Linear Antisymmetric Recurrent Neural Networks

Signe Moe SIGNE.MOE@SINTEF.NO
Dept. of Mathematics and Cybernetics, SINTEF Digital, Oslo, Norway

Filippo Remonato FILIPPO.REMONATO@SINTEF.NO
Dept. of Mathematics and Cybernetics, SINTEF Digital, Oslo, Norway

Esten I. Grøtli ESTENINGAR.GROTLI@SINTEF.NO
Dept. of Mathematics and Cybernetics, SINTEF Digital, Trondheim, Norway

Jan Tommy Gravdahl JAN.TOMMY.GRAVDAHL@NTNU.NO
Dept. Engineering Cybernetics, NTNU, Trondheim, Norway

Editors: A. Bayen, A. Jadbabaie, G. J. Pappas, P. Parrilo, B. Recht, C. Tomlin, M. Zeilinger

Abstract

Recurrent Neural Networks (RNNs) have a form of memory where the output from a node at one timestep is fed back as input the next timestep in addition to data from the previous layer. This makes them highly suitable for timeseries analysis. However, standard RNNs have known weaknesses such as struggling with long-term memory. In this paper, we suggest a new recurrent network structure called Linear Antisymmetric RNN (LARNN). This structure is based on the numerical solution to an Ordinary Differential Equation (ODE) with stability properties resulting in a stable solution, which corresponds to long-term memory. Three different numerical methods are suggested to solve the ODE: Forward and Backward Euler and the midpoint method. The suggested structure has been implemented in Keras and several simulated datasets have been used to evaluate the performance. In the investigated cases, the LARNN performs better or similar to the Long Short Term Memory (LSTM) network which is the current state of the art for RNNs.

Keywords: Recurrent Neural Network, Long-Term Memory, Timeseries Analysis

1. Introduction

RNNs are a standard, proven method for modeling sequential data (Goodfellow et al., 2016). They have a form of memory which result in the ability to learn long-term dependencies in data, which is relevant in for instance language modeling (Józefowicz et al., 2016) and physical systems with slow dynamics. A standard RNN layer has the following structure for the output

$$\boldsymbol{h}_t = f(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}), \tag{1}$$

where $\boldsymbol{h}_i$ is the output of the layer and $\boldsymbol{x}_i$ is the layer input at time i, $\boldsymbol{W}$, $\boldsymbol{V}$ and $\boldsymbol{b}$ are the layer weight matrices and bias vector and $f(\cdot)$ is a nonlinear activation function. These networks have known weaknesses such as exploding/vanishing gradient during training and they struggle with a long-term memory (Hochreiter, 1998). As a result, more complex gated structures such as LSTM networks and gated recurrent units have been proposed. These have a higher trainability (Collins et al., 2017). However, these only solve the problem to some extent and must be combined with techniques such as normalization layers and gradient clipping to achieve good performance (Chang et al., 2019).

The memory of a network is related to the sensitivity of the network prediction with respect to a perturbation in the input (Samarasinghe, 2016). Thus, an input-sensitive network implies that the output is unstable with respect to the input, i.e. a small change in the input results in a large change in the state $\boldsymbol{h}$. Similarly an input-insensitive network implies that the output is insensitive with respect to the input. For RNNs in particular, the input to the network is a sequence of length k (often referred to as the number of steps) $\boldsymbol{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_k]$. To ensure *long-term memory*, the final layer output $\boldsymbol{h}_k$ must be affected also by the first elements of $\boldsymbol{x}$, i.e. a perturbation $\Delta\boldsymbol{x}$ in $\boldsymbol{x}_1$ (or any step) should lead to a change $\Delta\boldsymbol{h}$ in the final state $\boldsymbol{h}_k$. In addition, to ensure a *consistent* memory, similar input sequences should result in similar layer outputs, i.e. the magnitude of $\Delta\boldsymbol{h}$ should reflect the magnitude of $\Delta\boldsymbol{x}$.

In this paper, a new structure for RNNs is proposed based on the numerical solution to an ODE. The stability properties of this ODE are analyzed and proven to provide a network layer with a stable long-term memory. The proposed structure is inspired by the AntisymmetricRNN (ARNN) (Chang et al., 2019), but addresses some of its shortcomings.

This paper is organized as follows; Section 2 discusses the stability of ODE solutions and relates this to long-term memory of RNNs. The previously suggested ARNN is presented and discussed in Section 3 before the LARNN is presented and analyzed in Section 4 and implementation and evaluation are described in Section 5. Finally, conclusions are given in Section 6.

2. Stability of ODEs as a basis for RNNs

To achieve a network structure with the desired properties for consistent long-term memory, we consider the stability of solutions of ODEs.

Definition 1 *A solution $\boldsymbol{y}(t)$ of an ODE*

$$\dot{\boldsymbol{y}}(t) = f(t, \boldsymbol{y}(t)) \tag{2}$$

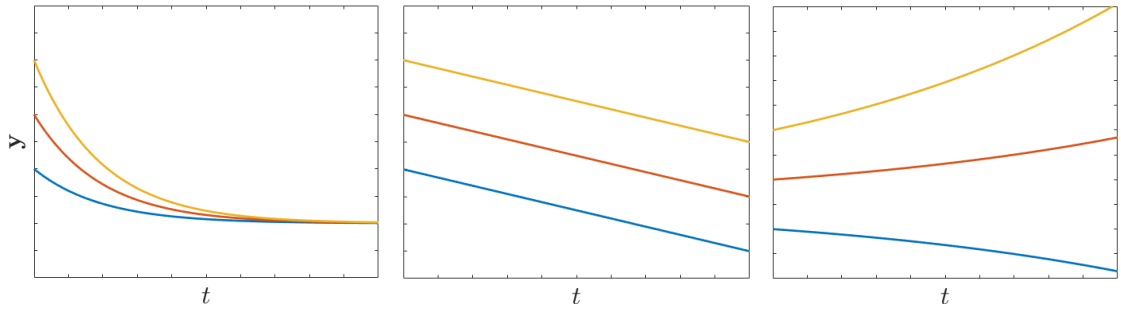
with initial condition $\boldsymbol{y}(0)$ is stable *if for any $\xi > 0$ there exists a $\delta > 0$ such that any other solution $\bar{\boldsymbol{y}}(t)$ with initial condition $\bar{\boldsymbol{y}}(0)$ satisfying $||\boldsymbol{y}(0) - \bar{\boldsymbol{y}}(0)|| \leq \delta$ also satisfies $||\boldsymbol{y}(t) - \bar{\boldsymbol{y}}(t)|| \leq \xi \ \forall\, t \geq 0$. It is* asymptotically stable *if $||\boldsymbol{y}(t) - \bar{\boldsymbol{y}}(t)|| \to 0$ as $t \to \infty$.*

Here, $\dot{\boldsymbol{y}}$ denotes the derivative of $\boldsymbol{y}$ with respect to time. Figure 1 illustrates the behavior of asymptotically stable, stable and unstable solutions $\boldsymbol{y}(t)$. An asymptotically stable solution has no long-term memory since the long-term solution is independent of the initial condition, whereas a small perturbation in the initial condition of an unstable solution might lead to infinitely large differences in the state evolution over time. For regression tasks, similar input sequences should result in similar, but not equal, states $\boldsymbol{h}$. Thus, it is desirable to design a RNN structure which possesses the qualities of a stable ODE solution as by Definition 1.

3. Antisymmetric RNNs

This section describes the AntisymetricRNNs as proposed by (Chang et al., 2019). The ARNN structure is defined as

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \varepsilon f(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}), \tag{3}$$



(*a*) Asymptotically stable (*b*) Stable (*c*) Unstable

Figure 1: Stability of ODE solutions with different initial conditions: *(a)* asymptotically stable - the solutions converge into one another, *(b)* stable - the solutions remain close to one another, *(c)* unstable - the solutions diverge away from each other.

which is the Forward Euler numerical solution with timestep $\varepsilon > 0$ to the ODE

$$\dot{\boldsymbol{h}} = f(\boldsymbol{W}_h \boldsymbol{h} + \boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}). \tag{4}$$

Here,

$$\boldsymbol{W}_h = \boldsymbol{W} - \boldsymbol{W}^T, \tag{5}$$

for some matrix $\boldsymbol{W}$, so $\boldsymbol{W}_h$ is by definition an *antisymmetric matrix* which has eigenvalues strictly on the imaginary axis. Furthermore, only *hyperbolic tangent* is considered as the activation function f in (Chang et al., 2019).

There are several possible improvements to the proposed structure (3): First, the theorems applied in (Chang et al., 2019) to analyze the stability of the solutions to the ODE (4) are from Chapter 3 in (Ascher et al., 1994) and are valid for ODEs on the form $\dot{\boldsymbol{y}} = f(t, \boldsymbol{y})$, i.e. an ODE without input terms. Eq. (4) is on the form $\dot{\boldsymbol{y}} = f(\boldsymbol{y}, \boldsymbol{x})$. Hence, the theorems are not actually applicable to this system.

Second, the proposed network structure (3) is based on the Forward Euler method, which is stable only in the region given in Figure 2a. Given that the system is designed so the eigenvalues are always on the imaginary axis, Forward Euler results in numerical instability regardless of timestep ε. To remedy this, the authors suggest adding a diffusion term to move the eigenvalues slightly into the left half plane. However, this results in an extra hyperparameter to tune in the implementation which results in a trade-off between numerical stability and long-term memory. Implicit numerical methods such as Backward Euler and midpoint method (also known as the trapezoidal rule) (Egeland and Gravdahl, 2002) have larger stability regions which include the imaginary axis (Figure 2b-2c), but these cannot be solved explicitly for $\boldsymbol{h}_t$ for the ODE (4).

Finally, we note that the state $\boldsymbol{h}(t)$ is unstable for other common activation functions such as *sigmoid* $\in (0,1)$ (globally unstable) and *relu* $\in [0, \infty)$ (locally unstable). For f as sigmoid, $\dot{\boldsymbol{h}} > \boldsymbol{0}$ in (4) regardless of the state $\boldsymbol{h}$ or input $\boldsymbol{x}$ and it is thus unbounded. Similarily, for relu $\dot{\boldsymbol{h}} \geq \boldsymbol{0}$ regardless of input, state or antisymmetric weight matrix.

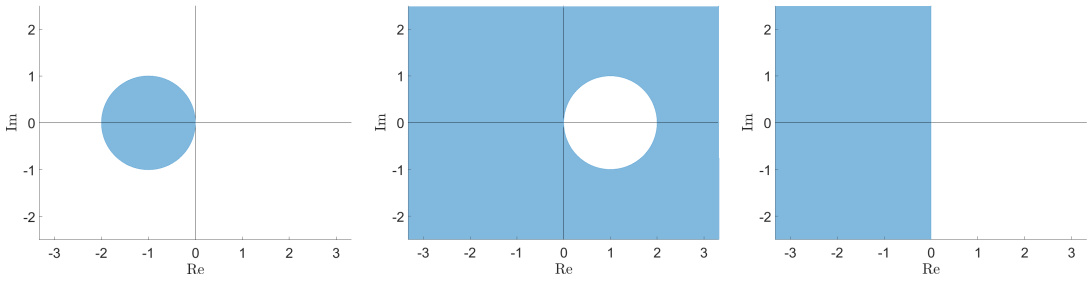


Figure 2: Stability region (blue) of various numerical methods. To ensure numerical stability, $\varepsilon\lambda_i$ must lie within the blue region for all eigenvectors λ_i. *(a)* Forward Euler method (explicit), *(b)* Backward Euler method (implicit) and *(c)* midpoint method (implicit).

4. Linear Antisymmetric RNNs

This section introduces a new RNN structure, also based on an antisymmetric weight matrix, which improves the stability and numerical properties of the ARNN. We suggest basing the network structure on the ODE

$$\dot{\boldsymbol{h}} = \boldsymbol{W}_h\boldsymbol{h} + f(\boldsymbol{V}\boldsymbol{x}+\boldsymbol{b}). \tag{6}$$

This ODE is linear in the state $\boldsymbol{h}(t)$ and is therefore referred to as a Linear Antisymmetric RNN (LARNN). Consider first the long-term memory of this ODE. Note that (6) is a Hammerstein system, which consists of a cascade of linear and nonlinear parts. Such systems are used for system identification and modeling (Mete et al., 2016). For two solutions $\boldsymbol{h}(t)$ and $\bar{\boldsymbol{h}}(t)$ with non-equal initial conditions $\boldsymbol{h}(0)$ and $\bar{\boldsymbol{h}}(0)$ presented with the same input $\boldsymbol{x}(t)$,

$$\begin{aligned}\Delta\dot{\boldsymbol{h}} = \dot{\boldsymbol{h}} - \dot{\bar{\boldsymbol{h}}} &= \boldsymbol{W}_h\boldsymbol{h} + f(\boldsymbol{V}\boldsymbol{x}+\boldsymbol{b}) - \boldsymbol{W}_h\bar{\boldsymbol{h}} - f(\boldsymbol{V}\boldsymbol{x}+\boldsymbol{b}) \\ &= \boldsymbol{W}_h\Delta\boldsymbol{h}.\end{aligned} \tag{7}$$

We apply Theoreom 4.5 (Khalil, 2002) to prove that the equilibrium point $\Delta\boldsymbol{h} = \boldsymbol{0}$ is stable. Since $\boldsymbol{W}_h \in \mathbb{R}^{n\times n}$ is antisymmetric, it is a normal operator over $\mathbb{R}^n$ and all its eigenvalues satisfy $\mathrm{Re}(\lambda_i) = 0$. By applying the Spectral Theorem we can conclude that $\boldsymbol{W}_h$ has a full basis of (orthogonal) eigenvectors, so that for each λ_i its geometric multiplicity g_i equals the algebraic multiplicity q_i. Thus, $\mathrm{Rank}(\boldsymbol{W}_h - \lambda_i I) = n - Null(\boldsymbol{W}_h - \lambda_i I) = n - g_i = n - q_i$, where the first equality comes from the Rank+Nullity Theorem. Thus, $\Delta\boldsymbol{h} = \boldsymbol{0}$ is stable (Khalil, 2002).

For practical purposes, the hidden state of RNNs has initial condition $\boldsymbol{h}(0) = f(\boldsymbol{V}\boldsymbol{x}(0)+\boldsymbol{b})$. Thus, if the ODE (6) is used as a basis for a RNN structure and two input sequences $\boldsymbol{x}(t) \equiv \bar{\boldsymbol{x}}(t)\ \forall\, t \neq 0$ are given to the network, we can conclude that the two solutions $\boldsymbol{h}(t)$ and $\bar{\boldsymbol{h}}(t)$ will neither converge into one another (thereby infinitely preserving the memory of the initial state) nor diverge away from one another (thereby having a consistent memory where similar input sequences yield similar output).

Furthermore, since the ODE (6) is linear in the state $\boldsymbol{h}$, it is possible to apply implicit numerical methods to solve the ODE and thereby achieve a RNN layer structure which is numerically stable

also on the imaginary axis. We propose the following three structures:

$$\boldsymbol{h}_t = (\boldsymbol{I} + \varepsilon \boldsymbol{W}_h)\boldsymbol{h}_{t-1} + \varepsilon f(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}), \tag{8}$$

$$(\boldsymbol{I} - \varepsilon \boldsymbol{W}_h)\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \varepsilon f(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}), \tag{9}$$

$$(\boldsymbol{I} - \frac{\varepsilon}{2}\boldsymbol{W}_h)\boldsymbol{h}_t = (\boldsymbol{I} + \frac{\varepsilon}{2}\boldsymbol{W}_h)\boldsymbol{h}_{t-1} + \varepsilon f(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}), \tag{10}$$

Eq. (8) is based on the Forward Euler method and consequently suffers from numerical instability as (3). On the other hand, (9)–(10) are based on the Backward Euler and the implicit midpoint method, respectively, and are numerically stable. They do, however, require inversion of the left-hand side matrix (see Proposition 2). It is also possible to apply other numerical methods such as higher order Runge-Kutta. However, these lead to more complex and computationally heavy network structures. Also note that Backwards Euler method is known to be dissipative, whereas the implicit midpoint method, being a *symmetric* method, preserves orbits without the need for a diffusion term, (Hairer et al., 2006).

Proposition 2 *The* $\mathbb{R}^{n\times n}$ *matrix*

$$\boldsymbol{I} + \delta \boldsymbol{W}_h \tag{11}$$

is always invertible for $\delta \in \mathbb{R}$.

Proof For $\delta = 0$, the expression is trivial. For $\delta \neq 0$, assume that $(\boldsymbol{I} + \delta \boldsymbol{W}_h)$ is not invertible. Then $\lambda = 0$ is a root of $\det(\boldsymbol{I} + \delta \boldsymbol{W}_h - \lambda \boldsymbol{I}) = \det(\delta \boldsymbol{W}_h - (\lambda - 1)\boldsymbol{I}) = \delta^n \det(\boldsymbol{W}_h - \frac{(\lambda-1)}{\delta}\boldsymbol{I})$. Thus,

$$\det\left(\boldsymbol{W}_h - \left(-\frac{1}{\delta}\right)\boldsymbol{I}\right) = 0,$$

which implies that $(-1/\delta) \in \mathbb{R}$ is an eigenvalue of $\boldsymbol{W}_h$. However, by definition $\boldsymbol{W}_h$ only has eigenvalues on the imaginary axis. Thus, the matrix $(\boldsymbol{I} + \delta \boldsymbol{W}_h)$ is invertible. ■

Finally, the solution of the ODE (6) does not grow monotonically for sigmoid and relu activation functions since the feedback term $\boldsymbol{W}_h\boldsymbol{h}$ is separate from the activation function. Proving input-to-state stability remains a topic for future work.

5. Implementation and evaluation

The proposed architectures (8)-(10) have been implemented in Keras so that it can be readily adopted and tested by other users. This code will be released as open source in the near future.

The parameter $\varepsilon > 0$ represents the timestep of the numerical methods and must be tuned as a hyperparameter. A small ε 1) puts more weight on the current state $\boldsymbol{h}_i$ (the memory) relative to the new input $\boldsymbol{x}_i$, 2) results in a more smooth evolution of $\boldsymbol{h}_i$ and 3) mitigates the effects of numerical instability if Forward Euler is applied as the numerical method (8). In theory, ε can take any positive value, but if the LARNN is to represent a dynamic system knowledge about the dynamics, sampling frequency, signal noise, expected number/magnitude of outlier data and system time constants should be considered when choosing ε. Similarly, the increased computational complexity of (9)-(10) should be weighed against the numerical stability they provide.

In the remainder of this section we compare the performance of the proposed architectures (8)-(10), the LSTM and the ARNN (3) on two cases of simulated timeseries data. In all cases,

the networks have the same structure, specifically one recurrent layer (either LSTM, ARNN (3) or LARNN (8)-(10)) of 32 units, one Dense (feedforward) layer of 8 units and relu activation function and finally a Dense layer of 1 unit and a linear activation function. Both cases have been run for tanh, sigmoid and relu as the activation function of the recurrent layer. For all networks, a mean square error loss and the Adam optimizer is applied. Furthermore, 80% of the data is used for training and 20% for testing. Hyperparameters for the two cases are given in Table 1.

	Case 1	Case 2
Steps in input sequence	64	100
Epochs	80	200
Batch size	64	64
Learning rate	0.0005	0.001
Timestep ε	0.05	0.1

Table 1: Hyperparameters for the two testcases.

Case 1 For this case, the network should predict the next value of a one-dimensional time series $y(t)$ given a sequence of the 64 previous values. The data consists of 2000 data points and is given by

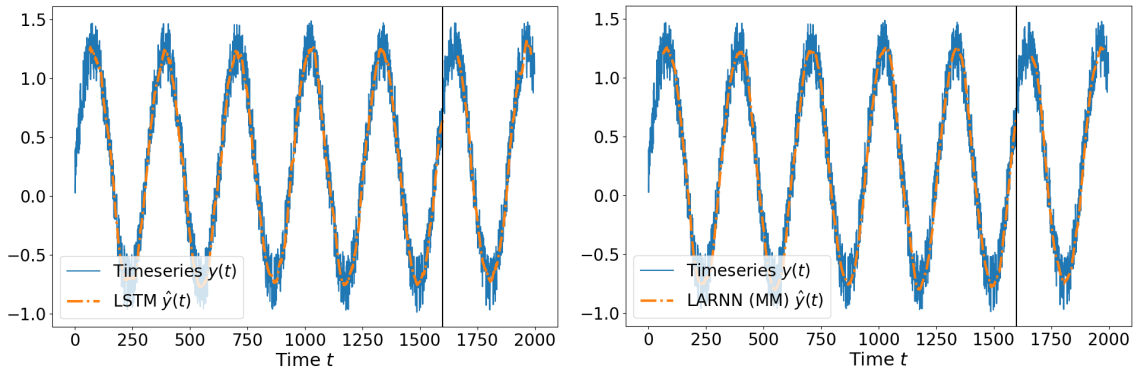
$$y(t) = \sin(0.02t) + \sigma[0, 0.5). \tag{12}$$

Here, $\sigma[a,b)$ represents noise from a uniform distribution over $[a,b)$. The results are summarized in Table 2. For this test case, all network structures give a similar performance both on the training and test data. The training time for ARNN (3) and LARNN using Forward Euler (8) are similar, whereas the LARNN based on implicit methods (9)-(10) take longer to train due to the more complex structure. These are in the same order of magnitude as the LSTM. For illustrative purposes, the timeseries $y(t)$ is plotted along with the predictions $\hat{y}(t)$ of the LSTM and LARNN based on the midpoint method (10) for the tanh activation function.

	Tanh			**Sigmoid**			**Relu**		
	Train and test loss [10e-3]		Train time	Train and test loss [10e-3]		Train time	Train and test loss [10e-3]		Train time
LSTM	22,58	22,45	53,42	26,74	26,22	66,37	24,41	23,19	63,72
ARNN	26,05	25,00	20,55	25,71	25,33	34,74	**22,03**	21,09	28,08
LARNN (FE)	22,62	22,00	**20,41**	23,20	23,02	**30,93**	27,27	24,11	**25,19**
LARNN (BE)	23,51	22,68	51,85	21,80	21,55	57,14	22,20	**20,94**	61,40
LARNN (MM)	**22,35**	**21,51**	58,18	**21,52**	**21,30**	67,10	23,06	21,50	60,80

Table 2: The train/test loss and training time in seconds for Case 1 with hyperparameters given by Tab. 1 and various activation functions.

Case 2 In case 2, the network should predict the next value of a one-dimensional time series $y(t)$ given a sequence of the 100 previous values of another time series $x(t)$, which consists of $N = 3000$ data points. The output $y(t)$ is the sum of the first 32 values in the input sequence multiplied by a constant, i.e. the network must have a long-term memory to find the correct mapping between input



(*a*) (*b*)

Figure 3: Actual timeseries $y(t)$ and prediction $\hat{y}(t)$ of the LSTM *(a)* and LARNN *(b)* based on the midpoint method (10) and tanh activation function for Case 1. Training data and test data are separated by the black line. The two networks have a similar performance.

and output. This is motivated by one of the test cases in (Chang et al., 2019).

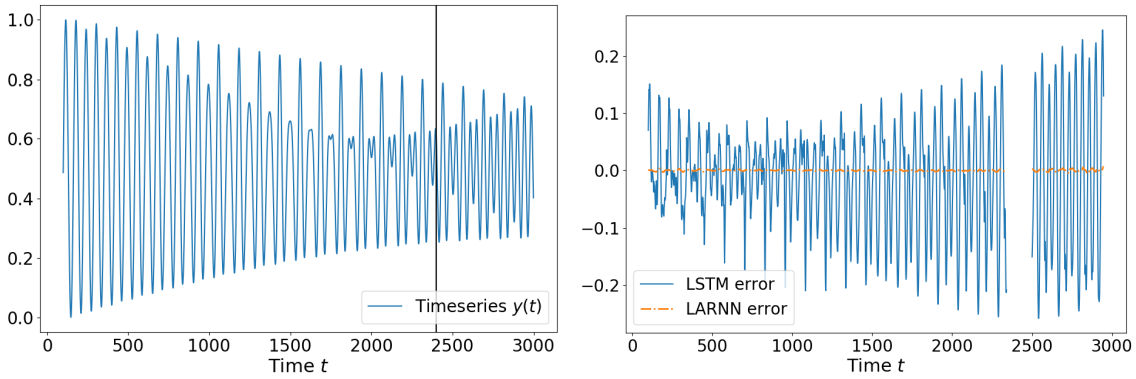
$$\begin{aligned} x(t) &= 0.03(t)\sin(0.15t) - 0.03(N-t)\cos(0.1t) \\ y(t) &= 0.0002 \sum_{i=t-100}^{t-68} x(i) \end{aligned} \tag{13}$$

Before training and testing, both $x(t)$ and $y(t)$ are scaled. This scaler is fitted so that the training data of both signals range between 0 and 1 and is then applied also to the test data. The results are summarized in Table 3. For hyperbolic tangent, all networks result in a similar performance, whereas for sigmoid and relu the LARNN have several orders of magnitude lower loss both on training and test data and outperforms the LSTM in particular. With regards to training time, the results from Case 1 are confirmed: Methods based on Forward Euler are faster whereas the structures based on implicit methods are slower and comparable to LSTM. For illustrative purposes, the timeseries $y(t)$ is plotted along with the predictions $\hat{y}(t)$ of the LSTM and LARNN based on the midpoint method (10) for the relu activation function.

	Tanh			**Sigmoid**			**Relu**		
	Train and test loss [10e-5]		Train time	Train and test loss [10e-5]		Train time	Train and test loss [10e-5]		Train time
LSTM	1,87	3,67	319,67	44,28	30,16	346,09	737,14	2294,17	313,79
ARNN	1,24	2,59	153,36	16,63	34,01	**129,63**	9,21	19,85	144,74
LARNN (FE)	0,72	0,90	**148,91**	1,84	2,53	146,80	0,52	2,88	**133,27**
LARNN (BE)	**0,07**	**0,30**	280,99	**0,16**	**0,51**	281,70	0,65	1,71	318,57
LARNN (MM)	0,19	0,83	346,26	0,22	**0,51**	324,55	**0,11**	**0,54**	355,73

Table 3: The train/test loss and training time in seconds for Case 2 with hyperparameters given by Tab. 1 and various activation functions.

The above cases illustrate that the LARNN structures (8)-(10) perform better or equally well as the LSTM and ARNN in addition to having improved theoretical properties regarding stability and



(a) (b)

Figure 4: The timeseries $y(t)$ *(a)* and the prediction error $y(t) - \hat{y}(t)$ of the LSTM and LARNN *(b)* based on the midpoint method (10) and relu activation function for Case 2. Training data and test data are separated by the black line in *(a)*. In this case, the LSTM has significantly worse performance.

long-term memory. It is worth noting that in some of the investigated exampled, the LSTM achieves a similar performance given longer training. Thus, if limited resources are available for training, the LARNN structure is preferable. Furthermore, although the ARNN (3) and LARNN based on Forward Euler (3) are subject to numerical instability, this effect will be small for small values of ε and/or short input sequences. Similarly, for practical purposes the state $\boldsymbol{h}$ for the ARNN will not be infinitely large in spite of being (globally and locally) unstable for sigmoid and relu activation functions. However, for cases with long input sequences and/or where the system dynamics and sampling frequency suggest a larger value for ε, the LARNN based on implicit methods (9)-(10) is recommended.

6. Conclusion

This paper presents the LARNN, which is a network structure specifically designed to ensure long-term memory of time series analysis. An ODE is designed and proven to have a stable solution, which corresponds to a long-term, consistent memory. The proposed network structures are based on numerical solutions to the ODE, using explicit (Forward Euler) or implicit (Backward Euler and midpoint method) approaches. The implicit methods ensure numerical stability, but result in a more complex network structure and require longer time to train given the same hyperparameters. Two test cases are investigated where the LARNN is compared to other recurrent network structures. In these cases, the LARNN performs equally well or better than the other networks.

Acknowledgments

This work was supported by the industry partners Borregaard, Elkem, Hydro, Yara and the Research Council of Norway through the project TAPI: Towards Autonomy in Process Industries, project number 294544.

References

U.M. Ascher, R.M.M. Mattheij, and R.D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1994.

B. Chang, M. Chen, E. Haber, and E. H. Chi. AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks. In *Proc. 2019 International Conference on Learning Representations (ICLR)*, 2019.

J. Collins, J. Sohl-Dickstein, and D. Sussillo. Capacity and trainability in recurrent neural networks. In *Proc. 2017 International Conference on Learning Representations (ICLR)*, 2017.

O. Egeland and J. T. Gravdahl. *Modeling and Simulation for Automatic Control*. Marine Cybernetics, Trondheim, Norway, 1st edition, 2002.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Massachusetts, USA, 2016. http://www.deeplearningbook.org.

E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration, Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer, 2006.

S. Hochreiter. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2), 1998.

R. Józefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. *ArXiv*, abs/1602.02410, 2016.

H. K. Khalil. *Nonlinear systems*. Prentice Hall PTR, New Jersey, USA, 2002.

S. Mete, S. Ozer, and H. Zorlu. System identification using Hammerstein model optimized with differential evolution algorithm. *AEU - International Journal of Electronics and Communications*, 70(12), 2016.

S. Samarasinghe. *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*. CRC Press, Florida, USA, 2016.