On the Role of Depth in the Expressivity of RNNs

Maude Lizaire
Mila & DIRO
Université de Montréal

Michael Rizvi-Martel
Mila & DIRO
Université de Montréal

Éric Dupuis
Independent

Guillaume Rabusseau
Mila & DIRO, CIFAR
Université de Montréal

Abstract

The benefits of depth in feedforward neural networks are well known: composing multiple layers of linear transformations with nonlinear activations enables complex computations. While similar effects are expected in recurrent neural networks (RNNs), it remains unclear how depth interacts with recurrence to shape expressive power. Here, we formally show that depth increases RNNs' memory capacity efficiently with respect to the number of parameters, thus enhancing expressivity both by enabling more complex input transformations and improving the retention of past information. We extend our analysis to 2RNNs, a generalization of RNNs with multiplicative interactions between inputs and hidden states. Unlike RNNs, which remain linear without nonlinear activations, 2RNNs perform polynomial transformations whose maximal degree grows with depth. We further show that multiplicative interactions cannot, in general, be replaced by layerwise nonlinearities. Finally, we validate these insights empirically on synthetic and real-world tasks.

1 INTRODUCTION

It is well known that Feedforward neural networks (FNNs) are universal approximators [Hornik et al., 1989, Cybenko, 1989]. In practice however, their capacity is limited as they may require impractically large hidden sizes to compute complex functions. That is where the benefit of depth comes to play. Indeed, stacking multiple layers results in a composition of nonlinear transformations, and it is understood that increasing the number of such compositions allows the network to represent more complex functions [Telgarsky, 2016, Rolnick and Tegmark, Montufar et al., 2014, Håstad and Goldmann, 1991]. In the absence of nonlinearities, however, deep feedforward neural networks essentially collapse into shallow ones, as the composition of linear functions results in just another linear transformation.

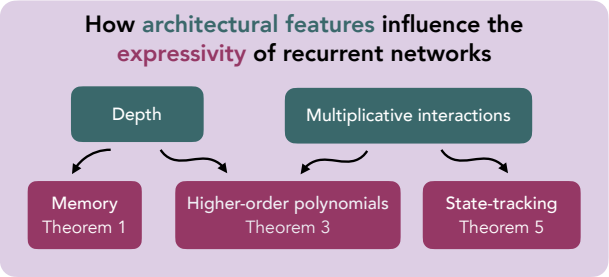

Figure 1: Theoretical insights overview of the effects of architectural choices on the expressivity of RNNs.

A similar situation occurs in neural networks computing functions over sequences. Recurrent neural networks (RNNs) are known to be Turing complete [Siegelmann and Sontag, 1992, 1994, Siegelmann, 1996], but their remarkable expressive power relies on impracticable assumptions, namely infinite precision or unbounded computation time [Weiss et al., 2018]. The impact of depth in recurrent networks is however less straightforward than in feedforward networks. While the benefits of depth in FNNs can be extended to RNNs by simply ignoring recurrence, this overlooks the complex dynamics introduced by recurrent connections. Indeed, in sequence modeling, the expressivity of a model involves not only to transform inputs in meaningful representations, but also the capacity to propagate and combine information through time in useful ways. In order to fundamentally understand how depth influences the expressive power of RNNs over sequences, the subtle interplay between recurrence and depth ought to be examined.

In this work, we investigate how depth influences the expressivity of RNNs. First, we focus on *linear* RNNs

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

to isolate the interplay between recurrence and depth from the expressive gain arising by composing nonlinear activations. We formally show that deep linear RNNs are strictly more expressive than shallow ones as they have a greater ability to memorize information. Furthermore, we prove that increasing depth, rather than hidden size, is a more parameter-efficient approach to enhance the network's memory.

Second, we explore how the effect of depth manifests in models with multiplicative interactions between inputs and hidden states, which we refer to as second-order recurrent neural networks (2RNNs). Here, stacking layers has a similar effect as the composition of nonlinear activations: it expands the class of functions the model can represent. Specifically, linear 2RNNs compute polynomials of their inputs, whose degree increases with the number of layers. As a result, deep linear 2RNNs are strictly more expressive than their shallow counterparts. We further consider models whose bilinear terms are parameterized by a CP decomposition, CPRNNs, and show that depth does not alter the expressive gain obtained by increasing the rank of the decomposition.

Third, we investigate how the gain in expressive power from stacking nonlinear activations differs from the one provided by multiplicative interactions, showing that there exist functions computable by single-layer 2RNNs (specifically those requiring state-tracking) that cannot be realized by deep RNNs with nonlinear activations applied only depth-wise.

We study how these theoretical findings translate in practice with gradient descent optimization through synthetic and real data experiments on RNNs, 2RNNS and SSMs (S4). Empirically, RNNs capacity to memorize and copy information with respect to depth supports our theoretical analysis, even when nonlinearity is added. When tested on parity, a task not requiring memory, but rather the ability to state-track via temporal multiplicative interactions, and found that the impact of depth is highly dependent on the way nonlinearities are applied (i.e. recurrently or only in depth). On real datasets, whether performing language modeling on tiny Shakespeare or testing Long Rang Arena benchmarks, we find that depth improves performance quite consistently, while the parameter efficiency benefit is task-dependent as predicted theoretically.

Our contributions can be summarized as follows:

- Our theoretical analysis reveals that even in the absence of nonlinearities, depth strictly increases the expressivity of RNNs (Theorem 1). More precisely, it enlarges the hidden capacity of the network, and we show that for certain tasks, such as those requiring memory, depth provides a parameter-efficient means to improve expressivity (Theorem 2).
- Whereas linear RNNs compute only linear transformations of their inputs regardless of depth, we show that in 2RNNs, increasing the number of layers directly enables the computation of higher-order polynomials (Theorem 3). This result extends to CPRNNs, and we further show that depth does not alter the effect of the rank on the network's capacity (Theorem 4).
- We establish a fundamental separation between the effects of nonlinear activations and multiplicative interactions, by showing that certain functions computed by single-layer 2RNNs cannot be realized by deep RNNs with nonlinear activations only in depth (Theorem 5).
- Finally, we provide a diverse set of experiments that validates and illustrates the insights revealed by our theoretical analysis*.

Related work RNNs [Elman, 1990] are a natural choice for sequence modeling. Many gated variants of such models were introduced to mitigate vanishing/exploding gradient issues [Hochreiter and Schmidhuber, 1997, Cho et al., 2014, Chung et al., 2014]. Recently, the advent of SSMs has brought recurrent models back into the limelight [Gu et al., 2021, Gu and Dao, 2023, Gu et al., 2020, Nguyen et al., 2022, Gupta et al., 2022]. This renewed interest has prompted recent studies on the expressivity and limitations of SSMs [Wang and Xue, 2023, Wang et al., 2024, Grazzi et al.]. In particular, it was shown that these models, who only have nonlinear activations depth-wise cannot state-track [Merrill et al., 2024]. Early research on expressive power of sequence models [Siegelmann and Sontag, 1992, Steijvers and Grünwald, 2019, Bodén et al., 1999] showed that recurrent models can recognize regular and other formal languages and are even Turing complete (with unbounded precision and compute time). More recently, novel hierarchies have been used to analyze the expressivity of RNNs [Merrill et al., 2020] and theoretical work have highlighted fundamental differences between RNNs and transformers [Bhattamishra et al., 2024]. Previous work has largely overlooked the role of depth in RNNs, which is the focus of this study. In deep FNNs, depth has been shown to provide exponential gains in efficiency for certain functions [Eldan and Shamir, 2016, Telgarsky, 2015]. Similarly, depth in CNNs has been extensively studied using tensor network analysis [Cohen and Shashua, 2016a, Cohen et al., 2016, Cohen and Shashua, 2016b, Sharir and Shashua, 2017, Alexander et al., 2024, Razin et al., 2024]. For RNNs, analysis based on surrogate tensor decomposition models demonstrated the benefit

*Code base for this paper can be found at `https://github.com/MaudeLiz/Role_of_Depth_in_RNNs`

of depth, even for tasks that do not share a sequential inductive bias [Khrulkov et al., 2017, 2019, Levine et al., 2018]. Finally, second-order interactions have been used to make deep learning models both more interpretable [Pearce et al., 2024] and more expressive [Irsoy and Cardie, 2014, Jayakumar et al., 2020, Cheng et al., 2024]. This led to the introduction of various multiplicative RNN architectures [Tjandra et al., 2016, Krause et al., 2017, Sutskever et al., 2011, Wu et al., 2016, Su et al., 2024]. The expressivity of 2RNNs and their variants has been the object of theoretical studies [Li et al., 2022, Lizaire et al., 2024].

2 PRELIMINARIES

We begin by introducing the notation used in this work as well as the models studied.

Notation Our notation conventions are as follows: vectors, matrices and higher-order tensors are respectively denoted by bold lowercase letters $\mathbf{v} \in \mathbb{R}^d$, bold uppercase letters $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$, and bold calligraphic letters $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_p}$. A diagonal matrix with vector $\mathbf{v}$ on the diagonal is noted $\text{diag}(\mathbf{v})$. The n-mode product of a tensor with a vector[†] is defined by: $(\boldsymbol{\mathcal{T}} \times_n \mathbf{v})_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_p} = \sum_{i_n=1}^{d_n} \boldsymbol{\mathcal{T}}_{i_1,\ldots,i_p} \mathbf{v}_{i_n}$. Lastly, $[n]$ denotes the set of integers from 1 to n and $\lceil x \rceil$ is the smallest integer greater or equal to x.

Models We present formal definitions of RNNs and 2RNNs, then we introduce CPRNNs and BIRNNs based on how they relate to the definitions provided.

Definition 1 (RNN)**.** *A Recurrent Neural Network of depth L and hidden size n is parameterized by initial hidden state vectors $\mathbf{h}_0^{(l)} \in \mathbb{R}^n$, weight matrices $\mathbf{U}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{n \times n}$ (except $\mathbf{U}^{(1)} \in \mathbb{R}^{n \times d}$), bias terms $\mathbf{b}^{(l)} \in \mathbb{R}^n$ and activation functions $\sigma^{(l)} : \mathbb{R}^n \to \mathbb{R}^n$ for $l \in [L]$. For any sequence length T, an RNN maps a sequence of inputs $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^d$, to a sequence of hidden states $\mathbf{h}_1^{(L)}, \ldots, \mathbf{h}_T^{(L)} \in \mathbb{R}^n$ via the following computation at each time step $t \in [T]$ and layer $l \in [L]$ with $\mathbf{h}_t^{(0)} = \mathbf{x}_t$ for all t:*

$$\mathbf{h}_t^{(l)} = \sigma^{(l)}(\mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{U}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)}). \quad (1)$$

The activation functions are applied element-wise. They generally consist of nonlinear functions such as tanh or ReLU. In this work, we focus on linear architectures, where $\sigma^{(l)}$ is the identity for all l. We call these *linear RNNs*. Section 3.3 will also study RNNs with nonlinear activations applied only in depth:

$$\mathbf{h}_t^{(l)} = \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{U}^{(l)}\sigma^{(l-1)}\left(\mathbf{h}_t^{(l-1)}\right) + \mathbf{b}^{(l)}. \quad (2)$$

[†]Note that this notation differs from the one in the reference [Kolda and Bader, 2009] where $\times_n$ denotes the n-mode product of a tensor with a *matrix*.

Second-order RNNs are obtained by adding a bilinear term between inputs and hidden states to Definition 1.

Definition 2 (2RNN)**.** *A Second-order Recurrent Neural Network of depth L and hidden size n is parameterized by initial hidden state vectors $\mathbf{h}_0^{(l)} \in \mathbb{R}^n$, weight tensors $\boldsymbol{\mathcal{A}}^{(l)} \in \mathbb{R}^{n \times n \times n}$ (except for $\boldsymbol{\mathcal{A}}^{(1)} \in \mathbb{R}^{n \times d \times n}$), weight matrices $\mathbf{U}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{n \times n}$ (except $\mathbf{U}^{(1)} \in \mathbb{R}^{n \times d}$), bias terms $\mathbf{b}^{(l)} \in \mathbb{R}^n$ and activation functions $\sigma^{(l)} : \mathbb{R}^n \to \mathbb{R}^n$ for $l \in [L]$. For any sequence length T, a 2RNN maps a sequence of inputs $\mathbf{x}_1, \ldots, \mathbf{x}_T$ to a sequence of hidden states $\mathbf{h}_1^{(L)}, \ldots, \mathbf{h}_T^{(L)}$ via the following computation at each time step $t \in [T]$ and layer $l \in [L]$:*

$$\sigma^{(l)}(\boldsymbol{\mathcal{A}}^{(l)} \times_1 \mathbf{h}_{t-1}^{(l)} \times_2 \mathbf{h}_t^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{U}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)}).$$

We also consider CPRNNs, which provide a practical alternative to 2RNNs whose weight tensors $\boldsymbol{\mathcal{A}}^{(l)}$ scale cubically. It consists in parameterizing the bilinear term using a CP decomposition instead of the full third-order tensor. Thus, $\boldsymbol{\mathcal{A}}^{(l)}$ is replaced by a sum of outer products $\sum_{r=1}^{R} \mathbf{a}_r^{(l)} \circ \mathbf{b}_r^{(l)} \circ \mathbf{c}_r^{(l)}$ with $\mathbf{a}_r^{(l)}, \mathbf{b}_r^{(l)}, \mathbf{c}_r^{(l)} \in \mathbb{R}^n$ for all $l \in [L]$ (expect $\mathbf{b}_r^{(1)} \in \mathbb{R}^d$). The rank R is an additional hyperparameter of the model. For a comprehensive formal definition of this architecture, we refer the reader to [Lizaire et al., 2024]. Finally, we call BIRNNs second-order RNNs with only a bilinear term (i.e. $\mathbf{U}^{(l)}, \mathbf{V}^{(l)}$ and $\mathbf{b}^{(l)}$ are all null). Similarly, CPRNNs with only their multiplicative term are CP-BIRNNs, and we use the notation CP(BI)RNNs when both CPRNNs and CPBIRNNs are referenced at the same time. Figure 2 presents how recurrent architectures unroll across depth and time. It also illustrates that in linear RNNs, the overall computation reduces to a linear map of the inputs, whereas in second-order RNNs (2RNNs), it corresponds to a polynomial function which will be discussed in Sections 3.1 and 3.2.

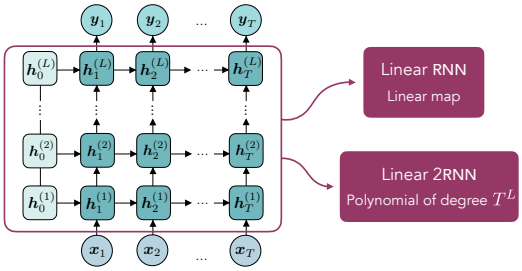

Figure 2: Unrolled deep recurrent architecture. Linear RNNs compute linear mappings of the inputs, while linear 2RNNs produce polynomial ones.

3 THEORETICAL RESULTS

To analyze the effect of depth on the expressivity of RNNs, we introduce a formal definition of the set of functions they can represent.

Definition 3. *For any $n \geq 1$ and $L \geq 1$, $\mathcal{H}_{\text{RNN}}(n, L)$ denotes the set of functions h mapping input sequences of arbitrary length to the corresponding hidden state sequences computed by RNNs of hidden size n and depth L, as given by Def. 1: $h(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T) = (\mathbf{h}_1^{(L)}, \mathbf{h}_2^{(L)}, ..., \mathbf{h}_T^{(L)})$.*

Similarly we define $\mathcal{H}_{\text{2RNN}}(n, L)$ for 2RNNs based on Def. 2 and $\mathcal{H}_{\text{CPRNN}}(n, L, R)$ for CPRNNs of rank R.

3.1 Role of Depth in (linear) RNNs: Increasing Memory Efficiently

In this section, we focus on linear RNNs, since RNNs with nonlinear activations naturally inherit the same benefits from composing nonlinearities as FNNs. We begin our analysis by observing that linear RNNs perform *linear* transformations of their inputs, regardless of the network's depth (See Appendix A.2 for formalization of this statement). Although this observation is reminiscent of the equivalence between deep and shallow linear FNNs, for linear RNNs, there is no collapse of the structure to a single-layer network. In fact, the following theorem states that depth makes linear RNNs strictly more expressive. More precisely, even though linear RNNs compute linear transformations, adding a layer will always make the model strictly more expressive, i.e. able to compute (linear) functions that could not be computed with one layer less.

Theorem 1. *For any $n > 1$ and $L \geq 1$, $\mathcal{H}_{\text{RNN}}(n, L) \subsetneq \mathcal{H}_{\text{RNN}}(n, L+1)$ for linear RNNs.*

It is worth taking a moment to contemplate why this result is non-trivial and surprising. Indeed, composing linear functions does not increase expressiveness; the composition remains linear. It is the sequential nature of the model which leads to the strict inclusion in Theorem 1. Intuitively, adding a layer to an RNN increases its memory capacity, and the network can thus retain information longer. Although only the hidden vector of the last layer is outputted, the network maintains L hidden vectors, allowing the information to be staged and propagated towards the deeper layers at later time steps.

Sketch of proof Proving the inclusion simply consists in finding an explicit parameterization for a model of depth $L+1$ to reproduce the computation of depth L RNN. To show strict inclusion, i.e. $\mathcal{H}_{\text{RNN}}(n, L) \not\supset \mathcal{H}_{\text{RNN}}(n, L+1)$, we look at the capacity of a linear RNN to memorize and propagate information. We thus introduce the function f_p that copies the first component of the input $\mathbf{x}_t$ p steps forward:

$$f_p(\mathbf{x}_1, \cdots, \mathbf{x}_t) = \begin{cases} (\mathbf{x}_{t-p})_1 & \text{if } t - p > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

First, we give an explicit construction (illustrated in Figure 3) proving that for any p and any n, there exists an RNN of hidden size n and depth $L = \lceil p/(n-1) \rceil$ that computes f_p. Then, looking at the RNN as a flow graph in which information is propagated, we derive an upper bound on the value of p for which an RNN of hidden dimension n and depth L can compute f_p, namely p can be at most $L(n-1)$. Combining these two facts together, we obtain that for any n and L, by setting $p = (L+1)(n-1)$, f_p can be computed with $L+1$ layers of n neurons but not with L layers.

We observe that while the increased memory is the key element of our proof, there may exist other forms of expressivity gains coming from depth in linear RNNs.

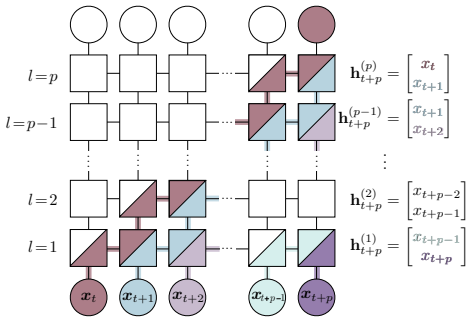

Figure 3: Information flow of RNN's computation of f_p for $n = 2$ and $L = p$.

Since the proof of strict inclusion for Theorem 1 relies on increasing memory capacity via the additional hidden units of the extra layer, a natural next step is to compare deep and shallow RNNs with latent representations of the same dimension, i.e., the same total number of hidden units. Concretely, we compare a deep network of hidden dimension n and depth L to a shallow network with hidden dimension nL. The following proposition offers a perspective that differs from Theorem 1.

Proposition 1 (informal[‡]). *For $n > 1$ and $L > 1$, a single-layer linear RNN of hidden size nL has a greater latent representational capacity than a linear RNN of size n and depth L.*

This proposition indicates that, provided the same hidden capacity (i.e., number of hidden units), a shallow linear RNN is strictly more expressive than a deeper one. This result provides insight into the inner mechanics of RNNs. The latent representation of the shallow network is a vector in $\mathbb{R}^{nL}$, whereas that of the deep network is a concatenation of L vectors in $\mathbb{R}^n$. This block structure imposed by depth constrains the expressive power of the latent representation, giving the shallow network a larger representational capacity, which is in direct contrast with Theorem 1.

[‡]See Appendix A.4 for formal statement and its proof.

However, this gain in capacity comes at a higher parameter cost. While the number of hidden units grows linearly with both hidden size and depth, the number of parameters in an RNN increases linearly with L but quadratically with n. The last theorem of this section shows that, if we consider parameter counts instead of number of hidden units, increasing depth can lead to a strict gain in expressiveness. More precisely, the theorem shows that, for any depth, as soon as the hidden size is large enough, a linear RNN can compute functions that cannot be computed by shallower RNN with (at most) the same number of parameters.

Theorem 2. *Considering linear RNNs with input dimension $d = 1$, let $\#\text{params}(n, L)$ denote the number of parameters of a L-layers RNN with hidden size n* [§]*.*

For any depth L and any hidden size $n \geq 4$, there exists a function $f \in \mathcal{H}_{\text{RNN}}(n, L)$ such that, for all $\tilde{L} < L$ and $\tilde{n}$, if $f \in \mathcal{H}_{\text{RNN}}(\tilde{n}, \tilde{L})$ then $\#\text{params}(\tilde{n}, \tilde{L}) > \#\text{params}(n, L)$.

Sketch of proof To prove this theorem, we show that the difference in number of parameters $\#\text{params}(\tilde{n}, \tilde{L}) - \#\text{params}(n, L)$ required to compute the copy function f_p introduced in Equation 3 with $p = (L+1)(n-1)$ evolves as an upward parabola with respect to n, and thus will be positive as soon as $n \geq 4$. The complete proof is in the appendix.

Overall, Theorem 2 states that there are functions, in particular those requiring memory, for which increasing the number of layer is the parameter-efficient approach to increase the capacity of the network. However, the optimal balance between number of layers and hidden size depends on the task at hand. It is worth emphasizing that the gain in efficiency brought by depth is, in essence, independent of the activation of the network. Therefore, Theorem 2 can be relevant to other recurrent architectures. Figure 4 summarizes the theoretical findings on linear RNNs presented in this section.

3.2 Role of Depth in BIRNNs: Increasing Complexity via Higher-Order Interactions

We now turn to analyzing the benefits of depth in second-order RNNs. As in the previous section, we consider linear architectures to abstract away the effect of composing nonlinear activation functions. Additionally, in order to isolate the effect of multiplicative interactions, our formal results focus on linear BIRNNs, rather than 2RNNs. We begin by observing that even in the absence of nonlinear activations, shallow BIRNNs compute polynomial functions of their inputs (See App. A.6 for formal statement). In contrast with linear RNNs,

[§]One can check that $\#\text{params}(n, L) = (2L-1)n^2 + (L+1)n$ (excluding initial hidden states as parameters count, though including them would not change the result).

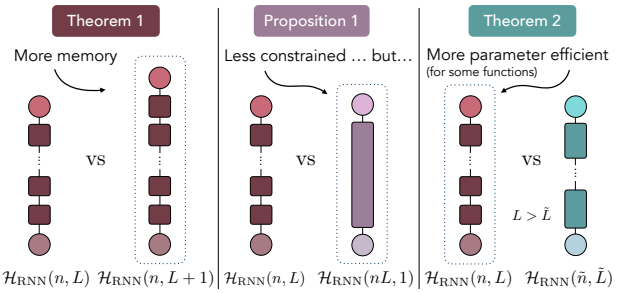

Figure 4: Summary of theoretical results on linear RNNs: Adding layers increases memory capacity (Thm. 1), a single layer (for fixed number of units) offers more flexibility (Prop. 1), but raising depth rather than width (n) can be more parameter-efficient (Thm. 2).

which compute linear transformations regardless of the network depth, increasing the number of layers in BIRNNs broadens the class of functions they can represent. Specifically, one can show that the maximum degree of the resulting polynomials grows exponentially with depth. Thus, by increasing the number of layers in a BIRNN we can compute higher-order polynomials. This is the key observation to prove the following theorem showing that, similarly to RNNs, there is a strict gain in expressivity with depth in linear BIRNNs.

Theorem 3. *For $n > 1$ and $L \geq 1$, $\mathcal{H}_{\text{BIRNN}}(n, L) \subsetneq \mathcal{H}_{\text{BIRNN}}(n, L+1)$ for linear BIRNNs.*

Sketch of proof To prove inclusion, we show that a BIRNN with $L+1$ layers can produce in its L^{th} layer the same latent vectors $\mathbf{h}_t^{(L)}$'s as a BIRNN with L layers, and transfer these hidden states without further modification to the last layer. The proof of the strict inclusion relies on showing that in BIRNNs of depth L, the polynomial dependency of the second hidden vectors in the first inputs is of degree at most L. We then consider a specific case that achieves the upper bound for a network of depth $L+1$, and conclude that this function cannot be computed with fewer layers. See Appendix A.7 for the complete proof.

Note that increasing the hidden size of BIRNNs does not affect the maximal degree of the input's polynomial representation the way depth does. Consequently, the expressive power brought by depth in these models is always increasing, regardless of hidden size. This contrasts with Theorem 1 for linear RNNs, where the gain in memory capacity arises from the number of hidden units, allowing hidden size to compensate for depth. Since 2RNNs are RNNs augmented by a bilinear term, i.e. a BIRNN, they benefit from both the increased memory capacity provided by extra hidden units (Theorem 1) and the higher-order interactions coming from multiplicative terms (Theorem 3). One might say, we are far from the shallow now.

It is worth noting that, while 2RNNs are not commonly used in practice, they provide a formal framework generalizing multiplicative interactions in recurrent architectures. As a result, the insights from Theorem 3 can inform our understanding of models incorporating multiplicative mechanisms, such as gated RNNs (e.g. LSTMs [Hochreiter and Schmidhuber, 1997], GRUs Cho et al. [2014]), time-variant SSMs (e.g. Mamba [Gu and Dao, 2023]), Multiplicative Integration RNNs (MIRNNs [Wu et al., 2016, Levine et al., 2018]), and CP(BI)RNNs [Lizaire et al., 2024, Sutskever et al., 2011]. In particular, for CPBIRNNs we can infer from Thm. 3 that increasing depth leads to a strict inclusion, as formalized in the following corollary (Proof in App. A.8).

Corollary 1. *For $n > 1$, $L \geq 1$ and any $\tilde{R} \geq R > 1$, $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subsetneq \mathcal{H}_{\text{CPBIRNN}}(n, L{+}1, \tilde{R})$ for linear CPBIRNNs.*

Corollary 1 introduces the additional hyperparameter influencing CP(BI)RNNs expressivity: R, the rank of the CP decomposition parameterizing their second-order term. This corollary states that, as long as the rank is maintained at least at the same level, adding layers strictly increases the expressive capacity of the network. For single-layer CP(BI)RNNs, it has been shown that the rank is an effective tuning parameter for strictly increasing expressivity, up to a saturation point $R_{\max}$, the maximal CP rank for a family of tensors sharing the same dimensions, $R_{\max}^{d_1,d_2,d_3} = \max\{\text{rank}_{\text{CP}}(\boldsymbol{\mathcal{T}}) \mid \boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}\}$. The following theorem verifies that this effect of the rank is not altered by depth.

Theorem 4. *For $n > 1$, $L \geq 1$ and any $R \leq R_{max}$, $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subsetneq \mathcal{H}_{\text{CPBIRNN}}(n, L, R{+}1)$ for linear CPBIRNNs if $R < n$.*

Sketch of proof To prove inclusions it suffices to find explicit parameterizations such that there is equality between the latent vectors $\mathbf{h}^t$'s, as in previous sketches of proof. To show strict inclusion, that is $\mathcal{H}_{\text{CPBIRNN}}(n, L, R+1) \not\subset \mathcal{H}_{\text{CPBIRNN}}(n, L, R)$, the idea is to consider the dimension of the image of $\mathbf{h}_t$. Independently of depth, it is bounded by R for CPBIRNN of rank R while we can find a CPBIRNN of rank $R+1$ whose hidden vector has an image of dimension $R+1$. The complete proof can be found in Appendix A.9.

Theorem 4 generalizes the known effect of the rank in single-layer CP(BI)RNNs to deeper architectures, showing that increasing depth does not diminish the expressive benefits provided by a higher rank. In other words, the rank remains an effective lever for controlling expressivity even as the network becomes deeper, allowing to tune both depth and rank independently to achieve the desired expressive power.

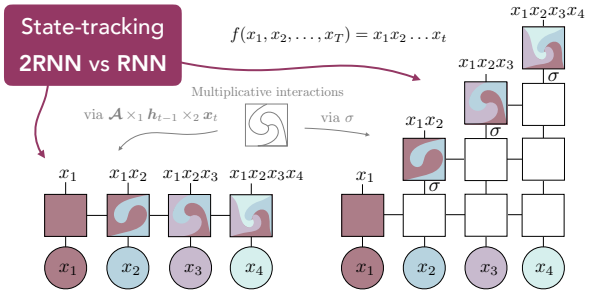

Figure 5: State-tracking information flow. In 2RNNs, multiplicative interactions are performed by bilinear products, while in RNNs they need nonlinear activations σ, thus moving up a layer if applied only in depth.

3.3 Depth and Multiplicative Interactions: Two Distinct Forms of Expressivity

The previous section showed that multiplicative interactions with depth impact expressivity in a manner similar to the composition of nonlinear activations. Both expand the class of functions a model can compute by enabling more complex computations. This naturally raises the question of whether the effects of these two architectural designs on expressive power differ in some way. An avenue to address this question is to consider the connections 2RNNs have with automata theory and formal languages. In particular, linear BIRNNs are equivalent to weighted finite automata (WFA) [Li et al., 2022] whose computation involves *state-tracking*, i.e., maintaining a vector state that evolves multiplicatively with each input to summarize and aggregate information over the sequence. Interestingly, Merrill et al. [2024] showed that single-layer state space models, which are linear recurrent networks, cannot perform state-tracking. A natural question to ask is if depth changes this limitation. Here, we investigate whether the composition of nonlinear activation functions along depth (i.e. across layers but not time, as in Equation 2), enables an RNN to perform the same computations as a 2RNN. In particular, whether it could state-track. The following theorem provides a negative answer to this question.

Theorem 5. *There exists a function computed by a single-layer 2RNN that cannot be computed by any RNN of arbitrary (constant, finite) depth and width with nonlinear activation applied only in depth.*

Sketch of proof The demonstration of this theorem relies on the observation that, in (first-order) RNNs, any multiplicative interaction must occur through the application of a nonlinear activation function, i.e. it requires moving up a layer. Since 2RNNs perform one such multiplicative interaction at each time step, an RNN would need a depth (or hidden size) proportional to the sequence length to replicate this behavior as

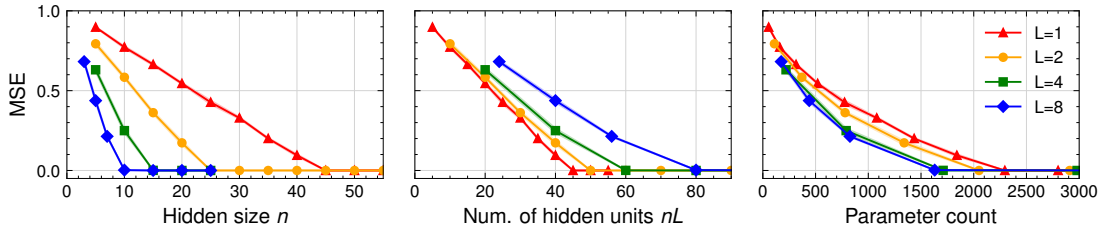

Figure 6: Mean squared error (MSE) on test set for the copy task (lag 8) of linear RNNs with respect to hidden size (left), number of hidden units (center) and number of parameters (right) for varying depth.

illustrated in Figure 5. Consequently, no deep RNN with nonlinear activations applied only in depth can perform state-tracking on arbitrarily long sequences. The complete proof can be found in Appendix A.10.

The functions implied in Theorem 5 are state-tracking ones. Consequently, this result shows that deep RNNs applying nonlinear activations only in depth cannot, in general, perform state-tracking tasks, regardless of the number of layers. Importantly, this holds for arbitrarily complex nonlinearities. This result reveals a fundamental separation in the expressivity conferred by multiplicative interactions versus nonlinear activations, highlighting that these two architectural designs impose different inductive biases.

4 EXPERIMENTS

We assess the practical implications of our theoretical results through experiments on synthetic and real data using RNNs, CPRNNs, and S4 models of varying depth. CPRNNs are employed, rather than BIRNNs or 2RNNs, to avoid handling the third-order tensor growth. First, to validate the theory from Section 3.1, we test linear RNNs on the memorization task f_p (Eq. 3), then we evaluate the nonlinear case using a sinusoidal transformation. Next, we explore the insights from Section 3.3 with the parity task, which emphasizes multiplicative state-tracking over memorization, and compare the effects of nonlinearities in recurrence versus depth.

We then evaluate RNNs, CPRNNs, and S4 on language modeling using the tiny Shakespeare dataset [Karpathy, 2015]. Lastly, we analyze the impact of depth in S4 models across multiple tasks using the Long Range Arena benchmark [Tay et al., 2021]. All models were trained with Adam optimizer [P. Kingma and Ba, 2015] and early stopping. Plots show averages over 3–5 random seeds, with shaded areas indicating standard deviations. S4 weights follow [Gu et al., 2021]; others are initialized with $\mathcal{U}[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. Details specific to each experiments are given in the appendix.

4.1 Copy, sinus, & sinus-copy tasks

Our first synthetic experiment is the copy task f_p from Eq. 3, modified to output all input dimensions. We use a lag $p = 8$ and inputs in $\mathbb{R}^5$ sampled from $\mathcal{N}(0, 1)$. The training set contains 10,000 sequences of length 16; validation and test sets each contain 2000 sequences.

As predicted by Theorem 1, Figure 6 shows that deeper models require a smaller hidden size to achieve zero loss. Indeed, depth strictly increases expressivity and, empirically, leads to better performances. However, the

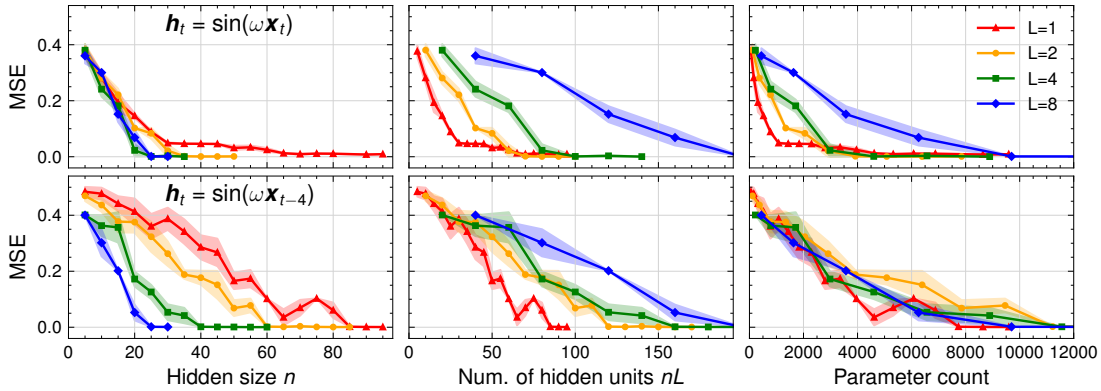

Figure 7: Test mean squared error (MSE) for sinus (top) and copy-sinus (lag 4, bottom) of *tanh* activated RNNs in depth as a function of hidden size (left), number of units (center) and parameters (right) for varying depth.

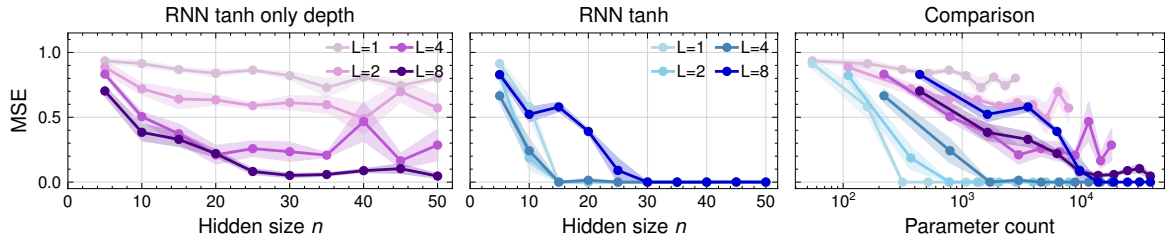

Figure 8: Test MSE for the parity task versus hidden size n on RNNs with *tanh* activations in depth (left) or recurrence (center) for varying depth. Right panel compares activations by number of parameters.

hidden capacity is also augmented by depth. When plotting the mean square error (MSE) against the number of hidden units (center), the trend reverses: shallower models outperform deeper ones for a fixed unit budget, corroborating Proposition 1. Finally, Theorem 2 is validated empirically for this task by the right panel, where the models performance with respect to the number of parameters gradually increases with depth.

We then study the extent to which these trends hold in a nonlinear setting, using a sinusoidal transformation of the copy task (Figure 7). With $p = 0$, the task becomes $\mathbf{h}_t = \sin(\omega \mathbf{x}_t)$, which requires no memory and serves as a baseline for the copy-sinus case with a lag of 4, $\mathbf{h}_t = \sin(\omega \mathbf{x}_{t-4})$. The models have tanh activation in depth, keeping the recurrence linear.

For $p = 0$ (top), depth reduces the hidden size needed to learn the task, with a marked gap between $L = 1$ and 2, suggesting that the sinus transformation is hard to learn by a one-layer model. Nonetheless, $L = 2$ is the first to achieve a zero-loss with respect to the number of parameters, which empirically shows that depth does not provide a parameter efficiency benefit for this specific task.

The sinus-copy case (bottom) combines memory and nonlinearity, both imposing thresholds on the minimum hidden size required for convergence towards a solution. The former gives an advantage in terms of the number of parameters while the latter does not. As shown in the bottom-left panel, all models perform worse than in the memoryless case, but deeper ones ($L = 4$, 8) degrade less than shallower ones ($L = 1$, 2). Consequently, for an equal parameters budget (right panel) the deeper RNNs become competitive, $L = 8$ achieving zero loss with the least amount of parameter.

4.2 Parity task

We now examine parity, a state-tracking task that does not involve memory, to corroborate the theory from Section 3.3. At each time step, the model must identify whether the number of -1s seen so far in a sequence of 1s and -1s is even or odd. We evaluate the MSE on sequences of length 20 with 5-dimensional inputs, where the task is performed independently in each dimension. Theorem 5 states that this problem cannot be compensated by depth-wise activations. It is instead biased towards second-order interactions between input and hidden state, which is confirmed empirically by linear 2RNNs solving it with a hidden size as low as 5.

We compare RNNs with tanh activations applied recurrently (as in Definition 1) or only in depth (as in Equation 2). As shown in Figure 8 (left), shallow models with only depth-wise activations ($L = 1$ and 2) fail to learn the task, while deeper ones ($L = 4$ and 8) approach zero loss for hidden sizes over 50. In contrast, RNNs with recurrent activations learn the task with fewer than 20 hidden units, even with only one layer (center panel), validating empirically Theorem 5.

Comparing the two activation schemes (right panel), we observe distinct trends: recurrently activated RNNs gain parameter efficiency with reduced depth, while depth-only models show no clear depth dependency. This suggests models are not copying input's information, consistent with the task not requiring memory. It also indicates that parity is not among the functions behind the strict inclusion in Theorem 2.

4.3 Language modeling on the tiny Shakespeare dataset

We study the effect of depth on RNNs, CPRNNs, and S4's ability to model natural language using the tiny Shakespeare dataset [Karpathy, 2015], composed of 40,000 lines from the author's corpus. Models were trained at the character level using negative log likelihood. RNNs and CPRNNs use tanh activation applied recurrently, and S4 follows the architecture used in [Gu et al., 2021] for language modeling.

Figure 9 shows a clear and consistent benefit to depth in RNNs, seen both in terms of hidden size (first panel) and parameter count (fourth), where deep RNNs outperform all others almost everywhere. This resembles the copy task behavior (Fig. 6), suggesting that language modeling either involves memorization or another expressive aspect with similar parameter efficiency.

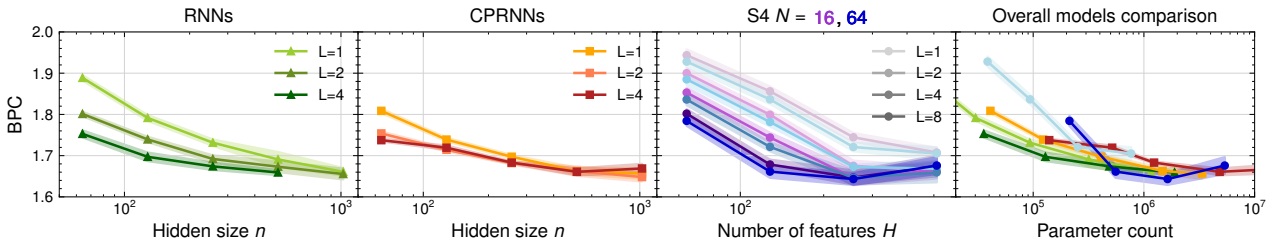

Figure 9: Test bits-per-character (BPC) of RNNs (first), CPRNNs (second) versus hidden size n and S4 (third) versus features H across depths on tiny Shakespeare, fourth panel compares models by parameter count.

For CPRNNs (second panel), the benefit of added layers is much less prominent. There is some improvement from $L = 1$ to $L = 2$, but it saturates at $L = 4$, suggesting that the polynomial class generated by two layers suffices. Note however that the rank was not finely tuned. Also, for $n = 1024$ and all $L = 4$ models, slower optimization was needed for stability, possibly explaining the lower performance. Still, for $L = 1, 2$ and $n = 64$–512, CPRNNs outperform RNNs.

For S4, we use depths up to $L = 8$ and state dimensions $N = 16$ and 64. Depth brings consistent gains across feature counts H (third panel), but with fixed parameter budgets (fourth), more layers offer no clear benefit. Below 1M parameters, S4 underperforms both RNNs and 2RNNs regardless of depth which is reminiscent of the gap between RNNs activated recurrently or only in depth in the parity task (Figure 8), suggesting a similar bias toward multiplicative state-tracking. Overall, deep S4 with at least 256 features (1M parameters) perform best, reaching BPCs similar to deep RNNs.

4.4 Long Range Arena

To assess the role of depth on a variety of tasks, we conclude our empirical investigation by comparing S4 models on Long Range Arena problems [Tay et al., 2021]. The same experimental configurations as in [Gu et al., 2021] were adopted. Table 1 shows that the effect of depth is not the same across all tasks. Performance on Images and ListOps are improved by increasing the number of layers, but eventually saturates, whereas Pathfinders exhibits continued improvement with depth. Interestingly, the efficiency of the Retrieval task is mostly independent of the number of layers. Within our theoretical framework, this contrast can be understood as a difference in memory requirement. Indeed, Retrieval tests a model's ability to store compressed information which is less memory intensive in comparison to Pathfinders which requires remembering all paths in an image. Overall, these results support the task-dependency paradigm of depth's benefit in recurrent architectures inferred by Theorems 1 and 2.

L	Retrieval	Images	ListOps	Pathfinder
2	0.9139	0.8581	0.5806	0.8534
3	**0.9170**	0.8765	0.6154	0.9328
6	0.9099	**0.8980**	**0.6159**	0.9619
8	0.9118	0.8913	0.6154	**0.9673**

Table 1: Test accuracies on Long Range Arena of S4 for varied depth with fixed parameter budget.

5 CONCLUSION

We studied how depth affects the expressivity of recurrent networks by isolating it from nonlinear activations, using linear RNNs to analyze depth–recurrence interactions and linear BIRNNs for depth with multiplicative interactions. We formally show that adding layers increases memory capacity and allows higher-order interactions, and that some single-layer 2RNN functions, like state-tracking, cannot be realized by deep RNNs with nonlinearities only in depth. Experiments on synthetic and real tasks confirm that depth generally improves performance, with gains and parameter efficiency depending on task. This work motivates further explorations of how architectural choices, such as gating mechanisms or time-variant versus time-invariant structures in SSMs, affect expressivity. Another important direction is to deepen our understanding of how these designs interact with optimization. Such insights could clarify how robust the gains in expressive power are in practice and provide principled guidance to design models that are both theoretically expressive and practically effective.

ACKNOWLEDGEMENT

M. Lizaire's research is supported by NSERC (Vanier Scholarship) and IVADO (PhD Excellence Scholarship); G. Rabusseau's by NSERC and the CIFAR AI Chair program. We also acknowledge NVIDIA for providing computational resources. We are grateful to Pascal Jr Tikeng Notsawo for his valuable feedback and support.

References

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

D Rolnick and M Tegmark. The power of deeper networks for expressing natural functions. arxiv 2017. *arXiv preprint arXiv:1705.05502*.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *computational complexity*, 1:113–129, 1991.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

Hava T Siegelmann and Eduardo D Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360, 1994.

Hava T Siegelmann. Recurrent neural networks and finite automata. *Computational intelligence*, 12(4): 567–574, 1996.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. *arXiv preprint arXiv:1805.04908*, 2018.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

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

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4nd: Modeling images and videos as multidimensional signals with state spaces. *Advances in neural information processing systems*, 35:2846–2861, 2022.

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

Shida Wang and Beichen Xue. State-space models with layer-wise nonlinearity are universal approximators with exponential decaying memory. *Advances in Neural Information Processing Systems*, 36:74021–74038, 2023.

Peihao Wang, Ruisi Cai, Yuehao Wang, Jiajun Zhu, Pragya Srivastava, Zhangyang Wang, and Pan Li. Understanding and mitigating bottlenecks of state space models through the lens of recency and oversmoothing. *arXiv preprint arXiv:2501.00658*, 2024.

Riccardo Grazzi, Julien Siems, Jorg KH Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues. 2024. *URL https://api. semanticscholar. org/CorpusID*, 274141450.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

Mark Steijvers and Peter Grünwald. A recurrent network that performs a context-sensitive prediction task. In *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*, pages 335–339. Routledge, 2019.

Mikael Bodén, Janet Wiles, Bradley Tonkes, and Alan Blair. Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99.(Conf. Publ. No. 470)*, volume 1, pages 359–364. IET, 1999.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A Smith, and Eran Yahav. A formal hierarchy of rnn architectures. *arXiv preprint arXiv:2004.08500*, 2020.

Satwik Bhattamishra, Michael Hahn, Phil Blunsom, and Varun Kanade. Separations in the representational capabilities of transformers and recurrent

architectures. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

Matus Telgarsky. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *International conference on machine learning*, pages 955–963. PMLR, 2016a.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on learning theory*, pages 698–728. PMLR, 2016.

Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. *arXiv preprint arXiv:1605.06743*, 2016b.

Or Sharir and Amnon Shashua. On the expressive power of overlapping architectures of deep learning. *arXiv preprint arXiv:1703.02065*, 2017.

Yotam Alexander, Nimrod De La Vega, Noam Razin, and Nadav Cohen. What makes data suitable for a locally connected neural network? a necessary and sufficient condition based on quantum entanglement. *Advances in Neural Information Processing Systems*, 36, 2024.

Noam Razin, Tom Verbin, and Nadav Cohen. On the ability of graph neural networks to model interactions between vertices. *Advances in Neural Information Processing Systems*, 36, 2024.

Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811*, 2017.

Valentin Khrulkov, Oleksii Hrinchuk, and Ivan Oseledets. Generalized tensor models for recurrent neural networks. *arXiv preprint arXiv:1901.10801*, 2019.

Yoav Levine, Or Sharir, and Amnon Shashua. Benefits of depth for long-term memory of recurrent networks. 2018.

Michael T Pearce, Thomas Dooms, Alice Rigg, Jose M Oramas, and Lee Sharkey. Bilinear mlps enable weight-based mechanistic interpretability. *arXiv preprint arXiv:2410.08417*, 2024.

Ozan Irsoy and Claire Cardie. Modeling compositionality with multiplicative recurrent neural networks. *CoRR*, abs/1412.6577, 2014.

Siddhant M. Jayakumar, Wojciech M. Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rylnK6VtDH`.

Yixin Cheng, Grigorios G Chrysos, Markos Georgopoulos, and Volkan Cevher. Multilinear operator networks. *arXiv preprint arXiv:2401.17992*, 2024.

Andros Tjandra, Sakriani Sakti, Ruli Manurung, Mirna Adriani, and Satoshi Nakamura. Gated recurrent neural tensor network. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 448–455. IEEE, 2016.

Ben Krause, Iain Murray, Steve Renals, and Liang Lu. Multiplicative LSTM for sequence modelling. *ICLR Workshop track*, 2017. URL `https://openreview.net/forum?id=SJCS5rXFl`.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1017–1024, 2011.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Russ R Salakhutdinov. On multiplicative integration with recurrent neural networks. *Advances in neural information processing systems*, 29, 2016.

Zhan Su, Yuqin Zhou, Fengran Mo, and Jakob Grue Simonsen. Language modeling using tensor trains. *arXiv preprint arXiv:2405.04590*, 2024.

Tianyu Li, Doina Precup, and Guillaume Rabusseau. Connecting weighted automata, tensor networks and recurrent neural networks through spectral learning. *Machine Learning*, pages 1–35, 2022.

Maude Lizaire, Michael Rizvi-Martel, Marawan Gamal Abdel Hameed, and Guillaume Rabusseau. A tensor decomposition perspective on second-order rnns. *arXiv preprint arXiv:2406.05045*, 2024.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Andrej Karpathy. char-rnn. `https://github.com/karpathy/char-rnn`, 2015.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=qVyeW-grC2k`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] The models studied are presented in the Preliminary section.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] We discuss the complexity of models through comparison of the number of hidden units and parameters. Information on dataset sizes and training time are provided in the appendix.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] Access to the GitHub repository used to run the experiments will be linked to the paper after the blind review process.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes] Included in the theorems statements.
   (b) Complete proofs of all theoretical results. [Yes] The main text presents sketches of proofs for all theoretical results and we provide the complete proofs in the appendix.
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes & No] We did not include yet the code in the supplementary material, as we plan on releasing the URL for the GitHub repository with the camera ready version. All experimental setups are described in details in the appendix.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] All details are given, either in the main paper or the appendix.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] Specifics on the memory and time requirements are provided in the Implementation details section of the appendix.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes] All datasets, code and models used are referenced, in accordance with their respective license.
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes & No] The URL for the GitHub repository will be included after blind review process.
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

On the Role of Depth in the Expressivity of RNNs: Supplementary Materials

A Proofs of theoretical results

A.1 Preliminaries for Proofs

Before presenting the proofs of our theoretical results, we introduce some additional notation, a lemma and its corollary that will be used subsequently as well as a formal definition for CP(BI)RNNs.

A.1.1 Notation

- $\mathcal{L}^{p,q}$ denotes the space of linear maps from $\mathbb{R}^p$ to $\mathbb{R}^q$.
- $\delta_{i,j}$ denotes the Kronecker delta, that is $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.
- $[n, m]$ for $n, m \in \mathbb{N}$ with $m > n$ denotes the interval of integers from n to m.

A.1.2 Expressing $\mathbf{h}_t^{(l)}$ in terms of $\mathbf{x}_t$

We now introduce Lemma 1 which is used in the proof of Theorem 3, and then its corollary used to prove Proposition 1.

Lemma 1. *The hidden states in a 2RNN at time t and layer l can be expressed as*

$$\mathbf{h}_t^{(l)} = \left(\prod_{i=1}^{l} \bar{\mathbf{U}}_{t-1}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l}\left(\prod_{k=j+1}^{l} \bar{\mathbf{U}}_{t-1}^{(k)}\right)\bar{\mathbf{b}}_{t-1}^{(j)},$$
$$\text{where } \bar{\mathbf{U}}_t^{(l)} \equiv (\boldsymbol{\mathcal{A}}^{(l)} \times_1 \mathbf{h}_t^{(l)})^\top + \mathbf{U}^{(l)}, \quad \bar{\mathbf{b}}_t^{(l)} \equiv \mathbf{V}^{(l)}\mathbf{h}_t^{(l)} + \mathbf{b}^{(l)}.$$

Proof. We first note that the bilinear term in the 2RNN (Def. 2) can be expressed as a matrix-vector product, $(\boldsymbol{\mathcal{A}}^{(l)} \times_1 \mathbf{h}_{t-1}^{(l)})^\top \mathbf{h}_t^{(l-1)}$. Inserting this in the complete definition of the 2RNN and reorganizing the terms, we obtain

$$\begin{aligned}\mathbf{h}_t^{(l)} &= [(\boldsymbol{\mathcal{A}}^{(l)} \times_1 \mathbf{h}_{t-1}^{(l)})^\top + \mathbf{U}^{(l)}]\mathbf{h}_t^{(l-1)} + [\mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)}] \\ &\equiv \bar{\mathbf{U}}_{t-1}^{(l)}\mathbf{h}_t^{(l-1)} + \bar{\mathbf{b}}_{t-1}^{(l)} \end{aligned} \tag{4}$$

- For $l = 1$: From the expression above we have $\mathbf{h}_t^{(1)} = \bar{\mathbf{U}}_{t-1}^{(1)}\mathbf{h}_t^{(0)} + \bar{\mathbf{b}}_{t-1}^{(1)}$, which indeed corresponds to the evaluation of Lemma 1 for $l = 1$.
- For l, we assume that this claim holds for $l - 1$

$$\mathbf{h}_t^{(l-1)} = \left(\prod_{i=1}^{l-1} \bar{\mathbf{U}}_{t-1}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l-1}\left(\prod_{k=j+1}^{l-1} \bar{\mathbf{U}}_{t-1}^{(k)}\right)\bar{\mathbf{b}}_{t-1}^{(j)}$$

and insert this result in Eq. 4

$$\begin{aligned}\mathbf{h}_t^{(l)} &= \bar{\mathbf{U}}_{t-1}^{(l)}\left[\left(\prod_{i=1}^{l-1}\bar{\mathbf{U}}_{t-1}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l-1}\left(\prod_{k=j+1}^{l-1}\bar{\mathbf{U}}_{t-1}^{(k)}\right)\bar{\mathbf{b}}_{t-1}^{(j)}\right] + \bar{\mathbf{b}}_{t-1}^{(l)} \\ &= \left(\prod_{i=1}^{l}\bar{\mathbf{U}}_{t-1}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l-1}\left(\prod_{k=j+1}^{l}\bar{\mathbf{U}}_{t-1}^{(k)}\right)\bar{\mathbf{b}}_{t-1}^{(j)} + \bar{\mathbf{b}}_{t-1}^{(l)} \\ &= \left(\prod_{i=1}^{l}\bar{\mathbf{U}}_{t-1}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l}\left(\prod_{k=j+1}^{l}\bar{\mathbf{U}}_{t-1}^{(k)}\right)\bar{\mathbf{b}}_{t-1}^{(j)}\end{aligned}$$

This corresponds to Lemma 1, which completes the induction and proves the result is true for any l. □

The following corollary to Lemma 1 is obtained by setting $\boldsymbol{\mathcal{A}}^{(l)} = 0$, thus considering a RNN instead of a 2RNN. In this case, $\bar{\mathbf{U}}_{t-1}^{(i)} = \mathbf{U}^{(i)}$, and by expanding $\bar{\mathbf{b}}_t^{(l)}$, we obtain the following result.

Corollary 2. *The hidden states in a RNN at time t and layer l can be expressed as*

$$\mathbf{h}_t^{(l)} = \left(\prod_{i=1}^{l}\mathbf{U}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l}\left(\prod_{k=j+1}^{l}\mathbf{U}^{(k)}\right)\mathbf{V}^{(j)}\mathbf{h}_{t-1}^{(j)} + \sum_{j=1}^{l}\left(\prod_{k=j+1}^{l}\mathbf{U}^{(k)}\right)\mathbf{b}^{(j)}$$

A.1.3 Formal definition CP(BI)RNNs

We begin by reviewing the basics of CP decomposition, before introducing a formal definition of CPRNNs, from which we also derive the definition of CPBIRNNs. More details on the single-layer versions of these models can be found in Lizaire et al. [2024].

First, recall that a CP decomposition of rank R expresses a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ as a sum of R rank-one tensors: $\boldsymbol{\mathcal{T}} = \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \equiv [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ where $\mathbf{a}_r \in \mathbb{R}^{d_1}$, $\mathbf{b}_r \in \mathbb{R}^{d_2}$ and $\mathbf{c}_r \in \mathbb{R}^{d_3}$ (see, e.g., [Kolda and Bader, 2009]). The factor matrices $\mathbf{A} \in \mathbb{R}^{d_1 \times R}, \mathbf{B} \in \mathbb{R}^{d_2 \times R}, \mathbf{C} \in \mathbb{R}^{d_3 \times R}$ have the vectors $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r$ as columns. The CP decomposition reduces the parameter count from $\mathcal{O}(d^3)$ to $\mathcal{O}(Rd)$ where $d = max\{d_1, d_2, d_3\}$. We now turn to the definition of (multi-layer) CPRNNs.

Definition 4 (CPRNN)**.** *A CP Recurrent Neural Network of depth L, hidden size n and rank R is parameterized by initial hidden state vectors $\mathbf{h}_0^{(l)} \in \mathbb{R}^n$, weight matrices $\mathbf{A}^{(l)}, \mathbf{B}^{(l)}, \mathbf{C}^{(l)} \in \mathbb{R}^{n \times R}$ (except for $\mathbf{B}^{(1)} \in \mathbb{R}^{d \times R}$), $\mathbf{U}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{n \times n}$ (except for $\mathbf{U}^{(1)} \in \mathbb{R}^{d \times n}$), bias terms $\mathbf{b}^{(l)} \in \mathbb{R}^n$ and activation functions $\sigma^{(l)} : \mathbb{R}^n \to \mathbb{R}^n$ for $l \in [L]$. Given a sequence of inputs $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$, a CPRNN outputs a sequence of hidden states $(\mathbf{h}_1^{(L)}, \mathbf{h}_2^{(L)}, ..., \mathbf{h}_T^{(L)})$ via the following computation at each time step $t \in [T]$ and layer $l \in [L]$:*

$$\mathbf{h}_t^{(l)} = \sigma^{(l)}([\![\mathbf{A}^{(l)}, \mathbf{B}^{(l)}, \mathbf{C}^{(l)}]\!] \times_1 \mathbf{h}_{t-1}^{(l)} \times_2 \mathbf{h}_t^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{U}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)}).$$

As in Definition 1 and 2, the activation functions $\sigma^{(l)}$ are applied element-wise. CPBIRNN is obtained from this definition by setting the first-order parameters $(\mathbf{V}^{(l)}, \mathbf{U}^{(l)}, \mathbf{b}^{(l)})$ to zero. Moreover, note that the bilinear term of CP(BI)RNNs can be expressed using matrix product:

$$[\![\mathbf{A}^{(l)}, \mathbf{B}^{(l)}, \mathbf{C}^{(l)}]\!] \times_1 \mathbf{h}_{t-1}^{(l)} \times_2 \mathbf{h}_t^{(l-1)} = \left[\mathbf{C}^{(l)}\text{diag}(\mathbf{A}^{(l)\top}\mathbf{h}_{t-1}^{(l)})\mathbf{B}^{(l)\top}\right]\mathbf{h}_t^{(l-1)}. \tag{5}$$

A.2 Formalization of "linear RNNs compute linear transformations"

The following proposition formalizes the statement *linear RNNs perform linear transformations of their inputs, regardless of the network's depth.*

Proposition 2. *Let $h \in \mathcal{H}_{\text{RNN}}(n, L)$ be the function computed by a linear RNN. Then, for any T, the function $f : \mathbb{R}^{dT} \to \mathbb{R}^{nT}$ mapping (concatenations of) sequences of inputs $(\mathbf{x}_1, \cdots, \mathbf{x}_T)$ to (concatenations of) sequences of hidden states $(\mathbf{h}_1^{(L)}, \cdots, \mathbf{h}_T^{(L)})$ is linear.*

Proof. We show that $\mathbf{h}_t^{(L)} \in \mathcal{L}^{dT,n}$ for all $t \in [T]$, which implies that their concatenation is in $\mathcal{L}^{dT,nT}$[¶]. The proof consists of a nested induction over l and t.

- Base case $l = 1$: We show that $\mathbf{h}_t^{(1)} \in \mathcal{L}^{dT,n}$ for all t.
  - Base case $t = 1$: $\mathbf{h}_1^{(1)} = \mathbf{U}^{(1)}\mathbf{x}_1 + \mathbf{V}^{(1)}\mathbf{h}_0^{(1)} + \mathbf{b}^{(1)} \in \mathcal{L}^{dT,n}$
  - Inductive step on t (proving for t assuming true for $t-1$): $\mathbf{h}_t^{(1)} = \mathbf{U}^{(1)}\mathbf{x}_t + \mathbf{V}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{b}^{(1)} \in \mathcal{L}^{dT,n}$ since by induction hypothesis $\mathbf{h}_{t-1}^{(1)} \in \mathcal{L}^{dT,n}$.
- Inductive step on l: We show that $\mathbf{h}_t^{(l)} \in \mathcal{L}^{dT,n}$ assuming $\mathbf{h}_t^{(l-1)} \in \mathcal{L}^{dT,n}$.
  - Base case $t = 1$: $\mathbf{h}_1^{(l)} = \mathbf{U}^{(l)}\mathbf{h}_1^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_0^{(l)} + \mathbf{b}^{(l)} \in \mathcal{L}^{dT,n}$ since $\mathbf{h}_1^{(l-1)} \in \mathcal{L}^{dT,n}$ by induction hypothesis.
  - Inductive step on t (proving for t assuming true for $t-1$): $\mathbf{h}_t^{(l)} = \mathbf{U}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)} \in \mathcal{L}^{dT,n}$ since $\mathbf{h}_t^{(l-1)} \in \mathcal{L}^{dT,n}$ by induction hypothesis on l and $\mathbf{h}_{t-1}^{(l)} \in \mathcal{L}^{dT,n}$ by induction hypothesis on t.

□

A.3 Proof of Theorem 1

Theorem. *For any $n > 1$ and $L \geq 1$, $\mathcal{H}_{\text{RNN}}(n, L) \subsetneq \mathcal{H}_{\text{RNN}}(n, L+1)$ for linear RNNs.*

Proof.

Inclusion: We first show that for any $h \in \mathcal{H}_{\text{RNN}}(n, L)$, there exists a function $\tilde{h} \in \mathcal{H}_{\text{RNN}}(n, L+1)$ such that $h = \tilde{h}$, that is $\mathcal{H}_{\text{RNN}}(n, L) \subseteq \mathcal{H}_{\text{RNN}}(n, L+1)$. Consider the parameters of the RNN computing h : $\{\mathbf{U}^{(l)}, \mathbf{V}^{(l)}, \mathbf{b}^{(l)}, \mathbf{h}_0^{(l)}\}_{l=1}^{L}$. First, we set the parameters of the first L layers of the RNN computing $\tilde{h}$ the same as the one computing h so that the hidden states of the L^{th} layer are the same for both functions:

$$\tilde{\mathbf{U}}^{(l)} = \mathbf{U}^{(l)}, \quad \tilde{\mathbf{V}}^{(l)} = \mathbf{V}^{(l)}, \quad \tilde{\mathbf{b}}^{(l)} = \mathbf{b}^{(l)}, \quad \tilde{\mathbf{h}}_0^{(l)} = \mathbf{h}_0^{(l)} \quad \forall l \in [L]$$

$$\implies \tilde{\mathbf{h}}_t^{(l)} = \mathbf{h}_t^{(l)} \quad \forall t \text{ and } \forall l \in [L].$$

Then, we set the last layer $L+1$, such that it computes the identity over the hidden vectors of the previous layer $\tilde{\mathbf{h}}_t^{(L)}$:

$$\tilde{\mathbf{U}}^{(L+1)} = \mathbf{I}_n, \quad \tilde{\mathbf{V}}^{(L+1)} = \mathbf{0}, \quad \tilde{\mathbf{b}}^{(L+1)} = \mathbf{0}, \quad \tilde{\mathbf{h}}_0^{(L+1)} = \mathbf{0}$$

$$\implies \tilde{\mathbf{h}}_t^{(L+1)} = \mathbf{I}_n\, \tilde{\mathbf{h}}_t^{(L)} + \mathbf{0}\, \tilde{\mathbf{h}}_{t-1}^{(L+1)} + \mathbf{0} = \tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)}.$$

This parameterization is such that $\tilde{\mathbf{h}}_t^{(L+1)} = \mathbf{h}_t^{(L)}$ for all t, which implies that $\tilde{h} = h$ and proves the inclusion $\mathcal{H}_{\text{RNN}}(n, L) \subseteq \mathcal{H}_{\text{RNN}}(n, L+1)$. Note that this proof also works for nonlinear RNNs by applying the same principles: using the same nonlinearities for the first L layers and leaving the last layer linear[‖].

Strict inclusion: To prove strict inclusion of this theorem, we consider the (real-valued) function f_p defined for any p by

$$f_p(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t) = \begin{cases} (\mathbf{x}_{t-p})_1 & \text{if } t - p > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Informally, the function f_p copies the first component of $\mathbf{x}_t$ p time steps forward. To simplify the notation, let $x_t = (\mathbf{x}_t)_1$ be the first component of the input at time t. We first show in the following lemma that for any n and any p the function f_p can be computed by a linear RNN as soon as it is deep enough (at least $p/(n-1)$ layers).

[¶]Here (and thereafter), we use the abuse of notation $\mathbf{h}_t^{(L)} \in \mathcal{L}^{dT,n}$ to mean that the function that maps $(\mathbf{x}_1, \cdots, \mathbf{x}_T)$ to $\mathbf{h}_t^{(L)}$ is linear.

[‖]Even in the case where the last layer has to be non-linear, one could find parameters such that the last layer approximates the identity operator to an arbitrary precision (assuming bounded domain of the inputs and infinite precision to represent floating operations).

Lemma 2. *For any $n > 1$ and $p \geq 1$, $f_p \in \mathcal{L}^{n,1} \circ \mathcal{H}_{\text{RNN}}(n, \lceil p/(n-1) \rceil)$.*

Proof. The lemma is proved by construction. We consider an RNN computing a function h and a linear mapping w such that $w \circ h = f_p$. In this RNN, each hidden vectors is the concatenation of the first dimension of n consecutive inputs. At each time step, the input occupying the first units of the hidden vector jumps to the next layer while the others are sent to the next (in time) hidden state through the recurrent connection:

$$\mathbf{h}_t^{(1)} = \begin{bmatrix} x_{t-(n-1)} \\ \vdots \\ x_t \end{bmatrix}, \quad \mathbf{h}_t^{(2)} = \begin{bmatrix} x_{t-2(n-1)} \\ \vdots \\ x_{t-(n-1)} \end{bmatrix}, \quad \dots, \quad \mathbf{h}_t^{(L)} = \begin{bmatrix} x_{t-L(n-1)} \\ \vdots \\ x_{t-(L-1)(n-1)} \end{bmatrix}$$

Since the depth of the network is $L = \lceil \frac{p}{n-1} \rceil$, the component $1 + L(n-1) - p$ of $\mathbf{h}_t^{(L)}$ contains x_{t-p}. Moreover, it is ensured that, at all time, the first dimension of the previous p inputs required for future computations are contained within the L hidden vectors, see Figure 10.

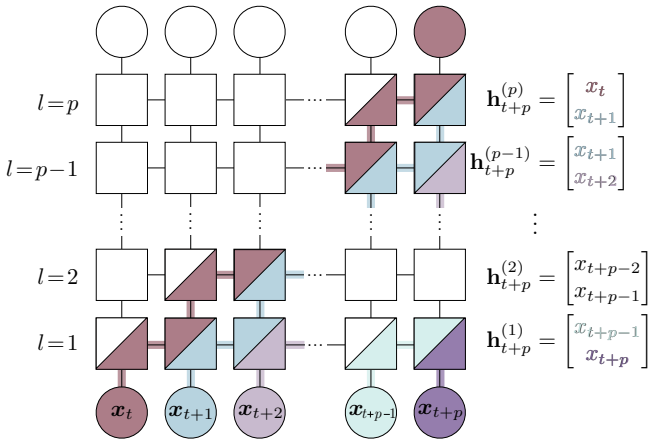

Figure 10: Schematic representation of RNN's computation of f_p for $n = 2$ and $L = p$.

This computation is achieved with the following weight matrices, null biases and null initial hidden vectors :

$$\mathbf{U}^{(1)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{U}^{(l)} = \left[\begin{array}{c|ccc} 0 & & & \\ \vdots & & \mathbf{0} & \\ \hline 1 & 0 & \cdots & 0 \end{array}\right] \forall l \in [2, L] \quad \text{and} \quad \mathbf{V}^{(l)} = \left[\begin{array}{c|ccc} 0 & & & \\ \vdots & & \mathbf{I}_{n-1} & \\ \hline 0 & 0 & \cdots & 0 \end{array}\right] \forall l \in [L].$$

One can verify that by composing h with the linear map $w \in \mathcal{L}^{n,1}$ defined by $w(\mathbf{h}) = [\mathbf{h}]_1$, we obtain f_p, which verifies the statement of the lemma. □

We now show in the following lemma how the number of layers and hidden neurons needed for an RNN to compute the function f_p are lower bounded by the lag value p.

Lemma 3. *If $f_p \in \mathcal{L}^{n,1} \circ \mathcal{H}_{\text{RNN}}(n, L)$, then $p \leq L(n-1)$.*

Proof. In order to prove the lemma, we formalize the computation of f_p by an RNN as a flow graph in which information is propagated. The graph is a grid where the horizontal axis correspond to the time t and the vertical axis to the depth l. There is a total of $L + 2$ layers: one for the input ($l = 0$), one for the output ($l = L + 1$) and L hidden layers in between. Each node can be identified by its layer l and time step t. Given the recurrent nature of the function, information can only flow through depth from l to $l + 1$ or through the recurrent connection from t to $t + 1$.

In order for an RNN to compute f_p, for each time step i, the information of x_i needs to flow through the graph up to $\mathbf{h}_{i+p}^{(L)}$ in the last layer p time steps later. Thus, there exists a path in the graph between $\mathbf{h}_i^{(1)}$ and $\mathbf{h}_{i+p}^{(L)}$ through which the information of x_i is propagated. Each step in this path is either horizontal or vertical, meaning either the information is propagated through recurrence, from $\mathbf{h}_t^{(l)}$ to $\mathbf{h}_{t+1}^{(l)}$, or through depth, from $\mathbf{h}_t^{(l)}$ to $\mathbf{h}_t^{(l+1)}$. See Figure 11 (center) for an illustration. Note that we will use both t and i to denote time steps, but we will use is for information flowing through the network and ts for the actual time steps of the computation.

Now observe that for the information of x_i to have been propagated through this path without loss of information, x_i must have appeared as one of the components (neuron) of each of the hidden state appearing on this path**. We now introduce a notation to formalize the flow of information through the computation network of the RNN.

Definition: For each $i, t \geq 1$ and $1 \leq l < L$, we let $f^{\rightarrow}_{i,t,l} = 1$ if the information of x_i flows from t to $t+1$ at layer l, and 0 otherwise; similarly, we let $f^{\uparrow}_{i,t,l} = 1$ if the information of x_i flows from l to $l+1$ at time t, and 0 otherwise.

This definition is illustrated in Figure 11 (left). The computation f_p is schematized in the center part of the figure: the information of x_i introduced at time $t = i$ must flow out from the last layer of the RNN as the output at time $t = i + p$. Since it is introduced at time $t = i$, x_i cannot flow out of the input at other times or flow horizontally from earlier times.

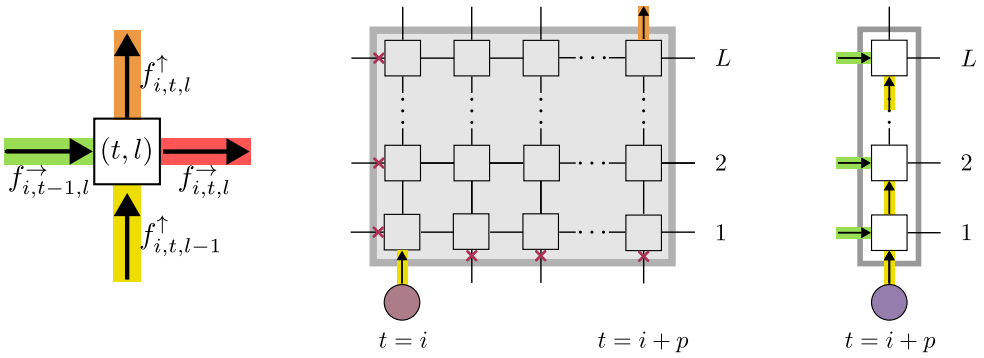

Figure 11: Left panel: Information flow through $\mathbf{h}_t^{(l)}$. Center panel: Schematization of computation f_p. Right panel: The information capacity needed for all the hidden states across the layers at some time t is obtained by considering the sum of information flows that enter the nodes.

Lemma 3 is derived by inspecting the total information capacity needed across all layers at time t, which is constrained by the number of neurons in the hidden states. At each time step t, the sum of all information flowing in all the neurons in the hidden states cannot exceed the total size of the hidden states nL (see the right panel in Figure 11). Formally, for all $t \geq 1$, we have

$$nL \geq \sum_{l=1}^{L} \sum_{i=1}^{t-1} f^{\rightarrow}_{i,t-1,l} + \sum_{l=1}^{L} \sum_{i=1}^{t} f^{\uparrow}_{i,t,l-1}. \tag{6}$$

The first part of the r.h.s. corresponds to the information that is flowing from the previous time step through recurrence (at the same layer), and the second one corresponds to the information flowing between two consecutive layers through depth (at time step t).

The first double sum in Eq. (6) is simplified by observing that to compute f_p, all input information x_i where $i \in [t-p, t-1]$ must flow from $t-1$ to t. To achieve this, each input must flow horizontally in (at least) one of the layers, meaning that $\sum_{l=1}^{L} f^{\rightarrow}_{i,t-1,l} \geq 1$ for all $i \in [t-p, t-1]$. Hence, the first double sum can be lower

**One may argue that information about x_i could have been separated between 2 or more neurons, but this would result in a sub-optimal construction since more than 2 neurons would have been used to store what could more efficiently be stored in a single neuron.

bounded as $\sum_{l=1}^{L}\sum_{i=1}^{t-1} f^{\rightarrow}_{i,t-1,l} \geq p$, leading to

$$nL \geq p + \sum_{l=1}^{L}\sum_{i=1}^{t} f^{\uparrow}_{i,t,l-1} \tag{7}$$

for all time step $t \geq 1$. Since this inequality is true for any time step t, we can sum up these inequalities for each time step up to an arbitrary time step K to obtain

$$KnL \geq Kp + \sum_{t=1}^{K}\sum_{l=1}^{L}\sum_{i=1}^{t} f^{\uparrow}_{i,t,l-1} \ , \quad \text{for any } K. \tag{8}$$

To lower bound the last term, first observe that to make its way from the input layer $l = 0$ to the last layer $l = L$, a given information x_i must flow vertically at least once in each layer between the input and output times $t \in [i, i + p]$. Formally, we must have that, for each time step i, $\sum_{t=i}^{i+p} f^{\uparrow}_{i,t,l-1} \geq 1$ for all $l \in [L]$, hence $\sum_{l=1}^{L}\sum_{t=i}^{i+p} f^{\uparrow}_{i,t,l-1} \geq L$. Again, since this inequality holds for each time step i, we can sum up inequalities up to an arbitrary time step. In particular, we have

$$\sum_{i=1}^{K-p}\sum_{l=1}^{L}\sum_{t=i}^{i+p} f^{\uparrow}_{i,t,l-1} \geq (K-p)L \ , \quad \text{for any } K \geq p.$$

Now, we can show that

$$\sum_{t=1}^{K}\sum_{l=1}^{L}\sum_{i=1}^{t} f^{\uparrow}_{i,t,l-1} \geq \sum_{i=1}^{K-p}\sum_{l=1}^{L}\sum_{t=i}^{i+p} f^{\uparrow}_{i,t,l-1}.$$

Indeed, every term appearing in the r.h.s. is of the form $f^{\uparrow}_{\tilde{i},\tilde{i}+\tau,l-1}$ with $\tilde{i} \in [1, K-p]$ and $\tau \in [0, p]$, and thus appears in the l.h.s. as well when $t = \tilde{i} + \tau$ (since in this case $\tilde{i}$ is the range of the sum over i of the l.h.s., which is $[1, t] = [1, \tilde{i} + \tau]$).

Finally, we thus have that $\sum_{t=1}^{K}\sum_{l=1}^{L}\sum_{i=1}^{t} f^{\uparrow}_{i,t,l-1} \geq (K-p)L$, which allows us to obtain from Eq. (8) the final inequality

$$KnL \geq Kp + (K-p)L$$

which can be re-arranged as

$$p \leq L(n-1) + \frac{pL}{K}.$$

Since this equality is true for any K, we can choose K large enough for $\frac{pL}{K}$ to be strictly smaller than one, which implies the result, $p \leq L(n-1)$.

□

To prove the theorem, it only remains to combine the two lemmas. Let $n > 1$ and $L \geq 1$. We choose $p = (L+1)(n-1)$; from Lemma 2, $f_p \in \mathcal{L}^{n,1} \circ \mathcal{H}_{\text{RNN}}(n, L+1)$, from Lemma 3, $f_p \notin \mathcal{L}^{n,1} \circ \mathcal{H}_{\text{RNN}}(n, L)$. This shows that $\mathcal{H}_{\text{RNN}}(n, L) \subsetneq \mathcal{H}_{\text{RNN}}(n, L+1)$.

□

A.4 Formalization of Proposition 1

Proposition. *For $n > 1$ and $L > 1$, a single-layer linear RNN of hidden size nL has a greater latent representational capacity than a linear RNN of size n and depth L.*

The following theorem formalizes the previous proposition.

Theorem 6. *Let $\mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,l)$ denote the set of functions $h^{(\|)}$ mapping an input sequence to the concatenation of hidden vectors computed at each layers of an RNN of hidden size n and depth L:*

$$h^{(\|)}(\mathbf{x}_1,\dots,\mathbf{x}_T) = (\mathbf{h}_1^{(\|)},\dots,\mathbf{h}_T^{(\|)}) \quad \textit{with} \quad \mathbf{h}_t^{(\|)} = \begin{bmatrix}\mathbf{h}_t^{(1)}\\ \vdots \\ \mathbf{h}_t^{(L)}\end{bmatrix}.$$

For any $n>1$ and $L\geq 1$, $\mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L) \subsetneq \mathcal{H}_{\mathrm{RNN}}(nL,1)$, for linear RNNs.

Proof.

Inclusion: For any $h^{(\|)} \in \mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L)$, we show there exists a $\tilde{h} \in \mathcal{H}_{\mathrm{RNN}}(nL,1)$ such that $\tilde{h} = h^{(\|)}$, that is $\mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L) \subseteq \mathcal{H}_{\mathrm{RNN}}(nL,1)$. The idea is for the hidden vectors $\tilde{\mathbf{h}}_t$ of the function $\tilde{h}$ to be composed of L blocks of n dimensions. Each block reproduces the computation of one of the layer hidden vector $\mathbf{h}_t^{(l)}$ that is concatenated in the function $h^{(\|)}$. The parameters of $\tilde{h}$ are thus also separated in blocks as follow:

$$\tilde{\mathbf{h}}_t = \tilde{\mathbf{U}}\mathbf{x}_t + \tilde{\mathbf{V}}\tilde{\mathbf{h}}_{t-1} + \tilde{\mathbf{b}} \rightarrow \begin{bmatrix}\tilde{\mathbf{h}}_{t,1}\\ \vdots \\ \tilde{\mathbf{h}}_{t,L}\end{bmatrix} = \begin{bmatrix}\tilde{\mathbf{U}}_1\\ \vdots \\ \tilde{\mathbf{U}}_L\end{bmatrix}\mathbf{x}_t + \begin{bmatrix}\tilde{\mathbf{V}}_{11} & \dots & \tilde{\mathbf{V}}_{1L}\\ \vdots & \ddots & \vdots \\ \tilde{\mathbf{V}}_{L1} & \dots & \tilde{\mathbf{V}}_{LL}\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{h}}_{t-1,1}\\ \vdots \\ \tilde{\mathbf{h}}_{t-1,L}\end{bmatrix} + \begin{bmatrix}\tilde{\mathbf{b}}_1\\ \vdots \\ \tilde{\mathbf{b}}_L\end{bmatrix} \tag{9}$$

with $\tilde{\mathbf{h}}_{t,l} \in \mathbb{R}^n$, $\tilde{\mathbf{U}}_l \in \mathbb{R}^{n\times d}$, $\tilde{\mathbf{V}}_{lj} \in \mathbb{R}^{n\times n}$ and $\tilde{\mathbf{b}}_l \in \mathbb{R}^n$ for all $l \in [L]$.

For each $l \in [L]$, we want the computation $\tilde{\mathbf{h}}_{t,l} = \tilde{\mathbf{U}}_l\mathbf{x}_t + \sum_{j=1}^{L}\tilde{\mathbf{V}}_{lj}\tilde{\mathbf{h}}_{t-1,j} + \tilde{\mathbf{b}}_l$ to correspond to $\mathbf{h}_t^{(l)} = \mathbf{U}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)}$.

As shown in Corollary 2, $\mathbf{h}_t^{(l)}$ can be expressed in terms of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}^{(\bar{l})}$ with $\bar{l} \in [l]$ as :

$$\mathbf{h}_t^{(l)} = \left(\prod_{i=1}^{l}\mathbf{U}^{(i)}\right)\mathbf{x}_t + \sum_{j=1}^{l}\left(\prod_{k=j+1}^{l}\mathbf{U}^{(k)}\right)\mathbf{V}^{(j)}\mathbf{h}_{t-1}^{(j)} + \sum_{j=1}^{l}\left(\prod_{k=j+1}^{l}\mathbf{U}^{(k)}\right)\mathbf{b}^{(j)}$$

It then suffices to set the parameters of an RNN computing $\tilde{h}$ as

$$\tilde{\mathbf{U}}_l = \prod_{i=1}^{l}\mathbf{U}^{(i)}, \quad \tilde{\mathbf{V}}_{lj} = \begin{cases}\left(\prod_{k=j+1}^{l}\mathbf{U}^{(k)}\right)\mathbf{V}^{(j)} & j \leq l\\ 0 & j > l\end{cases}, \quad \tilde{\mathbf{b}}_l = \sum_{j}\left(\prod_{k=j+1}^{l}\mathbf{U}^{(k)}\right)\mathbf{b}^{(j)}$$

in order to have $\tilde{\mathbf{h}}_{t,l} = \mathbf{h}_t^{(l)}$ for all $l \in [L]$, and therefore $\tilde{h} = h^{(\|)}$. This concludes the proof for $\mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L) \subseteq \mathcal{H}_{\mathrm{RNN}}(nL,1)$.

Strict inclusion: We show there exists a function $\tilde{h} \in \mathcal{H}_{\mathrm{RNN}}(nL,1)$ that cannot be computed by any function in $h \in \mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L)$, that is $\mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L) \not\supset \mathcal{H}_{\mathrm{RNN}}(nL,1)$.

Consider the function $\tilde{h} \in \mathcal{H}_{\mathrm{RNN}}(nL,1)$ computed by an RNN parameterized with $\tilde{\mathbf{V}} = \mathbf{0}$, $\tilde{\mathbf{b}} = \mathbf{0}$ and $\tilde{\mathbf{U}}$ block parameterized as in Eq. 9 with $\mathrm{rank}(\tilde{\mathbf{U}}_1) = 1$ and $\mathrm{rank}(\tilde{\mathbf{U}}_l) = 2$ for all $l \in [2,l]$[††]. Suppose there exists $h \in \mathcal{H}^{(\|)}_{\mathrm{RNN}}(n,L)$ such that $h = \tilde{h}$. Then, we would have equality between the first hidden vectors $\tilde{\mathbf{h}}_1$ and $\mathbf{h}_1^{(\|)}$ which implies on one hand,

$$\begin{aligned}\tilde{\mathbf{h}}_{1,1} = \mathbf{h}_1^{(1)} &\implies \tilde{\mathbf{U}}_1\mathbf{x}_1 = \mathbf{U}^{(1)}\mathbf{x}_1 \implies \tilde{\mathbf{U}}_1 = \mathbf{U}^{(1)}\\ &\implies \mathrm{rank}(\mathbf{U}^{(1)}) = \mathrm{rank}(\tilde{\mathbf{U}}_1) = 1,\end{aligned}$$

[††]In fact, the result works for any parameterization such that $\mathrm{rank}(\tilde{\mathbf{U}}_1) = 1$ while $\mathrm{rank}(\tilde{\mathbf{U}}_l) > 1$ for at least one $l \in [2,l]$

but on the other hand,

$$\begin{aligned}\tilde{\mathbf{h}}_{1,2} = \mathbf{h}_1^{(2)} &\implies \tilde{\mathbf{U}}_2\mathbf{x}_1 = \mathbf{U}^{(2)}\mathbf{U}^{(1)}\mathbf{x}_1 \implies \tilde{\mathbf{U}}_2 = \mathbf{U}^{(2)}\mathbf{U}^{(1)} \\ &\implies \operatorname{rank}(\mathbf{U}^{(2)}\mathbf{U}^{(1)}) = \operatorname{rank}(\tilde{\mathbf{U}}_2) = 2 \\ &\implies \operatorname{rank}(\mathbf{U}^{(1)}) \geq 2.\end{aligned}$$

The existence of $h \in \mathcal{H}^{(\|)}_{\text{RNN}}(n, L)$ such that $h = \tilde{h}$ leads to a contradiction, proving that $h \notin \mathcal{H}_{\text{RNN}}(nL, 1)$ and $\mathcal{H}^{(\|)}_{\text{RNN}}(n, L) \not\supset \mathcal{H}_{\text{RNN}}(nL, 1)$. □

A.5 Proof of Theorem 2

Theorem. *Considering linear RNNs with input dimension $d = 1$, let $\#\text{params}(n, L)$ denote the number of parameters of a L-layers RNN with hidden size n* [‡‡].

For any depth L and for any hidden size $n \geq 4$, there exists a function $f \in \mathcal{H}_{\text{RNN}}(n, L)$ such that, for all $\tilde{L} < L$ and $\tilde{n}$, if $f \in \mathcal{H}_{\text{RNN}}(\tilde{n}, \tilde{L})$ then $\#\text{params}(\tilde{n}, \tilde{L}) > \#\text{params}(n, L)$.

Proof. Let $L > 1$ and $n > 1$. We set $p = L(n-1)$ and show that the function f_p defined in Eq. (3) satisfies the result. From Lemma 2, we know that $f_p \in \mathcal{L}^{n,1} \circ \mathcal{H}_{\text{RNN}}(n, L)$. Now, let $\tilde{L}$ be an integer such that $1 \leq \tilde{L} < L$ and $f_p \in \mathcal{L}^{\tilde{n},1} \circ \mathcal{H}_{\text{RNN}}(\tilde{n}, \tilde{L})$ for some integer $\tilde{n}$. Then, from Lemma 3, we must have $p \leq \tilde{L}(\tilde{n} - 1)$, i.e.,

$$L(n-1) \leq \tilde{L}(\tilde{n} - 1).$$

It follows that

$$\tilde{n} \geq \frac{L}{\tilde{L}}(n-1) + 1.$$

Since $\#\text{params}(n, L) = (2L-1)n^2 + (L+1)n$ is monotonous increasing in n for $n \geq 1, L \geq 1$, a lower bound can be obtained as

$$\#\text{params}(\tilde{n}, \tilde{L}) \geq \#\text{params}\left(\frac{L}{\tilde{L}}(n-1) + 1, \tilde{L}\right) = (2\tilde{L} - 1)\left(\frac{L}{\tilde{L}}\right)^2 n^2 + bn + c$$

where b and c are constants that depend only on L and $\tilde{L}$. It follows that

$$\#\text{params}(\tilde{n}, \tilde{L}) - \#\text{params}(n, L) \geq \left((2\tilde{L} - 1)\left(\frac{L}{\tilde{L}}\right)^2 - 2L + 1\right) n^2 + b'n + c \tag{10}$$

where b' does not depend on n.

The r.h.s. of Eq. (10) is a second order polynomial in n and its leading coefficient is positive. To see this, this coefficient can be developped and re-arranged as

$$a = \frac{(L - \tilde{L})(2\tilde{L}L - L - \tilde{L})}{\tilde{L}^2} > 0. \tag{11}$$

All factors $\tilde{L}^{-2}$, $L - \tilde{L}$ and $2\tilde{L}L - (\tilde{L} + L)$ are positive for $1 \leq \tilde{L} < L$, showing that the leading coefficient is indeed positive. Since the r.h.s. of Eq. (10) is a second order polynomial with a strictly positive leading term (its graph is an upward-opening parabola), there exists an N for which this polynomial is positive for any $n > N$, i.e., $\#\text{params}(\tilde{n}, \tilde{L}) > \#\text{params}(n, L)$ for all $n > N$. □

[‡‡] One can check that $\#\text{params}(n, L) = (2L-1)n^2 + (L+1)n$ (excluding initial hidden states as parameters count, though including them would not change the result).

A.5.1 Critical N

It is in fact possible to find the critical value N where the r.h.s. of Eq. (10) vanishes, that is we solve $an^2+b'n+c=0$. The leading coefficient a is shown in Eq. (11), and other coefficients can be found easily. The coefficients we obtain have common factors and the equation can be factorized as follows: $(L-\tilde{L})\tilde{L}^{-2}(\tilde{a}n^2+\tilde{b}n+\tilde{c})=0$, where

$$\tilde{a}=2L\tilde{L}-L-\tilde{L},\quad \tilde{b}=-2\tilde{a}-\tilde{L},\quad \tilde{c}=\tilde{a}-\tilde{L}(3\tilde{L}-1).$$

It is sufficient to solve $\tilde{a}n^2+\tilde{b}n+\tilde{c}=0$. The largest root is given as a function of $L,\tilde{L}$

$$n_0^+(L,\tilde{L})=1+\frac{\tilde{L}(1+\sqrt{12\tilde{a}+1})}{2\tilde{a}}.$$

where $\tilde{a}$'s dependence in L and $\tilde{L}$ is not noted explicitly to keep the notation compact. The maximal $n_0^+(L,\tilde{L})$ among pairs $(L,\tilde{L})$ respecting $1\leq\tilde{L}<L$ gives N. We observe that the gradient in L is always negative

$$\frac{\partial n_0^+}{\partial L}=-\frac{\left(6\tilde{a}+\sqrt{12\tilde{a}+1}+1\right)\tilde{L}\left(2\tilde{L}-1\right)}{2\tilde{a}^2\sqrt{12\tilde{a}+1}}<0$$

which is easy to see by noting that $\tilde{a}>0$ and $\tilde{L}\geq 1$. This implies that n_0^+ is maximized by setting L to its smallest possible value, $\tilde{L}+1$

$$n_0^+(\tilde{L}+1,\tilde{L})=1+\frac{\tilde{L}\left(1+\sqrt{24\tilde{L}^2-11}\right)}{4\tilde{L}^2-2}.$$

This expression decreases monotonically in $\tilde{L}$ for $\tilde{L}\geq 1$. Thus, n_0^+ is again maximized by choosing the appropriate boundary, $\tilde{L}=1$. The maximum is $n_0^+(L=2,\tilde{L}=1)=(3+\sqrt{13})/2\approx 3.30$, which is rounded up to 4. Therefore, for any depth L and any hidden size $n\geq 4$, there exists a function computable by an RNN with L layers and n neurons which cannot be computed by a shallower network without increasing the number of parameters.

A.6 Formalization of "linear BIRNNs compute polynomial transformations"

Proposition. *Let $h\in\mathcal{H}_{\text{BIRNN}}(n,L)$ be the function computed by a BIRNN. Then, for any T, the function $f:\mathbb{R}^{dT}\to\mathbb{R}^{nT}$ mapping (concatenations of) sequences of inputs $(\mathbf{x}_1,\cdots,\mathbf{x}_T)$ to (concatenations of) sequences of hidden states $(\mathbf{h}_1^{(L)},\cdots,\mathbf{h}_T^{(L)})$ is a multivariate polynomial whose order is at most T^L.*

Proof. First, it is easy to check that f is a polynomial since the update function of a linear 2RNN is a polynomial map of order 2:

$$\mathbf{h}_t^{(l)}=\boldsymbol{\mathcal{A}}^{(l)}\times_1\mathbf{h}_{t-1}^{(l)}\times_2\mathbf{h}_t^{(l-1)}+\mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)}+\mathbf{U}^{(l)}\mathbf{h}_t^{(l-1)}+\mathbf{b}^{(l)}.$$

Thus, each $\mathbf{h}_t^{(l)}$ is obtained by composing polynomial maps of the inputs, and f is itself a polynomial map.

We now proceed to show that the order of the polynomial is at most T^L by considering the recurrence on L. For sake of simplicity, we assume that only the second order parameters $\boldsymbol{\mathcal{A}}^{(l)}$ of the 2RNN are non-zero, since the polynomial interactions of highest order in $\mathbf{h}_t^{(l)}$ are the second order ones. For $L=1$, for any t, we have

$$\mathbf{h}_t^{(1)}=\boldsymbol{\mathcal{T}}\times_1\mathbf{h}_0^{(1)}\times_2\mathbf{x}_1\times_3\cdots\times_{t+1}\mathbf{x}_t$$

where the $(t+2)$th order tensor $\boldsymbol{\mathcal{T}}\in\mathbb{R}^{n\times d\times\cdots\times d\times n}$ is defined by

$$\boldsymbol{\mathcal{T}}_{i,j_1,\cdots,j_t,k}=\sum_{r_1=1}^{n}\sum_{r_2=1}^{n}\cdots\sum_{r_2=1}^{n}(\mathbf{h}_0^{(1)})_i\boldsymbol{\mathcal{A}}^{(1)}_{i,j_1,r_1}\boldsymbol{\mathcal{A}}^{(1)}_{r_1,j_2,r_2}\boldsymbol{\mathcal{A}}^{(1)}_{r_2,j_3,r_3}\cdots\boldsymbol{\mathcal{A}}^{(1)}_{r_n,j_n,k}$$

for all $i,k\in[n],j_1,\cdots,j_n\in[d]$. This shows that, for $L=1$, f is a polynomial map of order at most t. Indeed, recall that

$$(\boldsymbol{\mathcal{T}}\times_1\mathbf{h}_0^{(1)}\times_2\mathbf{x}_1\times_3\cdots\times_{t+1}\mathbf{x}_t)_k=\sum_{i,j_1,\cdots,j_n}\boldsymbol{\mathcal{T}}_{i,j_1,\cdots,j_n,k}(\mathbf{h}_0^{(1)})_i(\mathbf{x}_1)_{j_1}\cdots(\mathbf{x}_t)_{j_t}.$$

Now, assuming the result true for all integers strictly less than L, the output of the $(L-1)$th layer, $(\mathbf{h}_1^{(L-1)}, \cdots, \mathbf{h}_T^{(L-1)})$, is a polynomial map of the inputs of order at most T^{L-1}. Using the same argument as for the case $L = 1$, it is straightforward to show that the output of the Lth layer is a polynomial map of $(\mathbf{h}_1^{(L-1)}, \cdots, \mathbf{h}_T^{(L-1)})$ of order at most T. Since the composition of two polynomial maps of orders p_1 and p_2, respectively, is at most $p_1 p_2$, it follows that the output of the Lth layer is a polynomial map of the inputs of order at most $T^{L-1}T = T^L$. □

A.7 Proof of Theorem 3

Theorem. *For $n > 1$ and $L \geq 1$, $\mathcal{H}_{\text{BIRNN}}(n, L) \subsetneq \mathcal{H}_{\text{BIRNN}}(n, L+1)$ for linear BIRNNs.*

Proof.

Inclusion: We begin by showing that for any $h \in \mathcal{H}_{\text{BIRNN}}(n, L)$, there exists a function $\tilde{h} \in \mathcal{H}_{\text{BIRNN}}(n, L+1)$ such that $h = \tilde{h}$, that is $\mathcal{H}_{\text{BIRNN}}(n, L) \subseteq \mathcal{H}_{\text{BIRNN}}(n, L+1)$. Let the function h be computed by a BIRNN parameterized by $\{\boldsymbol{\mathcal{A}}^{(l)}, \mathbf{h}_0^{(l)}\}_{l=1}^{L}$. We set the first L layers of the BIRNN computing $\tilde{h}$ the same as the one computing h such that $\tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$:

$$\tilde{\boldsymbol{\mathcal{A}}}^{(l)} = \boldsymbol{\mathcal{A}}^{(l)}, \quad \tilde{\mathbf{h}}_0^{(l)} = \mathbf{h}_0^{(l)} \quad \forall l \in [L].$$

Then, for the last layer $L+1$, it suffices to add residual connections from the Lth layer and set $\tilde{\boldsymbol{\mathcal{A}}}_{ijk} = \mathbf{0}$ and $\tilde{\mathbf{h}}_0^{(L+1)} = \mathbf{0}$ to have $\tilde{\mathbf{h}}_t^{(L+1)} = \tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$ which implies $\tilde{h} = h$ and proves the inclusion $\mathcal{H}_{\text{BIRNN}}(n, L) \subseteq \mathcal{H}_{\text{BIRNN}}(n, L+1)$.

Strict Inclusion: We show there exists a function $\tilde{h} \in \mathcal{H}_{\text{BIRNN}}(n, L+1)$ that cannot be computed by any function $h \in \mathcal{H}_{\text{BIRNN}}(n, L)$, that is $\mathcal{H}_{\text{BIRNN}}(n, L) \not\supseteq \mathcal{H}_{\text{BIRNN}}(n, L+1)$. Let $poly(\mathbf{x}, k)$ denote the set of functions having a polynomial dependency in $\mathbf{x}$ of degree up to k, irrespective of dependencies in other variables. First, we prove by induction that for any $h \in \mathcal{H}_{\text{BIRNN}}(n, L)$, the second time step output $\mathbf{h}_2^{(L)}$ has a polynomial dependency in $\mathbf{x}_1$ of degree at most L, i.e., $\mathbf{h}_2^{(L)} \in poly(\mathbf{x}_1, L)$ (using a slight abuse of notation):

- First, observe from Lemma 1 that the hidden states in the first time step are given by

$$\mathbf{h}_1^{(l)} = \left(\prod_{i=1}^{l} (\boldsymbol{\mathcal{A}}^{(i)} \times_1 \mathbf{h}_0^{(i)})^\top \right) \mathbf{x}_1$$

  which means that $\mathbf{h}_1^{(l)} \in poly(\mathbf{x}_1, 1)\ \forall l$.

- In the second time step, Lemma 1 can similarly be used to obtain

$$\mathbf{h}_2^{(l)} = \left(\prod_{i=1}^{l} (\boldsymbol{\mathcal{A}}^{(i)} \times_1 \mathbf{h}_1^{(i)})^\top \right) \mathbf{x}_2.$$

  $\mathbf{h}_2^{(l)}$ contains terms with a maximal number of l factors in $\{\mathbf{h}_1^{(i)}\}_{i=1}^{l}$, which, given that $\mathbf{h}_1^{(l)} \in poly(\mathbf{x}_1, 1)$, implies that $\mathbf{h}_2^{(l)} \in poly(\mathbf{x}_1, l)$.

Now we prove there exists a function $\tilde{h} \in \mathcal{H}_{\text{BIRNN}}(n, L+1)$ such that $\tilde{\mathbf{h}}_2^{(L+1)} = \text{diag}(\mathbf{x}_1)^{L+1}\mathbf{x}_2$. We begin with the case $n = d$ and generalize after. Consider the function $\tilde{h}$ parameterized by $\{\tilde{\boldsymbol{\mathcal{A}}}^{(l)} = \delta_{ijk}, \tilde{\mathbf{h}}_0^{(l)} = [1, \ldots, 1]^\top\}_{l=1}^{L}$. From Lemma 1 and by observing that for any $\mathbf{y} \in \mathbb{R}^n$, $\tilde{\boldsymbol{\mathcal{A}}}^{(l)} \times_1 \mathbf{y} = \text{diag}(\mathbf{y})$, we obtain:

$$\tilde{\mathbf{h}}_t^{(l)} = (\prod_{i=1}^{l} \text{diag}(\tilde{\mathbf{h}}_{t-1}^{(i)}))\mathbf{x}_t.$$

Given the hidden states initialization, we have $\text{diag}(\tilde{\mathbf{h}}_0^{(l)}) = \mathbf{I} \quad \forall l$ and the hidden states in the first time step become $\tilde{\mathbf{h}}_1^{(l)} = \mathbf{x}_1$. Inserting in the expression for $\tilde{\mathbf{h}}_2^{(l)}$, we obtain

$$\tilde{\mathbf{h}}_2^{(l)} = \text{diag}(\mathbf{x}_1)^l \mathbf{x}_2.$$

This proves that $\tilde{h}$ is such that $\tilde{\mathbf{h}}_2^{(L+1)} = \text{diag}(\mathbf{x}_1)^{L+1}\mathbf{x}_2$. This function can not be computed by any $h \in \mathcal{H}_{\text{BIRNN}}(n, L)$ as for any such h, $\mathbf{h}_2^{(L)} \in poly(\mathbf{x}_1, L)$.

In the case $n > d$, we set $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = \delta_{ijk} \forall i, j, k \in [1, d]$ and $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = 0 \quad \forall i, k \in [d+1, n]$. The result will be the same, but with 0 padding on the extra dimensions. In the case $n < d$, we set $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = \delta_{ijk} \forall j \in [1, n]$ and $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = 0 \quad \forall j \in [n+1, d]$. The result will be the same, but for the first n dimensions of the inputs : $\mathbf{h}_2^{(L+1)} = \text{diag}([\mathbf{x}_1]_{1:n})^{L+1}[\mathbf{x}_2]_{1:n}$.

□

A.7.1 Theorem 3 for 2RNNs

In the main text, we focused on BIRNNs to isolate the effect of bilinear interactions, but the same proof technique can be used to prove the result for 2RNNs. Here is the adaptation of Theorem 3 for 2RNNs and its proof.

Theorem. *For $n > 1$ and $L \geq 1$, $\mathcal{H}_{\text{2RNN}}(n, L) \subsetneq \mathcal{H}_{\text{2RNN}}(n, L+1)$ for linear BIRNNs.*

Proof.

Inclusion: We begin by showing that for any $h \in \mathcal{H}_{\text{2RNN}}(n, L)$, there exists a function $\tilde{h} \in \mathcal{H}_{\text{2RNN}}(n, L+1)$ such that $h = \tilde{h}$, that is $\mathcal{H}_{\text{2RNN}}(n, L) \subseteq \mathcal{H}_{\text{2RNN}}(n, L+1)$. Let the function h be computed by a 2RNN parameterized by $\{\boldsymbol{\mathcal{A}}^{(l)}, \mathbf{U}^{(l)}, \mathbf{V}^{(l)}, \mathbf{b}^{(l)}, \mathbf{h}_0^{(l)}\}_{l=1}^{L}$. We set the first L layers of the 2RNN computing $\tilde{h}$ the same as the one computing h such that $\tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$:

$$\tilde{\boldsymbol{\mathcal{A}}}^{(l)} = \boldsymbol{\mathcal{A}}^{(l)}, \quad \tilde{\mathbf{U}}^{(l)} = \mathbf{U}^{(l)}, \quad \tilde{\mathbf{V}}^{(l)} = \mathbf{V}^{(l)}, \quad \tilde{\mathbf{b}}^{(l)} = \mathbf{b}^{(l)}, \quad \tilde{\mathbf{h}}_0^{(l)} = \mathbf{h}_0^{(l)} \quad \forall l \in [L].$$

Then, the last layer $L+1$ is set to perform the identity on $\tilde{\mathbf{h}}_t^{(L)}$:

$$\tilde{\boldsymbol{\mathcal{A}}} = \mathbf{0}, \quad \tilde{\mathbf{U}}^{(L+1)} = \mathbf{I}_n, \quad \tilde{\mathbf{V}}^{(L+1)} = \mathbf{0}, \quad \tilde{\mathbf{b}}^{(L+1)} = \mathbf{0}, \quad \tilde{\mathbf{h}}_0^{(L+1)} = \mathbf{0}$$

We thus have $\tilde{\mathbf{h}}_t^{(L+1)} = \tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$ which implies $\tilde{h} = h$ and proves the inclusion $\mathcal{H}_{\text{2RNN}}(n, L) \subseteq \mathcal{H}_{\text{2RNN}}(n, L+1)$.

Strict Inclusion: We show there exists a function $\tilde{h} \in \mathcal{H}_{\text{2RNN}}(n, L+1)$ that cannot be computed by any function $h \in \mathcal{H}_{\text{2RNN}}(n, L)$, that is $\mathcal{H}_{\text{2RNN}}(n, L) \not\supset \mathcal{H}_{\text{2RNN}}(n, L+1)$. Let $poly(\mathbf{x}, k)$ denote the set of functions having a polynomial dependency in $\mathbf{x}$ of degree up to k, irrespective of dependencies in other variables. First, we prove by induction that for any $h \in \mathcal{H}_{\text{2RNN}}(n, L)$, the second time step output $\mathbf{h}_2^{(L)}$ has a polynomial dependency in $\mathbf{x}_1$ of degree at most L, i.e., $\mathbf{h}_2^{(L)} \in poly(\mathbf{x}_1, L)$ (using a slight abuse of notation):

- First, observe from Lemma 1 that the hidden states in the first time step are given by

$$\mathbf{h}_1^{(l)} = (\prod_{i=1}^{l} (\boldsymbol{\mathcal{A}}^{(i)} \times_1 \mathbf{h}_0^{(i)})^\top + \mathbf{U}^{(i)})\mathbf{x}_1 + \sum_{j=1}^{l} (\prod_{k=j+1}^{l} (\boldsymbol{\mathcal{A}}^{(k)} \times_1 \mathbf{h}_0^{(k)})^\top + \mathbf{U}^{(k)})(\mathbf{V}^{(j)}\mathbf{h}_0^{(j)} + \mathbf{b}^{(j)})$$

  which means that $\mathbf{h}_1^{(l)} \in poly(\mathbf{x}_1, 1) \; \forall l$.

- In the second time step, Lemma 1 can similarly be used to obtain

$$\mathbf{h}_2^{(l)} = (\prod_{i=1}^{l} (\boldsymbol{\mathcal{A}}^{(i)} \times_1 \mathbf{h}_1^{(i)})^\top + \mathbf{U}^{(i)})\mathbf{x}_2 + \sum_{j=1}^{l} (\prod_{k=j+1}^{l} (\boldsymbol{\mathcal{A}}^{(k)} \times_1 \mathbf{h}_1^{(k)})^\top + \mathbf{U}^{(l)})(\mathbf{V}^{(j)}\mathbf{h}_1^{(j)} + \mathbf{b}^{(j)}).$$

$\mathbf{h}_2^{(l)}$ contains terms with a maximal number of l factors in $\{\mathbf{h}_1^{(i)}\}_{i=1}^{l}$, which, given that $\mathbf{h}_1^{(l)} \in poly(\mathbf{x}_1, 1)$, implies that $\mathbf{h}_2^{(l)} \in poly(\mathbf{x}_1, l)$.

Now we prove there exists a function $\tilde{h} \in \mathcal{H}_{\text{2RNN}}(n, L+1)$ such that $\tilde{\mathbf{h}}_2^{(L+1)} = \text{diag}(\mathbf{x}_1)^{L+1}\mathbf{x}_2$. We begin with the case $n = d$ and generalize after. Consider the function $\tilde{h}$ parameterized by $\{\tilde{\boldsymbol{\mathcal{A}}}^{(l)} = \delta_{ijk}, \tilde{\mathbf{U}}^{(l)} = \mathbf{0}, \tilde{\mathbf{V}}^{(l)} = \mathbf{0}, \tilde{\mathbf{b}}^{(l)} = \mathbf{0}, \tilde{\mathbf{h}}_0^{(l)} = [1, \dots, 1]^\top\}_{l=1}^{L}$. By keeping only $\tilde{\boldsymbol{\mathcal{A}}}^{(l)}$ in Lemma 1, we obtain that $\tilde{\mathbf{h}}_t^{(l)} = (\prod_{i=1}^{l} \tilde{\boldsymbol{\mathcal{A}}}^{(i)} \times_1 \tilde{\mathbf{h}}_{t-1}^{(i)})^\top \mathbf{x}_t$. By observing that for any $\mathbf{y} \in \mathbb{R}^n$, $\tilde{\boldsymbol{\mathcal{A}}}^{(l)} \times_1 \mathbf{y} = \text{diag}(\mathbf{y})$, this is simplified to

$$\tilde{\mathbf{h}}_t^{(l)} = (\prod_{i=1}^{l} \text{diag}(\tilde{\mathbf{h}}_{t-1}^{(i)}))\mathbf{x}_t.$$

Given the hidden states initialization, we have $\text{diag}(\tilde{\mathbf{h}}_0^{(l)}) = \mathbf{I} \quad \forall l$ and the hidden states in the first time step become $\tilde{\mathbf{h}}_1^{(l)} = \mathbf{x}_1$. Inserting in the expression for $\tilde{\mathbf{h}}_2^{(l)}$, we obtain

$$\tilde{\mathbf{h}}_2^{(l)} = \text{diag}(\mathbf{x}_1)^l \mathbf{x}_2.$$

This proves that $\tilde{h}$ is such that $\tilde{\mathbf{h}}_2^{(L+1)} = \text{diag}(\mathbf{x}_1)^{L+1}\mathbf{x}_2$. This function can not be computed by any $h \in \mathcal{H}_{\text{2RNN}}(n, L)$ as for any such h, $\mathbf{h}_2^{(L)} \in poly(\mathbf{x}_1, L)$.

In the case $n > d$, we set $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = \delta_{ijk} \forall i, j, k \in [1, d]$ and $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = 0 \quad \forall i, k \in [d+1, n]$. The result will be the same, but with 0 padding on the extra dimensions. In the case $n < d$, we set $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = \delta_{ijk} \forall j \in [1, n]$ and $\tilde{\boldsymbol{\mathcal{A}}}_{ijk}^{(1)} = 0 \quad \forall j \in [n+1, d]$. The result will be the same, but for the first n dimensions of the inputs : $\mathbf{h}_2^{(L+1)} = \text{diag}([\mathbf{x}_1]_{1:n})^{L+1}[\mathbf{x}_2]_{1:n}$.

□

A.8 Proof of Corollary 1

The same proof technique presented in Appendix A.7 for Theorem 3 can be applied to CPBIRNNs. It suffices to replace the parameter weight tensor $\boldsymbol{\mathcal{A}}$ by the CP decomposition matrices $[\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ (the formal definition for CP(BI)RNN can be found in Appendix A.1.3).

Corollary. *For $n > 1$, $L \geq 1$ and any $\tilde{R} \geq R > 1$, $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subsetneq \mathcal{H}_{\text{CPBIRNN}}(n, L+1, \tilde{R})$ for linear CPBIRNNs.*

Proof.

Inclusion: We begin by showing that for any $h \in \mathcal{H}_{\text{CPBIRNN}}(n, L, R)$, there exists a function $\tilde{h} \in \mathcal{H}_{\text{CPBIRNN}}(n, L+1, R)$ such that $h = \tilde{h}$, that is $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subseteq \mathcal{H}_{\text{CPBIRNN}}(n, L+1, R)$. Let the function h be computed by a CPBIRNN parameterized by $\{\mathbf{A}^{(l)}, \mathbf{B}^{(l)}, \mathbf{C}^{(l)}, \mathbf{h}_0^{(l)}\}_{l=1}^{L}$. We set the first L layers of the CPBIRNN computing $\tilde{h}$ the same as the one computing h such that $\tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$:

$$\tilde{\mathbf{A}}^{(l)} = \mathbf{A}^{(l)}, \quad \tilde{\mathbf{B}}^{(l)} = \mathbf{B}^{(l)}, \quad \tilde{\mathbf{C}}^{(l)} = \mathbf{C}^{(l)}, \quad \tilde{\mathbf{h}}_0^{(l)} = \mathbf{h}_0^{(l)} \quad \forall l \in [L].$$

Then, for the last layer $L+1$, it suffices to add residual connections from the Lth layer and set $\tilde{\mathbf{A}}^{(L+1)} = \mathbf{0}, \tilde{\mathbf{B}}^{(L+1)} = \mathbf{0}, \tilde{\mathbf{C}}^{(L+1)} = \mathbf{0}$ and $\tilde{\mathbf{h}}_0^{(L+1)} = \mathbf{0}$ to have $\tilde{\mathbf{h}}_t^{(L+1)} = \tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$ which implies $\tilde{h} = h$ and proves the inclusion $\mathcal{H}_{\text{CPBIRNN}}(n, L) \subseteq \mathcal{H}_{\text{CPBIRNN}}(n, L+1)$.

Strict Inclusion: We show there exists a function $\tilde{h} \in \mathcal{H}_{\text{CPBIRNN}}(n, L+1, R)$ that cannot be computed by any function $h \in \mathcal{H}_{\text{CPBIRNN}}(n, L, R)$, that is $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \not\supset \mathcal{H}_{\text{CPBIRNN}}(n, L+1, R)$. Let $poly(\mathbf{x}, k)$ denote the set of functions having a polynomial dependency in $\mathbf{x}$ of degree up to k, irrespective of dependencies in other variables. First, we prove by induction that for any $h \in \mathcal{H}_{\text{CPBIRNN}}(n, L, R)$, the second time step output $\mathbf{h}_2^{(L)}$ has a polynomial dependency in $\mathbf{x}_1$ of degree at most L, i.e., $\mathbf{h}_2^{(L)} \in poly(\mathbf{x}_1, L)$ (using a slight abuse of notation):

- First, observe from Lemma 1 that the hidden states in the first time step are given by

$$\mathbf{h}_1^{(l)} = \left(\prod_{i=1}^{l} ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^{(i)} \times_1 \mathbf{h}_0^{(i)})^\top \right) \mathbf{x}_1$$

  which means that $\mathbf{h}_1^{(l)} \in poly(\mathbf{x}_1, 1)\ \forall l$.

- In the second time step, Lemma 1 can similarly be used to obtain

$$\mathbf{h}_2^{(l)} = \left(\prod_{i=1}^{l} ([\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]^{(i)} \times_1 \mathbf{h}_1^{(i)})^\top \right) \mathbf{x}_2.$$

  $\mathbf{h}_2^{(l)}$ contains terms with a maximal number of l factors in $\{\mathbf{h}_1^{(i)}\}_{i=1}^{l}$, which, given that $\mathbf{h}_1^{(l)} \in poly(\mathbf{x}_1, 1)$, implies that $\mathbf{h}_2^{(l)} \in poly(\mathbf{x}_1, l)$.

Now we prove there exists a function $\tilde{h} \in \mathcal{H}_{\text{CPBIRNN}}(n, L+1, R)$ such that $\tilde{\mathbf{h}}_2^{(L+1)} = \text{diag}(\mathbf{x}_1)^{L+1}\mathbf{x}_2$. We begin with the case $n = d = R$ and generalize after. Consider the function $\tilde{h}$ parameterized by $\{\tilde{\mathbf{A}}^{(l)} = \mathbf{I}, \tilde{\mathbf{B}}^{(l)} = \mathbf{I}, \tilde{\mathbf{C}}^{(l)} = \mathbf{I}, \tilde{\mathbf{h}}_0^{(l)} = [1, \ldots, 1]^\top\}_{l=1}^{L}$. From Lemma 1 and by observing that for any $\mathbf{y} \in \mathbb{R}^n$, $[\![\tilde{\mathbf{A}}^{(l)}, \tilde{\mathbf{B}}^{(l)}, \tilde{\mathbf{C}}^{(l)}]\!] \times_1 \mathbf{y} = \text{diag}(\mathbf{y})$, we obtain:

$$\tilde{\mathbf{h}}_t^{(l)} = \left(\prod_{i=1}^{l} \text{diag}(\tilde{\mathbf{h}}_{t-1}^{(i)}) \right) \mathbf{x}_t.$$

Given the hidden states initialization, we have $\text{diag}(\tilde{\mathbf{h}}_0^{(l)}) = \mathbf{I} \quad \forall l$ and the hidden states in the first time step become $\tilde{\mathbf{h}}_1^{(l)} = \mathbf{x}_1$. Inserting in the expression for $\tilde{\mathbf{h}}_2^{(l)}$, we obtain

$$\tilde{\mathbf{h}}_2^{(l)} = \text{diag}(\mathbf{x}_1)^l \mathbf{x}_2.$$

This proves that $\tilde{h}$ is such that $\tilde{\mathbf{h}}_2^{(L+1)} = \text{diag}(\mathbf{x}_1)^{L+1}\mathbf{x}_2$. This function can not be computed by any $h \in \mathcal{H}_{\text{CPBIRNN}}(n, L, R)$ as for any such h, $\mathbf{h}_2^{(L)} \in poly(\mathbf{x}_1, L)$.

In the case $n > d$ and $R > d$, we set $\tilde{\mathbf{A}}_{ij}^{(1)} = \tilde{\mathbf{C}}_{ij}^{(1)} = \tilde{\mathbf{B}}_{ij}^{(1)} = \delta_{ij} \forall i, j \in [1, d]$, and 0 in the other rows and columns. The result will be the same, but with 0 padding on the extra dimensions. In the case $n < d$ or $R < d$, let $k = \min(n, R)$, we set $\tilde{\mathbf{A}}_{ij}^{(1)} = \tilde{\mathbf{C}}_{ij}^{(1)} = \tilde{\mathbf{B}}_{ij}^{(1)} = \delta_{ij} \forall i, j \in [1, k]$, and 0 in the other rows and columns. The result will be the same, but for the first k dimensions of the inputs : $\mathbf{h}_2^{(L+1)} = \text{diag}([\mathbf{x}_1]_{1:k})^{L+1}[\mathbf{x}_2]_{1:k}$. □

A.9 Proof of Theorem 4

Theorem. *For $n > 1$, $L \geq 1$ and any $R \leq R_{max}$, $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subsetneq \mathcal{H}_{\text{CPBIRNN}}(n, L, R+1)$ for linear CPBIRNNs if $R < n$.*

Proof.

Inclusion: We begin by showing that for any $h \in \mathcal{H}_{\text{CPBIRNN}}(n, L, R)$, there exists a function $\tilde{h} \in \mathcal{H}_{\text{CPBIRNN}}(n, L, R+1)$ such that $h = \tilde{h}$, that is $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subseteq \mathcal{H}_{\text{CPBIRNN}}(n, L, R+1)$. Let the function h be computed by a CPBIRNN of rank R parameterized by $\{\mathbf{A}^{(l)}, \mathbf{B}^{(l)}, \mathbf{C}^{(l)}, \mathbf{h}_0^{(l)}\}_{l=1}^{L}$. For a CPBIRNN of rank $R+1$ computing $\tilde{h}$, we set:

$$\begin{aligned}
\tilde{\mathbf{A}}_{ij}^{(l)} &= \mathbf{A}_{ij}^{(l)} \quad \forall i \in [n],\ j \geq R \\
\tilde{\mathbf{B}}_{ij}^{(l)} &= \mathbf{B}_{ij}^{(l)} \quad \forall i \in [d], j \geq R, \\
\tilde{\mathbf{C}}_{ij}^{(l)} &= \mathbf{C}_{ij}^{(l)} \quad \forall i \in [n], j \geq R, \\
\tilde{\mathbf{h}}_0^{(l)} &= \mathbf{h}_0^{(l)},
\end{aligned}$$

and all other dimensions of $\mathbf{A}^{(l)}, \mathbf{B}^{(l)}$ and $\mathbf{C}^{(l)}$ to 0, $\forall l \in [L]$. We thus have $\tilde{\mathbf{h}}_t^{(L)} = \mathbf{h}_t^{(L)} \forall t$ which implies $\tilde{h} = h$ and proves the inclusion $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \subseteq \mathcal{H}_{\text{CPBIRNN}}(n, L, R+1)$. □

Strict Inclusion: To show strict inclusion, i.e. $\mathcal{H}_{\text{CPBIRNN}}(n, L, R) \not\supset \mathcal{H}_{\text{CPBIRNN}}(n, L, R+1)$, first observe that from Lemma 1 and Equation 5,

$$\mathbf{h}_1^{(l)} = \left(\prod_{i=1}^{l} (\llbracket \mathbf{A}, \mathbf{B}, \mathbf{C} \rrbracket^{(i)} \times_1 \mathbf{h}_0^{(i)})^\top\right) \mathbf{x}_1 = \left(\prod_{i=1}^{l} \left[\mathbf{C}^{(i)} \text{diag}(\mathbf{A}^{(i)\top} \mathbf{h}_{t-1}^{(i)}) \mathbf{B}^{(i)\top}\right]\right) \mathbf{x}_1.$$

One can easily check that, independently of l, for any CPBIRNN of rank R, the image of the map $\mathbf{x}_1 \mapsto \mathbf{h}^{(L)}$ has dimension at most R, i.e. $\text{Im}(h_1(\mathbb{R}^d)) \leq R$. To complete the proof for strict inclusion, we only have to show there exists a CPBIRNN of rank $R+1$ that reaches that limit ($\text{Im}(h_1(\mathbb{R}^d)) = R+1$), since no CPBIRNNs of rank R could compute this map. It suffices to consider any CPBIRNNs such that $\text{diag}(\mathbf{A}^{(i)\top} \mathbf{h}_0^{(l)}), \mathbf{B}^{(l)}, \mathbf{C}^{(l)}$ for $l \in [L]$ are full rank weight matrices.

A.10 Proof of Theorem 5

Before presenting the proof of Theorem 5, we introduce a lemma that will be used in the proof of Theorem 5.

Lemma 4. *Let f be a function over sequences of d-dimensional vectors computed by a 2RNN with n states ($n \geq d$) and parameters $\boldsymbol{\mathcal{A}}, \mathbf{U} = \mathbf{0}, \mathbf{V} = \mathbf{0}, \mathbf{b} = \mathbf{0}, \mathbf{h}_0 = \mathbf{f}_0 \neq \mathbf{0}$ with $\boldsymbol{\mathcal{A}}$ generic.*

Then, the function f cannot be computed by any RNN with one layer of any width and nonlinear activations applied only in depth.

Proof. The computation of a 1 layer RNN of hidden size $\tilde{n}$ with nonlinear activations ϕ applied only in depth at each time step t is given by $\mathbf{y}_t = \phi(\mathbf{h}_t)$ where $\mathbf{h}_t = \mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{b}$. Here we assume ϕ is a homeomorphism as are most common activation functions (tanh, sigmoid, ReLU on $\mathbb{R}_{\geq 0}$). For such a RNN to *compute* f, means there exists a linear map $\ell \in \mathcal{L}^{\tilde{n}, n}$ such that $\mathbf{f}_t = \ell(\mathbf{y}_t)$ for all t.

Observe that $\mathbf{h}_t = h(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ with $h \in \mathcal{L}^{dt, \tilde{n}}$ a linear map.

The function f is a multi-linear mapping of $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$. Given its recursive nature, it can be computed as a bilinear map between $\mathbf{x}_t$ and $\mathbf{f}_{t-1}$, i.e. $f : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}^n$, with $f(\mathbf{x}_t, \mathbf{f}_{t-1}) = (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_t)\mathbf{f}_{t-1}$, but computing f_{t-1} involves a nonlinear operation between $(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$. Indeed, the entries of a matrix obtained by multiplying slices of a generic tensor (i.e. $\boldsymbol{\mathcal{A}}$) are polynomial functions of the tensor's entries. Since a nonzero polynomial (over $\mathbb{R}$) vanishes only on a set of Lebesgue measure zero (unless it is identically zero) and $\boldsymbol{\mathcal{A}}$ is generic, all entries of this matrix are nonzero with probability 1. Thus, $\mathbf{f}_{t-1}$ is a polynomial of $(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$:

$$\mathbf{f}_{t-1} = \left(\sum_{i_1, \ldots, i_{t-1}} \underbrace{\boldsymbol{\mathcal{A}}_{:, i_{t-1}, :} \ldots \boldsymbol{\mathcal{A}}_{:, i_1, :}}_{\neq 0} (\mathbf{x}_{t-1})_{i_{t-1}} \ldots (\mathbf{x}_1)_{i_1}\right) \mathbf{f}_0$$

Since ϕ is a homeomorphism, h must encode $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ in order for $\mathbf{f}_t = \ell(\mathbf{y}_t)$ for all t, which can only be the case if $\tilde{n} \geq dt$. However, for any (arbitrarily large) $\tilde{n}$, there will always be a t sufficiently large such that $\tilde{n} < dt$, and therefore, no such RNN can compute f.

□

Theorem. *There exists a function computed by a single-layer 2RNN that cannot be computed by any RNN of arbitrary (finite) depth and width with nonlinear activation applied only in depth.*

Proof. We consider a 1 layer 2RNN of hidden size n with parameters $\boldsymbol{\mathcal{A}}, \mathbf{U} = \mathbf{0}, \mathbf{V} = \mathbf{0}, \mathbf{b} = \mathbf{0}, \mathbf{h}_0 = \mathbf{f}_0$ satisfying the conditions of Lemma 4, computing the function $f : (\mathbb{R}^d)^n \to \mathbb{R}^n$ mapping $(\mathbf{x}_1, \ldots, \mathbf{x}_T) \mapsto (\mathbf{f}_1, \ldots, \mathbf{f}_T)$ with the computation at each time step being $\mathbf{f}_t = (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_t)\mathbf{f}_{t-1}$. By unrolling the computation, we obtain $\mathbf{f}_t = (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_t)(\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_{t-1}) \ldots (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_1)\mathbf{f}_0$. The function f can be viewed as a multi-linear mapping of $(\mathbf{x}_1, \ldots, \mathbf{x}_T)$, and given its recursive nature, it can also be computed as a bilinear map between $\mathbf{x}_T$ and $\mathbf{f}_{T-1}$, i.e. $f : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}^n$, with $f(\mathbf{x}_T, \mathbf{f}_{T-1}) = (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_T)\mathbf{f}_{T-1}$. More generally, we can compute $\mathbf{f}_T = (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_T) \ldots (\boldsymbol{\mathcal{A}} \times_2 \mathbf{x}_{T-i+1})\mathbf{f}_{T-i}$, so $f : ((\mathbb{R}^d)^i \times \mathbb{R}^n) \to \mathbb{R}^n$ is a multi-linear map of $(\mathbf{x}_T, ..., \mathbf{x}_{T-i+1}, \mathbf{f}_{T-i})$.

Now consider a RNN of hidden size $\tilde{n}$, depth L and nonlinear activations $\phi^{(l)}$ applied only in depth that computes at each time step $t \in [T]$ and layer $l \in [L]$: $\mathbf{y}_t^{(l)} = \phi^{(l)}(\mathbf{h}_t^{(l)})$ where $\mathbf{h}_t^{(l)} = \mathbf{U}^{(l)}\mathbf{y}_t^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)}$. We assume $\phi^{(l)}$'s are homeomorphisms (as are the common activations tanh, sigmoid and ReLU on $\mathbb{R}_{\geq 0}$).

We will suppose this RNN is capable of computing f for any sequence length T and show that it leads to a contradiction. We write an output $\mathbf{y}_t^{(L)}$ that would achieve the computation as $\bar{\mathbf{f}}_t \in \mathbb{R}^{\tilde{n}}$ and suppose that there exists a linear map $\ell \in \mathcal{L}^{\tilde{n},n}$ to complete the computation $\mathbf{f}_t = \ell(\bar{\mathbf{f}}_t)$ for all $t \in [T]$.

Assume $\mathbf{y}_t^{(L)} = \phi^{(L)}(\mathbf{h}_t^{(L)}) = \bar{\mathbf{f}}_t \ \forall t$. Then, either (1) $\mathbf{h}_t^{(L)}$ encodes the full sequence $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$ or (2) $\mathbf{h}_t^{(L)}$ encodes the 2RNN state from some previous time step along with all subsequent inputs, i.e., $(\mathbf{x}_t, \ldots, \mathbf{x}_{t-i+1}, \mathbf{f}_{t-i})$ for some $i \in [t-1]$. In case (1), since $\phi^{(l)}$ is a homeomorphism at all layers l and the dimension of the space of all possible input sequences of length t, $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$, is dt, if $\mathbf{h}_t^{(L)}$ encodes $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$, it is necessary that $\mathbf{h}_t^{(L)}$ lies in a space of dimension at least dt. However, we can always find t such that the dimension of $\mathbf{h}_t^{(L)}$, $\tilde{n} < dt$, contradicting the dimension condition required for $\mathbf{h}_t^{(L)}$ to encode $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$. We thus have to turn to case (2): $\mathbf{h}_t^{(L)}$ must encode $(\mathbf{x}_t, \ldots, \mathbf{x}_{t-i+1}, \mathbf{f}_{t-i})$. Recall that $\mathbf{h}_t^{(L)}$ is a linear map of $(\mathbf{y}_1^{(L-1)}, \ldots, \mathbf{y}_t^{(L-1)})$. The information $\mathbf{f}_{t-i}$ is thus encoded in (at least) one of the inputs $(\mathbf{y}_{t-i}^{(L-1)}, \ldots, \mathbf{y}_t^{(L-1)})$. Without loss of generality, let $\mathbf{y}_{t-j}^{(L-1)}$ with $j \leq i$ encode $\mathbf{f}_{t-i}$. We have that $\mathbf{y}_{t-j}^{(L-1)}$ encodes all the information of $(\mathbf{x}_{t-j}, \ldots \mathbf{x}_{t-i}, \mathbf{f}_{t-i})$. This implies that one can find $\tilde{\phi}^{(L-1)}$ such that $\mathbf{y}_{t-j}^{(L-1)} = \bar{\mathbf{f}}_{t-j}$, which in turn implies that a RNN could compute $\bar{\mathbf{f}}_t$ with $L-1$ layers. The same argument can then be used to show that if it can be computed with $L-1$ layers, it can be done with $L-2$, and so on, until 1 layer.

However, by Lemma 4, the function f cannot be computed by any 1-layer RNN with activation only in depth: a contradiction.

□

B Experimental details

B.1 Copy, copy-sinus and sinus tasks (synthetic experiments)

Given inputs $\mathbf{x}_t \in \mathbb{R}^5$ sampled from $\mathcal{N}(0,1)$, the copy task presented in Section 4.1 consists of predicting $\mathbf{x}_{t-p}$ at each time step, with a lag set to $p = 8$. Similarly, the copy-sinus and sinus tasks consist in predicting $\sin(\omega \mathbf{x}_{t-p})$, with frequency $\omega = 3$ and lags $p = 4$ and 0, respectively. The training sets contain 10,000 sequences of length 16; the validation and test sets each contain 200 sequences. Models were trained with a batch size of 128, learning rates between 0.001 and 0.002 and early stopping on the validation set with a patience of 400. For the copy task, we studied a linear RNNs, while for the copy-sinus and sinus tasks, we used a RNN with tanh activation applied only in depth. Weights were initialized from a random uniform distribution, $\mathcal{U}[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. Figures 6 and 7 show the average values over 3–5 random seeds, with the standard deviation indicated by the shaded area. All experiments were run using one GPU (RTX8000, L40S or V100) with 32GB of memory. Generating one point took less than 30 minutes for the shortest experiments (copy) and less than 1.5 hours for the longest (copy-sinus).

B.2 Parity (synthetic experiments)

To produce inputs for the parity task, we take the signs of $\mathbf{x}_t \in \mathbb{R}^5$ sampled from $\mathcal{N}(0,1)$, resulting in vectors of filled with -1s and 1s. The training set contains 10,000 sequences of length 20; the validation and test sets each contain 2000 sequences. The targets consist in element-wise products of the input sequences: $\mathbf{y}_t = \mathbf{x}_1 ... \mathbf{x}_t$. Models were trained with a batch size of 128, a learning rate 0.001 (0.0005 for few runs) and early stopping with a patience of 400 on the validation set. Figure 8 displays the mean values computed over 3–5 random seeds, with the shaded region representing the standard deviation. We studied RNNs with tanh activation applied either in depth or in recurrence, with weights were initialized from a uniform distribution, $\mathcal{U}[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. Experiments were run on a single GPU (RTX8000, L40S or V100) with 32GB of memory, and it took between 10 minutes and 2 hours for each point.

B.3 Language modeling experiments - Tiny Shakespeare

Language modeling experiments were conducted on the Tiny Shakespeare dataset Karpathy [2015] at the character level, using an embedding size of 107. The dataset was split into training, validation and test sets with a ratio of 0.8, 0.1 and 0.1, respectively. Models were trained on sequences of length 64, with early stopping based on validation performance and a patience of 200 epochs. If the training loss plateaued before showing any learning (i.e., decreasing a reasonable amount), optimization was restarted. In Figure 9, we present the average values over 3–5 random seeds, with the standard deviation shown as a shaded area. All experiments were run on a single GPU (RTX8000, L40S or V100) with 32GB of memory, and training times ranged from 40 minutes to 6 hours.

RNNs and CPRNNs were initialized with weights drawn from a uniform distribution, $\mathcal{U}[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$, and had tanh activation in the recurrent connections. For RNNs, a batch size of 128 and a learning rate of 0.001 were set across all depths and hidden sizes. For CPRNNs, two optimization regimes were adopted to ensure stability: one with a higher learning rate (0.0005 - 0.001) and a batch size of 128, and another with a lower learning (0.0001) and batch size of 32. The former was used for all hidden sizes at $L = 1$, as well as for $n = [64, 128, 256]$ at $L = 2$ and $L = 4$. The latter was required for the larger models with $n = [512, 1024]$ at $L = 2$ and $L = 4$. Similarly, the rank was set to twice the hidden size for all configurations at $L = 1$, except for $n = 1024$, and for $n = [64, 128]$ at $L = 2$ and $L = 4$. In all other cases, the rank was set equal to the hidden size. S4 models used the diagonal-plus-low-rank kernel Gu et al. [2022], GeLU activation, input normalization, and a dropout rate of 0.1. These models were trained with a learning rate of 0.0001 and a batch size of 32.

B.4 Long Range Arena Benchmark on S4

All experiments on the Long Range Arena benchmark Tay et al. [2021] were conducted using S4 models based on the official implementation from the original S4 paper Gu et al. [2021]. We used an RTX8000 GPU with 48GB of memory. Each run for the shortest experiment (ListOps) took between 5 and 10 hours, while for Retrieval and Images it ranged between 15 to 35 hours. The longest, Pathfinder, required between 30 and 60 hours.

All parameters, except for the depth L and number of features H, were kept the same as the original settings for each tasks. The depth used in the original paper was $L = 6$, for our purposes, we varied the depth from 2 to 8 and adjusted the number of features H to maintain a constant parameter count (see Table 2).

Table 2: S4 parameters for Long Range Arena experiments: depth L, number of features H, and the resulting number of parameters. State dimension N is fixed (4 for Retrieval and ListOps, 64 for Images and Pathfinder). Rows corresponding to $L = 6$ reflect configurations from Gu et al. [2021].

L	Parameter	Retrieval	Images	ListOps	Pathfinder
6	H	256	512	256	256
	#params	808k	3.6M	808k	1.3M
2	H	446	922	446	484
	#params	808k	3.6M	808k	1.3M
3	H	364	745	364	385
	#params	810k	3.6M	810k	1.3M
8	H	221	440	221	220
	#params	806k	3.6M	806k	1.3M