximated by linear RNNs, interleaved with MLPs. Our notation and results are based on Korda and Mezić (2018); Mauroy et al. (2020).

Basic theory. Consider a discrete-time nonlinear dynamical system $x_{k+1} = S(x_k)$, where $S : \mathbb{R}^n \to \mathbb{R}^n$ is a sufficiently regular map. The Koopman operator $\mathcal{K}_S$ for the dynamical system S prescribes the evolution of any observable (measurement) $f : \mathbb{R}^n \to \mathbb{C}$:

$$(\mathcal{K}_S f)(x) := f(S(x)). \tag{17}$$

For instance, let us consider $n = 1$ and the observable $f(x) = \sin(x)$: the Koopman operator is the map that takes $\sin(\cdot) \overset{\mathcal{K}_S}{\mapsto} \sin(S(\cdot))$, i.e. *advances the measurement* f one step forward in time.
The crucial property of the Koopman operator is that it is **linear** and bounded (Mauroy et al., 2020): let f_1, f_2 be two observables, then

$$\mathcal{K}_S(\alpha f_1 + \beta f_2)(x) = (\alpha f_1 + \beta f_2)(S(x)) \tag{18}$$
$$= \alpha f_1(S(x)) + \beta f_2(S(x)) \tag{19}$$
$$= \alpha(\mathcal{K}_S f_1)(x) + \beta(\mathcal{K}_S f_2)(x). \tag{20}$$

If S is regular enough, i.e. if the Hilbert space of observables can be chosen such that $\mathcal{K}$ only has point spectrum, then the spectral theory of bounded linear operators in Hilbert spaces implies that $\mathcal{K}_S$ is diagonalizable — i.e. any observable f can be expanded in terms of eigenfunctions of $\mathcal{K}_S$, where the Koopman acts linearly. We recall the definition: $\phi_\lambda : \mathbb{C}^n \to \mathbb{C}$ is an eigenfunction of $\mathcal{K}_S$ with eigenvalue $\lambda \in \mathbb{C}$ if $\mathcal{K}_S \phi_\lambda = \lambda \phi_\lambda$ — i.e if the system measured on ϕ evolves linearly. Since the eigenfunctions of $\mathcal{K}_S$ form a basis for L_2, for any observable $f : \mathbb{C}^n \to \mathbb{C}$, there exist complex numbers $\nu_1, \nu_2, \cdots$ such that one can write (Mauroy and Mezić, 2016)

$$\mathcal{K}_S f(x) = \mathcal{K}_S \left(\sum_{j=1}^{\infty} \nu_j \phi_j \right)(x) = \sum_{j=1}^{\infty} \lambda_k \nu_j \phi_j(x). \tag{21}$$

Since also the identity measurement map $x \mapsto x$ can be decomposed into eigenfunctions of $\mathcal{K}_S$ coordinate-wise, we have the following: assuming $x_{k+1} = S(x_k)$, with $x \in \mathbb{R}^n$, for any $k \in \mathbb{N}$ we have

$$x_k = V \Lambda^k \Phi(x_0), \tag{22}$$

where, with slight abuse of notation, $\Phi : \mathbb{R}^n \to \mathbb{C}^\infty$ is a vector of functions with the j coordinate defined as $(\Phi)_j := x \mapsto \phi_j(x)$, and $V \in \mathbb{C}^{n \times \infty}$ (often named the Koopman modes matrix) is the infinite dimensional matrix such that, for the observable $f_i : x \mapsto x_i$, one has $f_i(x) = \sum_{j=1}^{\infty} V_{ij} \phi_j(x)$.

Basic Theory Summary. In essence, Koopman operator theory, provides the following guarantee: ***any sufficiently regular nonlinear autonomous dynamical system can be made linear under a high-dimensional nonlinear blow-up of the state-space. Sounds familiar? This is exactly what a wide MLP + Linear RNN can do***. Moreover, to take the system back to the original coordinate system, one just needs a linear projection with matrix V. In practice, for identification and diagnosis of nonlinear systems (e.g. in machanical engineering), this approach is used in a truncated version, where the finite class of dominant eigenfunctions is constructed by using the dynamic mode decomposition (DMD) algorithm from Hermite Polynomials (Kaiser et al., 2021; Schmid, 2010).

Extension to nonlinear systems with inputs. Several options exist for extending Koopman operator theory to systems with inputs (Kaiser et al., 2021; Korda and Mezić, 2020; Proctor et al., 2018; Surana, 2016). Here, we briefly outline the approach of (Korda and Mezić, 2020). Let $S : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ be a nonlinear function which evolves the state of the system as $x_{k+1} = S(x_k, u_k)$, where $(u_k)_{k=1}^{\infty} \in \ell_2(\mathbb{R}^m)$ is the input sequence. We wish to take this nonlinear dynamical system with inputs to linear form in the infinite-dimensional space of observables f of the form $\mathbb{R}^n \times \ell_2(\mathbb{R}^m) \to \mathbb{C}$. Let $\mathcal{L}$ denote the left shift operator $\tilde{u} = (u_0, u_1, \dots) \mapsto \mathcal{L}(\tilde{u}) = (u_1, u_2, \dots)$, then one can define the Koopman operator for any observable f as follows:

$$\mathcal{K}_S f(x, \tilde{u}) = f(S(x, u_0), \mathcal{L}(\tilde{u})). \tag{23}$$

This operator is again linear and bounded for regular enough S (Korda and Mezić, 2020) — hence the analysis in the autonomous setting carries out also in this case. In particular, using the notation in the last paragraph:

$$x_k = V \Lambda_{(x,u)}^k \Phi(x_0, \tilde{u}), \tag{24}$$

where $\Lambda_{(x,u)}$ is a diagonal complex infinite-dimensional matrix which contains the eigenvalues corresponding to the eigenfunctions of the extended state $\Phi(x_0, \tilde{u})$.

Implication for deep RNNs. In essence, Koopman operator theory, provides the following guarantee: *any regular nonlinear dynamical system is representable by a linear RNN after proper nonlinear reparameterization of the inputs* — which can be performed by an MLP. While we believe this connection is conceptually solid and gives substantial insights into our architecture, a quantitative discussion would require substantial technical efforts perhaps linked to recent contributions from the statistical learning community (Kostic et al., 2022).

E.2. Optimization of recurrent blocks

In this subsection we back-up some of our claims about optimization of linear RNNs with experimental findings on toy examples. Our purpose is to confirm validity of our intuition outside the deep learning setting, without architecture-dependent confounders: i.e on vanilla RNNs with one layer.

Recurrent nonlinearities slow down gradient descent. In §3 and §E.1 we showed how linear RNNs can be used as elementary recurrent blocks for the purpose of modeling complex nonlinear dynamics when stacked in deep architectures. Similarly, the results in (Li et al., 2022a) indicate that, to achieve S4 performance, one can equivalently replace the recurrent core with a collection of convolutions parametrized by filters. While a single-layer level, a (dense) RNNs (Eq.1) with tanh or sigmoid activation can express convolutions with filters (Wang et al., 2022), the results in Tb. 1 (and Fig. 1(a) in Wang et al. (2022)) indicate an advantage on test accuracy from dropping such nonlinearities in the recurrence — i.e. of making the RNN linear. Motivated by this, in Fig. 7 we consider the problem of learning a single one-dimensional convolution kernel with a single layer RNN, and compare performance of linear and tanh activations. The sequence length in this problem was 100, and our data consists in 32 input-output one-dimensional trajectories, where the output is the result of a convolution with the kernel of elements $h_k := \frac{1}{10} \exp(-0.015 \cdot k) \cos(0.04 \cdot k)^2$, which induces moderate-length dependencies in the data (see bump in the kernel in Figure 7 at $k = 70$). The 32 input sequences are generated sampling random a, c parameters on a range and have form $\sin(0.05 \cdot a \cdot k) \cos(0.05 \cdot c \cdot k)^2$. Outputs are generated by convolving each input by h. Learning is performed using the Adam optimizer (Kingma and Ba, 2014) with standard momentum parameters.

Interestingly, already on this simple task, linear RNNs outperforms the tanh variant even after careful tuning of the stepsize. While the input-output map the system had to approximate is linear (i.e. a convolution), this result still indicates that on deep architectures, where the MLPs interleaving RNNs can quickly perform position-wise nonlinearities lifting the function approximation class (see §E.1), linear RNNs are preferrable.

Benefits of exponential parameterization. Our experimental results in §3.3 indicete that linear RNN cores can be more effectively learned under exponential parameterization of the eiganvalues: $\lambda = \exp(-\nu + i\theta)$. To understand the reason behind this phenomenon, we go back at the classical (hard) problem of learning powers (Bengio et al., 1994), crucially linked with linear RNN models (see Eq. (4)). For a specific planted solution $\lambda^* = \lambda_r^* + i\lambda_i^* = \exp(-\nu^* + i\theta^*)$, we consider the problem of minimizing the loss $L(\hat{\lambda}) = \frac{1}{2}|\hat{\lambda}^k - (\lambda^*)^k|^2$, where $k = 100$ and $\hat{\lambda}$ is generated from two real parameters following standard (real + imaginary) or

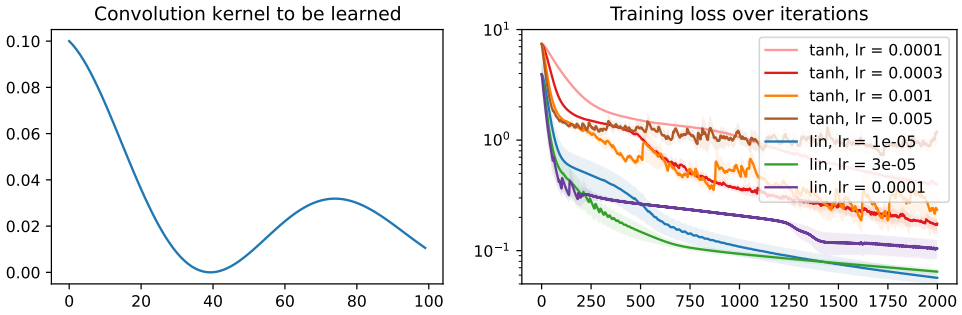

Figure 7 | *Learning with Adam a one-dimensional convolution with a length-*100 *kernel using a single-layer RNNs with linear or* tanh *recurrent activations and 100-dimensional hidden state. Initialization is performed using Glorot on all quantities for both options. For all learning rates in our grid, the linear variant is faster to converge.*

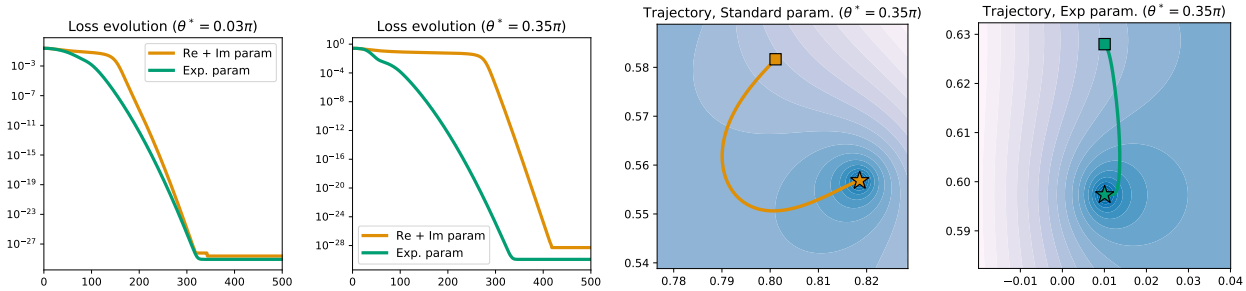

Figure 8 | *Exponential parametrization helps when learning a single complex eigenvalue* $\lambda^* = \exp(-\nu^* + i\theta^*)$*, exponentiated* 100 *times. As* λ^* *gets close to the purely imaginary setting* $\theta^* = \pi/2$*, the geometry of the loss landscape under standard real+imaginary parametrization becomes suboptimal for the Adam optimizer, which works best in the axis-aligned setting (exponential parametrization). In the plot, the square denotes initialization , while the star denotes the solution after 500 iterations.*

exponential parameterization. Note that in this paragraph $\lambda^* \in \mathbb{C}$ denotes the solution, not the complex conjugate of λ. In Fig. 8, we show that as the target phase θ^* approaches $\pi/2$ (i.e. λ^* gets close to the imaginary axis), standard parameterization slows down learning, as the corresponding landscape gets non-axis-aligned — a feature that does not match well the inner workings of the Adam optimizer[16], which is a diagonal preconditioner (Kingma and Ba, 2014). Instead, under exponential parameterization, the effects of phase and magnitude parameters on the powers of λ are more efficiently decouped: for example, while the real part of λ^k is simply $\exp(-k\nu)$ using exponential parameterization, if standard parameterization is used, $\mathrm{Re}\left[\lambda^k\right]$ is a function of both λ_r and λ_i. We noticed that the performance difference gets most pronounced when the system has to learn how to "turn": i.e. the initialization magnitude is correct, but the position on the complex plane is not (this is the precise setting for Figure 8): while for standard parameterization changing the phase θ^* requires a careful balance between real and imaginary components, for exponential parameterization gradients are fully aligned with the phase parameter. This makes the learning more flexible, a feature which we observed necessary in our experiments on the Long Range Arena, see §3.3 and Tb.2.

E.3. On alternatives to using complex numbers

In this subsection, we show how to derive the canonical real form for a non-symmetric real-valued matrix A, which we assume to be diagonalizable in the complex domain (always true up to arbitrary small perturbation of the entries (Axler, 1997)). This derivation is classical and can be found in many textbooks under the context of *real Jordan form* (more general), see e.g. Weintraub (2009). Here, we present a simplified discussion.

After diagonalizing A, we retrieve a set of purely real eigenvalues (each with multiplicity 1 up to vanishing perturbations) with corresponding *real* eigenvectors, and pairs of complex conjugate eigenvalues, with corresponding *complex conjugate* eigenvectors.

[16]For this problem, vanilla gradient descent cannot be effectively used as the landscape is highly non-convex, with challenging curvature vanishing as $|\lambda|$ approaces 0.

We recall a proof for the facts above: let * denote the elementwise complex conjugate of any complex quantity. This operation clearly commutes with multiplication. If $\lambda \in \mathbb{C}$ is an eigenvalue of $A \in \mathbb{R}^{N\times N}$ with eigenvector $v \in \mathbb{C}^N$, then since A is real-valued we have $Av^* = (A^*v)^* = (Av)^* = (\lambda v)^* = \lambda^* v^*$. Hence, λ^* is an eigenvalue with eigenvector v^*. This also shows that there always does exist a real eigenvector corresponding to each real eigenvalue: let $v \in \mathbb{C}^N$ be a complex eivengvector with real eigenvalue λ, then $v + v^* \in \mathbb{R}^N$ is an eigenvector with eigenvalue λ since, again using the fact that A is real, $A(v + v^*) = Av + Av^* = Av + (Av)^* = \lambda(v + v^*)$.

The action of A on its real eigenvectors (with real eigenvalues) is trivial and analogous to the symmetric case — this corresponds to a diagonal entry in the diagonalized version of A. For the subspaces spanned by complex eigenvalues, the discussion is more interesting: let λ, λ^* be a pair of conjugate eigenvalues with corresponding eigenvectors v, v^*. Collect v, v^* in a $N \times 2$ matrix V, then

$$AV = V \begin{pmatrix} \lambda & 0 \\ 0 & \lambda^* \end{pmatrix} =: V\Lambda \tag{25}$$

Let us now choose a different *real* basis for the columns of V, the real and imaginary parts of v: $\tilde{V} = [\mathrm{Re}(v), \mathrm{Im}(v)]$. Note that this is a basis, since v, v^* are linearly independent and can be both written as (complex-weighted) linear combination of real and imaginary parts of v. Now note that

$$\begin{aligned} A \cdot \mathrm{Re}(v) &= \frac{1}{2} A(v + v^*) \\ &= \frac{1}{2} (\lambda v + \lambda^* v^*) \\ &= \mathrm{Re}(\lambda v) \\ &= \mathrm{Re}\left[(\mathrm{Re}(\lambda) + i\mathrm{Im}(\lambda))(\mathrm{Re}(v) + i\mathrm{Im}(v))\right] \\ &= \mathrm{Re}(\lambda)\mathrm{Re}(v) - \mathrm{Im}(\lambda)\mathrm{Im}(v). \end{aligned}$$

Similarly,

$$\begin{aligned} A \cdot \mathrm{Im}(v) &= \frac{1}{2} A(v - v^*) \\ &= \frac{1}{2} (\lambda v - \lambda^* v^*) \\ &= \mathrm{Im}(\lambda v) \\ &= \mathrm{Im}\left[(\mathrm{Re}(\lambda) + i\mathrm{Im}(\lambda))(\mathrm{Re}(v) + i\mathrm{Im}(v))\right] \\ &= \mathrm{Re}(\lambda)\mathrm{Im}(v) + \mathrm{Im}(\lambda)\mathrm{Re}(v). \end{aligned}$$

This shows that the action of A on the new *real* basis $\tilde{V}$ is of simple form:

$$A\tilde{V} = \tilde{V} \begin{pmatrix} \mathrm{Re}(\lambda) & -\mathrm{Im}(\lambda) \\ \mathrm{Im}(\lambda) & \mathrm{Re}(\lambda) \end{pmatrix} =: \tilde{V}\tilde{\Lambda} \tag{26}$$

This discussion shows that there exist a simple invertible change of basis (from V to $\tilde{V}$ for all pairs of conjugate eigenvalues) which makes takes the system back to a simple decomposition in the real domain, both in terms of eigenvalues and eigenvectors — one simply has to replace all diagonal blocks of form Λ with 2×2 matrices $\tilde{\Lambda}$.

The careful reader might recognize that, in the resulting system, matrix multiplication for the 2×2 blocks is algebraically equivalent to multiplication of the corresponding complex numbers. Hence, while complex numbers are not *per-se* needed to find a simple representation of non-symmetric matrices, they are convenient to work with since the matrix in Eq. (26) is structured: has 4 entries but can be represented using just two — real and imaginary parts, exactly what a complex number stores in memory.

F. Proofs

In this section we provide proofs for the propositions listed in the main paper.

F.1. Proof of Lemma 3.2

We provide here a proof for the following sampling lemma.

Lemma 3.2. *Let u_1, u_2 be independent uniform random variables on the interval $[0, 1]$. Let $0 \leq r_{\min} \leq r_{\max} \leq 1$. Compute $\nu = -\frac{1}{2}\log\left(u_1(r_{\max}^2 - r_{\min}^2) + r_{\min}^2\right)$ and $\theta = 2\pi u_2$. Then $\exp(-\nu + i\theta)$ is uniformly distributed on the ring in $\mathbb{C}$ between circles of radii $r_{\min}$ and $r_{\max}$.*

Proof. First, note that one can sample phase and magnitude independently by symmetry of the target distribution. Phase sampling can trivially performed through scaling a uniform distribution.

Next, we consider sampling the magnitude. The area of the ring in between $r_{\min}$ and $r_{\max}$ is $\pi(r_{\max}^2 - r_{\min}^2)$, while the cumulative distribution function for the radius distribution is such that $F_r(r_{\min}) = 0$, $F_r(r_{\max}) = 1$ and for $r \in [r_{\min}, r_{\max}]$ we therefore have

$$F(r) = \frac{r^2 - r_{\min}^2}{r_{\max}^2 - r_{\min}^2}. \tag{27}$$

Under parametrization of r using the exponential, $r = e^{-\nu}$, one gets

$$F(r) = \frac{e^{-2\nu} - r_{\min}^2}{r_{\max}^2 - r_{\min}^2}. \tag{28}$$

Finally, we use the inverse sampling theorem (see e.g. Vogel (2002)): one can sample ν using the formula $\nu = F^{-1}(u)$, where u is uniform on $[0, 1]$. By setting

$$u = \frac{e^{-2\nu} - r_{\min}^2}{r_{\max}^2 - r_{\min}^2}, \tag{29}$$

we get

$$e^{-2\nu} = (r_{\max}^2 - r_{\min}^2)u + r_{\min}^2, \tag{30}$$

from which follows that $\nu = -\frac{1}{2}\log((r_{\max}^2 - r_{\min}^2)u + r_{\min}^2)$. □

F.2. Proof of Proposition 3.3

Validity of this proposition is verified numerically in Figure 9.

Proposition 3.3 (Forward-pass blow-up)**.** *Let Λ be diagonal with eigenvalues sampled uniformly on the ring in $\mathbb{C}$ between circles of radii $r_{\min} < r_{\max} < 1$. Then, under constant or white-noise input and Glorot input projection, we have that the squared norm of the state x_k converges as $k \to \infty$ to the following quantity.*

$$\mathbb{E}[\|x_\infty\|_2^2] = \frac{1}{r_{\max}^2 - r_{\min}^2} \log\left(\frac{1 - r_{\min}^2}{1 - r_{\max}^2}\right) \mathbb{E}[\|Bu\|_2^2].$$

Proof. Assume first (most difficult case) that u_k is constant, i.e. such that $Bu_k =: \tilde{u}$ for all k. Then,

$$\|x_\infty\|_2^2 = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} \tilde{u}_{k-m}^* \left(\Lambda^m\right)^* \Lambda^n \tilde{u}_{k-n} \tag{31}$$

$$= \tilde{u}^* \left[\sum_{n=1}^{\infty} \sum_{m=1}^{\infty} \left(\Lambda^m\right)^* \Lambda^n\right] \tilde{u}. \tag{32}$$

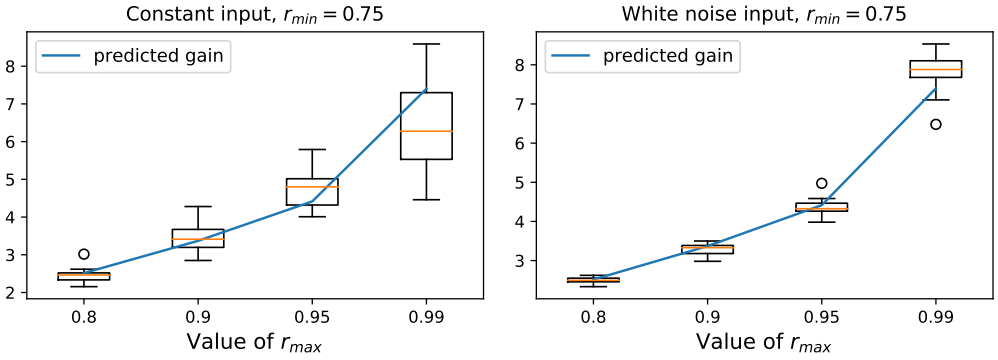

Figure 9 | *Numerical simulation for gain formula derived in Proposition 3.3. Here we chose* $N = 500$, $L = 10k$ *(sequence length) and plotted statistics for 10 runs with boxplot indicating median and (5,95) percentile. Indicated in blue line is our prediction. The formula holds both for constant and random input, yet we notice that it is more accurate in the random input setting.*

Note that $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_N)$ is diagonal with equally distributed entries on the disk between radii $r_{\min}$ and $r_{\max}$. One can then sample a generic entry λ using the change of variables formula for probabilities (Jeffreys, 1998) as follows (see also Lemma 3.2):

$$\lambda = r^{\frac{1}{2}} e^{i2\pi\theta}, \quad r \sim \mathcal{U}[r_{\min}^2, r_{\max}^2], \quad \theta \sim \mathcal{U}[0,1], \tag{33}$$

Where crucially r and θ are independent. Let $\mathbb{T}(r_{\min}, r_{\max}) = \{\lambda \in \mathbb{C} : |\lambda| \in [r_{\min}, r_{\max}]\}$. We need to study the following quantity:

$$\mathbb{E}_{\lambda\sim\mathbb{T}(r_{\min},r_{\max})}\left[\sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\lambda^n(\lambda^m)^*\right] = \mathbb{E}_{r,\theta}\left[\sum_{n=1}^{\infty}\sum_{m=1}^{\infty} r^{\frac{1}{2}(n+m)} e^{i2\pi(n-m)\theta}\right] \tag{34}$$

$$= \sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\mathbb{E}_r\left[r^{\frac{1}{2}(n+m)}\right]\mathbb{E}_\theta\left[e^{i2\pi(n-m)\theta}\right] \tag{35}$$

The expectation w.r.t θ is non-zero only if $n = m$, therefore

$$\mathbb{E}_{\lambda\sim\mathbb{T}(r_{\min},r_{\max})}\left[\sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\lambda^n(\lambda^m)^*\right] = \sum_{n=1}^{\infty}\mathbb{E}_r\left[r^n\right] \tag{36}$$

$$= \mathbb{E}_r\left[\sum_{n=1}^{\infty} r^n\right] \tag{37}$$

$$= \mathbb{E}_r\left[\frac{1}{1-r}\right] \tag{38}$$

$$= \frac{1}{r_{\max}^2 - r_{\min}^2}\int_{r_{\min}^2}^{r_{\max}^2}\frac{1}{1-r}dr \tag{39}$$

$$= \frac{1}{r_{\max}^2 - r_{\min}^2}(-\log(|1-r_{\max}^2|) + \log(|1-r_{\min}^2|)) \tag{40}$$

$$= \frac{1}{r_{\max}^2 - r_{\min}^2}\log\left(\frac{1-r_{\min}^2}{1-r_{\max}^2}\right). \tag{41}$$

The *white noise input* case is simpler. Let us start from $\|x_\infty\|_2^2 = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\tilde{u}_{k-m}^*(A^m)^* A^n \tilde{u}_{k-n}$. Now, we can retrieve the single sum by the fact that A is diagonal and $\mathbb{E}[\tilde{u}_{k-m}^*\tilde{u}_{k-n}] = 0$ for $m \neq n$. The rest of the proof is identical, and presented in the main paper for the one-simensional setting. □