The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)

System Identification with Time-Aware Neural Sequence Models

Thomas Demeester
Internet Technology and Data Science Lab, Ghent University - imec
thomas.demeester@ugent.be

Abstract

Established recurrent neural networks are well-suited to solve a wide variety of prediction tasks involving discrete sequences. However, they do not perform as well in the task of dynamical system identification, when dealing with observations from continuous variables that are unevenly sampled in time, for example due to missing observations. We show how such neural sequence models can be adapted to deal with variable step sizes in a natural way. In particular, we introduce a 'time-aware' and stationary extension of existing models (including the Gated Recurrent Unit) that allows them to deal with unevenly sampled system observations by adapting to the observation times, while facilitating higher-order temporal behavior. We discuss the properties and demonstrate the validity of the proposed approach, based on samples from two industrial input/output processes.

1 Introduction

The field of system identification, aiming to build mathematical models of dynamical systems based on observed data, has been a large active research area for many years, with several specialized sub-fields (Ljung 2013). Within this general field, the topic of research in this paper is the joint application of numerical methods developed to solve systems of differential equations, with established techniques from the field of artificial neural networks.

On the one hand, neural networks provide a highly flexible tool to train unknown parameterized functions to fit available data (Goodfellow, Bengio, and Courville 2016). In particular, recurrent neural networks (RNNs), and especially variants such as Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber 1997) or Gated Recurrent Units (GRU) (Cho et al. 2014), have become important general-purpose tools for modeling discrete sequential data, for example in the area of natural language processing (Young, Hazarika, and Poria 2017). These models are however not naturally suited to deal with sampled time series where the interval between consecutive samples may not be constant over time. Yet, so-called *unevenly spaced time series* occur often in practice, due to missing data after discarding invalid measurements, or because of variations in the sampling times (Eckner 2014). For example, a patient's clinical variables may only be measured at irregular moments. Or, observations from systems in meteorology, economics, finance, or geology might only be possible at irregular points in time. Multivariate data consisting of individual time series with different sample rates are also naturally treated as unevenly spaced time series. Eckner (2014) further summarizes a number of disadvantages of transforming unevenly spaced data through resampling into evenly spaced data.

On the other hand, numerical methods such as the Runge-Kutta schemes (Butcher 2016) lead to highly accurate solutions for dynamical systems with known differential equations, and are *by construction* able to deal with varying time intervals.

The main research question that we tackle is therefore the following. Given a set of dynamical system observations, how can we build a model that can make use of the full expressive power of general-purpose RNNs, but that naturally deals with unevenly spaced time series?

It has already been pointed out that the Euler scheme for numerically solving differential equations bears similarities with artificial neural network models containing residual connections (E 2017; Zhu and Fu 2018; Chang et al. 2019). Naively applying this numerical scheme to RNNs for time series modeling, would indeed allow correctly dealing with uneven time intervals. However, unlike traditional RNNs, such schemes are incremental in nature, due to the residual connections along the time dimension, and therefore ill-suited for modeling stationary time series. We will show how this can be resolved, leading to models that take advantage of both the bounded character and expressive power of well-known RNNs, and the time-aware nature of numerical schemes for differential equations.

The paper makes the following contributions:

- We introduce an extension of general-purpose RNNs to deal with unevenly spaced time series (Section 3.1), and which can be used in higher-order numerical schemes (Section 3.2). In particular, we propose a solution for the so-called 'unit root' problem related to incremental recurrent schemes, to make the proposed approach suited for modeling stationary dynamical systems.
- We introduce a time-aware and higher-order extension of

the Gated Recurrent Unit and Antisymmetric RNN (Section 3.3).

- We provide insights into the introduced models with experiments on data from two industrial input/output systems (Section 4). The code required to run the presented experiments is publicly available[1].

2 Related Work

This work is situated in between the fields of system identification, numerical techniques for differential equations, and deep neural networks. We refer the reader to standard works in these areas (Ljung 2013; Butcher 2016; Goodfellow, Bengio, and Courville 2016), and focus in the following paragraphs on specifically related works.

System Identification with Neural Networks More than two decades ago, Wang and Lin (1998) introduced 'Runge Kutta Neural Networks', to predict trajectories of time-invariant ODEs with unknown right-hand side. Their core idea of combining neural networks and Runge-Kutta schemes still applies to the underlying work.

Several authors have recently put forward techniques for learning to compose partial differential equations from data. Rudy et al. (2017) presented an efficient method to select fitting nonlinear terms, from a library of potential candidate functions and their derivatives. Raissi and Kariadakis (2018) proposed to use Gaussian Processes for learning system parameters, to balance model complexity and data fitting. Raissi (2018) introduced another interesting approach whereby two neural network models are simultaneously trained to model the solution as well as the right-hand side of unknown partial differential equations.

Rudy, Kutz, and Brunton (2019) presented a new paradigm on system identification from noisy data. Rather than assuming ideal trajectories, they proposed simultaneously predicting the measurement noise on the training data, while learning the system dynamics. For the latter, they used multilayer feed-forward neural networks in a Runge-Kutta scheme, which allows dealing with non-uniform timesteps. They provided results on well-known autonomous dynamical systems including the chaotic Lorenz system and double pendulum. We focus on the use of general-purpose recurrent neural networks, which allow modeling input/output systems. Their ideas on predicting the measurement noise can however be applied in our setting, which provides an interesting future research direction.

Ayed et al. (2019) proposed a general model for continuous state-space dynamics, based on the adjoint state method, and with the explicit Euler scheme for stepping through time. As in most of the related work mentioned above, they focus on initial-value problems, while we target input/output systems. Importantly, their model is able to capture the entire state dynamics even if these are only partly observed. In our work, we also consider systems for which the entire state is not captured by the observations alone, an ideal setting for RNNs which maintain a hidden state through time.

[1] https://github.com/tdmeeste/TimeAwareRNN

The work by Zhu et al. (2017) is related to our work in the endeavor to extend RNN models with temporal information. They propose the Time-LSTM, engineered for applications in the context of recommendation engines. Its so-called 'time gates' are designed to model intervals over time, to capture users' short- and long-term interests.

The use of RNNs in modeling time series with missing data was explored by Che et al. (2018). More in particular, they leveraged 'informative missingness', the fact that patterns of missing values are often correlated with the target labels. They introduced the GRU-D, an extension of the GRU with a mask indicating which input variables are missing, as well as as a decay mechanism to model the fading influence of missing variables over time.

Recent Advances in Machine Learning The successful use of neural networks in the field of dynamical systems, has in turn inspired some recent developments in the area of deep learning.

E (2017) shared ideas on using dynamical systems to model high-dimensional nonlinear functions in machine learning, pointing out the link between the Euler scheme and deep residual networks.

Zhu and Fu (2018) started from the same link, in the area of image recognition, and suggested to extend the residual building blocks by multi-stage blocks, whereby suitable coefficients are learned jointly with the model (rather than relying on existing Runge-Kutta schemes).

Recently, Chen et al. (2018) introduced *Neural ODEs*, a new family of neural networks where the output is computed using a black-box ODE solver (which could involve explicit Runge-Kutta methods, or even more stable implicit methods). They provide proof-of-concept experiments on a range of applications. These include the 'continuous depth' setting, where ODE solvers (including a Runge-Kutta integrator) are shown to be able to replace residual blocks for a simple image classification task. They also discuss the 'continuous time' setting, and propose a generative time-series model. That model represents the considered time series, which may be unevenly spaced in time, as latent trajectories. These are initialized by encoding the observed time series using an RNN, and can be generated by the ODE solver at arbitrary points forwards or backwards in time. Training of the generative model is done as a variational auto-encoder. The complementary value of our work is in the fact that we provide an input/output dynamical systems formulation, and we believe that our extension from discrete RNNs to time-aware RNNs for dynamical systems makes it straightforward to apply.

Chang et al. (2019) introduced the Antisymmetric RNN (henceforth written ASRNN), again based on knowledge of the Euler scheme for dynamical systems. The use of an anti-symmetric hidden-to-hidden weight matrix in the incremental recurrent scheme (as well as a small diffusion term), ensured stability of the system. They demonstrated their model's ability to capture long-term dependencies, for a number of image recognition tasks cast as sequence classification problems. What's more, the ASRNN model is naturally suited for modeling unevenly sampled time series:

while Chang et al. view the step size as a fixed hyperparameter to be tuned during model selection, it can just as well be used as the actual time step, which we will further describe in Section 3.3. We will adapt the ASRNN to become better suited for modeling stationary systems.

Finally, we want to point out that our use of the term 'higher-order' models, refers to the order of the corresponding Runge-Kutta method. The term 'higher order neural networks' has also been used for neural network layers with multiplicative, rather than additive, combinations of features at the input (Zhang 2012). Also, 'higher order recurrent neural networks' may refer to RNNs where the current hidden state has access to multiple previous hidden states, rather than only the last one.

3 Time-Aware RNNs

After a general treatment on adapting recurrent neural networks to deal with unevenly spaced data (Section 3.1), we show how to use such models in higher-order Runge-Kutta schemes (Section 3.2), and introduce the higher-order time-aware extensions for the GRU and ASRNN (Section 3.3).

3.1 RNNs and ODEs

An RNN that models a discrete sequence of N inputs $\{\boldsymbol{x}_n\}_{n=1}^N$ and outputs $\{\boldsymbol{y}_n\}_{n=1}^N$, can generally be described as follows

$$\begin{aligned} \boldsymbol{h}_n &= \boldsymbol{f}_{\text{RNN}}(\boldsymbol{x}_n, \boldsymbol{h}_{n-1}) \\ \boldsymbol{y}_n &= \boldsymbol{g}(\boldsymbol{h}_n) \end{aligned} \tag{1}$$

The quantity $\boldsymbol{h}_n$ is called the hidden state at time step n, and is obtained by combining the input $\boldsymbol{x}_n$ with the hidden state $\boldsymbol{h}_{n-1}$ from the previous time step through the cell function $\boldsymbol{f}_{\text{RNN}}$, which contains the recurrent cell's trainable parameters. The initial hidden state $\boldsymbol{h}_0$ can be fixed to the zero vector, or trained with the model parameters. The output function $\mathbf{g}$, which converts the hidden state $\boldsymbol{h}_n$ to the output $\boldsymbol{y}_n$, is typically a basic neural network designed for classification or regression, depending on the type of output data. The nature of the recurrent network is determined by $\boldsymbol{f}_{\text{RNN}}$. It could be for instance an Elman RNN (Elman 1990) or a GRU. Variations to Eq. (1) are possible, for example for an LSTM where an additional internal cell state is passed between time steps.

Now consider an unknown continuous-time dynamical system, with inputs $\boldsymbol{x}(t)$ and outputs $\boldsymbol{y}(t)$ at time t. These do not necessarily cover the entire state space, as argued in (Ayed et al. 2019). We therefore explicitly introduce the state variable $\boldsymbol{h}(t)$ which, when observed at one point in time, allows determining the future system behavior, provided the system equations and future inputs are known. Many dynamical systems in science and technology can be described by an n'th order ordinary differential equation (ODE) (Coddington and Levinson 1955). Assuming this holds for the considered system, its system equations can be reduced to a first-order ODE in the n-dimensional state variable $\boldsymbol{h}$

$$\begin{aligned} \frac{d\boldsymbol{h}(t)}{dt} &= \boldsymbol{F}\big(\boldsymbol{x}(t), \boldsymbol{h}(t)\big) \\ \boldsymbol{y}(t) &= \boldsymbol{G}(\boldsymbol{h}(t)) \end{aligned} \tag{2}$$

whereby $\mathbf{G}$ represents a mapping from the latent state space to the output space. We assume time-invariant systems, i.e., the system equations only indirectly depend on the time, through the state $\boldsymbol{h}(t)$ and a potential source $\boldsymbol{x}(t)$.

If we discretize time into a sequence $\{t_n\}_{n=0}^N$, with potentially irregular step sizes $\delta_n = (t_{n+1} - t_n)$, and make use of the differential form of the derivative over a single time step, we can discretize the system equations as

$$\begin{aligned} \boldsymbol{h}_{n+1} &= \boldsymbol{h}_n + \delta_n \boldsymbol{F}\big(\boldsymbol{x}_n, \boldsymbol{h}_n\big) \\ \boldsymbol{y}_{n+1} &= \boldsymbol{G}(\boldsymbol{h}_{n+1}) \end{aligned} \tag{3}$$

in which $\boldsymbol{h}_n$ is shorthand for $\boldsymbol{h}(t_n)$ and similarly for the other quantities.

Because of the similarities between the ODE discretization formulation (Eq. (3)) and the standard RNN formulation (Eq. (1)), we will be able to use RNNs to model the system equations from the considered unknown system. However, there are two important differences between both formulations.

Predicting 'current' vs. 'next' output First of all, where the RNN allows predicting the output $\boldsymbol{y}_n$ from the input $\boldsymbol{x}_n$ at the *same* position in the sequence (combined with the previous state $\boldsymbol{h}_{n-1}$), Eq. (3) only allows predicting the output $\boldsymbol{y}_{n+1}$ at the *next* time step t_{n+1}, given the input and state at the current time step t_n. For evenly sampled data, shifting the entire output sequence over one position still allows training a model that predicts the output $\boldsymbol{y}_n$ at the same point in time as the input $\boldsymbol{x}_n$ while using a valid ODE discretization scheme. This is no longer possible for unevenly spaced data. However, our preliminary experiments on the considered datasets (see Section 4) show very little difference in prediction effectiveness, if we train a model that predicts either $\boldsymbol{y}_{n+1}$ or $\boldsymbol{y}_n$ from $\boldsymbol{x}_n$ (in the evenly spaced case). Yet, it should be implemented with care in the case of irregular spacing, to avoid ending up with incorrect discretization schemes.

Stationarity Secondly, the trainable function $\boldsymbol{F}$ in Eq. (3) is only responsible for the 'residual' part besides $\boldsymbol{h}_n$ when calculating $\boldsymbol{h}_{n+1}$. In other words, the term $\boldsymbol{h}_n$ acts as a skip connection. Such skip connections form a key element in deep residual neural networks which have been highly successful for image recognition tasks (He et al. 2016). However, in the time dimension they can have unwanted side effects. Naively extending a standard RNN with cell function $\boldsymbol{f}_{\text{RNN}}$ to deal with uneven time steps according to the Euler scheme, yields the following update equation

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \delta_n \boldsymbol{f}_{\text{RNN}}(\boldsymbol{x}_n, \boldsymbol{h}_n) \tag{4}$$

Consider for example an LSTM, its cell function bounded between -1 and $+1$. When starting from a zero initial state, the bounds of the state $\boldsymbol{h}_n$ after n time steps become $\pm \sum_{\nu=0}^{n-1} \delta_\nu$, or approximately $\pm\, n\, \mu_\delta$, with μ_δ the average step size. This does not necessarily lead to numerical problems, but a model with a potentially linearly growing state is ill-suited for modeling stationary time series. As will be demonstrated in Section 4, it leads to sub-optimal results.

A similar problem arises for the well-studied discrete-time stochastic process of first order defined by

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \epsilon_n \tag{5}$$

with ϵ_n a serially uncorrelated zero-mean stochastic process with constant variance. The 'unit root' is a solution to the process' characteristic equation. It leads to a trend in the mean, and hence a non-stationary process (Guidolin and Pedio 2018). However, the 'differenced time series' $\Delta\boldsymbol{h}_{n+1} = \boldsymbol{h}_{n+1} - \boldsymbol{h}_n = \epsilon_n$ no longer has this unit root, and is stationary.

Our update equation (4) similarly suffers from the unit root problem. We can therefore apply the idea of the differenced time series, but at the same time need to adhere to the general scheme (3) to correctly deal with variable sample intervals. We hence propose to use the following function $\boldsymbol{F}$ in system equation (3)

$$\boldsymbol{F}\big(\boldsymbol{x}, \boldsymbol{h}\big) = \frac{1}{\mu_\delta}\big(\boldsymbol{f}_{\text{RNN}}(\boldsymbol{x}, \boldsymbol{h}) - \boldsymbol{h}\big) \tag{6}$$

with μ_δ again denoting the average step size. The first-order update equation for the state can then be written as

$$\boldsymbol{h}_{n+1} = \big(1 - \frac{\delta_n}{\mu_\delta}\big)\boldsymbol{h}_n + \frac{\delta_n}{\mu_\delta}\boldsymbol{f}_{\text{RNN}}(\boldsymbol{x}_n, \boldsymbol{h}_n)$$

The unit root introduced by the term $\boldsymbol{h}_n$ in Eq. (4) has indeed disappeared because the expected value of $(1-\delta_n/\mu_\delta)$ is zero. Note that unlike ϵ_n in Eq. (5), the RNN cell function in Eq. (4) is no zero-mean stochastic process, nor are its consecutive outputs uncorrelated. There is therefore no theoretical guarantee that the resulting model will be stationary. However, the neural network no longer has to explicitly 'learn' how to compensate for the unit root problem induced by the skip connection, and we hypothesize that it is therefore more naturally suited to deal with stationary data. For simplicity, in the remainder this will be called the 'stationary' formulation, in contrast to the incremental models with the unit root, for simplicity denoted as the 'non-stationary' one.

Combining equations (3) and (6) results in an *extension* of the standard RNN scheme towards unevenly spaced time series, in the sense that for evenly spaced samples ($\delta_n = \mu_\delta$) they reduce to the standard RNN in Eq. (1), or strictly speaking, one applied to predicting $\boldsymbol{y}_{n+1}$ from $\boldsymbol{x}_n$ and $\boldsymbol{h}_n$.

Note that the sample rate at inference time might not entirely correspond to μ_δ, leading to a remnant of the unit root effect. Our experimentation indicates that this is less of a problem than for instance the presence of outliers (i.e., occasionally very large gaps between consecutive samples).

3.2 Higher Order Neural Sequence Models

A well-known family of higher order iterative ODE methods are the explicit Runge-Kutta methods (Butcher 2016). Applying an s-stage Runge-Kutta method to the ODE in Eq. (2) leads to the update equation

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \delta_n \sum_{i=1}^{s} b_i\, \boldsymbol{k}_i \tag{7}$$

Table 1: Non-zero coefficients of selected explicit Runge-Kutta methods.

NAME (order)	c_2	c_3	c_4	b_1	b_2	b_3	b_4	a_{21}	a_{31}	a_{32}	a_{43}
EULER (1)				1							
MIDPOINT (2)	1/2				1			1/2			
KUTTA3 (3)	1/2	1		1/6	2/3	1/6		1/2	-1	2	
RK4 (4)	1/2	1/2	1	1/6	1/3	1/3	1/6	1/2		1/2	1

where $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_s$ are found recursively over s stages, as

$$\boldsymbol{k}_1 = \boldsymbol{F}\big(\boldsymbol{x}_n, \boldsymbol{h}_n\big)$$

$$\boldsymbol{k}_i = \boldsymbol{F}\Big(\boldsymbol{x}(t_n + c_i\delta_n), \boldsymbol{h}_n + \delta_n \sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j\Big),\ i \in \{2, \ldots, s\}$$

for predefined values of the coefficients b_i, c_i, and a_{ij}. The error in predicting $\boldsymbol{h}_{n+1}$ from a correct $\boldsymbol{h}_n$ is called the *local truncation error*. A Runge-Kutta method of order p denotes a method where the local truncation error is of the order $\mathcal{O}(\delta^{p+1})$. The coefficients for a number of well-known methods are listed in Table 1. For these methods, the number of stages equals their order. The update scheme initially proposed in Eq. (3) corresponds to Euler's method (EULER), the simplest Runge-Kutta method. We further consider the second-order Explicit Midpoint method (MIDPOINT), Kutta's third order method (KUTTA3), and the classical fourth-order Runge-Kutta method (RK4).

If the function $\boldsymbol{F}$ is differentiable, the model can be trained by backpropagating the gradient of the loss on the predicted output through the considered sequence, and within each time step, through the stages of the considered Runge-Kutta update scheme. Existing RNN models $\boldsymbol{f}_{\text{RNN}}$ can hence be directly extended towards higher-order models that are suited for unevenly spaced data, by combining the proposed function $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{h})$ in Eq. (6) with the Runge-Kutta update equation (7).

Some recent works have already applied numerical ODE methods in combination with neural networks to model dynamical systems (Chen et al. 2018; Rudy, Kutz, and Brunton 2019). However, they do not investigate input/output systems, the focus of this work. An important restriction to the use of higher-order methods for input/output systems, is the fact that the inputs $\boldsymbol{x}$ may be only available under the form of samples $\{\boldsymbol{x}(t_n)\}_{n=1}^N$. A valid Runge-Kutta update scheme however requires evaluating the input $\boldsymbol{x}(t_n + c_i\delta_n)$ at intermediate points in time (as per Eq. (7)). As shown experimentally in Section 4, this can be achieved through interpolation, but the approximation may counteract the effectiveness of higher-order methods. Another approach is to build a generative higher-order model for the input as well, which will be explored in future work.

3.3 Time-Aware GRU and ASRNN

To make the results from the previous sections more tangible, we write out the proposed time-aware extension for the GRU cell and for the ASRNN.

The cell function $\boldsymbol{f}_{\text{GRU}}$ for the GRU (Cho et al. 2014),

cast for the formulation $\boldsymbol{h}_{n+1} = \boldsymbol{f}_{\text{GRU}}(\boldsymbol{x}_n, \boldsymbol{h}_n)$, is given by

$$\boldsymbol{f}_{\text{GRU}} = (1 - \boldsymbol{z}_n) \odot \tanh\Big(W_h \boldsymbol{x}_n + U_h(\boldsymbol{r}_n \odot \boldsymbol{h}_n) + b_h\Big) + \boldsymbol{z}_n \odot \boldsymbol{h}_n$$

in which $\odot$ represents elementwise multiplication. The auxiliary vectors $\boldsymbol{z}_n$ and $\boldsymbol{r}_n$ are called the 'gates'

$$\boldsymbol{z}_n = \sigma(W_z \boldsymbol{x}_n + U_z \boldsymbol{h}_n + b_z)$$
$$\boldsymbol{r}_n = \sigma(W_r \boldsymbol{x}_n + U_r \boldsymbol{h}_n + b_r)$$

with $\sigma(.)$ the sigmoid function. With the hidden state dimension k_h and input size k_x, the trainable parameters are given by the weight matrices $W_h, W_z, W_r \in \mathbb{R}^{k_h \times k_x}$, $U_h, U_z, U_r \in \mathbb{R}^{k_h \times k_h}$, and biases $b_h, b_z, b_r \in \mathbb{R}^{k_h}$. Applying Eq. (6) to avoid the unit root, yields for the time-aware GRU

$$\begin{aligned}\boldsymbol{F}(\boldsymbol{x}_n, \boldsymbol{h}_n) &= \frac{1}{\mu_\delta}\big(\boldsymbol{f}_{\text{GRU}}(\boldsymbol{x}_n, \boldsymbol{h}_n) - \boldsymbol{h}_n\big) \\ &= \frac{\tilde{\boldsymbol{z}}_n}{\mu_\delta} \odot \Big(\tanh\big(W_h \boldsymbol{x}_n + U_h(\boldsymbol{r}_n \odot \boldsymbol{h}_n) + b_h\big) - \boldsymbol{h}_n\Big)\end{aligned}$$

whereby for convenience $(1 - \boldsymbol{z}_n)$ is replaced by a new gate $\tilde{\boldsymbol{z}}_n$. The function $\boldsymbol{F}(\boldsymbol{x}_n, \boldsymbol{h}_n)$ is to be used with Eq. (3) for the first-order scheme, or with Eq. (7) for its higher-order counterparts. These equations retain the expressiveness and amount of trainable parameters from the original GRU cell, but remain valid for unevenly spaced data without inducing the unit root problem, and can be used in higher-order schemes. In Section 4, the non-stationary formulation from Eq. (4), i.e., $\boldsymbol{F} = \boldsymbol{f}_{\text{GRU}}$, will be used as a baseline, to underline the importance of avoiding the unit root.

As a second example, we consider the gated ASRNN introduced by Chang et al. (2019). Its cell function $\boldsymbol{f}_{\text{ASRNN}}$ can be written as

$$\begin{aligned}&\boldsymbol{f}_{\text{ASRNN}} = \boldsymbol{z}_n \odot \tanh\big(W_h \boldsymbol{x}_n + A\,\boldsymbol{h}_n + b_h\big) \\ &\text{with the gate } \boldsymbol{z}_n = \sigma\big(W_z \boldsymbol{x}_n + A\,\boldsymbol{h}_n + b_z\big)\end{aligned}$$

The hidden-to-hidden matrix $A \in \mathbb{R}^{k_h \times k_h}$ can be written as $A = (W_h - W_h^T - \gamma I)$. It corresponds to an anti-symmetric matrix (i.e., the difference between a weight matrix W_h and its transpose W_h^T), with a small negative value on the diagonal (indicated by the non-negative 'diffusion' parameter γ and unit matrix I) to ensure stability. The original ASRNN formulation follows the incremental (i.e., non-stationary) first-order formulation, with the step size ϵ as a hyperparameter for weighting the residual term in the state update equation. We replace it by the actual step size δ_n / μ_δ (normalized by its mean) to deal with uneven sample times, and keep the scaling factor ϵ for tuning the model:

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \epsilon\, \frac{\delta_n}{\mu_\delta} \boldsymbol{f}_{\text{ASRNN}}(\boldsymbol{x}_n, \boldsymbol{h}_n) \tag{8}$$

Its stationary counterpart follows from Eq. (3) and Eq. (6)

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \frac{\delta_n}{\mu_\delta}\big(\epsilon\, \boldsymbol{f}_{\text{ASRNN}}(\boldsymbol{x}_n, \boldsymbol{h}_n) - \boldsymbol{h}_n\big) \tag{9}$$

with straightforward extension to higher-order schemes.

4 Experimental Validation

This section presents experimental results on two input/output datasets. The main research questions we want to investigate are (i) what is the impact of unevenly sampled data on prediction results with standard RNNs vs. with their time-aware extension, (ii) how important is the use of state update schemes that avoid the unit root issue, (iii) is there any impact from applying the time-aware models with higher-order schemes, and (iv) how are the output predictions affected while applying such higher-order schemes based only on *sampled* inputs?

After describing the selected datasets (Section 4.1), the model architecture, and the training and evaluation setup (Section 4.2), we describe the experiments and discuss the results (Section 4.3).

4.1 Datasets

We have selected two datasets from STADIUS's Identification Database "DaISy" (De Moor et al. 1997), a well-known database for system identification.

CSTR Dataset We use the *Continuous Stirred Tank Reactor* (CSTR) dataset[2]. It contains evenly sampled observations (10 samples per minute) from a model of an exothermic reaction in a continuous stirred tank reactor. There is a single piecewise constant input signal, the coolant flow, and two output signals, the resulting concentration and temperature. The data was first studied by Lightbody and Irwin (1997), who introduced a basic neural network model in the context of adaptive control of nonlinear systems. It consists of a sequence of in total 7,500 samples, of which we used the first 70% for training, the next 15% for validation, and the last 15% for testing. We used the original dataset for an evenly spaced baseline model, and generated a version with missing data, by randomly leaving out samples with a probability of $p_{\text{missing}} = 0.50$. The average interval between consecutive samples is doubled ($\mu_\delta = 0.2$ minutes), but the data now contains gaps up to 13 times the original gap ($\delta_n \in [0.1, 1.3]$). We normalize the input and outputs, by subtracting their respective mean value in the training data, and dividing by the standard deviation.

Winding Dataset We also use the data from a *Test Setup of an Industrial Winding Process*[3] (Winding). It contains a sequence of 2,500 evenly sampled measurements (10 samples per second). The test setup consists of 3 reels, the 'unwinding reel' from which a web is unwinded, after which it goes over a traction reel, and is rewinded onto the 'rewinding reel'. The inputs are the angular speed of the 3 reels, as well as the setpoint current of the motors for the unwinding and rewinding reel, i.e., five inputs in total. The two outputs correspond to measurements of the web's tension in between

[2] ftp://ftp.esat.kuleuven.be/pub/SISTA/data/process_industry/cstr.dat.gz

[3] ftp://ftp.esat.kuleuven.be/pub/SISTA/data/process_industry/winding.dat.gz

the reels. The data was introduced and first studied by Bastogne et al. (1997). We again created an artificial unevenly sampled data sequence based on real-world data, by randomly leaving out samples with probability $p_{\text{missing}} = 0.50$. We used the first 70% of the input sequence for training, then 15% for development, and the last 15% for testing.

4.2 Experimental Sfinaletup

The overall goal of this work is to demonstrate how standard RNNs can be applied to unevenly sampled data from input/output models. To keep the approach generic, we use the same overall architecture and training procedure for both datasets.

Model Architecture Modeling the data using a recurrent network with the same input dimension k_x as the number of observed system inputs, appeared not sufficient. We therefore use RNN cell functions with a potentially higher input size k and the same state size, and feed it with the observed input data (1 dimension for CSTR, 5 for Winding) extended to k dimensions by applying a trainable linear mapping from the inputs to k dimensions, followed by a $\tanh$ non-linearity. As output function $\mathbf{G}(\boldsymbol{h})$ (from Eq. (3)) we use a trainable linear mapping from k state dimensions to the observed output space (2 dimensions for both datasets).

Evaluation We report the root relative squared error (RRSE) averaged over the k_{out} output channnels. Being a relative metric, it allows comparing results between different models and datasets. The RRSE is defined as (Botchkarev 2018)

$$\text{RRSE} = \frac{1}{k_{\text{out}}} \sum_{i=1}^{k_{\text{out}}} \sqrt{\frac{\sum_{n=1}^{N} \left(\hat{y}_n^{(i)} - y_n^{(i)}\right)^2}{\sum_{n=1}^{N} \left(\hat{y}_n^{(i)} - \mu_y^{(i)}\right)^2}}$$

in which $\hat{y}_n^{(i)}$ represents the predicted test value for channel i at time step n, whereas $y_n^{(i)}$ is the corresponding ground truth value, with $\mu_y^{(i)} = \frac{1}{N} \sum_n y_n^{(i)}$ the channel mean.

Per output channel, the RRSE can be interpreted as the root mean squared (RMS) error of the prediction, normalized by the RMS error of the channel's average as a baseline. In other words, for an RRSE value of 100%, the model performs no better than predicting the mean.

The test value is obtained by evaluating the model through the entire sequence, to ensure that the test sequence receives the appropriate initial state, and calculating the RRSE on the test sequence only. All reported values are the mean (and standard deviation) over five training runs starting from a different random initialization of the network.

Training We train the model with backpropagation-through-time (Werbos 1988) over 20 time steps, and perform the optimization in parallel over mini-batches of (possibly overlapping) segments of 20 time steps. The hidden state $\boldsymbol{h}_0$ at the start of the entire sequence is randomly initialized, and trained with the model. After each training epoch over all segments, a forward pass through the entire train sequence is performed, and the resulting states are used as initial state for the corresponding training segments (i.e., we used the training 'scheme 4' as introduced by De Boom, Demeester, and Dhoedt (2018)). We minimize the mean squared error of the predicted outputs using the Adam optimizer (Kingma and Ba 2014), and apply early stopping by measuring the RRSE on the validation sequence.

Table 2: Hyperparameters tuned over ranges $b \in \{64, 512\}$, $k \in \{5, 10, 20, 30, 40, 60, 80, 100, 150\}$, and $\lambda \in \{0.001, 0.003, 0.01\}$.

hyperparameter	CSTR		Winding	
	GRU	ASRNN	GRU	ASRNN
minibatch size b	512	512	512	64
state size k	20	100	10	10
learning rate λ	0.001	0.001	0.003	0.01

In order to investigate whether the proposed models work out of the box, rather than requiring substantial tuning, we only tune the baseline GRU and ASRNN models, without missing data. The same hyperparameters are then adopted for the experiments in the uneven sampling setting. They are shown in Table 2.

During our preliminary experiments, we noticed that applying regularization through dropout gave higher prediction errors. Given the small amount of training data (i.e., a single sequence of a few thousand measurements), our hypothesis is that applying dropout while allowing for larger numbers of trainable parameters did not allow to better capture the system dynamics, while it made training more difficult. We therefore chose to tune the model complexity only through the hidden state size k.

4.3 Results and Discussion

Baselines The test error for the GRU and ASRNN baselines without missing data are shown in Table 3 (top two lines). As mentioned above, the hyperparameters from Table 2 are tuned on these. For both models, the formulation without unit root is used. For the GRU, this comes down to its standard formulation, whereas for the ASRNN, the stationary variant in Eq. (9) is used, which for evenly sampled data reduces to $\boldsymbol{h}_{n+1} = \epsilon\, \boldsymbol{f}_{\text{ASRNN}}(\boldsymbol{x}_n, \boldsymbol{h}_n)$.

Overall it can be seen that the prediction error for the CSTR data is much lower than for the Winding data. This is likely due to properties of the datasets. The CSTR dataset contains smooth simulation results, and has a relatively higher sample rate with respect to changes in the signal compared to the Winding data, which consists of actual measurements.

We created a number of baselines for the data with missing samples as well, also shown in Table 3. When the standard GRU is applied, ignoring the missing data, there is a substantial increase of the error ('standard, ignore missing' in Table 3). The results are not dramatic, though. We hypothesize that the model learns, to some extent, to compensate in its output for sudden larger gaps in the input (due to larger temporal gaps). From that perspective, it does make sense to

Table 3: Baseline results with missing data. Displaying test RRSE values in percentage points (mean $\pm$ std).

Model	CSTR	Winding
Baselines on all original samples		
GRU (standard)	$\mathbf{2.2 \pm 0.5}$	$\mathbf{20.1 \pm 0.6}$
ASRNN (stationary)	2.5 ± 0.2	27.4 ± 3.9
Baselines with missing data		
GRU (standard, ignore missing)	10.8 ± 0.6	30.2 ± 0.4
GRU (standard, extra input δ_n)	$\mathbf{9.2 \pm 1.4}$	$\mathbf{28.9 \pm 1.7}$
GRU (time-aware, non-station.)	79.7 ± 8.0	64.8 ± 10.2
ASRNN (time-aware, non-station.)	12.3 ± 1.1	41.5 ± 7.4

Table 4: Stationary time-aware higher-order GRU with constant vs. linear input interpolation for the datasets with missing data. Displaying test RRSE values in percentage points (mean $\pm$ std).

Scheme	CSTR	Winding	
	constant	constant	linear
EULER	12.1 ± 1.3	33.1 ± 0.6	33.1 ± 0.6
MIDPOINT	11.0 ± 4.1	35.3 ± 2.1	28.2 ± 1.8
KUTTA3	9.9 ± 4.7	34.7 ± 4.3	27.1 ± 1.0
RK4	$\mathbf{8.0 \pm 0.4}$	32.2 ± 6.6	$\mathbf{25.6 \pm 1.4}$

apply a standard RNN to unevenly sampled data. A straightforward way to augment standard RNN models with variable time steps, is by providing the step size δ_n (normalized) as an additional input signal to the model. This reduces the error by a few percentage points ('standard, extra input δ_n' in the table). More advanced models, in line with (Zhu et al. 2017), may provide an even better alternative.

Finally, we applied the incremental Euler scheme of Eq. (4) without compensation of the unit root, both for the GRU and the ASRNN (indicated as 'time-aware, non-station.'). This leads to an increased error, confirming the hypothesis from Section 3.1 that the presence of the unit root negatively affects the modeling of stationary data. Note however that the ASRNN seems less affected than the incremental GRU model. The step size ϵ and the diffusion parameter γ from the original ASRNN were both set to 1.0 for the baseline without missing data, but were now tuned over the same values as in (Chang et al. 2019), without much improvement.

Time-aware higher-order models The results for the stationary time-aware higher-order GRU model are shown in Table 4. The 'constant' vs. 'linear' input interpolation shown in the table is related to the issue identified in Section 3.2 that correct higher-order schemes require inputs evaluated in between samples, i.e., $\boldsymbol{x}(t_n + c_i\delta_n)$, with c_i depending on the specific Runge-Kutta scheme. For the CSTR data, the constant approximation $\boldsymbol{x}(t_n + c_i\delta_n) \approx \boldsymbol{x}_n$ is sufficient as the inputs are piecewise constant over several time steps, and higher-order schemes lead to lower errors. However, this is not the case for the Winding dataset, where some of the inputs correspond to continuous variables. The results in Table 4 indeed show that higher-order schemes are not beneficial when the inputs are assumed piecewise constant within each sample interval (column 'constant' for the Winding data). However, with a simple linear interpolation between consecutive inputs $\boldsymbol{x}(t_n + c_i\delta_n) \approx (1 - c_i)\,\boldsymbol{x}_n + c_i\,\boldsymbol{x}_{n+1}$, it seems the output error again decreases for the tested higher-order schemes. More advanced interpolation methods may be more suited still, but were considered out of scope for this work.

For both datasets, the EULER scheme performs a few percentage points worse than the standard GRU where the time steps are not explicitly encoded (see baseline 'ignore missing' in Table 3). This might be related to the absence of hyperparameter tuning for the time-aware models. However, increasing the order leads to gradually lower test errors for the time-aware methods with appropriate input interpolation. The 4'th order RK4 method leads to the overall lowest error for the missing data setting, even without tuning.

Due to space constraints, Table 4 only shows time-aware results for the stationary GRU. Note that the corresponding ASRNN errors would remain slightly higher, consistent with the baselines. Also, the non-stationary counterpart of Table 4 would show that the presence of the unit root annihilates the positive effect of higher-order schemes entirely.

Summary We now shortly look back at the research questions formulated at the start of this section. In our setting, randomly leaving out training and test samples from a sequence of input/output system measurements leads to an increase in output prediction error. However, standard RNNs can still make meaningful predictions, especially when the temporal information is explicitly provided as a feature. For the datasets under study, the proposed time-aware higher-order schemes have the potential to compensate even better for the missing data. Eliminating the unit root appears however important when applying a standard RNN cell in an incremental update scheme. Furthermore, for data with continuously valued input samples, the use of higher-order schemes only makes sense if a proper interpolation in the input space is performed.

5 Conclusions and Future Research Ideas

This paper focused on using neural sequence models for input/output system identification from unevenly spaced observations. We showed how to extend standard recurrent neural networks to naturally deal with unevenly spaced data by augmenting the update scheme with the local step size in a way that allows modeling stationary dynamical systems, and showed how the resulting model can be used in higher-order Runge-Kutta schemes.

We provided experimental results for two different input/output system datasets where we experimented with the impact of randomly leaving out data samples. Applying the time-aware model in higher-order schemes, gave better output predictions compared to ignoring the uneven sample times. The direct extension of RNNs with the incremental Euler scheme to correctly account for uneven sample times appeared to give inferior results. We hypothesized that this was due to the unit root problem, leading to an inherently

non-stationary model, and showed how to avoid that problem.

Future research includes looking into more complex input/output systems with non-uniform noisy data. One potential research direction could involve the application of adaptive sampling schemes during forecasting. For example, the introduced models could be readily used with the Runge-Kutta-Fehlberg methods with adaptive step size (Fehlberg 1969) in a computationally efficient way. A further promising research direction is in extending the proposed techniques for dynamical systems to generative sampling models for time series as proposed by Chen et al. (2018). A potentially interesting application domain would be in robotics, where light-weight dynamical system models with adaptive sample times could be of interest in terms of computational efficiency.

Acknowledgments

This research received funding from the Flemish Government under the *"Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen"* programme. I am grateful to Cedric De Boom for his feedback on the paper, and to the anonymous reviewers for their useful suggestions.

References

Ayed, I.; de Bézenac, E.; Pajot, A.; Brajard, J.; and Gallinari, P. 2019. Learning dynamical systems from partial observations. *arXiv:1902.11136*.

Bastogne, T.; Noura, H.; Richard, A.; and Hittinger, J.-M. 1997. Application of subspace methods to the identification of a winding process. *1997 European Control Conference (ECC)* 2168–2173.

Botchkarev, A. 2018. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv:1809.03006*.

Butcher, J. C. 2016. *Numerical methods for ordinary differential equations, 3rd edition*. Wiley.

Chang, B.; Chen, M.; Haber, E.; and Chi, E. H. 2019. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations (ICLR 2019)*.

Che, Z.; Purushotham, S.; Cho, K.; Sontag, D.; and Liu, Y. 2018. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports* 8(1):6085.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS 2018)*, 6571–6583.

Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, (EMNLP 2014)*, 1724–1734.

Coddington, E. A., and Levinson, N. 1955. *Theory of ordinary differential equations*. New York: McGraw-Hill.

De Boom, C.; Demeester, T.; and Dhoedt, B. 2018. Character-level recurrent neural networks in practice: comparing training and sampling schemes. *Neural Computing and Applications*.

De Moor, B.; De Gersem, P.; De Schutter, B.; and Favoreel, W. 1997. DAISY: A database for identification of systems. *Journal A* 38(3):4–5.

E, W. 2017. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics* 5(1).

Eckner, A. 2014. A framework for the analysis of unevenly spaced time series data. Technical report, working paper.

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

Fehlberg, E. 1969. Low-order classical runge-kutta formulas with step size control and their application to some heat transfer problems. Technical report, NASA, United States.

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

Guidolin, M., and Pedio, M. 2018. Chapter 4 - unit roots and cointegration. In Guidolin, M., and Pedio, M., eds., *Essentials of Time Series for Financial Applications*. Academic Press. 113 – 149.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

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

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Lightbody, G., and Irwin, G. W. 1997. Nonlinear control structures based on embedded neural system models. *IEEE Transactions on Neural Networks* 8(3):553–567.

Ljung, L. 2013. *System Identification: An Overview*. London: Springer London. 1–20.

Raissi, M., and Kariadakis, G. E. 2018. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics* 125–141.

Raissi, M. 2018. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *J. Mach. Learn. Res.* 19(1):932–955.

Rudy, S. H.; Brunton, S. L.; Proctor, J. L.; and Kutz, J. N. 2017. Data-driven discovery of partial differential equations. *Science Advances* 3(4).

Rudy, S. H.; Kutz, J. N.; and Brunton, S. L. 2019. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics* 396:483 – 506.

Wang, Y.-J., and Lin, C.-T. 1998. Runge-kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks* 9(2):294–307.

Werbos, P. J. 1988. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks* 1(4):339 – 356.

Young, T.; Hazarika, D.; and Poria, S. 2017. Recent trends in deep learning based natural language processing. *arXiv:1708.02709*.

Zhang, M. 2012. *Artificial Higher Order Neural Networks for modeling and simulation*. IGI Global.

Zhu, M., and Fu, C. 2018. Convolutional neural networks combined with runge-kutta methods. *arXiv:1802.08831*.

Zhu, Y.; Li, H.; Liao, Y.; Wang, B.; Guan, Z.; Liu, H.; and Cai, D. 2017. What to do next: Modeling user behaviors by time-LSTM. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, 3602–3608. AAAI Press.