Mixed-Memory RNNs for Learning Long-term Dependencies in Irregularly-sampled Time Series

Mathias Lechner and Ramin Hasani
CSAIL, MIT
Cambridge, MA 02139
mlechner, rhasani @mit.edu

Abstract

Recurrent neural networks (RNNs) with continuous-time hidden states are a natural fit for modeling irregularly-sampled time series. These models, however, face difficulties when the input data possess long-term dependencies. We show that similar to standard RNNs, the underlying reason for this issue is the vanishing or exploding of the gradient during training. This phenomenon is expressed by the ordinary differential equation (ODE) representation of the hidden state, regardless of the ODE solver's choice. We provide a solution by equipping arbitrary continuous-time networks with a memory compartment separated from their time-continuous state. This way, we encode a continuous-time dynamic flow within the RNN, allowing it to respond to inputs arriving at arbitrary time lags while ensuring a constant error propagation through the memory path. We call these models Mixed-Memory-RNNs (mmRNNs). We experimentally show that Mixed-Memory-RNNs outperform recently proposed RNN-based counterparts on non-uniformly sampled data with long-term dependencies.

1 Introduction

Irregularly-sampled time series, routine data streams in medical and business settings, can be modeled effectively by a time-continuous version of recurrent neural networks (RNNs). Instead of explicitly defining a state update function, these ODE-RNNs identify an ordinary differential equation with no exogenous inputs [53] or with inputs [19, 36, 34] in the following form [19]:

$$\frac{\partial h}{\partial t} = f_\theta(x_{t+T}, h_t, T) - \tau h, \tag{1}$$

where x_t is the input sequence, h_t is an RNN's hidden state, and τ is an optional dampening factor. The time-lag T specifies at what times the inputs x_t have been sampled. ODE-RNNs were recently rediscovered [53] and have shown promise in approximating irregularly-sampled data, thanks to the implicit definition of *time* in their resulting dynamical systems. ODE-RNNs can be trained by backpropagation through time (BPTT) [54, 62, 63] through ODE solvers or by treating the solver as a black-box and applying the adjoint method [50] to gain memory efficiency [9].

In this work, we first show that ODE-RNNs tend to suffer from vanishing/exploding gradients [29, 5], regardless of whether they are trained by reverse-mode automatic differentiation or the adjoint method [54, 50]. Consequently, ODE-RNNs are unable to learn long-term causal dependencies in the training data. Moreover, we show that the vanishing and exploding gradient problem persists even when the ODE is combined with a long short term memory network (LSTM) [30] or gated recurrent unit (GRU) [10] in the form of an ODE-LSTM or ODE-GRU. In particular, although the LSTM and GRU resolve the vanishing gradient problem in regularly sampled data by enforcing a near-constant error propagation through the hidden states, the presence of ODE dynamics in their inference pathway disrupts the near-constant error propagation.

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

In this paper, we also show that to simultaneously enjoy the rich modeling capability of ODE-RNNs (or any other time-continuous network) with the capability of learning long-term dependencies requires two separated signal pathways: one for the time-continuous ODE dynamics and one for the long-term memory transport. We call networks that satisfy this pattern mixed-memory RNNs (mmRNNs). We compare mmRNNs to standard and advanced continuous-time RNN variants on a set of synthetic and real-world sparse time-series tasks and discover consistently better performance.

2 Related works

Learning Irregularly-Sampled Data Statistical [49, 41, 4, 52] and functional analysis [17, 2, 35] tools have long been studying non-uniformly-spaced data. A natural fit for this problem is the use of time-continuous RNNs [53]. We showed that although ODE-RNNs are performant models in these domains, their performance tremendously drops when the incoming samples have long-range dependencies. We solved this shortcoming by introducing mmRNNs.

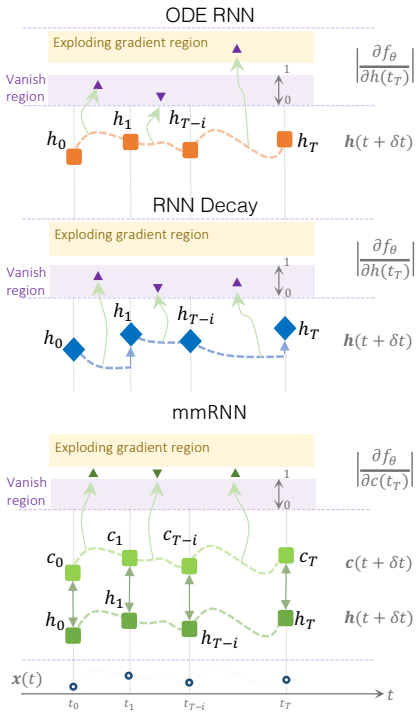

Figure 1: Error propagation in time-continuous RNNs express vanishing gradients (first two models). mmRNNs avoid these phenomena by separating time-continuous and memory pathways.

Learning Long-term Dependencies The notorious question of vanishing/exploding gradient [29, 5] was identified as the core reason for RNNs' lack of generalizability when trained by gradient descent [1, 58]. Recent studies used state-regularization [61] and long memory stochastic processes [21] to analyze long-range dependencies. Apart from the original LSTM model [30] and its variants [22] that solve the problem in the context of RNNs, very few alternative researches exist [8].

As the class of CT RNNs become steadily popularized [27, 37], it is important to characterize them better [39, 14, 15] and understand their applicability and limitations [32, 38, 26, 31, 51, 34, 28]. We proposed a method to enable ODE-based RNNs to learn long-term dependencies.

Further details and related works on time-continuous recurrent neural network architectures can be found in section A.

3 ODE-RNNs tend to suffer from vanishing or exploding gradient.

Hochreiter [29] discovered that the error-flow in the BPTT algorithm follows a power series that determines the effectiveness of the learning process [29, 30, 5, 48]. In particular, the state-to-previous-state Jacobian of an RNN:

$$\frac{\partial h_{t+T}(x_{t+T}, h_t, T)}{\partial h_t}, \tag{2}$$

governs whether the propagated error exponentially grows (explodes), exponentially vanishes, or stays constant.

Formally:

Definition 1 (Vanishing or exploding gradient)**.** *A recurrent neural network suffers from a vanishing or exploding gradient if its state-to-previous-state Jacobian* $\frac{\partial h_{t+T}}{\partial h_t}$ *has eigenvalues with absolute value less than 1 (vanishing) or greater than 1 (exploding).*

We say an RNN *tends* to suffer from a vanishing or exploding gradient if there is no functional mechanism that prevents the RNN from expressing a vanishing or exploding gradient during parts of the training.

Now, consider an ODE-RNN given by Eq. 1 is implemented either by an Explicit Euler discretization or by a Runge-Kutta method [55, 12]. We can formulate their state-previous state Jacobian in the following two lemmas:

Lemma 1. *For an ODE-RNN defined in Eq.1, the state-to-previous-state Jacobian of the explicit Euler is given by*

$$\frac{\partial h_{t+T}}{\partial h_t} = I + T\frac{\partial f}{\partial h}\Big|_{h=h_t} - \tau T I,$$

, with eigenvalues $1 - \tau T + Eig(T\frac{\partial f}{\partial h}\Big|_{h=h_t})$.

Lemma 2. *For an ODE-RNN defined in Eq.1, the state-to-previous-state Jacobian of the Runge-Kutta method is given by*

$$\frac{\partial h_{t+T}}{\partial h_t} = I + T\sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_i} - \tau T I,$$

with with eigenvalues $1 - \tau T + Eig(T\sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_i})$*, where* $\sum_{j=1}^{M} b_i = 1$ *and some* K_i.

The derivations are provided in Appendix B. Consequently, we have:

Theorem 1 (Informal)**. (ODE-RNNs tend to suffer from a vanish or exploding gradient)** *An ODE-RNN with uniformly Lipschitz continuous dynamics defined in Eq.1 simulated by an explicit Euler or Runge-Kutta method tends to suffer from vanishing and exploding gradients.*

The argument is given in full in Appendix B. In essence, a dynamics of $\frac{\partial f}{\partial h} - \tau = 0$ would enforce a constant error propagation by making the Jacobians equal to the identity. However, it also removes any meaningful dynamics from the ODE, as it would operate the ODE as a memory element. Intuitively, any interesting function f_θ pushes the eigenvalues away from a stable orbit, creating a vanishing or exploding gradient depending on f_θ.

Note that the Theorem 1 applies independently of the used RNN architecture. In particular, we can entail the following statement:

Corollary 1. *ODE-combined-with-GRU and ODE-combined-with-LSTM as presented in [53] (and defined in the row "ODE-RNN" in Table 1), also tend to suffer from vanish or exploding gradients.*

The derivation follows by applying the Theorems 1 with a GRU and LSTM. In Appendix B we also provide arguments for the vanishing and exploding gradient effects in ODE-RNNs regardless of ODE-solver choice and training mode (e.g., BPTT or adjoint method).

4 Mixed-Memory Recurrent Architectures

Instead of having a single state vector h_t that is processed by the discrete RNN and the time-continuous ODE, our Mixed-Memory architecture represents its hidden state by a pair (c_t, h_t). An update of the form

$$c_{t+1} = c_t \odot \sigma(g_\theta(h_t) + b_f) + z_\theta(h_t), \tag{3}$$

governs the memory cell component c_t of the mmRNN, where g_θ and z_θ are learnable gating functions and b_f a bias term. Although the update in Eq. (3) appears similar to that of a vanilla LSTM and GRU, there is one key difference: The gates of an LSTM/GRU are controlled by its previous hidden state, whereas the gating of an mmRNN is defined by a second process h_t, e.g., a neural ODE. Particularly, the update function for h_t is of the form

$$h_{t+1} = \text{ODE-Solve}(f_\theta, c_t, T), \tag{4}$$

where f_θ is the derivative of the state, c_t the initial state, and T the solving time.

The fundamental distinction of mmRNN from other continuous-time RNNs is the strict separation of memory and time-continuous hidden states. In particular, the memory update in Eq. (4) ensures a constant error propagation through c_t, while arbitrary Neural ODEs process the state h_t in a time-continuous fashion. Table 1 lists how the transition of the hidden states between two observations of the mmRNN differs from other architectures. Algorithm 1 in the Appendix implements an mmRNN.

mmRNNs allow controlling the vanishing gradient Initializing the weights of g_θ and z_θ close to 0 avoids the units c_t of the state pair (c_t, h_t) from the suffering of vanishing or exploding gradient at the beginning of the training process. Specifically, if g_θ and z_θ are close to 0 we can neglect them and get $\left|\sum_{j=1}^{N} \frac{\partial c_{t+T}^i(x_{t+T},(c_t,h_t),T)}{\partial c_t^j}\right| = \sigma(b_f) \approx 0.9943$, allowing a near-constant error propagation. For further details see Appendix B.

Empirical measurement of gradient norms. To emphasize the impact of our theoretical results, we performed an experiment comparing the hidden state gradient norms of an ODE-RNN instance compared to another ODE-RNN equipped with mmRNN for different sequence lengths (mean/std over three initialization seeds). The results show (as highlighted in Theorem 1 and 2) that the gradients of a standard ODE-RNN tend to exponentially increase with the length of the input sequence. This gradient issue is significantly improved when mmRNNs are used (See Table 2).

Model	State between observation
Standard RNN	h_t
GRU-D	$h_t e^{-T\tau}$
ODE-RNN	ODE-Solve(f_θ, h_t, T)
mmRNN	$\big(c_t, \text{ODE-Solve}(f_\theta, c_t, T)\big)$

Table 1: Change to the hidden states of an RNN between two observations t and $t+T$

5 Experimental evaluation

We perform an experimental evaluation to assess the generalization of time-continuous RNN architectures on datasets that are deliberately selected to express long-term dependencies and are of irregularly-sampled nature.

Baselines. We compare mmRNN to a large variety of continuous-time RNNs, including among others ODE-RNN [53] and CT-RNNs [19], and GRU-D [7]. An exhaustive list of evaluated RNN architectures for irregularly sampled time series can be found in Appendix A, and further details on the experimental settings are provided in Appendix C.

Sequence length	ODE-RNN	mmRNN
5	10.50 ± 5.47	0.97 ± 0.01
10	38.75 ± 33.20	0.96 ± 0.03
25	235.76 ± 383.04	1.01 ± 0.10
50	1214.94 ± 3750.62	1.25 ± 0.25

Table 2: Gradient Norms ODE-RNN vs mmRNN. (n =3)

Synthetic benchmark - Bit-stream sequence classification Our first two experiments concern a synthetic sequence classification task in the form of a time-series adaptation of the XOR problem [43]. We create two variants of the problem, one where the input is represented by a regular bit-after-bit sequence and an event-based encoding where the bit-stream is run-length encoded (e.g., 10001111 is represented as the three events 1T1.0T3.1T4).

The results in Table 3 show that, while most models can learn the correct function for the regularly coded bit-streams, all models struggle with the run-length encoded irregular data representation. Nonetheless, our mmRNN showed the best performance on the event-based data representation.

Algorithm 1 The mixed-memory RNN

Input: Datapoints and their timestamps $\{(x_t, t_i)\}_{i=1\dots N}$
Parameters: Weights θ, output weight and bias W_{output}, b_{output}
$h_0 = \mathbf{0}$ {ODE state}
$c_0 = \mathbf{0}$ {Memory cell}
for $i = 1 \dots N$ **do**
  $c_i = c_{i-1} \odot \sigma(g_\theta(h_{i-1}, x_i) + b_f) + z_\theta(h_{i-1}, x_i)$ {Memory cell update}
  $h_i = \text{ODESolve}(f_\theta, c_i, t_i - t_{i-1})$ {Time-continuous state update}
  $o_i = h_i W_{output} + b_{output}$
end for
Return $\{o_i\}_{i=1\dots N}$

Table 3: Test accuracy of our experimental evaluation (mean $\pm$ std, $N = 5$).

Model	Bit-stream classification		Real-world datasets	
	Dense encoding	Event-based encoding	Person Activity	Runlength-coded sMNIST
ODE-RNN [53]	50.47% $\pm$ 0.06	51.21% $\pm$ 0.37	80.43% $\pm$ 1.55	72.41% $\pm$ 1.69
CT-RNN [19]	50.42% $\pm$ 0.12	50.79% $\pm$ 0.34	83.65% $\pm$ 1.55	72.05% $\pm$ 0.71
Augmented LSTM [30]	**100.00% $\pm$ 0.00**	89.71% $\pm$ 3.48	**84.11% $\pm$ 0.68**	82.10% $\pm$ 4.36
CT-GRU [46]	**100.00% $\pm$ 0.00**	61.36% $\pm$ 4.87	79.48% $\pm$ 2.12	87.51% $\pm$ 1.57
RNN Decay [53]	60.28% $\pm$ 19.87	75.53% $\pm$ 5.28	62.89% $\pm$ 3.87	88.93% $\pm$ 4.06
Reciprocal RNN [3]	**100.00% $\pm$ 0.00**	90.17% $\pm$ 0.69	**83.85% $\pm$ 0.45**	94.43% $\pm$ 0.23
GRU-D [7]	**100.00% $\pm$ 0.00**	97.90% $\pm$ 1.71	83.57% $\pm$ 0.40	95.44% $\pm$ 0.34
PhasedLSTM [47]	50.99% $\pm$ 0.76	80.29% $\pm$ 0.99	83.33% $\pm$ 0.69	86.79% $\pm$ 1.57
GRU-ODE [53]	50.41% $\pm$ 0.40	52.52% $\pm$ 0.35	82.56% $\pm$ 2.63	80.95% $\pm$ 1.52
CT-LSTM [44]	97.73% $\pm$ 0.08	95.09% $\pm$ 0.30	**84.13% $\pm$ 0.11**	94.84% $\pm$ 0.17
iRNN [33]	49.99% $\pm$ 1.20	50.54% $\pm$ 0.94	74.56% $\pm$ 1.29	95.51% $\pm$ 1.95
coRNN [56]	**100.00% $\pm$ 0.00**	52.89% $\pm$ 1.25	78.89% $\pm$ 0.62	94.44% $\pm$ 0.24
Lipschitz RNN [16]	**100.00% $\pm$ 0.00**	52.84% $\pm$ 3.25	81.35% $\pm$ 0.60	95.92% $\pm$ 0.16
mmRNN (ours)	**100.00% $\pm$ 0.00**	**98.89% $\pm$ 0.26**	**84.15% $\pm$ 0.33**	**97.83% $\pm$ 0.37**

Real-world task - Person activity Our first irregularly sampled real-world dataset consider the person activity recognition dataset from the UCI repository [13]. The data capture inertial measurement sensors worn by a person who performs certain physical activities (e.g., walking, sitting). The objective of the ML model is to classify the correct activity from the inertial observations.

Real-world task - Run-length encoded sequential MNIST Our final evaluation concerns the sequential MNIST task. However, to make the dataset irregularly sampled, we performed a binarization of the sequence (with threshold 127/255), followed by a run-length encoding of the resulting bit-stream.

The results in Table 3 show that most time-continuous architectures achieve a good performance on the person activity but struggle with the run-length encoded sMNIST task with respect to one standard deviation of the best performing model. We hypothesize that the person activity dataset expresses much fewer long-term dependencies, which makes a vanishing gradient less of a problem here. Most notably, the mmRNN showed the best performance in both tasks, followed by GRU-D and the CT-LSTM, demonstrating that the mixed architecture is able to learn long-term dependencies and rich dynamic models simultaneously.

6 Discussions, Scope and Limitations

We proposed a solution to learn long-term dependencies in irregularly-sampled input data streams. To perform this, we designed a novel long short-term memory network that possesses a continuous-time output state and consequently modifies its internal dynamical flow to a continuous-time model. mmRNNs resolve the vanishing and exploding of the gradient problem of the class of ODE-RNNs while demonstrating an attractive performance in learning long-term dependencies on data arriving at non-uniform intervals.

References

[1] Zeyuan Allen-Zhu and Yuanzhi Li. Can sgd learn recurrent neural networks with provable generalization? In *Advances in Neural Information Processing Systems*, pages 10331–10341, 2019.

[2] José M Amigó, Roberto Monetti, Thomas Aschenbrenner, and Wolfram Bunk. Transcripts: An algebraic approach to coupled time series. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(1):013105, 2012.

[3] Sepideh Babaei, Amir Geranmayeh, and Seyyed Ali Seyyedsalehi. Protein secondary structure prediction using modular reciprocal bidirectional recurrent neural networks. *Computer methods and programs in biomedicine*, 100(3):237–247, 2010.

[4] Francois W Belletti, Evan R Sparks, Michael J Franklin, Alexandre M Bayen, and Joseph E Gonzalez. Scalable linear causal inference for irregularly sampled time series with long range dependencies. *arXiv preprint arXiv:1603.03336*, 2016.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[7] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

[8] Dexiong Chen, Laurent Jacob, and Julien Mairal. Recurrent kernel networks. In *Advances in Neural Information Processing Systems*, pages 13431–13442, 2019.

[9] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

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

[11] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, pages 7377–7388, 2019.

[12] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[13] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[14] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

[15] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, pages 7509–7520, 2019.

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

[17] Grant Foster. Wavelets for period analysis of unevenly sampled time series. *The Astronomical Journal*, 112:1709, 1996.

[18] Peter K Friz and Nicolas B Victoir. *Multidimensional stochastic processes as rough paths: theory and applications*, volume 120. Cambridge University Press, 2010.

[19] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[20] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

[21] Alexander Greaves-Tunnell and Zaid Harchaoui. A statistical investigation of long memory in language and music. In *International Conference on Machine Learning*, pages 2394–2403, 2019.

[22] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[23] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Re. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33, 2020.

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

[25] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] YAN Hanshu, DU Jiawei, TAN Vincent, and FENG Jiashi. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2020.

[27] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. *arXiv preprint arXiv:2006.04439*, 2020.

[28] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. The natural lottery ticket winner: Reinforcement learning with ordinary neural circuits. In *Proceedings of the 2020 International Conference on Machine Learning*. JMLR. org, 2020.

[29] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen [in german] diploma thesis. *TU Münich*, 1991.

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

[31] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[32] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9843–9854, 2019.

[33] Anil Kag, Ziming Zhang, and Venkatesh Saligrama. Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*, 2019.

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

[35] Daniel R Kowal, David S Matteson, and David Ruppert. Functional autoregression for sparsely sampled data. *Journal of Business & Economic Statistics*, 37(1):97–109, 2019.

[36] Mathias Lechner, Radu Grosu, and Ramin M Hasani. Worm-level control through search-based reinforcement learning. *arXiv preprint arXiv:1711.03467*, 2017.

[37] Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10):642–652, 2020.

[38] Mathias Lechner, Ramin Hasani, Daniela Rus, and Radu Grosu. Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme. In *2020 International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[39] Mathias Lechner, Ramin Hasani, Manuel Zimmer, Thomas A Henzinger, and Radu Grosu. Designing worm-inspired neural networks for interpretable robotic control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 87–94. IEEE, 2019.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[41] Steven Cheng-Xian Li and Benjamin M Marlin. A scalable end-to-end gaussian process adapter for irregularly sampled time series classification. In *Advances in neural information processing systems*, pages 1804–1812, 2016.

[42] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062, 2018.

[43] Minsky Marvin and A Papert Seymour. *Perceptrons*. MIT Press, 1969.

[44] Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, pages 6754–6764, 2017.

[45] James Morrill, Patrick Kidger, Cristopher Salvi, James Foster, and Terry Lyons. Neural cdes for long time series via the log-ode method. *arXiv preprint arXiv:2009.08295*, 2020.

[46] Michael C Mozer, Denis Kazakov, and Robert V Lindsey. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*, 2017.

[47] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in neural information processing systems*, pages 3882–3890, 2016.

[48] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[49] Ronald Pearson, Gregory Goney, and James Shwaber. Imbalanced clustering for microarray time-series. In *Proceedings of the ICML*, volume 3, 2003.

[50] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[51] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan KoutnÃk. Snode: Spectral discretization of neural odes for system identification. In *International Conference on Learning Representations*, 2020.

[52] DP Roy and L Yan. Robust landsat-based crop time series modelling. *Remote Sensing of Environment*, 238:110810, 2020.

[53] Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

[54] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[55] Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

[56] T. Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (co{rnn}): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2021.

[57] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[58] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[59] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[60] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[61] Cheng Wang and Mathias Niepert. State-regularized recurrent neural networks. In *International Conference on Machine Learning*, pages 6596–6606, 2019.

[62] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

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

[64] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. *Advances in neural information processing systems*, 29:4880–4888, 2016.

[65] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR 119, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] Public datasets
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Related Works and Baseline Architectures

Time-continuous RNNs The notion of *CT-RNNs* [19] was introduced around three decades ago. It is identical to the ODE-RNN architecture [53] with an additional dampening factor τ. In our experiments we evaluated several time-continuous modeling mechanisms such as ODE-RNN [53] and CT-RNNs [19], CT-GRU [46], RNN Decay [53], GRU-D [7], CT-LSTM [44], and GRU-D [7], in addition to oscillatory models such as Phased-LSTM [47], CoRNNs [56], iRNNs [33], and Lipschitz RNNs [16]. Furthermore, we tested mmRNNs against intuitive time-gap modeling approaches we built here, termed an augmented LSTM topology as well as reciprocal RNNs [3]. Below, we highlight each method in more detail.

GRU-D encodes the dependence on the time-lags by a trainable decaying mechanism, similar to *RNN-decay* [53]. While this mechanism enables modeling irregularly sampled time series, it also introduces a vanishing gradient factor to the backpropagation path.

Similarly, *CT-GRU* [46] adds multiple decay factors in the form of extra dimensions to the RNN state. An attention mechanism inside the CT-GRU then selects which entry along the decay dimension to use for computing the next state update. The CT-GRU aims to avoid vanishing gradients by including a decay rate of 0, i.e., no decay at all. This mechanism nevertheless fails, as illustrated in the bit-stream task in Table 3.

Phased-LSTM [47] adds a learnable oscillator to LSTM. The oscillator modulates LSTM to create dependencies on the elapsed time but also introduces a vanishing factor in its gradients.

GRU-ODE [11] modifies the GRU [10] topology by incorporating a continuous dynamical system. First, GRU is expressed as a discrete difference equation and then transformed into a continuous ODE. This process makes the error-propagation time-dependent, i.e., the near-constant error propagation property of GRU is abolished.

Lipschitz RNN [16] constraints the hidden-to-hidden weight matrix of a continuous-time RNN, such that the underlying dynamic system globally converges to a stable equilibrium. As a result, Lipschitz RNNs cannot suffer from an exploding gradient problem. The constraint of the weight matrix is realized efficiently using a symmetric skew decomposition [64].

Log-ODE method [45] compresses the input time-series by time-continuous path signatures [18] before feeding them into ODE-RNNs. As the signatures are much shorter than the original input sequence, the ODE-RNNs can learn long-term dependencies in the original input sequence despite expressing a vanishing gradient.

CT-LSTM [44] combines the LSTM architecture with continuous-time neural Hawkes processes. At each time step, the RNN computes two alternative next state options of its hidden state. The actual hidden state is then computed by interpolating between these two hidden states depending on the elapsed time.

coRNN [56] uses an implicit-explicit Euler discretization of a second-order ODE modeling a controlled non-linear oscillator. The state-to-next state gradients of the resulting RNN are bounded in both directions, mitigating explosion and vanishing effects.

iRNN [33] parametrize an RNN by applying incremental updates to the steady state of an ODE. In the limit with infinitely many updates between the state and the next state, iRNN express a constant error propagation.

HiPPO [23] presents a framework for continuous-time function memorization. In particular, HiPPO projects the history of time-series to a high-order polynomial space and is, therefore, able to store the history efficiently in the form of coefficients. This mechanisms allows HiPPO-based models [25, 24] to memorize inputs over extremely long time horizons.

What if we feed in samples' time lag as an additional input feature to the network? The *Augmented LSTM* architecture we benchmarked against realizes this concept, which is a simplistic approach to making LSTMs compatible with irregularly sampled data. The RNN could then learn to make sense of the time input, for instance, by making its change proportional to the elapsed time. Nonetheless, the time characteristic of an augmented RNN depends purely on its learning process.

Consequently, we can only hope that the augmented RNN will generalize to unseen time-lags. Our experiments showed that an augmented LSTM performs reasonably well while being outperformed by models that explicitly declare their state by a continuous-time modality, such as mmRNNs.

Difference between reciprocal RNNs and mmRNN? A naive approach to tackling the issues of learning long-range dependencies while also being able to process irregularly sampled time series is a reciprocal RNN architecture [3]. A reciprocal architecture consists of two different types of RNNs reciprocally linked together in an auto-regressive fashion [3]. In our context, the first RNN could be designed to handle irregularly-sample time series while the second one is capable of learning long-term dependencies.

For example, an LSTM bidirectionally coupled with an ODE-RNN could, in principle, overcome both challenges. However, the use of heterogeneous RNN architectures might limit the learning process. In particular, due to different training convergence rates, the LSTM could already be overfitting long before the ODE-RNN has learned useful dynamics.

Contrarily, our mmRNN interlinks LSTMs and ODE-RNNs not in an autoregressive fashion but at an architectural level, avoiding the problem of learning at different speeds. Our experiments showed that mmRNNs consistently outperform a reciprocal LSTM-ODE-RNN architecture.

B Derivations and Additional Theoretical Analysis

Derivation of the Euler's method Jacobian Let $\dot{h} = f_\theta(x, h, T) - h\tau$ be an ODE-RNN. Then the explicit Euler's method with step-size T is defined as the discretization

$$h_{t+T} = h_t + T(f_\theta(x, h, T) - h\tau)\Big|_{h=h_t}. \tag{5}$$

Therefore, state-previous state Jacobian and eigenvalues is given by

$$\frac{\partial h_{t+T}}{\partial h_t} = I + T\frac{\partial f}{\partial h}\Big|_{h=h_t} - \tau T I, \tag{6}$$

and

$$\text{Eig}\Big(\frac{\partial h_{t+T}}{\partial h_t}\Big) = 1 - \tau T + T\ \text{Eig}(\frac{\partial f}{\partial h}\Big|_{h=h_t}) \tag{7}$$

Derivation of the Runge-Kutta Jacobian Let $\dot{h} = f_\theta(x, h, T) - h\tau$ be an ODE-RNN. Then the Runge-Kutta method with step-size T is defined as the discretization

$$h_{t+T} = h_t + T\sum_{j=1}^{M} b_i(f_\theta(x, h, T) - h\tau)\Big|_{h=K_i}, \tag{8}$$

where the coefficients b_i and the values K_i are taken according to the Butcher tableau with $\sum_{j=1}^{M} b_i = 1$ and $K_1 = h_t$.

Then state-previous state Jacobian of the Runge-Kutta method is given by the following equation :

$$\frac{\partial h_{t+T}}{\partial h_t} = I + T\sum_{j=1}^{M} b_i\frac{\partial f}{\partial h}\Big|_{h=K_i} - \tau T I., \tag{9}$$

and

$$\text{Eig}\Big(\frac{\partial h_{t+T}}{\partial h_t}\Big) = 1 - \tau T + T\ \text{Eig}(\sum_{j=1}^{M} b_i\frac{\partial f}{\partial h}\Big|_{h=K_i}) \tag{10}$$

Note that the explicit Euler method is an instance of the Runge-Kutta method with $M = 1$ and $b_1 = 1$.

Derivation of ODE-RNNs tend to suffer from vanishing or exploding gradients Let $\dot{h} = f_\theta(x, h, T) - h\tau$ be an ODE-RNN with latent dimension N. Let h_0 be the initial state at $t = 0$ and h_T denote the ODE state, which should be computed by a numerical ODE-solver. Then ODE-solvers,

including fixed-step methods [55] and variable-step methods such as the Dormand-Prince method [12], discretize the interval $[0, T]$ by a series $t_0, t_1, \ldots t_n$, where $t_0 = 0$ and $t_n = T$ and each h_{t_i} is computed by a single-step explicit Euler or Runge-Kutta method from $h_{t_{i-1}}$.

Our derivation closely aligns with the analysis in Hochreiter and Schmidhuber [30]. We refer the reader to [29, 5, 48] for a rigorous discussion on the vanishing and exploding gradients.

We first consider a scalar RNN, i.e., $n = 1$, and then extend the discussion to the general case. The error-flow per RNN step between $t = 0$ and $t = T$ is given by

$$\frac{\partial h_T}{\partial h_0} = \prod_{m=1}^{n} \Big(1 + (t_m - t_{m-1}) \sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_{m_i}} - \tau(t_m - t_{m-1})\Big), \tag{11}$$

which realizes a power series depending on the value

$$|1 + (t_m - t_{m-1}) \sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_{m_i}} - \tau(t_m - t_{m-1})|. \tag{12}$$

Obviously, the condition that this term is equal to 1 is not enforced during training and violated for any non-trivial f_θ, such as $f_\theta(h, x) = \sigma(W_h h + W_x x + \hat{b})$ with σ being a sigmoidal or rectified-linear activation function. The exact magnitude depends on the weights W_h, as

$$\frac{\partial f_\theta(h, x)}{\partial h} = W_h \sigma'(W_h h + W_x x + \hat{b}). \tag{13}$$

A non-zero time-constant τ pushes the gradient toward a vanishing region.

Note that the Equation (12) only becomes equal to 1, if $\sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_{m_i}} = \tau$. This would imply that $\frac{\partial h_{t_m}}{h_{t_{m-1}}} = 0$, i.e., when the change in ODE-state between two time-points is zero. A variable that does not change over time is a memory element. Thus the only solution for enforcing a constant-error propagation in a 1-dimensional ODE-RNN is to include an explicit memory element in the architecture [30] which does not change its value between two arbitrary time-points t_m and t_{m-1}.

For the general case $n \geq 1$, the error-flow per RNN step between $t = 0$ and $t = T$ is given by

$$\frac{\partial h_T}{\partial h_0} = \prod_{m=1}^{n} \Big(I + (t_m - t_{m-1}) \sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_{m_i}} - \tau(t_m - t_{m-1})I\Big). \tag{14}$$

As h is a vector, we need to consider all possible error-propagation paths. The error-flow from unit u to unit v is then given by summing all N^{n-1} possible paths between u to v,

$$\frac{\partial h_T^v}{\partial h_0^u} = \sum_{l_1}^{N} \cdots \sum_{l_{n-1}}^{N} \prod_{m=1}^{n} \Big(I + (t_m - t_{m-1}) \sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_{m_i}} - \tau(t_m - t_{m-1})I\Big)_{l_m, l_{m-1}}, \tag{15}$$

where $l_0 = u$ and $l_n = v$.

The arguments of the scalar case hold for every individual path in Equation (15). The only difference between the scalar case and the individual paths in the vectorized version is the non-diagonal connections in the general case do not include the constant one and τ. The error-propagation magnitude between u and v with $u \neq v$ is given by

$$|(t_m - t_{m-1})\Big(\sum_{j=1}^{M} b_i \frac{\partial f}{\partial h}\Big|_{h=K_{m_i}}\Big)_{u,v}|. \tag{16}$$

Again, for $f_\theta(h, x) = \sigma(W_h h + W_x x + \hat{b})$ we obtain an error-flow that depends on the weights W_h and can be either vanishing or exploding, depending on its magnitude.

Theorem 2 (Informal)**. (ODE-RNNs tend to suffer from vanishing/exploding gradients regardless of their choice of ODE-solver)** *Let $\dot{h} = f_\theta(x, h, T) - h\tau$, with f_θ being uniformly Lipschitz continuous. Moreover, let $h(t)$ be the solution to the initial value problem with initial state h_0. Then, the ODE-RNN tends to suffer from vanishing/exploding gradients regardless of their choice of ODE-solver.*

Let $\dot{h} = f_\theta(x, h, T) - h\tau$ be an ODE-RNN with latent dimension N, with f_θ being uniformly Lipschitz continuous. Without loss of generality, let h_0 be the initial state at $t = 0$ and h_T denote the ODE state, which should be computed by a numerical ODE-solver. We approximate the interval $[0, T]$ by a uniform discretization grid, i.e. $t_i - t_{i-1} = t_j - t_{j-1} = T/n$ for all i, j $t_0, t_1, \dots t_n$, where $t_0 = 0$ and $t_n = T$ and each h_{t_i} is computed by a single-step explicit Euler from $h_{t_{i-1}}$.

Even when making the discretization grid $t_0, t_1, \dots t_n$ finer and finer, the gradient propagation issue is not resolved. Let h_i denote the intermediate values computed by the Picard-iteration, i.e., the explicit Euler. We know that the Picard-iteration h_T converges to the true solution $h(T)$.

First, we have a look at the 1-dimenional case, i.e., $N = 1$. We assume there exists a bound $\xi > 0$ such that $\xi \leq \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau$ for all m. Note that this situation can naturally occur if we have a $f_\theta(h, x) = \sigma(W_h h + W_x x + \hat{b})$. In the limit $n \to \infty$ we get

$$\begin{aligned}
\lim_{n\to\infty} \frac{\partial h_T}{\partial h_0} &= \lim_{n\to\infty} \prod_{m=1}^{n} \Big(1 + (t_m - t_{m-1}) \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau(t_m - t_{m-1})\Big) \\
&= \lim_{n\to\infty} \prod_{m=1}^{n} \Big(1 + \frac{T}{n} \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau\frac{T}{n})\Big) \\
&\geq \lim_{n\to\infty} \prod_{m=1}^{n} \Big(1 + \frac{T}{n}\xi\Big), \text{ with some } 0 < \xi \leq \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau \text{ for all } m \\
&= \lim_{n\to\infty} \Big(1 + \frac{T}{n}\xi\Big)^n \\
&= e^{T\xi} \\
&> 1,
\end{aligned}$$

i.e., we have an exploding gradient.

Conversely, lets assume there exists a $\xi < 0$ such that $\xi \geq \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau$ for all m. Note that this situation can also naturally occur, for instance, if $\tau > 0$ and regions where f' is small. In the limit $n \to \infty$ we get

$$\begin{aligned}
\lim_{n\to\infty} \frac{\partial h_T}{\partial h_0} &= \lim_{n\to\infty} \prod_{m=1}^{n} \Big(1 + (t_m - t_{m-1}) \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau(t_m - t_{m-1})\Big) \\
&= \lim_{n\to\infty} \prod_{m=1}^{n} \Big(1 + \frac{T}{n} \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau\frac{T}{n})\Big) \\
&\leq \lim_{n\to\infty} \prod_{m=1}^{n} \Big(1 + \frac{T}{n}\xi\Big), \text{ with some } 0 > \xi \geq \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau \text{ for all } m \\
&= \lim_{n\to\infty} \Big(1 + \frac{T}{n}\xi\Big)^n \\
&= e^{T\xi} \\
&< 1,
\end{aligned}$$

i.e., we have a vanishing gradient.

Similar to the argument above, we can extend the scalar case to the general case. However, summing over all possible paths might not be trivial, as the number of possible paths also grows to infinity.

$$\lim_{n\to\infty} \frac{\partial h_T^v}{\partial h_0^u} = \lim_{n\to\infty} \sum_{l_1}^{N} \cdots \sum_{l_{n-1}}^{N} \prod_{m=1}^{n} \Big(I + (t_m - t_{m-1}) \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau(t_m - t_{m-1})I\Big)_{l_m, l_{m-1}}. \quad (17)$$

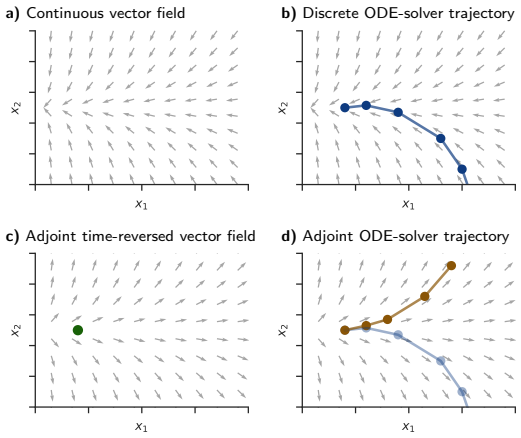

Figure 2: The adjoint method makes a numerical error when computing the gradients. **a)** Continuous vector field implied by an ODE. **b)** Numerical ODE-solvers realize a discrete trajectory on the vector field. **c)** The adjoint ODE creates a time-reversed vector field. **d)** Discrete trajectory of the adjoint ODE-solver diverges from the trajectory of the forward simulation due to discretization and numerical imprecision.

Instead, we assume $u = v = l_1 = \dots l_n - 1$, i.e., we only look at the error-propagation through the diagonal element u.

$$\begin{aligned}
\lim_{n\to\infty} \frac{\partial h_T^v}{\partial h_0^u} &= \lim_{n\to\infty} \prod_{m=1}^{n} \left(I + (t_m - t_{m-1}) \frac{\partial f}{\partial h}\Big|_{h=h_m} - \tau(t_m - t_{m-1}) I \right)_{u,u} \\
&= \lim_{n\to\infty} \prod_{m=1}^{n} \left(1 + (t_m - t_{m-1}) \frac{\partial f^u}{\partial h^u}\Big|_{h^u=h_m^u} - \tau^u (t_m - t_{m-1}) \right) \\
&= \lim_{n\to\infty} \prod_{m=1}^{n} \left(1 + \frac{T}{n} \frac{\partial f^u}{\partial h^u}\Big|_{h^u=h_m^u} - \tau^u \frac{T}{n}) \right),
\end{aligned}$$

which is equivalent to the scalar case. For an interesting f such as $f_\theta(h, x) = \sigma(W_h h + W_x x + \hat{b})$, the term $\frac{f}{h}$ depends on the value $W_h^{u,u}$. By assuming $W^{w,z}$ for any $(w, z) \neq (u, u)$ is negligibly small, we can infer that the effects of the gradient by any other path in Equation (17) is negligibly small. Thus the global error flow depends on $W_h^{u,u}$, which can make the error flow either explode or vanish depending on its value.

Note that this argument is similar to arguing that as the multi-dimensional case properly contains the scalar case, the multi-dimensional case can express an exploding or vanishing gradient too.

Does the adjoint method solve the vanishing gradient problem? Adjoint sensitivity method [50] allows for performing memory-efficient reverse-mode automatic differentiation for training neural networks with their hidden states defined by ODEs [9]. The method, however, possesses *lossy* reverse-mode integration steps, as it forgets the computed steps during the forward-pass [65]. Consequently, at each reverse-mode step, the backward gradient pass diverges from the true forward pass [65, 20]. Figure 2 schematically depicts this numerical instability, where at each reverse-mode step, the backward gradient pass diverges from the true forward pass.

This is because the auxiliary differential equation in the adjoint sensitivity method, $a(t)$, still contains state-dependent components at each reverse step, which depends on the historical values of the hidden states' gradient. In the extreme case, reverse steps completely diverge from the hidden states of the forward-time solution, resulting in incorrect gradients. Therefore, both vanilla BPTT and the adjoint method face difficulties in learning long-term dependencies. n the next section, we propose a solution.

A more detailed explanation that mmRNNs do not suffer from vanishing or exploding gradient at the beginning of the training

At the beginning of the training, we can assume that weights of g_θ and z_θ are initialized close to 0 and that the weights do not differ significantly from their initialized values. Moreover, for the derivation, we assume that g_θ and z_θ are standard multi-layer perceptron modules.

We have

$$\begin{aligned}\frac{\partial c_{t+1}}{\partial c_t} &= \frac{\partial c_t \odot \sigma(g_\theta(h_t) + b_f) + z_\theta(h_t)}{\partial c_t} \\ &= \frac{\partial \sigma(g_\theta(h_t) + b_f)}{\partial c_t}\text{diag}(c_t) + \text{diag}(\sigma(g_\theta(h_t) + b_f)) + \frac{\partial z_\theta(h_t)}{\partial c_t}.\end{aligned}$$

For the derivatives of the first term, we can simply apply the chain rule and get

$$\begin{aligned}\frac{\partial \sigma(g_\theta(h_t) + b_f)^v}{\partial c_t^u} &= \sigma'(g_\theta(h_t) + b_f)^v g_\theta(h_t)^v \frac{\partial g_\theta(h_t)^v}{\partial c_t^u} \\ &\approx 0,\end{aligned}$$

because we assumed $g_\theta(h_t) \approx 0$ due to its initialization.

A similar argument holds for the derivative of the last term, where we assumed that z_θ is initialized close to 0.

$$\begin{aligned}\frac{\partial z_\theta(h_t)^v}{\partial c_t^u} &= \text{f}'_{act}(W_\theta \hat{z}_t + b_\theta) W_\theta \frac{\partial \hat{z}_t^v}{\partial c_t^u} \\ &\approx 0,\end{aligned}$$

where f_{act} is the final activation function of z_θ, W_θ and b_θ the weights and bias parametrizing the last layer of z_θ and $\hat{z}_t$ the last hidden vector of z_θ. Note that we assumed $W_\theta \approx 0$.

Consequently, with a proper weight initialization, the Jacobian simplifies to

$$\frac{\partial c_{t+1}}{\partial c_t} \approx \text{diag}(\sigma(g_\theta(h_t) + b_f)).$$

We assumed that g_θ is initialized close to 0. Hence,

$$\begin{aligned}\text{diag}(\sigma(g_\theta(h_t) + b_f))^v &=\approx \sigma(b_f) \\ &= \sigma(3) \\ &\approx 0.994,\end{aligned}$$

as we initialized b_f to 3.

Hence, we have

$$\Big|\sum_{j=1}^{N} \frac{\partial c_{t+1}^i}{\partial c_t^j}\Big| \approx 0.994,$$

, which is close to 1 (can be made arbitrary close by varying b_f) and ensures a near-constant error propagation at the beginning of the training process.

As already mentioned in the paper, the exact value of the error flow can be controlled by changing the bias term from its default value of 1. If the underlying data distribution contains dependencies with a very long time lag, we can bring the error flow factor closer to 1 by increasing forget gate bias; Thus enabling the mmRNN to learn even very long-term dependencies in the data.

C Experimental evaluation

C.1 Bit-stream classification

For this task, the model observes a block of binary data in the form of a bit-after-bit time series. The objective is to learn an XOR function of the incoming bit stream. This setup is equivalent to the binary classification of the input sequence, where the labels are obtained by applying an XOR function to the inputs.

While any non-linear recurrent neural network architecture can learn the correct function, training the network to do so is non-trivial. For the model to make an accurate prediction, all bits in an upcoming chunk are required to be taken into account. However, the error signal is only provided after the last bit is observed. Consequently, during learning, the prediction error needs to be propagated to the first input time step to precisely capture the dependencies.

We designed two modes, a dense encoding mode in which the input sequence is represented as a regular, periodically sampled time-series, and an event-based mode which encodes the data into irregularly sampled run-length encoded bit-streams, e.g., $1, 1, 1, 1$ is encoded as $(1, t = 4)$. In particular, only changes in the bit-stream are fed to the RNN.

In more detail, every data point is a block of 32 random bits. The binary labels are created by applying an XOR function on the bit block, i.e., class A if the number of 1s in the bit-stream are even, class B if the number of 1s in the bit-stream is odd. For training, a cross-entropy loss on these two classes is used. The training set consists of 100,000 samples, which are less than 0.0024% of all possible bit-streams that can occur. The test set consists of 10,000 samples.

For event-based encoding, we introduce a time dimension. The time is normalized such that the complete sequence equals 1 unit of time, i.e., 32 bits corresponds to exactly 1 second. An illustration of the two different encodings is shown in Figure 5.

Sequences of our event-based bit-stream classification task and event-based seqMNIST can have different lengths. To allow an arbitrary batching of several sequences, we pad all sequences to equal lengths and apply a binary mask during training and evaluation.

We observed in Table 3 that a considerable number of RNNs face difficulties in modeling these tasks, even in dense-encoding mode. In particular, ODE-RNNs, CT-RNNs, RNN-Decay, Phased-LSTM, and GRU-ODE could not solve the XOR problem in the first mode. Phased-LSTM and RNN-Decay improved their performance in the second modality, whereas ODE-RNNs, CT-RNNs, and GRU-ODE still could not solve the task.

The core reason for their low performance is the exploitation of the vanishing gradient problem during training. The rest of the RNN variants (except CT-GRU) were successful in solving the task in both modes, with mmRNN outperforming others in an event-based encoding scenario.

C.2 Person activity recognition with irregularly sampled time-series

We consider the person activity recognition dataset from the UCI repository [13]. This task's objective is to classify the current activity of a person from four inertial measurement sensors worn on the person's arms and feet. Even though the four sensors are measured at a fixed period of 211ms, the random phase shifts between them create an irregularly sampled time series. Rubanova et al. [53] showed that ODE-based RNN architectures perform remarkably well on this dataset. Here, we benchmarked the performance of the mmRNN model against other variants.

The dataset is comprised of 25 recordings of human participants performing different physical activities. The eleven possible activities are "lying down", "lying", "sitting down", "sitting", "standing up from lying", "standing up from", "sitting", "standing up from sitting on the ground", "walking", "falling", "on all fours", and "sitting on the ground". The objective of this task is to recognize the activity from inertial sensors worn by the participant, i.e., a per-time-step classification problem. We group the eleven activities listed above into seven different classes, as proposed by [53].

The input data consists of sensor readings from four inertial measurement units placed on the participant's arms and feet. The sensors are read at a fixed period of 211 ms but have different phase shifts in the 25 recordings. Therefore, we treat the data as an irregularly sampled time series.

The 25 recordings are split into partially overlapping sequences of length 32 to allow an efficient training of the machine learning models.

Our results are not directly comparable to the experiments in [53], as we use a different representation of the input features. While [53] represents each input feature as a value-mask pair, i.e., 24 input features, we represent the data in the form of a 7-dimensional feature vector. The first four entries of the input indicate the sensor ID, i.e., which arm or foot, whereas the remaining three entries contain the sensor reading.

This setting realizes a per-time-step classification problem. That is, a new error signal is presented to the network at every time step which makes the vanishing gradient less of an issue here. The results in Table 3 show that the mmRNN outperforms other RNN models on this dataset. While the significance of an evaluation on a single dataset is limited, it demonstrates that the supreme generalization ability of mmRNN architecture.

C.3 Event-based sequential MNIST

We determined a challenging sequence classification task by designing an event-based version for the sequential-MNIST dataset. The MNIST dataset consists of 70,000 data points split into 60,000 training and 10,000 test samples [40]. Each sample is a 28-by-28 grayscale image, quantized with 8-bits, and represents one out of 10 possible digits, i.e., a number from 0 to 10.

We pre-process each sample as follows: We first apply a threshold to transform the 8-bits pixel values into binary values. The threshold is 128, on a scale where 0 represents the lowest possible, and 255 is the largest possible pixel value. We further transform the 28-by-28 image into a time series of length 784. Next, we encode binary time series in an event-based format. Essentially, the encoding step gets rid of consecutive occurrences of the same binary value, i.e., $1, 1, 1, 1$ is transformed into $(1, t = 4)$. By introducing a time dimension, we can compress the sequences from 784 to an average of 53 time steps.

To allow efficient batching and training, we pad each sequence to a length of 256. Note that no information was lost during this process. We normalize the added time dimension such that 256 symbols correspond to 1 second or unit of time. The resulting task is a per-sequence classification problem of irregularly sampled time series.

Table 3 demonstrates that ODE-based RNN architectures, such as the ODE-RNN, CT-RNN, and the GRU-ODE [11] struggle to learn a high-fidelity model of this dataset. On the other hand, RNNs built based on a memory mechanism, such as the reciprocal RNN and GRU-D [7] perform reasonably well, while the performance of mmRNN surpasses others.

C.4 Walker2d kinematic simulation

In this additional experiment, we evaluated how well mmRNNs can model a physical dynamical system. We create a dataset based on the `Walker2d-v2` OpenAI gym [6] environment and the MuJoCo physics engine [60]. Our objective is to benchmark how well the RNN architecture can model kinematic dynamical systems in an irregularly sampled fashion. The learning setup is based on auto-regressive supervised learning, i.e., the model predicts the next state of the Walker2d environment based on the current state.

In order to obtain interesting simulation rollouts, we trained a non-recurrent policy by Proximal Policy Optimization (PPO) [57] using the Rllib [42] reinforcement learning framework. We then collect the training data for our benchmark by performing rollouts on the `Walker2d-v2` environment using our pre-trained policy. Note that because the policy is deterministic, there is no need to include the actions produced by the policy in the training data.

We introduce three sources of uncertainty to make this task more challenging. First of all, for each rollout, we uniformly sample a checkpoint of policy at 562, 822, 923, or 1104 PPO iterations. Secondly, we overwrite 1% of all actions by random actions. Thirdly, we exclude 10% of the time-steps, i.e., we simulate frame-skips/frame-drops. Note that the last step transforms the rollouts into an irregularly sampled time series and introduces a time dimension.

In total, we collected 400 rollouts, i.e., 300 used for training, 40 for validation, and 60 for testing. For efficient training, we align the rollouts into sequences of length 64. We use the mean-square error as training loss and evaluation metric. We train each RNN for 200 epochs and log the validation error after each training epoch. In the end, we restore the weights that achieved the best (lowest) validation error and evaluate them on the test set.

The results, shown in Table 5, indicate that mmRNNs can capture the kinematic dynamics of the physics engine better than other algorithms with a high margin.

For models containing differential equations, we used the ODE-solvers as listed in Table 4. Hyperparameter settings used for our evaluation is shown in Table 6.

Table 4: ODE-solvers used for the different RNN architectures involving ordinary differential equations

Model	ODE-solver	Time-step ratio
CT-RNN	4-th order Runge-Kutta	1/3
ODE-RNN	4-th order Runge-Kutta	1/3
GRU-ODE	Explicit Euler	1/4
mmRNN	Explicit Euler	1/4

Table 5: **Per time-step regression**. Walker2d kinematic dataset. (mean $\pm$ std, $N = 5$)

Model	Square-error
ODE-RNN	1.904 ± 0.061
CT-RNN	1.198 ± 0.004
Augmented LSTM	1.065 ± 0.006
CT-GRU	1.172 ± 0.011
RNN-Decay	1.406 ± 0.005
Reciprocal RNN	1.071 ± 0.009
GRU-D	1.090 ± 0.034
PhasedLSTM	1.063 ± 0.010
GRU-ODE	1.051 ± 0.018
CT-LSTM	1.014 ± 0.014
iRNN	1.732 ± 0.025
coRNN	3.241 ± 0.215
Lipschitz RNN	1.781 ± 0.013
mmRNN (ours)	$\mathbf{0.883 \pm 0.014}$

Table 6: Hyperparameters

Parameter	Value	Description
RNN latent dimension	64	number of neurons in the RNN
Minibatch size	256	
Optimizer	RMSprop [59]	
Learning rate	5e-3	
Training epochs	500/200	Synthetic/real datasets

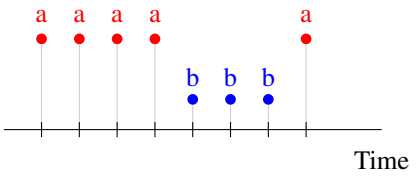

Figure 3: Dense coding

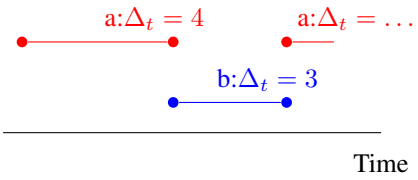

Figure 4: Event-based run-length encoding

Figure 5: Dense and event-based coding of the same time-series. An event-based coding is more efficient than dense coding at encoding sequences where the transmitted symbol changes only sparsely.