ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Seminar for Applied Mathematics SAM

UnICORNN: A recurrent model for learning very long time dependencies

T. K. Rusch and S. Mishra

Research Report No. 2021-10

March 2021
Latest revision: June 2021

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Funding ERC: 770880 COMANFLO

UnICORNN: A recurrent model for learning *very* long time dependencies

T. Konstantin Rusch [1] **Siddhartha Mishra** [1]

Abstract

The design of recurrent neural networks (RNNs) to accurately process sequential inputs with long-time dependencies is very challenging on account of the exploding and vanishing gradient problem. To overcome this, we propose a novel RNN architecture which is based on a structure preserving discretization of a Hamiltonian system of second-order ordinary differential equations that models networks of oscillators. The resulting RNN is fast, invertible (in time), memory efficient and we derive rigorous bounds on the hidden state gradients to prove the mitigation of the exploding and vanishing gradient problem. A suite of experiments are presented to demonstrate that the proposed RNN provides state of the art performance on a variety of learning tasks with (very) long-time dependencies.

1. Introduction

Recurrent Neural Networks (RNNs) have been very successful in solving a diverse set of learning tasks involving sequential inputs (LeCun et al., 2015). These include text and speech recognition, time-series analysis and natural language processing. However, the well-known *Exploding and Vanishing Gradient Problem* (EVGP) (Pascanu et al., 2013) and references therein, impedes the efficiency of RNNs on tasks that require processing (very) long sequential inputs. The EVGP arises from the fact that the backpropagation through time (BPTT) algorithm for training RNNs entails computing products of hidden state gradients over a large number of steps and this product can either be exponentially small or large as the number of recurrent interactions increases.

Different approaches to solve the EVGP has been suggested in recent years. These include the use of gating mechanisms, such as in LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Cho et al., 2014), where the additive structure of the gates mitigates the vanishing gradient problem. However, gradients might still explode, impeding the efficiency of LSTMs and GRUs on problems with very long time dependencies (LTDs) (Li et al., 2018). The EVGP can also be mitigated by constraining the structure of the recurrent weight matrices, for instance requiring them to be orthogonal or unitary (Henaff et al., 2016; Arjovsky et al., 2016; Wisdom et al., 2016; Kerg et al., 2019). Constraining recurrent weight matrices may lead to a loss of expressivity of the resulting RNN, reducing its efficiency in handling realistic learning tasks (Kerg et al., 2019). Finally, restricting weights of the RNN to lie within some prespecified bounds might lead to control over the norms of the recurrent weight matrices and alleviate the EVGP. Such an approach has been suggested in the context of *independent neurons* in each layer in (Li et al., 2018), and using a coupled system of damped oscillators in (Rusch & Mishra, 2021), among others. However, ensuring that weights remain within a pre-defined range during training might be difficult. Furthermore, *weight clipping* could also reduce expressivity of the resulting RNN.

In addition to EVGP, the learning of sequential tasks with very long time dependencies can require significant computational resources, for training and evaluating the RNN. Moreover, as the BPTT training algorithms entail storing all hidden states at every time step, the overall memory requirements can be prohibitive. Thus, *the design of a fast and memory efficient RNN architecture that can mitigate the EVGP is highly desirable for the effective use of RNNs in realistic learning tasks with very long time dependencies.* The main objective of this article is to propose, analyze and test such an architecture.

The basis of our proposed RNN is the observation that a large class of dynamical systems in physics and engineering, the so-called *Hamiltonian systems* (Arnold, 1989), allow for very precise control on the underlying states. Moreover, the fact that the phase space volume is preserved by the trajectories of a Hamiltonian system, makes such systems *invertible* and allows one to significantly reduce the storage requirements. Furthermore, if the resulting hidden state gradients also evolve according to a Hamiltonian dynamical system, one can obtain precise bounds on the hidden state

[1]Seminar for Applied Mathematics (SAM), D-MATH, ETH Zürich, Rämistrasse 101, Zürich-8092, Switzerland. Correspondence to: T. Konstantin Rusch <konstantin.rusch@sam.math.ethz.ch>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

gradients and alleviate the EVGP. We combine and extend these ideas into an RNN architecture that will allow us to prove rigorous bounds on the hidden states and their gradients, mitigating the EVGP. Moreover, our RNN architecture results in a fast implementation that attains state of the art performance on a variety of learning tasks with very long time dependencies.

2. The proposed RNN

Our proposed RNN is based on the time-discretization of the following system of *second-order ordinary differential equations* (ODEs),

$$\mathbf{y}'' = -[\sigma\left(\mathbf{w} \odot \mathbf{y} + \mathbf{V}\mathbf{u} + \mathbf{b}\right) + \alpha \mathbf{y}]. \tag{1}$$

Here, $t \in [0,1]$ is the (continuous) time variable, $\mathbf{u} = \mathbf{u}(t) \in \mathbb{R}^d$ is the time-dependent *input signal*, $\mathbf{y} = \mathbf{y}(t) \in \mathbb{R}^m$ is the *hidden state* of the RNN with $\mathbf{w} \in \mathbb{R}^m$ is a weight vector, $\mathbf{V} \in \mathbb{R}^{m \times d}$ a weight matrix, $\mathbf{b} \in \mathbb{R}^m$ is the bias vector and $\alpha \geq 0$ is a control parameter. The operation $\odot$ is the Hadamard product and the function $\sigma : \mathbb{R} \mapsto \mathbb{R}$ is the *activation function* and is applied component wise. For the rest of this paper, we set $\sigma(u) = \tanh(u)$.

By introducing the auxiliary variable $\mathbf{z} = \mathbf{y}'$, we can rewrite the second order ODE (1) as a first order ODE system:

$$\mathbf{y}' = \mathbf{z}, \quad \mathbf{z}' = -[\sigma\left(\mathbf{w} \odot \mathbf{y} + \mathbf{V}\mathbf{u} + \mathbf{b}\right) + \alpha \mathbf{y}]. \tag{2}$$

Assuming that $\mathbf{w}_i \neq 0$, for all $1 \leq i \leq m$, it is easy to see that the ODE system (2) is a *Hamiltonian system*,

$$\mathbf{y}' = \frac{\partial H}{\partial \mathbf{z}}, \quad \mathbf{z}' = -\frac{\partial H}{\partial \mathbf{y}}, \tag{3}$$

with the *time-dependent Hamiltonian*,

$$H(\mathbf{y}, \mathbf{z}, t) = \frac{\alpha}{2}\|\mathbf{y}\|^2 + \frac{1}{2}\|\mathbf{z}\|^2 + \sum_{i=1}^{m} \frac{1}{\mathbf{w}_i} \log(\cosh(\mathbf{w}_i \mathbf{y}_i + (\mathbf{V}\mathbf{u}(t))_i + \mathbf{b}_i)), \tag{4}$$

with $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$ denoting the Euclidean norm of the vector $\mathbf{x} \in \mathbb{R}^m$ and $\langle \cdot, \cdot \rangle$ the corresponding inner product.

The next step is to find a discretization of the ODE system (2). Given that it is highly desirable to ensure that the discretization respects the Hamiltonian structure of the underlying continuous ODE, the simplest such *structure preserving discretization* is the *symplectic Euler* method (Sanz Serna & Calvo, 1994; Hairer et al., 2003). Applying the symplectic Euler method to the ODE (2) results in the following discrete dynamical system,

$$\begin{aligned} \mathbf{y}_n &= \mathbf{y}_{n-1} + \Delta t \mathbf{z}_n, \\ \mathbf{z}_n &= \mathbf{z}_{n-1} - \Delta t[\sigma\left(\mathbf{w} \odot \mathbf{y}_{n-1} + \mathbf{V}\mathbf{u}_n + \mathbf{b}\right) + \alpha \mathbf{y}_{n-1}], \end{aligned} \tag{5}$$

for $1 \leq n \leq N$. Here, $0 < \Delta t < 1$ is the time-step and $\mathbf{u}_n \approx \mathbf{u}(t_n)$, with $t_n = n\Delta t$, is the input signal. It is common to initialize with $\mathbf{y}_0 = \mathbf{z}_0 = \mathbf{0}$.

We see from the structure of the discrete dynamical system (5) that there is *no interaction* between the neurons in the hidden layer of (5). Such an RNN will have very limited expressivity. Hence, we *stack* more hidden layers to propose the following deep or *multi-layer* RNN,

$$\begin{aligned} \mathbf{y}_n^\ell &= \mathbf{y}_{n-1}^\ell + \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot \mathbf{z}_n^\ell, \\ \mathbf{z}_n^\ell &= \mathbf{z}_{n-1}^\ell - \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot [\sigma(\mathbf{w}^\ell \odot \mathbf{y}_{n-1}^\ell + \mathbf{V}^\ell \mathbf{y}_n^{\ell-1} + \mathbf{b}^l) \\ &\quad + \alpha \mathbf{y}_{n-1}^\ell]. \end{aligned} \tag{6}$$

Here $\mathbf{y}_n^l, \mathbf{z}_n^l \in \mathbb{R}^m$ are hidden states and $\mathbf{w}^\ell, \mathbf{V}^\ell, \mathbf{b}^\ell$ are weights and biases, corresponding to layer $\ell = 1, \ldots, L$. We set $\mathbf{y}_n^0 = \mathbf{u}_n$ in the multilayer RNN (6).

In Fig. 1, we present a schematic diagram of the proposed multi-layer recurrent model UnICORNN.

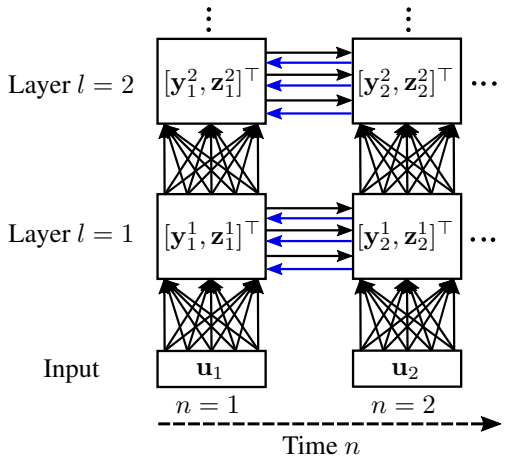

Figure 1. Schematic diagram of the multi-layer UnICORNN architecture, where the layers (respectively the input) are densely connected and the hidden states evolve independently in time. The invertibility of UnICORNN is visualized with blue arrows, emphasizing that the hidden states can be reconstructed during the backward pass and do not need to be stored.

Observe that we use the same step-size Δt for every layer, while multiplying a trainable parameter vector $\mathbf{c} \in \mathbb{R}^m$ to the time step. The action of $\mathbf{c}$ is modulated with the sigmoidal activation function $\hat{\sigma}(u) = 0.5 + 0.5 \tanh(u/2)$, which ensures that the time-step Δt is multiplied by a value between 0 and 1. We remark that the presence of this trainable vector $\mathbf{c}$ allows us to incorporate *multi-scale behavior* in the proposed RNN, as the effective time-step is learned during training and can be significantly different from the nominal time-step Δt. It is essential to point out that including this multi-scale time stepping is only possible, as each

neuron (within the same hidden layer) is independent of the others and can be integrated with a different effective time step. Finally, we also share the control hyperparameter α across the different layers, which results in a memory unit of L layers with a total of only 2 hyperparameters.

2.1. Motivation and background

The ODE system (2) is a model for a nonlinear system of uncoupled driven oscillators (Guckenheimer & Holmes, 1990). To see this, we denote $\mathbf{y}_i(t)$ as the displacement and $\mathbf{z}_i(t)$ as the velocity. Then, the dynamics of the i-th oscillator is determined by the frequency α and also by the *forcing* or *driving* term in the second equation of (2), where the forcing acts through the input signal $\mathbf{u}$ and is modulated by the weight $\mathbf{V}$ and bias $\mathbf{b}$. Finally, the weight $\mathbf{w}$ modulates the frequency α and allows each neuron to oscillate with its own frequency, rather than the common frequency α of the system. The structure of $\mathbf{w}$ implies that each neuron is independent of the others. A key element of the oscillator system (2) is the absence of any damping or friction term. This allows the system to possess a Hamiltonian structure, with desirable long time behavior. Thus, we term the resulting RNN (6), based on the ODE system (2) as **Un**damped **I**ndependent **C**ontrolled **O**scillatory **RNN** or **UnICORNN**. We remark that networks of oscillators are very common in science and engineering (Guckenheimer & Holmes, 1990; Strogatz, 2015) with prominent examples being pendulums in mechanics, electrical circuits in engineering, business cycles in economics and functional brain circuits such as cortical columns in neurobiology.

2.2. Comparison with related work.

UnICORNN lies firmly in the class of ODE-based or ODE-inspired RNNs, which have received considerable amount of attention in the machine learning literature in recent years. Neural ODEs, first proposed in (Chen et al., 2018), are a prominent example of using ODEs to construct neural networks. In this architecture, the continuous ODE serves as the learning model and gradients are computed from a sensitivity equation, which allows one to trade accuracy with computing time. Moreover, it is argued that these neural ODEs are invertible and hence, memory efficient. However, it is unclear if a general neural ODE, without any additional structure, can be invertible. Other RNN architectures that are based on discretized ODEs include those proposed in (E, 2017) and (Chang et al., 2018), where the authors proposed an *anti-symmetric* RNN, based on the discretization of a stable ODE resulting from a skew-symmetric hidden weight matrix, thus constraining the gradient dynamics.

Our proposed RNN (6) is inspired by two recent RNN architectures. The first one is *coRNN*, proposed recently in (Rusch & Mishra, 2021), where the underlying RNN architecture was also based on the use of a network of oscillators. As long as a constraint on the underlying weights was satisfied, coRNN was shown to mitigate the EVGP. In contrast to coRNN, our proposed RNN does not use a *damping* term. Moreover, each neuron, for any hidden layer, in UnICORNN (6) is independent. This is very different from coRNN where all the neurons were coupled together. Finally, UnICORNN is a multi-layer architecture whereas coRNN used a single hidden layer. These innovations allow us to admit a Hamiltonian structure for UnICORNN and facilitate a fast and memory efficient implementation.

Our proposed architecture was also partly inspired by *IndRNN*, proposed in (Li et al., 2018; 2019), where the neurons in each hidden layers were independent of each other and interactions between neurons were mediated by stacking multiple RNN layers, with output of each hidden layer passed on to the next hidden layer, leading to a deep RNN. We clearly use this construction of independent neurons in each layer and stacking multiple layers in UnICORNN (6). However in contrast to IndRNN, our proposed RNN is based on a discretized Hamiltonian system and we will not require any constraints on the weights to mitigate the EVGP.

Finally, we would like to point out that discrete Hamiltonian systems have already been used to design RNNs, for instance in (Greydanus et al., 2019) and also in (Chen et al.), where a symplectic time-integrator for a Hamiltonian system was proposed as the RNN architecture. However, these approaches are based on underlying time-independent Hamiltonians and are only relevant for mechanical systems as they cannot process time-dependent inputs, which arise in most realistic learning tasks. Moreover, as these methods enforce exact conservation of the Hamiltonian in time, they are not suitable for learning long-time dependencies, see (MacKay et al., 2018) for a discussion and experiment on that issue. Although we use a Hamiltonian system as the basis of our proposed RNN (6), our underlying Hamiltonian (4) is time-dependent and the resulting RNN can readily process any time-dependent input signal.

2.3. On the Memory Efficiency of UnICORNN

As mentioned in the introduction, the standard BPTT training algorithm for RNNs requires one to store all the hidden states at every time step. To see this, we observe that for a standard multi-layer RNN with L layers and a mini-batch size of b (for any mini-batch stochastic gradient descent algorithm), the storage (in terms of floats) scales as $\mathcal{O}(Nbd + LbmN)$, with input and hidden sequences of length N. This memory requirement can be very high. Note that we have ignored the storage of trainable weights and biases for the RNN in the above calculation.

On the other hand, as argued before, our proposed RNN is

a symplectic Euler discretization for a Hamiltonian system. Hence, it is invertible. In fact, one can explicitly write the *inverse* of UnICORNN (6) as,

$$\begin{aligned}\mathbf{y}_{n-1}^l &= \mathbf{y}_n^l - \Delta t\hat{\sigma}(\mathbf{c}^l)\odot\mathbf{z}_n^l,\\ \mathbf{z}_{n-1}^l &= \mathbf{z}_n^l + \Delta t\hat{\sigma}(\mathbf{c}^l)\odot[\sigma(\mathbf{w}^l\odot\mathbf{y}_{n-1}^l + \mathbf{V}^\ell\mathbf{y}_n^{\ell-1}+\mathbf{b}^l)\\ &\quad + \alpha\mathbf{y}_{n-1}^l].\end{aligned} \tag{7}$$

Thus, one can recover all the hidden states in a given hidden layer, only from the *stored* hidden state at the final time step, for that layer. Moreover, only the input signal needs to be stored as the other hidden states can be reconstructed from the formula (7). Hence, a straightforward calculation shows that the storage for UnICORNN scales as $\mathcal{O}(Nbd + Lbm)$. As $L << N$, we conclude that UnICORNN allows for a significant saving in terms of storage, when compared to a standard RNN.

3. Rigorous Analysis of UnICORNN

On the dynamics of the hidden state gradients for ODE (2). In order to investigate the EVGP for the proposed RNN (6), we will first explore the dynamics of the gradients of hidden states $\mathbf{y}, \mathbf{z}$ (solutions of the ODE (2)) with respect to the trainable parameters $\mathbf{w}, \mathbf{V}$ and $\mathbf{b}$. Denote any scalar parameter as θ and $f_\theta = \frac{\partial f}{\partial \theta}$, then differentiating the ODE (2) with respect to θ results in the ODE,

$$\begin{aligned}\mathbf{y}_\theta' &= \mathbf{z}_\theta,\\ \mathbf{z}_\theta' &= -\sigma'(\mathbf{A})\odot(\mathbf{w}\odot\mathbf{y}_\theta) - \alpha\mathbf{y}_\theta - \sigma'(\mathbf{A})\odot\mathbf{C}(t),\end{aligned} \tag{8}$$

where $\mathbf{A} = \mathbf{w}\odot\mathbf{y} + \mathbf{V}\mathbf{u} + \mathbf{b}$ is the pre-activation and the coefficient $\mathbf{C}\in\mathbb{R}^m$ is given by $\mathbf{C}_i = \mathbf{y}_i$ if $\theta = \mathbf{w}_i$, $\mathbf{C}_i = \mathbf{u}_j$ if $\theta = \mathbf{V}_{ij}$ and $\mathbf{C}_i = 1$ if $\theta = \mathbf{b}_i$, with all other entries of the vector $\mathbf{C}$ being zero.

It is easy to check that the ODE system (8) is a *Hamiltonian system* of form (3), with the following time-dependent Hamiltonian;

$$\begin{aligned}\mathbf{H}(\mathbf{y}_\theta,\mathbf{z}_\theta,t) &:= \frac{\alpha}{2}\|\mathbf{y}_\theta\|^2 + \frac{1}{2}\|\mathbf{z}_\theta\|^2\\ &+ \frac{1}{2}\sum_{i=1}^m \sigma'(\mathbf{A}_i)\mathbf{w}_i((\mathbf{y}_\theta)_i)^2 + \sum_{i=1}^m \sigma'(\mathbf{A}_i)\mathbf{C}_i(t)(\mathbf{y}_\theta)_i.\end{aligned} \tag{9}$$

Thus, by the well-known Liouville's theorem (Sanz Serna & Calvo, 1994), we know that the phase space volume of (8) is preserved. Hence, this system cannot have any asymptotically stable fixed points. This implies that $\{\mathbf{0},\mathbf{0}\}$ cannot be a stable fixed point for the hidden state gradients $(\mathbf{y}_\theta,\mathbf{z}_\theta)$. Thus, we can expect that the hidden state gradients with respect to the system of oscillators (2) do not remain near zero and suggest a possible mechanism for the mitigation of the vanishing gradient problem for UnICORNN (6), which is a structure preserving discretization of the ODE (2).

On the Exploding Gradient Problem for UnICORNN. We train the RNN (6) to minimize the loss function,

$$\mathcal{E} := \frac{1}{N}\sum_{n=1}^N \mathcal{E}_n, \quad \mathcal{E}_n = \frac{1}{2}\|\mathbf{y}_n^L - \bar{\mathbf{y}}_n\|_2^2, \tag{10}$$

with $\bar{\mathbf{y}}$ being the underlying ground truth (training data). Note that the loss function (10) only involves the output at the last hidden layer (we set the affine output layer to identity for the sake of simplicity). During training, we compute gradients of the loss function (10) with respect to the trainable weights and biases $\mathbf{\Theta} = [\mathbf{w}^\ell, \mathbf{V}^\ell, \mathbf{b}^\ell, \mathbf{c}^\ell]$, for all $1\le\ell\le L$, i.e.,

$$\frac{\partial\mathcal{E}}{\partial\theta} = \frac{1}{N}\sum_{n=1}^N \frac{\partial\mathcal{E}_n}{\partial\theta}, \quad \forall\,\theta\in\mathbf{\Theta}. \tag{11}$$

We have the following upper bound on the hidden state gradient,

Proposition 3.1. *Let the time step $\Delta t << 1$ be sufficiently small in the RNN (6) and let $\mathbf{y}_n^\ell, \mathbf{z}_n^\ell$, for $1\le\ell\le L$, and $1\le n\le N$ be the hidden states generated by the RNN (6). Then, the gradient of the loss function $\mathcal{E}$ (10) with respect to any parameter $\theta\in\mathbf{\Theta}$ is bounded as,*

$$\left|\frac{\partial\mathcal{E}}{\partial\theta}\right| \le \frac{1-(\Delta t)^L}{1-\Delta t} T(1+2\gamma T)\overline{\mathbf{V}}(\bar{Y}+\mathbf{F})\mathbf{\Delta}, \tag{12}$$

with $\bar{Y} = \max_{1\le n\le N}\|\bar{\mathbf{y}}_n\|_\infty$, be a bound on the underlying training data and other quantities in (12) defined as,

$$\gamma = \max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right) + \frac{\left(\max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right)\right)^2}{2},$$

$$\overline{\mathbf{V}} = \prod_{q=1}^L \max\{1, \|\mathbf{V}^q\|_\infty\}, \quad \beta = \max\{1+2\alpha, 4\alpha^2\}$$

$$\mathbf{F} = \sqrt{\frac{2}{\alpha}(1+2\beta T)}, \quad T = N\Delta t,$$

$$\mathbf{\Delta} = 2 + \sqrt{2(1+2\beta T)} + (2+\alpha)\sqrt{\frac{2}{\alpha}(1+2\beta T)}.$$

This proposition, proved in Appendix **C.2**, demonstrates that as long as the weights $\mathbf{w}^L, \mathbf{V}^q$ are bounded, there is a uniform bound on the hidden state gradients. This bound grows at most as $(N\Delta t)^3$, with N being the total number of time steps. Thus, there is no exponential growth of the gradient with respect to the number of time steps and the *exploding gradient problem* is mitigated for UnICORNN.

On the Vanishing Gradient Problem for UnICORNN. By applying the chain rule repeatedly to each term on the

right-hand-side of (11), we obtain

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{\ell=1}^{L} \sum_{k=1}^{n} \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}, \quad \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta},$$
$$\mathbf{X}_n^\ell = \left[\mathbf{y}_n^{\ell,1}, \mathbf{z}_n^{\ell,1}, \ldots, \mathbf{y}_n^{\ell,j}, \mathbf{z}_n^{\ell,j}, \ldots, \mathbf{y}_n^{\ell,m}, \mathbf{z}_n^{\ell,m}\right]. \tag{13}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k^\ell$ with respect to the parameter θ, while keeping the other arguments constant. The quantity $\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}$ denotes the contribution from the k-recurrent step at the l-th hidden layer of the deep RNN (6) to the overall hidden state gradient at the step n. The vanishing gradient problem (Pascanu et al., 2013) arises if $\left|\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}\right|$, defined in (13), $\to 0$ exponentially fast in k, for $k << n$ (long-term dependencies). In that case, the RNN does not have long-term memory, as the contribution of the k-th hidden state at the ℓ-th layer to error at time step t_n is infinitesimally small.

We have established that the hidden state gradients for the underlying continuous ODE (2) do not vanish. As we use a symplectic Euler discretization, the phase space volume for the discrete dynamical system (5) is also conserved (Sanz Serna & Calvo, 1994; Hairer et al., 2003). Hence, one can expect that the gradients of the multilayer RNN (6) do not vanish. However, these heuristic considerations need to be formalized. Observe that the vanishing gradient problem for RNNs focuses on the possible smallness of contributions of the gradient over a large number of recurrent steps. As this behavior of the gradient is independent of the number of layers, we focus on the vanishing gradient problem for a single hidden layer here, while presenting the multilayer results in Appendix **C.4**. Also, for the sake of definiteness, we set the scalar parameter $\theta = \mathbf{w}^{1,p}$ for some $1 \leq p \leq m$. Similar results also hold for any other $\theta \in \mathbf{\Theta}$.

We have the following representation formula (proved in Appendix **C.3**) for the hidden state gradients,

Proposition 3.2. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN (6). Then the gradient for long-term dependencies, i.e. $k << n$, satisfies the representation formula,*

$$\frac{\partial \mathcal{E}_{k,1}^{(n,1)}}{\partial \mathbf{w}^{1,p}} = -\Delta t \hat{\sigma}(\mathbf{c}^{1,p})^2 t_n \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p} \left(\mathbf{y}_n^{1,p} - \overline{\mathbf{y}}_n^p\right) + \mathcal{O}(\Delta t^2). \tag{14}$$

It is clear from the representation formula (14) that there is no k-dependence for the gradient. In particular, as long as all the weights are of $\mathcal{O}(1)$, the leading-order term in (14) is $\mathcal{O}(\Delta t)$. Hence, the gradient can be small but is independent of the recurrent step k. Thus, we claim that the *vanishing gradient problem*, with respect to recurrent connections, is mitigated for UnICORNN (6).

4. Experiments

The details of the training procedure for each experiment can be found in Appendix **A**. Code to replicate the experiments can be found at **https://github.com/tk-rusch/unicornn**.

Implementation The structure of UnICORNN (6) enables us to achieve a very fast implementation. First, the transformation of the input (i.e. $\mathbf{V}^\ell \mathbf{y}_n^{\ell-1}$ for all $l = 1, \ldots, L$), which is the most computationally expensive part of UnICORNN, does not have a sequential structure and can thus be computed in parallel over time. Second, as the underlying ODEs of the UnICORNN are uncoupled for each neuron, the remaining recurrent part of UnICORNN is solved independently for each component. Hence, inspired by the implementation of Simple Recurrent Units (SRU) (Lei et al., 2018) and IndRNN, we present in Appendix **B**, the details of an efficient CUDA implementation, where the input transformation is computed in parallel and the dynamical system corresponding to each component of (6) is an independent CUDA thread.

We benchmark the training speed of UnICORNN with $L = 2$ layers, against the fastest available RNN implementations, namely the cuDNN implementation (Appleyard et al., 2016) of LSTM (with 1 hidden layer), SRU and IndRNN (both with $L = 2$ layers and with batch normalization). Fig. 2 shows the computational time (measured on a GeForce RTX 2080 Ti GPU) of the combined forward and backward pass for each network, averaged over 100 batches with each of size 128, for two different sequence lengths, i.e. $N = 1000, 2000$. We can see that while the cuDNN LSTM is relatively slow, the SRU, IndRNN and the UnICORNN perform similarly fast. Moreover, we also observe that UnICORNN is about $30 - 40$ times faster per combined forward and backward pass, when compared to recently developed RNNs such as expRNN (Casado & Martínez-Rubio, 2019) and coRNN (Rusch & Mishra, 2021). We thus conclude that the UnICORNN is among the fastest available RNN architectures.

Permuted sequential MNIST A well-established benchmark for testing RNNs on input sequences with long-time dependencies is the permuted sequential MNIST (psMNIST) task (Le et al., 2015). Based on the classical MNIST data set (LeCun et al., 1998), the flattened grey-scale matrices are randomly permuted (based on a fixed random permutation) and processed sequentially by the RNN. This makes the learning task more challenging than sequential MNIST, where one only flattens the MNIST matrices without permuting them. In order to make different methods comparable, we use the same fixed seed for the random permutation, as in (Casado & Martínez-Rubio, 2019; Casado, 2019; Helfrich et al., 2018). Table 1 shows the results for UnICORNN with 3 layers, together with other recently pro-

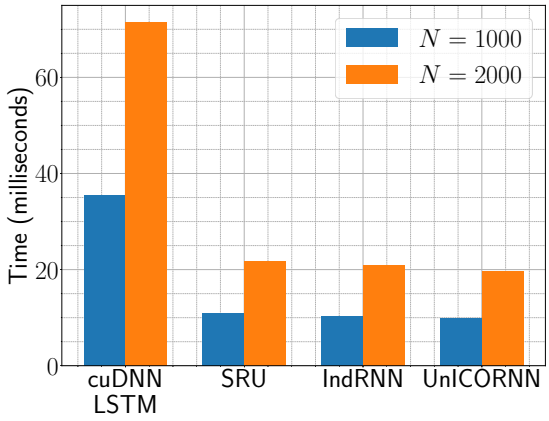

Figure 2. Measured computing time for the combined forward and backward pass for the UnICORNN as well as for three of the fastest available RNN implementations.

posed RNNs, which were explicitly designed to learn LTDs as well as two gated baselines. We see that UnICORNN clearly outperforms the other methods.

Table 1. Test accuracies on permuted sequential MNIST together with number of hidden units as well as total number of parameters M for each network. All other results are taken from the corresponding original publication, cited in the main text, except that we are using the results of (Chang et al., 2017) for GRU and of (Helfrich et al., 2018) for LSTM.

MODEL	TEST ACC.	#UNITS	M
LSTM	92.9%	256	270K
GRU	94.1%	256	200K
EXPRNN	96.6%	512	127K
CORNN	97.3%	256	134K
INDRNN (L=6)	96.0%	128	86K
DENSE-INDRNN (L=6)	97.2%	128	257K
UNICORNN (L=3)	97.8%	128	35K
UNICORNN (L=3)	**98.4%**	256	135K

Noise padded CIFAR-10 A more challenging test for the ability of RNNs to learn LTDs is provided by the recently proposed noise padded CIFAR-10 experiment (Chang et al., 2018). In it, the CIFAR-10 data points (Krizhevsky et al., 2009) are fed to the RNN row-wise and flattened along the channels resulting in sequences of length 32. To test long term memory, entries of uniform random numbers are added such that the resulting sequences have a length of 1000, i.e. the last 968 entries of each sequences are only noise to distract the RNNs. Table 2 shows the result of the UnICORNN with 3 layers together with the results of other recently proposed RNNs, namely for the LSTM, anti.sym. RNN and gated anti.sym. RNN (Chang et al., 2018), Lipschitz RNN (Erichson et al., 2021), Incremental RNN (Kag et al.), FastRNN (Kusupati et al., 2018) and coRNN (Rusch & Mishra, 2021). We conclude that the proposed RNN readily outperforms all other methods on this experiment.

Table 2. Test accuracies on noise padded CIFAR-10 together with number of hidden units as well as total number of parameters M for each network. All other results are taken from literature, specified in the main text.

MODEL	TEST ACC.	#UNITS	M
LSTM	11.6%	128	64K
INCREMENTAL RNN	54.5%	128	12K
LIPSCHITZ RNN	55.8%	256	158K
FASTRNN	45.8%	128	16K
ANTI.SYM. RNN	48.3%	256	36K
GATED ANTI.SYM. RNN	54.7%	256	37K
CORNN	59.0%	128	46K
UNICORNN (L=3)	**62.4%**	128	47K

EigenWorms The EigenWorms data set (Bagnall et al., 2018) is a collecting of 259 very long sequences, i.e. length of 17984, describing the motion of a worm. The task is, based on the 6-dimensional motion sequences, to classify a worm as either wild-type or one of four mutant types. We use the same train/valid/test split as in (Morrill et al., 2020), i.e. 70%/15%/15%. As the length of the input sequences is extremely long for this test case, we benchmark UnICORNN against three sub-sampling based baselines. These include the results of (Morrill et al., 2020), which is based on signature sub-sampling routine for neural controlled differential equations. Additionally after a hyperparameter fine-tuning procedure, we perform a random sub-sampling as well as truncated back-propagation through time (BPTT) routine using LSTMs, where the random sub-sampling is based on 200 randomly selected time points of the sequences as well as the BPTT is truncated after the last 500 time points of the sequences. Furthermore, we compare UnICORNN with three leading RNN architectures for solving LTD tasks, namely expRNN, IndRNN and coRNN, which are all applied to the full-length sequences. The results, presented in Table 3, show that while sub-sampling approaches yield moderate test accuracies, expRNN as well as the IndRNN yield very poor accuracies. In contrast, coRNN performs very well. However, the best results are obtained for UnICORNN as it reaches a test accuracy of more than 90%, while at the same time yielding a relatively low standard deviation, further underlining the robustness of the proposed RNN.

As this data set has only recently been proposed as a test for RNNs in learning LTDs, it is unclear if the input sequences

Table 3. Test accuracies on EigenWorms using 5 re-trainings of each best performing network (based on the validation set) together with number of hidden units as well as total number of parameters M for each network.

MODEL	TEST ACC.	#UNITS	M
T-BPTT LSTM	57.9% ± 7.0%	32	5.3K
SUB-SAMP. LSTM	69.2% ± 8.3%	32	5.3K
SIGN.-NCDE	77.8% ± 5.9%	32	35K
EXPRNN	40.0% ± 10.1%	64	2.8K
INDRNN (L=2)	49.7% ± 4.8%	32	1.6K
CORNN	86.7% ± 3.0%	32	2.4K
UNICORNN (L=2)	**90.3%** ± **3.0%**	32	1.5K

truly exhibit very long-time dependencies. To investigate this further, we train UnICORNN on a subset of the entries of the sequences. To this end, we consider using only the last entries as well as using a random subset of the entries. Fig. 3 shows the distributional results (10 re-trainings of the best performing UnICORNN) for the number of entries used in each sub-sampling routine, ranging from only using 1000 entries to using the full sequences for training. We can see that in order to reach a test accuracy of 80% when training on the last entries of the sequences, at least the last 10k entries are needed. Moreover, for both sub-sampling methods the test accuracy increases monotonically as the number of entries for training is increased. On the other hand, using a random subset of the entries increases the test accuracy significantly when compared to using only the last entries of the sequences. This indicates that the important entries of the sequences, i.e. information needed in order to classify them correctly, are uniformly distributed throughout the full sequence. We thus conclude that the EigenWorms data set indeed exhibits *very* long-time dependencies.

Healthcare application: Vital signs prediction We apply UnICORNN on two real-world data sets in health care, aiming to predict the vital signs of a patient, based on PPG and ECG signals. The data sets are part of the TSR archive (Tan et al., 2020) and are based on clinical data from the Beth Israel Deaconess Medical Center. The PPG and ECG signals are sampled with a frequency of 125Hz for 8 minutes each. The resulting two-dimensional sequences have a length of 4000. The goal is to predict a patient's respiratory rate (RR) and heart rate (HR) based on these signals. We compare UnICORNN to 3 leading RNN architectures for solving LTDs, i.e. expRNN, IndRNN and coRNN. Additionally, we present two baselines using the LSTM as well as the recently proposed sub-sampling method of computing signatures for neural controlled differential equations (NCDE) (Morrill et al., 2020). Following (Morrill et al., 2020), we split the 7949 sequences in a training set, validation set and testing set, using a 70%/15%/15% split. Table 4 shows the distributional results of all networks using 5 re-trainings of the best performing RNN. We observe that while the LSTM does not reach a low L^2 testing error in both experiments, the other RNNs approximate the vital signs reasonably well. However, UnICORNN clearly outperforms all other methods on both benchmarks. We emphasize that UnICORNN significantly outperforms all other state-of-the-art methods on estimating the RR, which is of major importance in modern healthcare applications for monitoring hospital inpatients as well as for mobile health applications, as special invasive equipment (for instance capnometry or measurement of gas flow) is normally needed to do so (Pimentel et al., 2016).

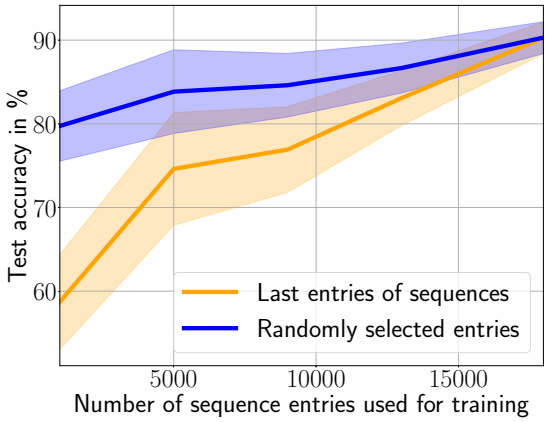

Figure 3. Test accuracy (mean and standard deviation) for the UnICORNN on EigenWorms for two types of sub-sampling approaches, i.e. using the last entries of the sequences as well as using a random subset of the entries. Both are shown for increasing number of entries used in each corresponding sub-sampling routine.

Table 4. L^2 test error on vital sign prediction using 5 re-trainings of each best performing network (based on the validation set), where the respiratory rate (RR) and heart rate (HR) is estimated based on PPG and ECG data.

MODEL	RR	HR
SIGN.-NCDE	1.51 ± 0.08	2.97 ± 0.45
LSTM	2.28 ± 0.25	10.7 ± 2.0
EXPRNN	1.57 ± 0.16	1.87 ± 0.19
INDRNN (L=3)	1.47 ± 0.09	2.10 ± 0.2
CORNN	1.45 ± 0.23	1.71 ± 0.1
UNICORNN (L=3)	**1.06** ± **0.03**	**1.39** ± **0.09**

Table 5. Test accuracies on IMDB together with number of hidden units as well as total number of parameters M (without embedding) for each network. All other results are taken from literature, specified in the main text.

MODEL	TEST ACC.	#UNITS	M
LSTM	86.8%	128	220K
SKIP LSTM	86.6%	128	220K
GRU	85.2%	128	99K
RELU GRU	84.8%	128	99K
SKIP GRU	86.6%	128	165K
CORNN	87.4 %	128	46K
UNICORNN (L=2)	**88.4**%	128	30K

Sentiment analysis: IMDB As a final experiment, we test the proposed UnICORNN on the widely used NLP benchmark data set IMDB (Maas et al., 2011), which consists of 50k online movie reviews with 25k reviews used for training and 25k reviews used for testing. This denotes a classical sentiment analysis task, where the model has to decide whether a movie review is positive or negative. We use 30% of the training set (i.e. 7.5k reviews) as the validation set and restrict the dictionary to 25k words. We choose an embedding size of 100 and initialize it with the pretrained 100d GloVe (Pennington et al., 2014) vectors. Table 5 shows the results for UnICORNN with 2 layers together with other recently proposed RNN architectures and gated baselines (which are known to perform very well on these tasks). The result of ReLU GRU is taken from (Dey & Salemt, 2017), of coRNN from (Rusch & Mishra, 2021) and all other results are taken from (Campos et al., 2018). We can see that UnICORNN outperforms the other methods while requiring significantly less parameters. We thus conclude, that the UnICORNN can also be successfully applied to problems, which do not necessarily exhibit long-term dependencies.

Further experimental results As stated before, UnICORNN has two hyperparameters, i.e. the maximum allowed time-step Δt and the damping parameter α. It is of interest to examine how sensitive the performance of UnICORNN is with respect to variations of these hyperparameters. To this end, we consider the noise padded CIFAR-10 experiment and perform an ablation study of the test accuracy with respect to variations of both α and Δt. Both hyperparameters are varied by an order of magnitude and the results of this study are plotted in Fig. 4. We observe from this figure, that the results are indeed somewhat sensitive to the maximum allowed time-step Δt and show a variation of approximately 15% with respect to to this hyperparameter. On the other hand, there is very little noticeable variation with respect to the damping parameter α. Thus, it can be set to a default value, for instance $\alpha = 1$, without impeding the performance of the underlying RNN.

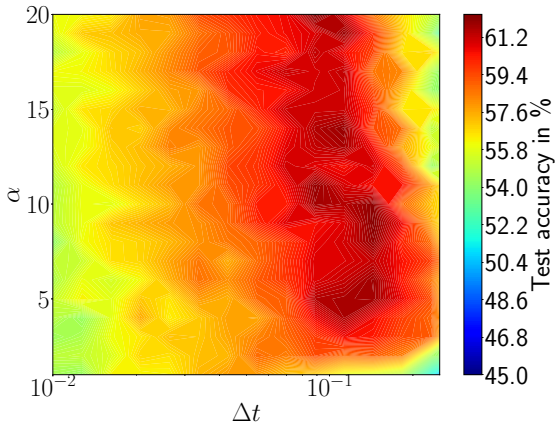

Figure 4. Ablation study on the hyperparameters Δt and α of UnICORNN (6) using the noise padded CIFAR-10 experiment.

Next, we recall that the design of UnICORNN (6) enables it to learn the effective time step (with a possible maximum of Δt) from data. It is instructive to investigate if this ability to express *multi-scale* behavior is realized in practice or not. To this end, we consider the trained UnICORNN of the psMNIST task with 3 layers and 256 neurons. Here, a maximum time step of $\Delta t = 0.19$ was identified by the hyperparameter tuning. In Fig. 5, we plot the effective time step $\Delta t \hat{\sigma}(\mathbf{c}_i^l)$, for each hidden neuron $i = 1, \ldots, 256$ and each layer $l = 1, 2, 3$. We observe from this figure that a significant variation of the effective time step is observed, both within the neurons in each layer, as well as between layers. In particular, the minimum effective time step is about 28 times smaller than the maximum allowed time step. Thus, we conclude from this figure, that UnICORNN exploits its design features to learn multi-scale behavior that is latent in the data. This perhaps explains the superior performance of UnICORNN on many learning tasks.

5. Discussion

The design of RNNs that can accurately handle sequential inputs with long-time dependencies is very challenging. This is largely on account of the exploding and vanishing gradient problem (EVGP). Moreover, there is a significant increase in both computational time as well as memory requirements when LTD tasks have to be processed. Our main aim in this article was to present a novel RNN architecture which is fast, memory efficient, *invertible* and mitigates the EVGP. To this end, we proposed UnICORNN (6), an RNN based on the symplectic Euler discretization of a Hamiltonian system of second-order ODEs (2) modeling a network of independent, undamped, controlled and driven oscillators.

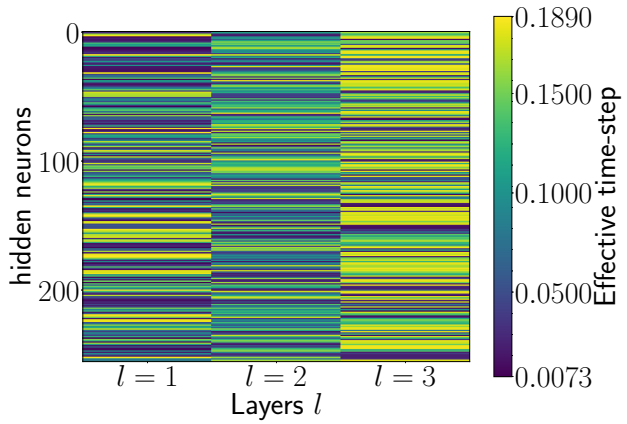

Figure 5. Effective time-step $\Delta t\hat{\sigma}(\mathbf{c}_i^l)$ for each hidden neuron $i = 1, \ldots, m$ and each layer $l = 1, \ldots, L$ of UnICORNN, after training on the psMNIST task using $m = 256$ hidden units and $L = 3$ layers.

In order to gain expressivity, we stack layers of RNNs and also endow this construction with a multi-scale feature by training the effective time step in (6).

Given the Hamiltonian structure of our continuous and discrete dynamical system, invertibility and volume preservation in phase space are guaranteed. Invertibility enables the proposed RNN to be memory efficient. The independence of neurons within each hidden layer allows us to build a highly efficient CUDA implementation of UnICORNN that is as fast as the fastest available RNN architectures. Under suitable smallness constraints on the maximum allowed time step Δt, we prove rigorous upper bounds (12) on the gradients and show that the exploding gradient problem is mitigated for UnICORNN. Moreover, we derive an explicit representation formula (14) for the gradients of (6), which shows that the vanishing gradient problem is also mitigated. Finally, we have tested UnICORNN on a suite of benchmarks that includes both synthetic as well as realistic learning tasks, designed to test the ability of an RNN to deal with long-time dependencies. In all the experiments, UnICORNN was able to show state of the art performance.

It is instructive to compare UnICORNN with two recently proposed RNN architectures, with which it shares some essential features. First, the use of coupled oscillators to design RNNs was already explored in the case of coRNN (Rusch & Mishra, 2021). In contrast to coRNN, neurons in UnICORNN are independent (uncoupled) and as there is no damping, UnICORNN possesses a Hamiltonian structure. This paves the way for invertibility as well as for mitigating the EVGP without any assumptions on the weights whereas the mitigation of EVGP with coRNN was conditional on restrictions on weights. Finally, UnICORNN provides even better performance on benchmarks than coRNN, while being significantly faster. While we also use independent neurons in each hidden layer and stack RNN layers together as in IndRNN (Li et al., 2018), our design principle is completely different as it is based on Hamiltonian ODEs. Consequently, we do not impose weight restrictions, which are necessary for IndRNN to mitigate the EVGP. Moreover, in contrast to IndRNNs, our architecture is invertible and hence, memory efficient.

This work can be extended in different directions. First, UnICORNN is a very flexible architecture in terms of stacking layers of RNNs together. We have used a fully connected stacking in (6) but other possibilities can be readily explored. See Appendix **C.5** for a discussion on the use of residual connections in stacking layers of UnICORNN. Second, the invertibility of UnICORNN can be leveraged in the context of normalizing flows (Papamakarios et al., 2019), where the objective is to parametrize a flow such that the resulting Jacobian is readily computable. Finally, our focus in this article was on testing UnICORNN on learning tasks with long-time dependencies. Given that the underlying ODE (2) models oscillators, one can envisage that UnICORNN will be very competitive with respect to processing different time series data that arise in healthcare AI such as EEG and EMG data, as well as seismic time series from the geosciences.

Acknowledgements

The research of TKR and SM was performed under a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 770880).

References

Appleyard, J., Kocisky, T., and Blunsom, P. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946*, 2016.

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Arnold, V. I. *Mathematical methods of classical mechanics*. Springer Verlag, New York, 1989.

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

Campos, V., Jou, B., Giró-i-Nieto, X., Torres, J., and Chang, S. Skip RNN: learning to skip state updates in recur-

rent neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

Casado, M. L. Trivializations for gradient-based optimization on manifolds. In *Advances in Neural Information Processing Systems*, pp. 9154–9164, 2019.

Casado, M. L. and Martínez-Rubio, D. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3794–3803, 2019.

Chang, B., Chen, M., Haber, E., and Chi, E. H. Antisymmetricrnn: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2018.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 77–87, 2017.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6571–6583, 2018.

Chen, Z., Zhang, J., Arjovsky, M., and Bottou, L. Symplectic recurrent neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Dey, R. and Salemt, F. M. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1597–1600. IEEE, 2017.

E, W. A proposal on machine learning via dynamical systems. *Commun. Math. Stat*, 5:1–11, 2017.

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

Gal, Y. and Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, 29:1019–1027, 2016.

Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pp. 15379–15389, 2019.

Guckenheimer, J. and Holmes, P. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. Springer Verlag, New York, 1990.

Hairer, E., Lubich, C., and Wanner, G. Geometric numerical integration illustrated by the störmer-verlet method. *Acta Numerica*, 14:399–450, 2003.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Helfrich, K., Willmott, D., and Ye, Q. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pp. 1969–1978. PMLR, 2018.

Henaff, M., Szlam, A., and LeCun, Y. Recurrent orthogonal networks and long-memory tasks. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2034–2042, 2016.

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

Kag, A., Zhang, Z., and Saligrama, V. Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Kerg, G., Goyette, K., Touzel, M. P., Gidel, G., Vorontsov, E., Bengio, Y., and Lajoie, G. Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics. In *Advances in Neural Information Processing Systems*, pp. 13591–13601, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., and Varma, M. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pp. 9017–9028, 2018.

Laurent, T. and von Brecht, J. A recurrent neural network without chaos. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April*

24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521:436–444, 2015.

Lei, T., Zhang, Y., Wang, S. I., Dai, H., and Artzi, Y. Simple recurrent units for highly parallelizable recurrence. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457–5466, 2018.

Li, S., Li, W., Cook, C., Gao, Y., and Zhu, C. Deep independently recurrent neural network (indrnn). *arXiv preprint arXiv:1910.06251*, 2019.

Lorenz, E. N. Predictability: A problem partly solved. In *Proc. Seminar on Predictability*, volume 1, 1996.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pp. 142–150. Association for Computational Linguistics, 2011.

MacKay, M., Vicol, P., Ba, J., and Grosse, R. B. Reversible recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 9029–9040, 2018.

Morrill, J., Kidger, P., Salvi, C., Foster, J., and Lyons, T. Neural cdes for long time series via the log-ode method. *arXiv preprint arXiv:2009.08295*, 2020.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762v1*, 2019.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *ICML'13*, pp. III–1310–III–1318. JMLR.org, 2013.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

Pimentel, M. A., Johnson, A. E., Charlton, P. H., Birrenkott, D., Watkinson, P. J., Tarassenko, L., and Clifton, D. A. Toward a robust estimation of respiratory rate from pulse oximeters. *IEEE Transactions on Biomedical Engineering*, 64(8):1914–1923, 2016.

Rusch, T. K. and Mishra, S. Coupled oscillatory recurrent neural network (cornn): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2021.

Sanz Serna, J. and Calvo, M. *Numerical Hamiltonian problems*. Chapman and Hall, London, 1994.

Strogatz, S. *Nonlinear Dynamics and Chaos*. Westview, Boulder CO, 2015.

Tan, C. W., Bergmeir, C., Petitjean, F., and Webb, G. I. Monash university, uea, ucr time series regression archive. *arXiv preprint arXiv:2006.10996*, 2020.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 4880–4888, 2016.

Supplementary Material for "UnICORNN: A recurrent model for learning *very* long time dependencies"

A. Training details

All experiments were run on GPU, namely NVIDIA GeForce GTX 1080 Ti and NVIDIA GeForce RTX 2080 Ti. The hidden weights $\mathbf{w}$ of the UnICORNN are initialized according to $\mathcal{U}(0,1)$, while all biases are set to zero. The trained vector $\mathbf{c}$ is initialized according to $\mathcal{U}(-0.1,0.1)$. The input weights $\mathbf{V}$ are initialized according to the Kaiming uniform initialization (He et al., 2015) based on the input dimension mode and the negative slope of the rectifier set to $a = 8$.

The hyperparameters of the UnICORNN are selected using a random search algorithm based on a validation set. The hyperparameters of the best performing UnICORNN can be seen in Table 6. The value for Δt and α is shared across all layers, except for the IMDB task and EigenWorms task, where we use a different Δt value for the first layer and the corresponding Δt value in Table 6 for all subsequent layers, i.e. we use $\Delta t = 6.6 \times 10^{-3}$ for IMDB and $\Delta t = 2.81 \times 10^{-5}$ for EigenWorms in the first layer. Additionally, the dropout column corresponds to variational dropout (Gal & Ghahramani, 2016), which is applied after each consecutive layer. Note that for the IMDB task also an embedding dropout with $p = 0.65$ is used.

We train the UnICORNN for a total of 50 epochs on the IMDB task and for a total of 250 epochs on the EigenWorms task. Moreover, we train UnICORNN for 650 epochs on psMNIST, after which we decrease the learning rate by a factor of 10 and proceed training for 3 times the amount of epochs used before reducing the learning rate. On all other tasks, UnICORNN is trained for 250 epochs, after which we decrease the learning rate by a factor of 10 and proceed training for additional 250 epochs. The resulting best performing networks are selected *based on a validation set*.

Table 6. Hyperparameters of the best performing UnICORNN architecture (based on a validation set) for each experiment.

Experiment	learning rate	dropout	batch size	Δt	α
noise padded CIFAR-10	3.14×10^{-2}	1.0×10^{-1}	30	1.26×10^{-1}	13.0
psMNIST (#units = 128)	1.14×10^{-3}	1.0×10^{-1}	64	4.82×10^{-1}	12.53
psMNIST (#units = 256)	2.51×10^{-3}	1.0×10^{-1}	32	1.9×10^{-1}	30.65
IMDB	1.67×10^{-4}	6.1×10^{-1}	32	2.05×10^{-1}	0.0
EigenWorms	8.59×10^{-3}	0.0	8	3.43×10^{-2}	0.0
Healthcare: RR	3.98×10^{-3}	1.0×10^{-1}	32	1.1×10^{-2}	9.0
Healthcare: HR	2.88×10^{-3}	1.0×10^{-1}	32	4.6×10^{-2}	10.0

B. Implementation details

As already described in the implementation details of the main paper, we can speed up the computation of the forward and backward pass, by parallelizing the input transformation and computing the recurrent part for each independent dimension in an independent CUDA thread. While the forward/backward pass for the input transformation is simply that of an affine transformation, we discuss only the recurrent part. Since we compute the gradients of each dimension of the UnICORNN independently and add them up afterwards to get the full gradient, we will simplify to the following one-dimensional system:

$$
\begin{aligned}
z_n &= z_{n-1} - \Delta t \hat{\sigma}(c)[\sigma\left(w y_{n-1} + x_n\right) + \alpha y_{n-1}], \\
y_n &= y_{n-1} + \Delta t \hat{\sigma}(c) z_n,
\end{aligned}
$$

where $x_n = (\mathbf{V}\mathbf{u}_n)_j$ is the transformed input corresponding to the respective dimension $j = 1, \ldots, m$.

Since we wish to train the UnICORNN on some given objective

$$\mathcal{E} := \sum_{n=1}^{N} \tilde{\mathcal{E}}(y_n), \tag{1}$$

where $\tilde{\mathcal{E}}$ is some loss function taking the hidden states y_n as inputs, for instance mean-square distance of (possibly transformed) hidden states y_n to some ground truth. During training, we compute gradients of the loss function (1) with respect to the following quantities $\mathbf{\Theta} = [w, \Delta t, x_n]$, i.e.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{n=1}^{N} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial \theta}, \quad \forall\, \theta \in \mathbf{\Theta}. \tag{2}$$

We can work out a recursion formula to compute the gradients in (2). We will exemplarily provide the formula for the gradient with respect to the hidden weight w. The computation of the gradients with respect to the other quantities follow similarly. Thus

$$\delta_k^z = \delta_{k-1}^z + \delta_{k-1}^y \Delta t \hat{\sigma}(c), \tag{3}$$

$$\delta_k^y = \delta_{k-1}^y - \delta_k^z \Delta t \hat{\sigma}(c)[\sigma'(w y_{N-k} + x_{N-k+1})w + \alpha] + \frac{\partial \tilde{\mathcal{E}}}{\partial y_{N-k}}, \tag{4}$$

with initial values $\delta_0^y = \frac{\partial \tilde{\mathcal{E}}}{\partial y_N}$ and $\delta_0^z = 0$. The gradient can then be computed as

$$\frac{\partial \mathcal{E}}{\partial w} = \sum_{k=1}^{N} a_k, \qquad \text{with } a_k = -\delta_k^z \Delta t \hat{\sigma}(c)\sigma'(w y_{N-k} + x_{N-k+1}) y_{N-k}. \tag{5}$$

Note that this recursive formula is a direct formulation of the back-propagation through time algorithm (Werbos, 1990) for the UnICORNN.

We can verify formula (3)-(5) by explicitly calculating the gradient in (2):

$$\begin{aligned}
\frac{\partial \mathcal{E}}{\partial w} &= \sum_{n=1}^{N} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} = \sum_{n=1}^{N-1} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + \frac{\partial \tilde{\mathcal{E}}}{\partial y_N}\left[\frac{\partial y_{N-1}}{\partial w} + \Delta t \hat{\sigma}(c)\left(\frac{\partial z_{N-1}}{\partial w} - \Delta t \hat{\sigma}(c)(\sigma'(w y_{N-1} + x_N)\right.\right. \\
&\left.\left.(y_{N-1} + w\frac{\partial y_{N-1}}{\partial w}) + \alpha \frac{\partial y_{N-1}}{\partial w}\right)\right] = \sum_{n=1}^{N-2} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + a_1 + \delta_1^z \frac{\partial z_{N-1}}{\partial w} + \delta_1^y \frac{\partial y_{N-1}}{\partial w} \\
&= \sum_{n=1}^{N-2} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + a_1 + \delta_1^y \frac{\partial y_{N-2}}{\partial w} + (\delta_1^y \Delta t \hat{\sigma}(c) + \delta_1^z)\left(\frac{\partial z_{N-2}}{\partial w} - \Delta t \hat{\sigma}(c)(\sigma'(w y_{N-2} + x_{N-1})\right. \\
&\left.(y_{N-2} + w\frac{\partial y_{N-2}}{\partial w}) + \alpha \frac{\partial y_{N-2}}{\partial w})\right) = \sum_{n=1}^{N-3} \frac{\partial \tilde{\mathcal{E}}(y_n)}{\partial w} + \sum_{k=1}^{2} a_k + \delta_2^z \frac{\partial z_{N-2}}{\partial w} + \delta_2^y \frac{\partial y_{N-2}}{\partial w}.
\end{aligned}$$

Iterating the same reformulation yields the desired formula (3)-(5).

C. Rigorous bounds on UnICORNN

We rewrite UnICORNN (Eqn. (6) in the main text) in the following form: for all $1 \le \ell \le L$ and for all $1 \le i \le m$

$$\begin{aligned}
\mathbf{y}_n^{\ell,i} &= \mathbf{y}_{n-1}^{\ell,i} + \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \mathbf{z}_n^{\ell,i}, \\
\mathbf{z}_n^{\ell,i} &= \mathbf{z}_{n-1}^{\ell,i} - \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \sigma(\mathbf{A}_{n-1}^{\ell,i}) - \alpha \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \mathbf{y}_{n-1}^{\ell,i}, \\
\mathbf{A}_{n-1}^{\ell,i} &= \mathbf{w}^{\ell,i} \mathbf{y}_{n-1}^{\ell,i} + \left(\mathbf{V}^{\ell} \mathbf{y}_n^{\ell-1}\right)^i + \mathbf{b}^{\ell,i}.
\end{aligned} \tag{6}$$

Here, we have denoted the i-th component of a vector $\mathbf{x}$ as $\mathbf{x}^i$.

We follow standard practice and set $\mathbf{y}_0^{\ell} = \mathbf{z}_0^{\ell} \equiv 0$, for all $1 \le \ell \le L$. Moreover for simplicity of exposition, we set $\alpha > 0$ in the following.

C.1. Pointwise bounds on hidden states.

We have the following bounds on the discrete hidden states,

Proposition C.1. *Let $\mathbf{y}_n^\ell, \mathbf{z}_n^\ell$ be the hidden states at the n-th time level t_n for UnICORNN (6), then under the assumption that the time step $\Delta t << 1$ is sufficiently small, these hidden states are bounded as,*

$$\max_{1\leq i\leq m} |\mathbf{y}_n^{\ell,i}| \leq \sqrt{\frac{2}{\alpha}\left(1+2\beta t_n\right)}, \quad \max_{1\leq i\leq m} |\mathbf{z}_n^{\ell,i}| \leq \sqrt{2\left(1+2\beta t_n\right)} \quad \forall n, \forall\, 1\leq \ell \leq L, \tag{7}$$

with the constant

$$\beta = \max\{1+2\alpha, 4\alpha^2\}.$$

Proof. We fix ℓ, n and multiply the first equation in (6) with $\alpha \mathbf{y}_{n-1}^{\ell,i}$ and use the elementary identity

$$b(a-b) = \frac{a^2}{2} - \frac{b^2}{2} - \frac{1}{2}(a-b)^2,$$

to obtain

$$\begin{aligned} \frac{\alpha(\mathbf{y}_n^{\ell,i})^2}{2} &= \frac{\alpha(\mathbf{y}_{n-1}^{\ell,i})^2}{2} + \frac{\alpha}{2}(\mathbf{y}_n^{\ell,i} - \mathbf{y}_{n-1}^{\ell,i})^2 + \alpha\Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}\mathbf{z}_n^{\ell,i}, \\ &= \frac{\alpha(\mathbf{y}_{n-1}^{\ell,i})^2}{2} + \frac{\alpha\Delta t^2}{2}(\hat{\sigma}(\mathbf{c}^{\ell,i}))^2(\mathbf{z}_n^{\ell,i})^2 + \alpha\Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}\mathbf{z}_n^{\ell,i}. \end{aligned} \tag{8}$$

Next, we multiply the second equation in (6) with $\mathbf{z}_n^{\ell,i}$ and use the elementary identity

$$a(a-b) = \frac{a^2}{2} - \frac{b^2}{2} + \frac{1}{2}(a-b)^2,$$

to obtain

$$\begin{aligned} \frac{(\mathbf{z}_n^{\ell,i})^2}{2} = \frac{(\mathbf{z}_{n-1}^{\ell,i})^2}{2} &- \frac{1}{2}(\mathbf{z}_n^{\ell,i} - \mathbf{z}_{n-1}^{\ell,i})^2 - \Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\sigma(\mathbf{A}_{n-1}^{\ell,i})\left(\mathbf{z}_n^{\ell,i} - \mathbf{z}_{n-1}^{\ell,i}\right) \\ &- \Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\sigma(\mathbf{A}_{n-1}^{\ell,i})\mathbf{z}_{n-1}^{\ell,i} - \alpha\Delta t\hat{\sigma}(\mathbf{c}^{\ell,i})\mathbf{y}_{n-1}^{\ell,i}\mathbf{z}_n^{\ell,i}. \end{aligned} \tag{9}$$

Adding (8) and (9) and using Cauchy's inequality yields,

$$\begin{aligned} \frac{\alpha(\mathbf{y}_n^{\ell,i})^2}{2} + \frac{(\mathbf{z}_n^{\ell,i})^2}{2} &\leq \frac{\alpha(\mathbf{y}_{n-1}^{\ell,i})^2}{2} + \frac{(1+\Delta t)(\mathbf{z}_{n-1}^{\ell,i})^2}{2} + \frac{\alpha\Delta t^2}{2}(\hat{\sigma}(\mathbf{c}^{\ell,i}))^2(\mathbf{z}_n^{\ell,i})^2 \\ &\quad + (\hat{\sigma}(\mathbf{c}^{\ell,i}))^2(\sigma(\mathbf{A}_{n-1}^{\ell,i}))^2\Delta t + \frac{\Delta t - 1}{2}(\mathbf{z}_n^{\ell,i} - \mathbf{z}_{n-1}^{\ell,i})^2 \\ \Rightarrow \alpha(\mathbf{y}_n^{\ell,i})^2 + (\mathbf{z}_n^{\ell,i})^2 &\leq \alpha(\mathbf{y}_{n-1}^{\ell,i})^2 + (1+\Delta t)(\mathbf{z}_{n-1}^{\ell,i})^2 + 2\Delta t + \alpha\Delta t^2(\mathbf{z}_n^{\ell,i})^2, \end{aligned}$$

where the last inequality follows from the fact that $|\sigma|, |\hat{\sigma}| \leq 1$ and $\Delta t < 1$. Using the elementary inequality,

$$(a+b+c)^2 \leq 4a^2 + 4b^2 + 2c^2,$$

and substituting for $\mathbf{z}_n^{\ell,i}$ from the second equation of (6) in the last inequality leads to,

$$\alpha(\mathbf{y}_n^{\ell,i})^2 + (\mathbf{z}_n^{\ell,i})^2 \leq (1+4\alpha^2\Delta t^4)\alpha(\mathbf{y}_{n-1}^{\ell,i})^2 + (1+\Delta t + 2\alpha\Delta t^2)(\mathbf{z}_{n-1}^{\ell,i})^2 + 2\Delta t + 4\alpha\Delta t^4.$$

Denoting $H_n = \alpha(\mathbf{y}_n^{\ell,i})^2 + (\mathbf{z}_n^{\ell,i})^2$ and

$$G := 1 + \beta\Delta t, \quad \beta = \max\{1+2\alpha, 4\alpha^2\}$$

yields the following inequality,

$$H_n \leq GH_{n-1} + 2\Delta t(1 + 2\alpha\Delta t^3). \tag{10}$$

Iterating the above n-times and using the fact that the initial data is such that $H_0 \equiv 0$ we obtain,

$$\begin{aligned} H_n &\leq \left(2\Delta t + 4\alpha\Delta t^4\right) \sum_{k=0}^{n-1} (1+\beta\Delta t)^k \\ &\leq \frac{(1+\beta\Delta t)^n}{\beta\Delta t}\left(2\Delta t + 4\alpha\Delta t^4\right) \\ &\leq \frac{1}{\beta}\left(1+2\beta n\Delta t\right)\left(2+4\alpha\Delta t^3\right) \quad \text{as } \Delta t << 1, \\ &\leq 2(1+2\beta t_n) \text{ (from definition of } \beta). \end{aligned} \tag{11}$$

The definition of H clearly implies the desired bound (7). □

C.2. On the exploding gradient problem for UnICORNN and Proof of proposition 3.1 of the main text.

We train the RNN (6) to minimize the loss function,

$$\mathcal{E} := \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}_n, \quad \mathcal{E}_n = \frac{1}{2}\|\mathbf{y}_n^L - \bar{\mathbf{y}}_n\|_2^2, \tag{12}$$

with $\bar{\mathbf{y}}$ being the underlying ground truth (training data). Note that the loss function (12) only involves the output at the last hidden layer (we set the affine output layer to identity for the sake of simplicity). During training, we compute gradients of the loss function (12) with respect to the trainable weights and biases $\mathbf{\Theta} = [\mathbf{w}^\ell, \mathbf{V}^\ell, \mathbf{b}^\ell, \mathbf{c}^\ell]$, for all $1 \leq \ell \leq L$ i.e.

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial \mathcal{E}_n}{\partial \theta}, \quad \forall\, \theta \in \mathbf{\Theta}. \tag{13}$$

We have the following bound on the gradient (13),

Proposition C.2. *Let the time step $\Delta t << 1$ be sufficiently small in the RNN (6) and let $\mathbf{y}_n^\ell, \mathbf{z}_n^\ell$, for $1 \leq \ell \leq L$, be the hidden states generated by the RNN (6). Then, the gradient of the loss function $\mathcal{E}$ (12) with respect to any parameter $\theta \in \mathbf{\Theta}$ is bounded as,*

$$\left|\frac{\partial \mathcal{E}}{\partial \theta}\right| \leq \frac{1-(\Delta t)^L}{1-\Delta t} T(1+2\gamma T)\overline{\mathbf{V}}(\overline{Y}+\mathbf{F})\mathbf{\Delta}, \tag{14}$$

with $\bar{Y} = \max\limits_{1\leq n\leq N} \|\bar{\mathbf{y}}_n\|_\infty$, be a bound on the underlying training data and other quantities in (14) defined as,

$$\begin{aligned} \gamma &= \max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right) + \frac{\left(\max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right)\right)^2}{2}, \\ \overline{\mathbf{V}} &= \prod_{q=1}^{L} \max\{1, \|\mathbf{V}^q\|_\infty\}, \\ \mathbf{F} &= \sqrt{\frac{2}{\alpha}(1+2\beta T)}, \\ \mathbf{\Delta} &= \left(2 + \sqrt{(1+2\beta T)} + (2+\alpha)\sqrt{\frac{2}{\alpha}(1+2\beta T)}\right). \end{aligned} \tag{15}$$

Proof. For any $1 \leq n \leq N$ and $1 \leq \ell \leq L$, let $\mathbf{X}_n^\ell \in \mathbb{R}^{2m}$ be the augmented hidden state vector defined by,

$$\mathbf{X}_n^\ell = \left[\mathbf{y}_n^{\ell,1}, \mathbf{z}_n^{\ell,1}, \ldots, \mathbf{y}_n^{\ell,i}, \mathbf{z}_n^{\ell,i}, \ldots, \mathbf{y}_n^{\ell,m}, \mathbf{z}_n^{\ell,m}\right]. \tag{16}$$

For any $\theta \in \mathbf{\Theta}$, we can apply the chain rule repeatedly to obtain the following extension of the formula of (Pascanu et al., 2013) to a deep RNN,

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{\ell=1}^{L}\sum_{k=1}^{n} \underbrace{\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L}\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell}\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}}_{\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}}. \tag{17}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k^\ell$ with respect to the parameter θ, while keeping the other arguments constant.

We remark that the quantity $\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}$ denotes the contribution from the k-recurrent step at the l-th hidden layer of the deep RNN (6) to the overall hidden state gradient at the step n.

It is straightforward to calculate that,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} = \left[\mathbf{y}_n^{L,1} - \overline{\mathbf{y}}_n^1, 0, \ldots, \mathbf{y}_n^{L,i} - \overline{\mathbf{y}}_n^i, 0, \ldots, \mathbf{y}_n^{L,m} - \overline{\mathbf{y}}_n^m, 0\right]. \tag{18}$$

Repeated application of the chain and product rules yields,

$$\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} = \prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \prod_{q=\ell+1}^{L} \frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}}. \tag{19}$$

For any j, a straightforward calculation using the form of the RNN (6) leads to the following representation formula for the matrix $\frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \in \mathbb{R}^{2m} \times \mathbb{R}^{2m}$:

$$\frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} = \begin{bmatrix} \mathbf{B}_j^{L,1} & 0 & \ldots & 0 \\ 0 & \mathbf{B}_j^{L,2} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & 0 & \mathbf{B}_j^{L,m} \end{bmatrix}, \tag{20}$$

with the block matrices $\mathbf{B}_j^{L,i} \in \mathbb{R}^{2\times 2}$ given by,

$$\mathbf{B}_j^{L,i} = \begin{bmatrix} 1 - (\hat{\sigma}(\mathbf{c}^{L,i}))^2 \Delta t^2 \left(\mathbf{w}^{L,i} \sigma'(\mathbf{A}_{j-1}^{L,i}) + \alpha\right) & \hat{\sigma}(\mathbf{c}^{L,i}) \Delta t \\ -\hat{\sigma}(\mathbf{c}^{L,i}) \Delta t \left(\mathbf{w}^{L,i} \sigma'(\mathbf{A}_{j-1}^{L,i}) + \alpha\right) & 1 \end{bmatrix}. \tag{21}$$

Similarly for any q, the matrix $\frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} \in \mathbb{R}^{2m\times 2m}$ can be readily computed as,

$$\frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} = \begin{bmatrix} \mathbf{D}_{11}^{q,k} & 0 & \mathbf{D}_{12}^{q,k} & 0 & \ldots & \ldots & \mathbf{D}_{1m}^{q,k} & 0 \\ \mathbf{E}_{11}^{q,k} & 0 & \mathbf{E}_{12}^{q,k} & 0 & \ldots & \ldots & \mathbf{E}_{1m}^{q,k} & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \mathbf{D}_{m1}^{q,k} & 0 & \mathbf{D}_{m2}^{q,k} & 0 & \ldots & \ldots & \mathbf{D}_{mm}^{q,k} & 0 \\ \mathbf{E}_{m1}^{q,k} & 0 & \mathbf{E}_{m2}^{q,k} & 0 & \ldots & \ldots & \mathbf{E}_{mm}^{q,k} & 0 \end{bmatrix}, \tag{22}$$

with entries given by,

$$\mathbf{D}_{i,\bar{i}}^{q,k} = -\Delta t^2 (\hat{\sigma}(\mathbf{c}^{q,i}))^2 \sigma' \left(\mathbf{A}_{k-1}^{q,i}\right) \mathbf{V}_{i\bar{i}}^q, \quad \mathbf{E}_{i,\bar{i}}^{q,k} = -\Delta t \hat{\sigma}(\mathbf{c}^{q,i}) \sigma' \left(\mathbf{A}_{k-1}^{q,i}\right) \mathbf{V}_{i\bar{i}}^q. \tag{23}$$

A direct calculation with (21) leads to,

$$\begin{aligned} \|\mathbf{B}_j^{L,i}\|_\infty &\leq \max\left(1 + \Delta t + (|\mathbf{w}^{L,i}| + \alpha)\Delta t^2, 1 + (|\mathbf{w}^{L,i}| + \alpha)\Delta t\right) \\ &\leq 1 + \max\left(2, |\mathbf{w}^{L,i}| + \alpha\right) \Delta t + \left(\max\left(2, |\mathbf{w}^{L,i}| + \alpha\right)\right)^2 \frac{\Delta t^2}{2}. \end{aligned} \tag{24}$$

Using the definition of the L^∞ norm of a matrix, we use (24) to the derive the following bound from (20),

$$\begin{aligned} \left\| \frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \right\|_\infty &\leq 1 + \max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right) \Delta t + \left(\max\left(2, \|\mathbf{w}^L\|_\infty + \alpha\right)\right)^2 \frac{\Delta t^2}{2} \\ &\leq 1 + \gamma \Delta t, \end{aligned} \tag{25}$$

with γ defined in (15).

As $\Delta t < 1$, it is easy to see that,

$$\left\| \frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} \right\|_\infty \leq \|\mathbf{V}^q\|_\infty \Delta t. \tag{26}$$

Combining (25) and (26) , we obtain from (19)

$$\begin{aligned} \left\| \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \right\|_\infty &\leq \prod_{j=k+1}^{n} \left\| \frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} \right\|_\infty \prod_{q=\ell+1}^{L} \left\| \frac{\partial \mathbf{X}_k^q}{\partial \mathbf{X}_k^{q-1}} \right\|_\infty \\ &\leq \Delta t^{L-\ell} \prod_{q=\ell+1}^{L} \|\mathbf{V}^q\|_\infty (1 + 2\gamma(n-k)\Delta t), \quad (\text{as } \Delta t << 1) \\ &\leq \overline{\mathbf{V}} \Delta t^{L-\ell} (1 + 2\gamma t_n), \end{aligned} \tag{27}$$

where the last inequality follows from the fact that $t_n = n\Delta t \leq T$ and the definition of $\overline{\mathbf{V}}$ in (15).

Next, we observe that for any $\theta \in \mathbf{\Theta}$

$$\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta} = \left[\frac{\partial^+ \mathbf{y}_k^{\ell,1}}{\partial \theta}, \frac{\partial^+ \mathbf{z}_k^{\ell,1}}{\partial \theta} \ldots, \ldots, \frac{\partial^+ \mathbf{y}_k^{\ell,i}}{\partial \theta}, \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta}, \ldots, \ldots, \frac{\partial^+ \mathbf{y}_k^{\ell,m}}{\partial \theta}, \frac{\partial^+ \mathbf{z}_k^{\ell,m}}{\partial \theta} \right]^\top. \tag{28}$$

For any $1 \leq i \leq m$, a direct calculation with the RNN (6) yields,

$$\begin{aligned} \frac{\partial^+ \mathbf{y}_k^{\ell,i}}{\partial \theta} &= \Delta t \hat{\sigma}'(\mathbf{c}^{\ell,i}) \frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \mathbf{z}_k^{\ell,i} + \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta}, \\ \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta} &= -\Delta t \hat{\sigma}'(\mathbf{c}^{\ell,i}) \frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \sigma(\mathbf{A}_{k-1}^{\ell,i}) - \Delta t \hat{\sigma}(\mathbf{c}^{\ell,i}) \sigma'(\mathbf{A}_{k-1}^{\ell,i}) \frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} - \alpha \Delta t \hat{\sigma}'(\mathbf{c}^{\ell,i}) \frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \mathbf{y}_{k-1}^{\ell,i}. \end{aligned} \tag{29}$$

Next, we have to compute explicitly $\frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta}$ and $\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta}$ in order to complete the expressions (29). To this end, we need to consider explicit forms of the parameter θ and obtain,

If $\theta = \mathbf{w}^{q,p}$, for some $1 \leq q \leq L$ and $1 \leq p \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} = \begin{cases} \mathbf{y}_{k-1}^{\ell,i}, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{30}$$

If $\theta = \mathbf{b}^{q,p}$, for some $1 \leq q \leq L$ and $1 \leq p \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} = \begin{cases} 1, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{31}$$

If $\theta = \mathbf{V}_{p,\bar{p}}^q$, for some $1 \leq q \leq L$ and $1 \leq p, \bar{p} \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} = \begin{cases} \mathbf{y}_k^{\ell-1,\bar{p}}, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{32}$$

If $\theta = \mathbf{c}^{q,p}$for any $1 \leq q \leq L$ and $1 \leq p \leq m$, then,

$$\frac{\partial \mathbf{A}_{k-1}^{\ell,i}}{\partial \theta} \equiv 0. \tag{33}$$

Similarly, if $\theta = \mathbf{w}^{q,p}$ or $\theta = \mathbf{b}^{q,p}$, for some $1 \leq q \leq L$ and $1 \leq p \leq m$, or If $\theta = \mathbf{V}_{p,\bar{p}}^q$, for some $1 \leq q \leq L$ and $1 \leq p, \bar{p} \leq m$, then

$$\frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} \equiv 0. \tag{34}$$

On the other hand, if $\theta = \mathbf{c}^{q,p}$for any $1 \leq q \leq L$ and $1 \leq p \leq m$, then

$$\frac{\partial \mathbf{c}^{\ell,i}}{\partial \theta} = \begin{cases} 1, & \text{if} \quad q = \ell, p = i, \\ 0, & \text{if} \quad \text{otherwise.} \end{cases} \tag{35}$$

For any $\theta \in \mathbf{\Theta}$, by substituting (30) to (35) into (29) and doing some simple algebra with norms, leads to the following inequalities,

$$\left| \frac{\partial^+ \mathbf{z}_k^{\ell,i}}{\partial \theta} \right| \leq \Delta t \left(1 + \alpha |\mathbf{y}_{k-1}^{\ell,i}| + \max \left(|\mathbf{y}_{k-1}^{\ell,i}|, |\mathbf{y}_k^{\ell-1,\bar{p}}|, 1 \right) \right), \tag{36}$$

and,

$$\left| \frac{\partial^+ \mathbf{y}_k^{\ell,i}}{\partial \theta} \right| \leq \Delta t |\mathbf{z}_k^{\ell,i}| + \Delta t^2 \left(1 + \alpha |\mathbf{y}_{k-1}^{\ell,i}| + \max \left(|\mathbf{y}_{k-1}^{\ell,i}|, |\mathbf{y}_k^{\ell-1,\bar{p}}|, 1 \right) \right), \tag{37}$$

for any $1 \leq \bar{p} \leq m$.

By the definition of L^∞ norm of a vector and some straightforward calculations with (37) yields,

$$\left\| \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta} \right\|_\infty \leq \Delta t \left(2 + \|\mathbf{z}_k^\ell\|_\infty + (1+\alpha) \|\mathbf{y}_{k-1}^\ell\|_\infty + \|\mathbf{y}_k^{\ell-1}\|_\infty \right). \tag{38}$$

From the pointwise bounds (7), we can directly bound the above inequality further as,

$$\left\| \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta} \right\|_\infty \leq \Delta t \left(2 + \sqrt{2\left(1 + 2\beta T\right)} + (2+\alpha) \sqrt{\frac{2}{\alpha}\left(1 + 2\beta T\right)} \right). \tag{39}$$

By (18) and the definition of $\overline{Y}$ as well as the bound (7) on the hidden states, it is straightforward to obtain that,

$$\left\| \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} \right\|_\infty \leq \overline{Y} + \sqrt{\frac{2}{\alpha}\left(1 + 2\beta T\right)} \tag{40}$$

From the definition in (17), we have

$$\left| \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} \right| \leq \left\| \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} \right\|_\infty \left\| \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \right\|_\infty \left\| \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta} \right\|_\infty. \tag{41}$$

Substituting (40), (39) and (25) into (41) yields,

$$\left| \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} \right| \leq \Delta t^{L-\ell+1} \left(1 + 2\gamma T\right) \overline{\mathbf{V}} (\overline{Y} + \mathbf{F}) \mathbf{\Delta}, \tag{42}$$

with $\mathbf{F}$ and $\mathbf{\Delta}$ defined in (15).

Therefore, from the fact that $\Delta t < 1, t_n = n\Delta t \leq T$ and (17), we obtain

$$\left| \frac{\partial \mathcal{E}_n}{\partial \theta} \right| \leq \frac{1 - (\Delta t)^L}{1 - \Delta t} T (1 + 2\gamma T) \overline{\mathbf{V}} (\overline{Y} + \mathbf{F}) \mathbf{\Delta}. \tag{43}$$

By the definition of the loss function (12) and the fact that the right-hand-side of (43) is independent of n leads to the desired bound (14).

□

C.3. On the Vanishing gradient problem for UnICORNN and Proof of Proposition 3.2 of the main text.

By applying the chain rule repeatedly to the each term on the right-hand-side of (13), we obtain

$$\frac{\partial \mathcal{E}_n}{\partial \theta} = \sum_{\ell=1}^{L} \sum_{k=1}^{n} \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}, \ \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^L} \frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_k^\ell} \frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}. \tag{44}$$

Here, the notation $\frac{\partial^+ \mathbf{X}_k^\ell}{\partial \theta}$ refers to taking the partial derivative of $\mathbf{X}_k^\ell$ with respect to the parameter θ, while keeping the other arguments constant. The quantity $\frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta}$ denotes the contribution from the k-recurrent step at the l-th hidden layer of the deep RNN (6) to the overall hidden state gradient at the step n. The vanishing gradient problem (Pascanu et al., 2013) arises if $\left| \frac{\partial \mathcal{E}_{k,\ell}^{(n,L)}}{\partial \theta} \right|$, defined in (44), $\to 0$ exponentially fast in k, for $k << n$ (long-term dependencies). In that case, the RNN does not have long-term memory, as the contribution of the k-th hidden state at the ℓ-th layer to error at time step t_n is infinitesimally small.

As argued in the main text, the vanishing gradient problem for RNNs focuses on the possible smallness of contributions of the gradient over a large number of recurrent steps. As this behavior of the gradient is independent of the number of layers, we start with a result on the vanishing gradient problem for a single hidden layer here. Also, for the sake of definiteness, we set the scalar parameter $\theta = \mathbf{w}^{1,p}$ for some $1 \leq p \leq m$. Similar results also hold for any other $\theta \in \mathbf{\Theta}$. Moreover, we introduce the following *order*-notation,

$$\begin{aligned} \beta = \mathcal{O}(\gamma), \text{ for } \gamma, \beta \in \mathbb{R}_+ \quad &\text{if there exists constants } \overline{C}, \underline{C} \text{ such that } \underline{C}\gamma \leq \beta \leq \overline{C}\gamma. \\ \mathbf{M} = \mathcal{O}(\gamma), \text{ for } \mathbf{M} \in \mathbb{R}^{d_1 \times d_2}, \gamma \in \mathbb{R}_+ \quad &\text{if there exists constant } \overline{C} \text{ such that } \|\mathbf{M}\| \leq \overline{C}\gamma. \end{aligned} \tag{45}$$

We restate Proposition 3.2 of the main text,

Proposition C.3. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN (6). Then the gradient for long-term dependencies, i.e. $k << n$, satisfies the representation formula,*

$$\frac{\partial \mathcal{E}_{k,1}^{(n,1)}}{\partial \mathbf{w}^{1,p}} = -\Delta t \hat{\sigma}(\mathbf{c}^{1,p})^2 t_n \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p} \left(\mathbf{y}_n^{1,p} - \overline{\mathbf{y}}_n^p \right) + \mathcal{O}(\Delta t^2). \tag{46}$$

Proof. Following the definition (44) and as $L = 1$ and $\theta = \mathbf{w}^{1,p}$, we have,

$$\frac{\partial \mathcal{E}_{k,1}^{(n,1)}}{\partial \mathbf{w}^{1,p}} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^1} \frac{\partial \mathbf{X}_n^1}{\partial \mathbf{X}_k^1} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}}. \tag{47}$$

We will explicitly compute all three expressions on the right-hand-side of (47). To start with, using (28), (29) and (30), we obtain,

$$\begin{aligned} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} &= \left[0, 0, \ldots, \ldots, \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}}, \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}}, \ldots, \ldots, 0, 0 \right]^\top, \\ \left(\frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} \right)_{2p-1} &= \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}} = -\Delta t^2 (\hat{\sigma}(\mathbf{c}^{1,p}))^2 \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p}, \\ \left(\frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} \right)_{2p} &= \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} = -\Delta t \hat{\sigma}(\mathbf{c}^{1,p}) \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p}. \end{aligned} \tag{48}$$

Using the product rule (19) we have,

$$\frac{\partial \mathbf{X}_n^1}{\partial \mathbf{X}_k^1} = \prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1}. \tag{49}$$

Observing from the expressions (20) and (21) and using the *order*-notation (45), we obtain that,

$$\frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} = \mathbf{I}_{2m \times 2m} + \Delta t \mathbf{C}_j^1 + \mathcal{O}(\Delta t^2), \tag{50}$$

with $\mathbf{I}_{k\times k}$ is the $k \times k$ Identity matrix and the matrix $\mathbf{C}_j^1$ defined by,

$$\frac{\partial \mathbf{X}_j^L}{\partial \mathbf{X}_{j-1}^L} = \begin{bmatrix} \mathbf{C}_j^{1,1} & 0 & \dots & 0 \\ 0 & \mathbf{C}_j^{1,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{C}_j^{1,m} \end{bmatrix}, \tag{51}$$

with the block matrices $\mathbf{C}_j^{1,i} \in \mathbb{R}^{2\times 2}$ given by,

$$\mathbf{C}_j^{1,i} = \begin{bmatrix} 0 & \hat{\sigma}(\mathbf{c}^{1,i}) \\ -\hat{\sigma}(\mathbf{c}^{1,i}) \left(\mathbf{w}^{1,i} \sigma'(\mathbf{A}_{j-1}^{1,i}) + \alpha \right) & 0 \end{bmatrix}. \tag{52}$$

By a straightforward calculation and the use of induction, we claim that

$$\prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} = \mathbf{I}_{2m\times 2m} + \Delta t \mathbf{C}^1 + \mathcal{O}(\Delta t^2), \tag{53}$$

with

$$\mathbf{C}^1 = \begin{bmatrix} \mathbf{C}^{1,1} & 0 & \dots & 0 \\ 0 & \mathbf{C}^{1,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{C}^{1,m} \end{bmatrix}, \tag{54}$$

with the block matrices $\mathbf{C}^{1,i} \in \mathbb{R}^{2\times 2}$ given by,

$$\mathbf{C}^{1,i} = \begin{bmatrix} 0 & (n-k)\hat{\sigma}(\mathbf{c}^{1,i}) \\ -(n-k)\alpha\hat{\sigma}(\mathbf{c}^{1,i}) - \hat{\sigma}(\mathbf{c}^{1,i})\mathbf{w}^{1,i} \sum\limits_{j=k+1}^{n} \sigma'(\mathbf{A}_{j-1}^{1,i}) & 0 \end{bmatrix}. \tag{55}$$

By the assumption that $k << n$ and using the fact that $t_n = n\Delta t$, we have that,

$$\Delta t \mathbf{C}^{1,i} = \begin{bmatrix} 0 & t_n\hat{\sigma}(\mathbf{c}^{1,i}) \\ -t_n\alpha\hat{\sigma}(\mathbf{c}^{1,i}) - \hat{\sigma}(\mathbf{c}^{1,i})\mathbf{w}^{1,i}\Delta t \sum\limits_{j=k+1}^{n} \sigma'(\mathbf{A}_{j-1}^{1,i}) & 0 \end{bmatrix}. \tag{56}$$

Hence, the non-zero entries in the block matrices can be $\mathcal{O}(1)$. Therefore, the product formula (53) is modified to,

$$\prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} = \mathbf{C} + \mathcal{O}(\Delta t), \tag{57}$$

with the $2m \times 2m$ matrix $\mathbf{C}$ defined as,

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}^1 & 0 & \dots & 0 \\ 0 & \mathbf{C}^2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \mathbf{C}^m \end{bmatrix}, \tag{58}$$

and,

$$\mathbf{C}^i = \begin{bmatrix} 1 & t_n\hat{\sigma}(\mathbf{c}^{1,i}) \\ -t_n\alpha\hat{\sigma}(\mathbf{c}^{1,i}) - \hat{\sigma}(\mathbf{c}^{1,i})\mathbf{w}^{1,i}\Delta t \sum\limits_{j=k+1}^{n} \sigma'(\mathbf{A}_{j-1}^{1,i}) & 1 \end{bmatrix}. \tag{59}$$

Thus by taking the product of (57) with (48), we obtain that,

$$\prod_{j=k+1}^{n} \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}} = \left[0, 0, \dots, \dots, \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}} + \mathbf{C}_{12}^p \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}}, \mathbf{C}_{21}^p \frac{\partial^+ \mathbf{y}_k^{1,p}}{\partial \mathbf{w}^{1,p}} + \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} \dots, \dots, 0, 0 \right]^\top + \mathcal{O}(\Delta t^2), \tag{60}$$

with $\mathbf{C}_{12}^p, \mathbf{C}_{21}^p$ are the off-diagonal entries of the corresponding block matrix, defined in (59). Note that the $\mathcal{O}(\Delta t^2)$ remainder term arises from the Δt-dependence in (48).

From (18), we have that,

$$\frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^1} = \left[\mathbf{y}_n^{1,1} - \overline{\mathbf{y}}_n^1, 0, \ldots, \mathbf{y}_n^{1,i} - \overline{\mathbf{y}}_n^i, 0, \ldots, \mathbf{y}_n^{1,m} - \overline{\mathbf{y}}_n^m, 0\right]. \tag{61}$$

Therefore, taking the products of (61) and (60) and substituting the explicit expressions in (48), we obtain the desired identity (46). □

C.4. On the vanishing gradient problem for the multilayer version of UnICORNN.

The explicit representation formula (46) holds for 1 hidden layer in (6). What happens when additional hidden layers are stacked together as in UnICORNN (6)? To answer this question, we consider the concrete case of $L = 3$ layers as this is the largest number of layers that we have used in the context of UnICORNN with fully connected stacked layers. As before, we set the scalar parameter $\theta = \mathbf{w}^{1,p}$ for some $1 \leq p \leq m$. Similar results also hold for any other $\theta \in \mathbf{\Theta}$. We have the following representation formula for the gradient in this case,

Proposition C.4. *Let $\mathbf{y}_n$ be the hidden states generated by the RNN (6). The gradient for long-term dependencies satisfies the representation formula,*

$$\frac{\partial \mathcal{E}_{k,1}^{(n,3)}}{\partial \mathbf{w}^{1,p}} = \Delta t^4 \hat{\sigma}(\mathbf{c}^{1,p}) t_n \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} \sum_{i=1}^m \bar{\mathbf{G}}_{2i-1,2p-1} \left(\mathbf{y}^{3,i} - \overline{\mathbf{y}}^i\right) + \mathcal{O}(\Delta t^6), \tag{62}$$

with the coefficients given by,

$$\begin{aligned} \frac{\partial^+ \mathbf{z}_k^{1,p}}{\partial \mathbf{w}^{1,p}} &= -\Delta t \hat{\sigma}(\mathbf{c}^{1,p}) \sigma'(\mathbf{A}_{k-1}^{1,p}) \mathbf{y}_{k-1}^{1,p}, \\ \bar{\mathbf{G}}_{2i-1,2p-1} = \sum_{j=1}^m \mathbf{G}_{ij}^3 \mathbf{G}_{jp}^2, \quad \forall\, 1 \leq i \leq m, \quad \mathbf{G}_{r,s}^q &= -(\hat{\sigma}(\mathbf{c}^{q,r}))^2 \sigma'\left(\mathbf{A}_{n-1}^{q,r}\right) \mathbf{V}_{rs}^q, \quad q = 2, 3. \end{aligned} \tag{63}$$

Proof. Following the definition (44) and as $L = 3$ and $\theta = \mathbf{w}^{1,p}$, we have,

$$\frac{\partial \mathcal{E}_{k,1}^{(n,3)}}{\partial \mathbf{w}^{1,p}} := \frac{\partial \mathcal{E}_n}{\partial \mathbf{X}_n^3} \frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_k^1} \frac{\partial^+ \mathbf{X}_k^1}{\partial \mathbf{w}^{1,p}}. \tag{64}$$

We will explicitly compute all three expressions on the right-hand-side of (64).

In (48), we have already explicitly computed the right most expression in the RHS of (64). Using the product rule (19) we have,

$$\frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_k^1} = \frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_n^2} \frac{\partial \mathbf{X}_n^2}{\partial \mathbf{X}_n^1} \prod_{j=k+1}^n \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1}. \tag{65}$$

Note that we have already obtained an explicit representation formula for $\prod_{j=k+1}^n \frac{\partial \mathbf{X}_j^1}{\partial \mathbf{X}_{j-1}^1}$ in (57).

Next we consider the matrices $\frac{\partial \mathbf{X}_n^3}{\partial \mathbf{X}_n^2}$ and $\frac{\partial \mathbf{X}_n^2}{\partial \mathbf{X}_n^1}$. By the representation formula (22), we have the following decomposition for any $1 \leq q \leq n$,

$$\frac{\partial \mathbf{X}_n^q}{\partial \mathbf{X}_n^{q-1}} = \Delta t^2 \mathbf{G}^{q,n} + \Delta t H^{q,n}, \tag{66}$$

with,

$$\mathbf{G}^{q,n} = \begin{bmatrix} \mathbf{G}_{11}^{q,n} & 0 & \mathbf{G}_{12}^{q,n} & 0 & \ldots & \ldots & \mathbf{G}_{1m}^{q,n} & 0 \\ 0 & 0 & 0 & 0 & \ldots & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \mathbf{G}_{m1}^{q,n} & 0 & \mathbf{G}_{m2}^{q,n} & 0 & \ldots & \ldots & \mathbf{G}_{mm}^{q,n} & 0 \\ 0 & 0 & 0 & 0 & \ldots & \ldots & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_{i,\bar{i}}^{q,k} = -(\hat{\sigma}(\mathbf{c}^{q,i}))^2 \sigma'\left(\mathbf{A}_{n-1}^{q,i}\right) \mathbf{V}_{i\bar{i}}^q, \tag{67}$$

and

$$\mathbf{H}^{q,n} = \begin{bmatrix} 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \mathbf{H}^{q,n}_{11} & 0 & \mathbf{H}^{q,n}_{12} & 0 & \dots & \dots & \mathbf{H}^{q,n}_{1m} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \dots & 0 & 0 \\ \mathbf{H}^{q,n}_{m1} & 0 & \mathbf{H}^{q,n}_{m2} & 0 & \dots & \dots & \mathbf{H}^{q,n}_{mm} & 0 \end{bmatrix}, \quad \mathbf{H}^{q,k}_{i,\bar{i}} = -\hat{\sigma}(\mathbf{c}^{q,i})\sigma'\left(\mathbf{A}^{q,i}_{n-1}\right)\mathbf{V}^{q}_{i\bar{i}}. \tag{68}$$

It is straightforward to see from (68) and (67) that,

$$\mathbf{H}^{3,n}\mathbf{H}^{2,n} \equiv \mathbf{0}_{2m\times 2m}, \quad \mathbf{G}^{3,n}\mathbf{H}^{2,n} \equiv \mathbf{0}_{2m\times 2m}, \tag{69}$$

and the entries of the $2m \times 2m$ matrix $\bar{\mathbf{G}} = \mathbf{G}^{3,n}\mathbf{G}^{2,n}$ are given by,

$$\bar{\mathbf{G}}_{2r-1,2s-1} = \sum_{j=1}^{m} \mathbf{G}^{3,n}_{r,j}\mathbf{G}^{2,n}_{j,s}, \quad \bar{\mathbf{G}}_{2r-1,2s} = \bar{\mathbf{G}}_{2r,2s-1} = \bar{\mathbf{G}}_{2r,2s} = 0, \quad \forall\, 1 \le r,s \le m, \tag{70}$$

while the entries of the $2m \times 2m$ matrix $\bar{\mathbf{H}} = \mathbf{H}^{3,n}\mathbf{G}^{2,n}$ are given by

$$\bar{\mathbf{H}}_{2r,2s-1} = \sum_{j=1}^{m} \mathbf{H}^{3,n}_{r,j}\mathbf{G}^{2,n}_{j,s}, \quad \bar{\mathbf{H}}_{2r-1,2s-1} = \bar{\mathbf{H}}_{2r-1,2s} = \bar{\mathbf{H}}_{2r,2s} = 0, \quad \forall\, 1 \le r,s \le m. \tag{71}$$

Hence we have,

$$\frac{\partial \mathbf{X}^3_n}{\partial \mathbf{X}^2_n}\frac{\partial \mathbf{X}^2_n}{\partial \mathbf{X}^1_n} = \Delta t^4(\bar{\mathbf{G}} + \Delta t^{-1}\bar{\mathbf{H}}). \tag{72}$$

Taking the matrix-vector product of (72) with (60), we obtain

$$\begin{aligned} \frac{\partial \mathbf{X}^3_n}{\partial \mathbf{X}^1_k}\frac{\partial^+ \mathbf{X}^1_k}{\partial \mathbf{w}^{1,p}} &= \Delta t^4 \left(\frac{\partial^+ \mathbf{y}^{1,p}_k}{\partial \mathbf{w}^{1,p}} + \mathbf{C}^p_{12}\frac{\partial^+ \mathbf{z}^{1,p}_k}{\partial \mathbf{w}^{1,p}}\right)\left[\bar{\mathbf{G}}_{1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2,2p-1}, \dots, \bar{\mathbf{G}}_{2m-1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2m,2p-1}\right]^\top + \mathcal{O}(\Delta t^6) \\ &= \Delta t^4 \mathbf{C}^p_{12}\frac{\partial^+ \mathbf{z}^{1,p}_k}{\partial \mathbf{w}^{1,p}}\left[\bar{\mathbf{G}}_{1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2,2p-1}, \dots, \bar{\mathbf{G}}_{2m-1,2p-1}, \Delta t^{-1}\bar{\mathbf{H}}_{2m,2p-1}\right]^\top + \mathcal{O}(\Delta t^6), \end{aligned} \tag{73}$$

where the last identify follows from the fact that $\frac{\partial^+ \mathbf{y}^{1,p}_k}{\partial \mathbf{w}^{1,p}} = \mathcal{O}(\Delta t^2)$.

Therefore, taking the products of (61) and (73), we obtain the desired identity (62).

□

An inspection of the representation formula (62) shows that as long as the weights are $\mathcal{O}(1)$ and from the bounds (7), we know that $\mathbf{y} \sim \mathcal{O}(1)$, the gradient

$$\frac{\partial \mathcal{E}^{(n,3)}_{k,1}}{\partial \mathbf{w}^{1,p}} \sim \mathcal{O}(\Delta t^5),$$

where the additional Δt stems from the Δt-term in (48). Thus the gradient does not depend on the recurrent step k. Hence, there is no vanishing gradient problem with respect to the number of recurrent connections, even in the multi-layer case.

However, it is clear from the representation formulas (46) and (62), as well as the proof of proposition C.4 that for L-hidden layers in UnICORNN (6), we have,

$$\frac{\partial \mathcal{E}^{(n,L)}_{k,1}}{\partial \mathbf{w}^{1,p}} \sim \mathcal{O}\left(\Delta t^{2L-1}\right). \tag{74}$$

Thus, the gradient can become very small if too many layers are stacked together. This is not at all surprising as such a behavior occurs even if there are no recurrent connections in UnICORNN (6). In that case, we simply have a fully connected deep neural network and it is well-known that the gradient can vanish as the number of layers increases, making it harder to train deep networks.

C.5. Residual stacking of layers in UnICORNN.

Given the above considerations, it makes imminent sense to modify the fully-connected stacking of layers in UnICORNN (6) if a moderately large number of layers ($L \geq 4$) are used. It is natural to modify the fully-connected stacking with a residual stacking, see (Li et al., 2019). We use the following form of residual stacking,

$$\mathbf{y}_n^\ell = \mathbf{y}_{n-1}^\ell + \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot \mathbf{z}_n^\ell, \tag{75}$$

$$\mathbf{z}_n^\ell = \mathbf{z}_{n-1}^\ell - \Delta t \hat{\sigma}(\mathbf{c}^\ell) \odot [\sigma\left(\mathbf{w}^\ell \odot \mathbf{y}_{n-1}^\ell + \mathbf{x}_n^\ell + \mathbf{b}^l\right) + \alpha \mathbf{y}_{n-1}^\ell], \tag{76}$$

where the input $\mathbf{x}_n^\ell$ corresponds to a residual connection skipping S layers, i.e.

$$\mathbf{x}_n^\ell = \begin{cases} \mathbf{\Lambda}^\ell \mathbf{y}_n^{\ell-S-1} + \mathbf{V}^\ell \mathbf{y}_n^{\ell-1}, & \text{for } l > S \\ \mathbf{V}^\ell \mathbf{y}_n^{\ell-1}, & \text{for } l \leq S \end{cases}.$$

The number of skipped layers is $2 \leq S$ and $\Lambda^\ell \in \mathbb{R}^{m \times m}$ is a trainable matrix.

The main advantages of using a residual staking such as (75) is to alleviate the vanishing gradient problem that arises from stacking multiple layers together and obtain a better scaling of the gradient than (74). To see this, we can readily follow the proof of proposition C.4, in particular the product,

$$\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_n^1} = \prod_{s=1}^{\nu} \frac{\partial \mathbf{X}_n^{L-(s-1)S}}{\partial \mathbf{X}_k^{L-sS}} \prod_{\ell=2}^{L-\nu S} \frac{\partial \mathbf{X}_n^\ell}{\partial \mathbf{X}_n^{\ell-1}} + \prod_{\ell=1}^{L-1} \frac{\partial \mathbf{X}_n^{\ell+1}}{\partial \mathbf{X}_k^\ell}, \tag{77}$$

with,

$$\nu = \begin{cases} \left[\frac{L}{S}\right], & \text{if} \quad L \bmod S \neq 0, \\ \left[\frac{L}{S}\right] - 1, & \text{if} \quad L \bmod S = 0. \end{cases} \tag{78}$$

Here $[x] \in \mathbb{N}$ is the largest natural number less than or equal to $x \in \mathbb{R}$.

Given the additive structure in the product of gradients and using induction over matrix products as in (69) and (70), we can compute that,

$$\frac{\partial \mathbf{X}_n^L}{\partial \mathbf{X}_n^1} = \mathcal{O}\left(\Delta t^{2(\nu+L-\nu S-1)}\right) + \mathcal{O}\left(\Delta t^{2(L-1)}\right). \tag{79}$$

By choosing S large enough, we clearly obtain that $\nu + L - \nu S - 1 < L - 1$. Hence by repeating the arguments of the proof of proposition C.4, we obtain that to leading order, the gradient of the residual stacked version of UnICORNN scales like,

$$\frac{\partial \mathcal{E}_{k,1}^{(n,L)}}{\partial \mathbf{w}^{1,p}} \sim \mathcal{O}\left(\Delta t^{2\nu+2L-2\nu S-1}\right). \tag{80}$$

Note that (80) is far more favorable scaling for the gradient than the scaling (74) for a fully connected stacking. As a concrete example, let us consider $L = 7$ i.e., a network of 7 stacked layers of UniCORNN. From (74), we see that the gradient scales like $\mathcal{O}(\Delta t^{13})$ in this case. Even for a very moderate values of $\Delta t < 1$, this gradient will be very small and will ensure that the first layer will have very little, if any, influence on the loss function gradients. On the other hand, for the same number of layers $L = 7$, let us consider the residual stacking (75) with $S = 3$ skipped connections. In this case $\nu = 2$ and one directly concludes from (80) that the gradient scales like $\mathcal{O}(\Delta t^5)$, which is significantly larger than the gradient for the fully connected version of UnICORNN. In fact, it is exactly the same as the gradient scaling for fully connected UnICORNN (6) with 3 hidden layers (62). Thus, introducing skipped connections enabled the gradient to behave like a shallower fully-connected network, while possibly showing the expressivity of a deeper network.

D. Chaotic time-series prediction: Lorenz 96 system

It is instructive to explore limitations of the proposed UnICORNN. It is straightforward to prove, along the lines of the proof of proposition C.1, that the UnICORNN architecture does not exhibit chaotic behavior with respect to changes in the input. While this property is highly desirable for many applications where a slight change in the input should not lead to a major

(possibly unbounded) change in the output, it impairs the performance on tasks where an actual chaotic system has to be learned.

Following the experiment in (Rusch & Mishra, 2021), we aim to predict future states of a dynamical system, following the Lorenz 96 system (Lorenz, 1996):

$$x_j' = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F, \tag{81}$$

where $x_j \in \mathbb{R}$ for all $j = 1, \dots, 5$ and F is an external force controlling the level of chaos in the system.

We consider two different choices for the external force, namely $F = 0.9$ and $F = 8$. While the first one produces non-chaotic trajectories, the latter leads to a highly chaotic system. We discretize the system exactly along the lines of (Rusch & Mishra, 2021), resulting in 128 sequences of length 2000 for each the training, validation and testing set. Table 7

Table 7. Test NRMSE on the Lorenz 96 system (81) for UnICORNN, coRNN and LSTM.

Model	$F = 0.9$	$F = 8$	# units	# params
LSTM (Rusch & Mishra, 2021)	2.0×10^{-2}	6.8×10^{-2}	44	9k
coRNN (Rusch & Mishra, 2021)	2.0×10^{-2}	9.8×10^{-2}	64	9k
UnICORNN (L=2)	2.2×10^{-2}	3.1×10^{-1}	90	9k

shows the normalized root mean square error (NRMSE) for UnICORNN as well as for coRNN and an LSTM, where all models have 9k parameters. We can see that UnICORNN performs comparably to coRNN and LSTM in the chaos-free regime (i.e. $F = 0.9$), while performing poorly compared to an LSTM when the system exhibits chaotic behavior (i.e. $F = 8$). This is not surprising, as LSTMs are shown to be able to exhibit chaotic behavior (Laurent & von Brecht, 2017), while coRNN and UnICORNN are not chaotic by design. This shows also numerically that UnICORNN should not be used for chaotic time-series prediction.

E. Further experimental results

As we compare the results of the UnICORNN to the results of other recent RNN architecture, where only the best results of each RNN were published for the psMNIST, noise padded CIFAR-10 and IMDB task, we as well show the best (based on a validation set) obtained results for the UnICORNN in the main paper. However, distributional results, i.e. statistics of several re-trainings of the best performing UnICORNN based on different random initialization of the trainable parameters, provide additional insights into the performance. Table 8 shows the mean and standard deviation of 10 re-trainings of the best performing UnICORNN for the psMNIST, noise padded CIFAR-10 and IMDB task. We can see that in all experiments the standard deviation of the re-trainings are relatively low, which underlines the robustness of our presented results.

Table 8. Distributional information (mean and standard deviation) on the results for the classification experiment presented in the paper, where only the best results is shown, based on 10 re-trainings of the best performing UnICORNN using different random seeds.

Experiment	Mean	Standard deviation
psMNIST (128 units)	97.7%	0.09%
psMNIST (256 units)	98.2%	0.22%
Noise padded CIFAR-10	61.5%	0.52%
IMDB	88.1%	0.19%

As emphasized in the main paper and in the last section, naively stacking of many layers for the UnICORNN might result in a vanishing gradient for the deep multi-layer model, due to the vanishing gradient problem of stacking many (not necessarily recurrent) layers. Following section C.5, one can use skipped residual connections and we see that the estimate on the gradients scale preferably when using residual connections compared to a naively stacking, when using many layers. To test this also numerically, we train a standard UnICORNN (6) as well as a residual UnICORNN (res-UnICORNN) (75), with $S = 2$ skipping layers, on the noise padded CIFAR-10 task. Fig. 6 shows the test accuracy (mean and standard deviation) of the best resulting model for different number of network layers $L = 3, \dots, 6$, for the standard UnICORNN and res-UnICORNN. We can see that while both models seem to perform comparably for using only few layers, i.e. $L = 3, 4$,

the res-UnICORNN with $S = 2$ skipping connections outperforms the standard UnICORNN when using more layers, i.e. $L = 5, 6$. In particular, we can see that the standard UnICORNN is not able to significantly improve the test accuracy when using more layers, while the res-UnICORNN seems to obtain higher test accuracies when using more layers.

Moreover, Fig. 6 also shows the test accuracy for a UnICORNN with an untrained time-step vector $\mathbf{c}$, resulting in a UnICORNN without the multi-scale property generated by the time-step. We can see that the UnICORNN without the multi-scale feature is inferior in performance, to the standard UnICORNN as well as its residual counterpart.

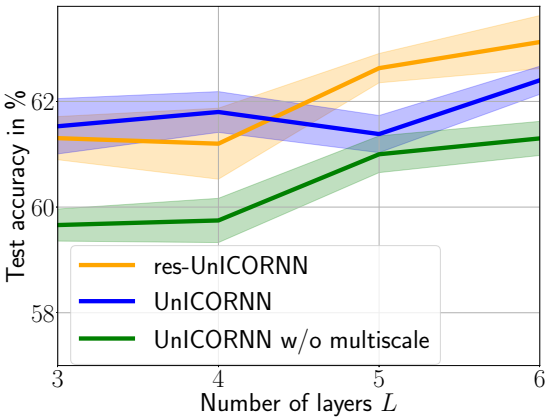

Figure 6. Test accuracies (mean and standard deviation of 10 re-trainings of the best performing model) of the standard UnICORNN, res-UnICORNN and UnICORNN without multi-scale behavior on the noise padded CIFAR-10 experiment for different number of layers L.

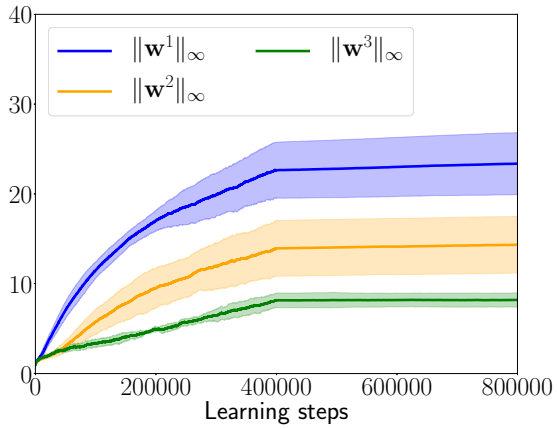

Figure 7. Norms (mean and standard deviation of 10 re-trainings) of the hidden weights $\|\mathbf{w}^l\|_\infty$, for $l = 1, 2, 3$, of the UnICORNN during training.

Finally, we recall that the estimate (14) on the gradients for UnICORNN (6) needs the weights to be bounded, see (15). One always initializes the training with bounded weights. However, it might happen that the weights explode during training. To check this issue, in Fig. 7, we plot the mean and standard deviation of the norms of the hidden weights $\mathbf{w}^l$ for $l = 1, 2, 3$ during training based on 10 re-trainings of the best performing UnICORNN on the noise padded CIFAR-10 experiment. We can see that none of the norms of the weights explode during training. In fact the weight norms seem to saturate, mostly on account of reducing the learning rate after 250 epochs. Thus, the upper bound (14) can be explicitly computed and it is finite, even after training has concluded.