Kaunas University of Technology

Faculty of Informatics

Recurrent Neural Network Approaches to Irregularly Sampled Data

Master’s Final Degree Project

Erik Schake

Project author

Assoc. Prof. Mantas Lukoševičius

Supervisor

Kaunas, 2022

Kaunas University of Technology

Faculty of Informatics

Erik Schake

Recurrent Neural Network Approaches to Irregularly Sampled Data

Declaration of Academic Integrity

I confirm the following:

1. I have prepared the final degree project independently and honestly without any violations of the copyrights or other rights of others, following the provisions of the Law on Copyrights and Related Rights of the Republic of Lithuania, the Regulations on the Management and Transfer of Intellectual Property of Kaunas University of Technology (hereinafter – University) and the ethical requirements stipulated by the Code of Academic Ethics of the University;

2. All the data and research results provided in the final degree project are correct and obtained legally; none of the parts of this project are plagiarised from any printed or electronic sources; all the quotations and references provided in the text of the final degree project are indicated in the list of references;

3. I have not paid anyone any monetary funds for the final degree project or the parts thereof unless required by the law;

4. I understand that in the case of any discovery of the fact of dishonesty or violation of any rights of others, the academic penalties will be imposed on me under the procedure applied at the University; I will be expelled from the University and my final degree project can be submitted to the Office of the Ombudsperson for Academic Ethics and Procedures in the examination of a possible violation of academic ethics.

Erik Schake

Confirmed electronically

Schake, Erik. Recurrent Neural Network Approaches to Irregularly Sampled Data. Master's Final Degree Project.
Supervisor Assoc. Prof. Mantas Lukoševičius; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Computer Sciences, Software Systems.

Keywords: artificial intelligence, neural differential equations, echo state networks, recurrent neural networks, time series.

Kaunas, 2022. 71 p.p.

Summary

Sequence data is a common pattern in real world observational datasets. The extra temporal dimension poses new challenges when applying deep learning techniques. Research leaps were made possible with the introduction of Long-Short Term Memory cells, as well as Gated Recurrent Units. However, these classical sequence data models all are based on the implicit assumption that sequences are sampled in regular intervals and are fully observed. In real world scenarios however, this is rarely the case. Most time series are irregularly sampled and often only partially observed.
It must therefore be the goal to have a universal model capable of handling arbitrary sequential data at low computational cost. Most of the proposed approaches in literature are heavily over-engineered in order to cope with certain properties. While working on this thesis however, a new method was proposed that utilizes controlled differential equations (CDEs) to create a continuous time recurrent neural network that is indifferent to partial observations. In this thesis, I apply the idea of CDEs to create a continuous time echo state network, which inherits the advantageous properties that echo state networks have in comparison to standard recurrent neural networks. This is the main contribution of this work. As a secondary contribution, a non-conclusive analysis of the most popular and powerful methods from literature is given.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ANN Artificial Neural Network. 14–18, 26

CDE Controlled Differential Equation. 26, 45, 56–58

ESN Echo State Network. 10, 12, 19–22, 42, 43, 48–53, 55, 57–60, 63–65

GRU Gated Recurrent Unit. 12, 19, 20, 33, 34, 38, 40, 41

LSTM Long-Short Term Memory. 12, 19, 20, 38, 39

MAR Missing at Random. 29–31

MCAR Missing Data Completely at Random. 29–31

MLP Multi Layer Perceptron. 16–18, 23, 55, 60

NDE Neural Differential Equation. 14, 23, 46, 48, 49

Neural CDE Neural Controlled Differential Equation. 45–51, 55–60, 63–65

Neural ODE Neural Ordinary Differential Equation. 23, 25, 26, 41, 43–45, 47, 55, 56

NMAR Not Missing at Random. 30

ODE Ordinary Differential Equation. 23–25, 44–48, 50, 55, 57–59, 63

PDE Partial Differential Equation. 23

RNN Recurrent Neural Network. 9, 12, 14, 18–21, 23, 26–28, 33, 36, 38, 41, 42, 45, 46, 48–53, 55, 57, 58, 63

1. Introduction

Research in the area of time series analysis has recently accelerated rapidly as yet another type of neural network architecture has been proposed by Chen et al. [1] (Neural Ordinary Differential Equations, see Chapter 2.3). While universal approaches to real world sequence data were only sparsely researched up until then, the area itself is by no means a new field of research. However, we now have methods that can naturally handle the universal case of irregularly sampled and partially observed time series in an expressive and powerful way. This leap was possible by utilizing the ever more powerful means of deep learning capabilities to represent complex dynamical systems using mathematical concepts that have been researched for over a hundred years (cf. [2]).

This work follows this trend of research by providing a two-fold contribution. First, an outline over the rapidly unfolding lines of research is given. This part aims to be a reference of categorizations over different methods' capabilities. The second and main contribution thereafter is the proposal and empirical evaluation of a novel method for universal time series analysis. This method builds upon the new understandings we now have thanks to recent research.

The remainder of this work is organized as follows. This present chapter introduces and motivates the topic of research. Afterwards, Chapter 2 entails the relevant foundations and theoretical backgrounds which this thesis is based upon. Chapter 3 presents the first major contribution as it outlines related work and thereby serves as a reference over the different lines of research proposed in literature. The next chapter proceeds to outline the main contribution by introducing a new method, which is subsequently and empirically evaluated. Chapter 5 concludes this work.

1.1. Motivation

Living in a world of three spatial and one temporal dimension, temporal sequence data is only natural to us and occurs everywhere. Examples are meteorological developments, stock prices, the spoken word, medical data, or dynamical systems in physics. Even enterprise workflows can be represented as temporal sequences [3]. Abstractly speaking, any time series is an ordered sequence of observations. In this work, all observations are considered to be real-valued (i.e., every observation is represented as a vector of real numbers). We will extend this to a formal definition in the ensuing section.

When applying Recurrent Neural Network (RNN) approaches to such data, it is usually assumed that observations are collected at regular time intervals. RNNs are specifically designed to handle this well and have been shown repeatedly to outperform classical approaches (e.g. [4, 5]). However, in reality, data is rarely collected at regular intervals. This is either due to irregular intervals at which data is actually collected (e.g., medical patient examinations or manually dated data such as archaeological data), or due to the fact that data collection is coupled to events that happen irregularly (e.g. user actions or economic transactions). Most classical and deep learning methods fail at this task as they cannot adequately represent the temporal dynamics of irregular intervals. To deal with this, data is often re-sampled by binning or interpolating the original observations to end up with regular time intervals. However, this approach leads to massive amounts of lost information and is therefore not the correct way to go. It is much more preferable to employ a system that can internally handle irregular time intervals in a natural manner without making assumptions on the data in between observations or even throwing entire bits of information away.

Recent advances now are capable of representing continuous dynamics using deep learning technologies. However, they still suffer from drawbacks, such as increased computational costs. Furthermore, as only early stage research is available, empirical evaluations and extensions to the basic ideas are still being actively investigated.

Another motivating factor are the recent rapid advances in the field. Building on top of the collective knowledge that was only recently published and contributing to this field is intrinsic motivation in itself. Furthermore, the connections between deep learning advances and long known mathematical concepts are highly intriguing to me.

A third and last (but not least) motivational factor are the topics of research of my supervisor. Echo State Networks (ESNs) shine in that they require less training time and less computationally expensive operations to yield good results. Applying the ideas from recent research with the solid knowledge of ESNs to reduce computational costs is therefore very promising.

1.2. The Fundamental Problems

We may now further specify the definition of a time series. As motivated before, observations in time series data are usually not sampled at regular intervals. In order to gain a complete picture of the recorded sequence, we must record the timestamp together with each observation. Therefore, the ith observation $(t_i, x^{<t_i>})$ is comprised of the time of observation (its *timestamp*) $t_i \in \mathbb{R}$ and the actual observed data $x^{<t_i>} \in \mathbb{R}^{d_x}$. In the case of regularly sampled sequences of length n, each $t_i - t_{i-1}$ is constant for every $i = 1, .., n$. In this case we would not need to record the timestamp along with every observation, as the information is redundant. However, in the general case, the time in between observations is irregular, even chaotic, prompting us to record every timestamp. This insight might be even extended to individual dimensions. The individual intervals in between observations across dimensions are not necessarily synchronized. Medical surveillance data for instance is often recorded one by one (e.g., first the heart rate is measured before blood pressure). This notion is illustrated in Figure 1. Hence, we introduce $\varnothing$ as a symbol representing missing data and must redefine $x^{<t_i>}$ as $x^{<t_i>} \in (\mathbb{R} \cup \{\varnothing\})^{d_x}$.

We can now define a time series in the most universal case as follows:

Definition 1.1 (Time Series)**.** Let $t_i \in \mathbb{R}$, $x^{<t_i>} \in (\mathbb{R} \cup \{\varnothing\})^{d_x}$ with $d_x \in \mathbb{N}$ the dimensionality of the observed data and $n \in \mathbb{N}$ the number of observations. Then $TS = ((t_1, x^{<t_1>}), (t_2, x^{<t_2>})), \dots, (t_n, x^{<t_n>})$ with $t_1 < t_2 < \dots < t_n$ is a time series[1].

[1]The notation in this thesis will follow mainly [2]

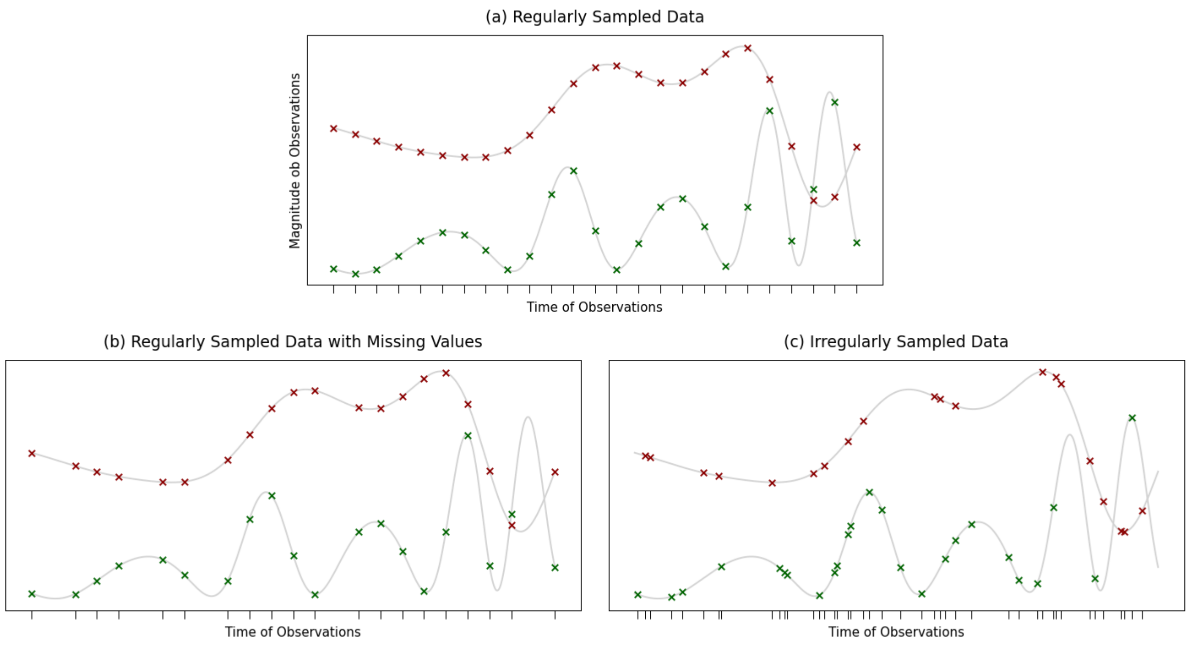


Figure 1. A taxonomy of time series sampling results. In (a) data is observed in regular intervals. In (b) some observations are missing, while intervals in between observations in (c) are completely chaotic. Notice the depiction of two dimensions, where times of observation across dimensions do not line up in (c).

This work focuses on three fundamental problems when dealing with multivariate universally sampled time series data. The methods proposed in literature will be classified in Chapter 3 with regard to their capabilities to cope with these problems. Chapter 4 proposes a novel method that is capable of processing sequences independently of how they have been sampled. Outlining methods – both from literature and new in this work – that can naturally handle universally sampled time series is the main contribution of this thesis. It must be stressed that only methods that can handle all three problems naturally can yield results based on all available information. If only one problem cannot be dealt with, this means that some information has to be disregarded.

The three fundamental practical issues to consider when processing irregularly sampled and partially observed data are as follows:

1. **Irregular Sampling.**
   As stated, the intervals $t_i - t_{i-1}$ in between observations are not necessarily constant. Furthermore, different time series in a dataset can have differing amounts of observations and the times of observation across different time series may not line up. To feed this information to the model, it is frequently proposed to concatenate the time of observation to the observed data and feed (t_i, x_i) to the model [2]. However, this is not sufficient, as the model itself must be able to distinguish this temporal information from observed data.
2. **Partial Observation.**
   This notion directly follows from (1) for multivariate data when applying (1) to every

dimension individually. Hence, at every timestamp t_i only a subset of the recorded dimensions may be observed. Since we still want to be able to represent interdimensional dependencies, we must feed all dimensions to the same model. Otherwise, we could use separate models for every dimension only dealing with problem (1).
A common proposal for feeding this information to the model is to use an observational mask $\psi(x_i)$ [2] and concatenating this to yield $(t_i, x_i, \psi(x_i))$. As in (1), feeding this mask in addition to the observed data and time of observation to the model is not sufficient. The model must be able to internally represent data coming in only partially.

3. **Informative Missingness.** Consider the recording of medical data at an ICU in hospital. Without knowledge about the data recorded, the simple fact that in a certain period of time assessments were collected at a higher frequency than before might by itself lead to the conclusion that the state of the patient has deteriorated. The fact that data is recorded at different frequencies therefore might carry information about the state of the observed system. This notion is known as *Informative Missingness*. This information can be fed to a model by concatenating the cumulative number of observations to the input vector.

A notion fundamental to the field of deep learning and especially in the domain of time series analysis is mostly only implicitly assumed by authors. Since it poses an essential basis for the goals of the present work as well, I decided to explicitly highlight the otherwise usually implicitly assumed.

Deep Learning methods rely on the idea that every observed system has a true latent state at every time. When having knowledge about this latent state, every desired information is available. This holds true for classification tasks, that can be deduced from this state, as well as for regressive or predictive tasks. It is therefore the goal of most machine learning approaches to estimate this latent state – also called hidden state – as best as possible. For instance, Echo State Networks (ESNs) usually employ an expansion of the observed variable into a higher dimensional space, the reservoir, to then learn the readout from this reservoir to do inference on the observed states. Standard RNNs and specializations thereof, such as LSTMs and GRUs, similarly keep a hidden state that is regularly updated once a new observation comes in.

As depicted in Figure 2a, this hidden state is updated discretely upon a new observation and kept constant in between observations. The perfect model (which of course is not achievable) however learns a representation of the true hidden state, which is continuous in time. It is the main focus of this work to outline existing and propose a new model that estimates a continuous hidden state and thereby allows for all the three problems depicted above to be tackled. A continuous hidden state also allows irregular generative sampling and classification from a learnt model.

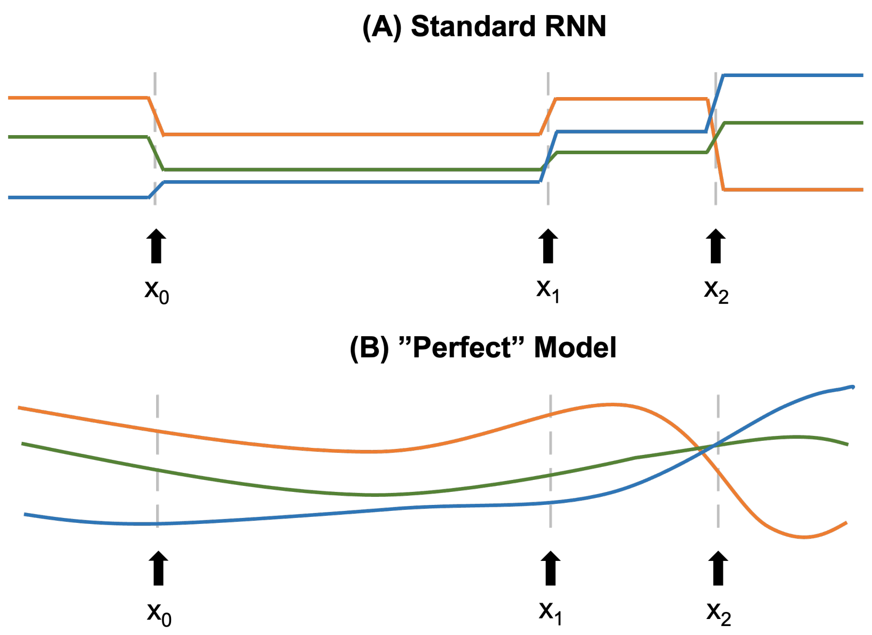


Figure 2. Discretely updated (A) vs continuously updated (B) hidden state in sequence models. Vertical lines depict times of observation. When doing inference in between observations, model (A) will yield the same result as during the last observation, whereas model (B) will allow evolving inference at any time.
Illustration adapted from [6]

1.3. Contribution and Scientific Importance

This thesis aims to give a two-fold contribution to current literature about continuous time series analysis. First, Chapter 3 outlines in detail different recently (and not so recently) proposed methods for dealing with irregularly sampled data. Every method will be classified and evaluated (not empirically). This section will serve as a reference over existing methods, how, and where to apply them to. Drawbacks and advantages are discussed. Furthermore, the chapter was written with the intent to provide intuition for all the presented methods. Such a comprehensive reference is, to the best of my knowledge, missing to this day in literature.

Second, the main contribution will be outlined in Chapter 4 where a novel method is proposed and empirically evaluated. As this method builds upon only recently gained knowledge about new approaches – especially their drawbacks – this serves as further research into these new methods, recombinations and extensions, as well es empirical evaluation thereof.

2. Foundations

This chapter outlines the relevant theoretical background necessary for the understanding of the reviewed and proposed methods in the upcoming chapters. It is organized in increasing order of complexity and puts a clear focus on topics relevant to time series models.

It starts with a very brief review of Artificial Neural Networks (ANNs) (assuming the reader is already familiar with the topic), before looking at RNNs and specializations thereof. The final section explains in greater detail the mathematical background of Neural Differential Equations (NDEs) assuming little familiarity on the side of the reader.

2.1. Artificial Neural Networks

Artificial Neural Networks are without question the dominant branch of Artificial Intelligence today. The reason being that ANNs are universal function approximators. Any smooth and differentiable function of arbitrary dimension can be approximated arbitrarily close by using one or the other form of ANNs. This was proven relatively early in 1989 by Hornik et al. [7] after earlier works have come to different conclusions (cf. [8]). The reason is that the true expressivity of ANNs is only apparent once enough layers of depth are added. Due to this approach the area of research and technology is commonly referred to as *Deep Learning*. However, as with everything, ANNs get no free lunch either, and the more expressive they are made, the higher the computational cost becomes. With infinite resources of computational power, one can theoretically build a huge neural network to approximate the laws of physics themselves and use this network to simulate the entire universe. However, this is and will be infeasible for the forseeable future. As such, the main bottleneck of ANNs today is their computational complexity. The exponential rise in available computational power in the early 2000s gave a rebirth to the field, as the fundamental ideas have already been researched in the middle of the 20th century. At that time, the available computational resources unfortunately were insufficient to put the theoretical works into practice.

Today, the enormous amounts of available resources allow *Deep Learning* to be utilized for complex tasks, such as image classification as well as generation, human language communication, or medical diagnosis assistance. However, since computational power still is the main bottleneck, research still focuses on reducing the costs by clever techniques and algorithms.

Neural Networks can be used for different types of objectives, such as classification (mapping an input to a class label), regression (predicting some value based on an input) or dimensionality reduction (mapping input data from d_x dimensions to d_h dimensions with $d_x >> d_h$). Any neural network therefore approximates a function $f_{true} : X \to Y$ that maps data from the input space $X \in \mathbb{R}^{d_x}$ onto the output space $Y \in \mathbb{R}^{d_y}$. The function f_{true} is unknown and must therefore be approximated by f_θ with θ being the parameters of the neural network. The network is *trained* by supplying it with many pairs of observations (x, y), for instance x being an image of dog and y being the number 1 meaning that the image is classified as containing a dog.

Throughout this work, $\hat{y}$ will denote a prediction given some input x, while y denotes the true output.

This section motivates Artificial Neural Networks (ANNs) and defines the algorithms used to run and train them. Most of the notation derivation is inpired by [9, 10].

2.1.1. Motivation – Logistic Regression

Consider the simple case of a linear regression problem, such as predicting housing prices based on multiple parameters. We have a set of observations which we can use to train the model we build. Under the assumption that the data follows a linear dependency, and given the input and output dimensionality d_x and d_y respectively, the goal is to estimate the true function $f_{true} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ by the affine function $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$. θ are the parameters of the function, the weights $W \in \mathbb{R}^{d_y \times d_x}$ and the bias $b \in d_y$, which we need to find, such that $f_{true} \approx f_\theta$. We define f_θ as follows:

$$\hat{y} = f_\theta(x) = Wx + b \tag{1}$$

This simple linear regression problem can be solved – finding parameters W and b that best approximate f_{true} – using a one-shot analytic solver. We will see later that finding analytic solutions will not be feasible any more for more complex models and we have to fall back to heuristic solvers, such as stochastic gradient descent (cf. Ch. 2.1.3).

Now suppose that we want to solve a binary classification problem, hence $d_y \in \{0, 1\}$. Since we are building continuous models, our goal is to find the probability $\hat{y} \in [0, 1]$ of y being equal to 1 given any input x. As before: $x \in \mathbb{R}^{d_x}$. Parameterizing the function f_θ to this problem is called *Logistic Regression*. The function is defined as before, only now wrapped with the sigmoid function) $\sigma : \mathbb{R} \to [0, 1]$ (see Fig. 5):

$$f_\theta(x) = P(y = 1|x) = \sigma(Wx + b) \tag{2}$$

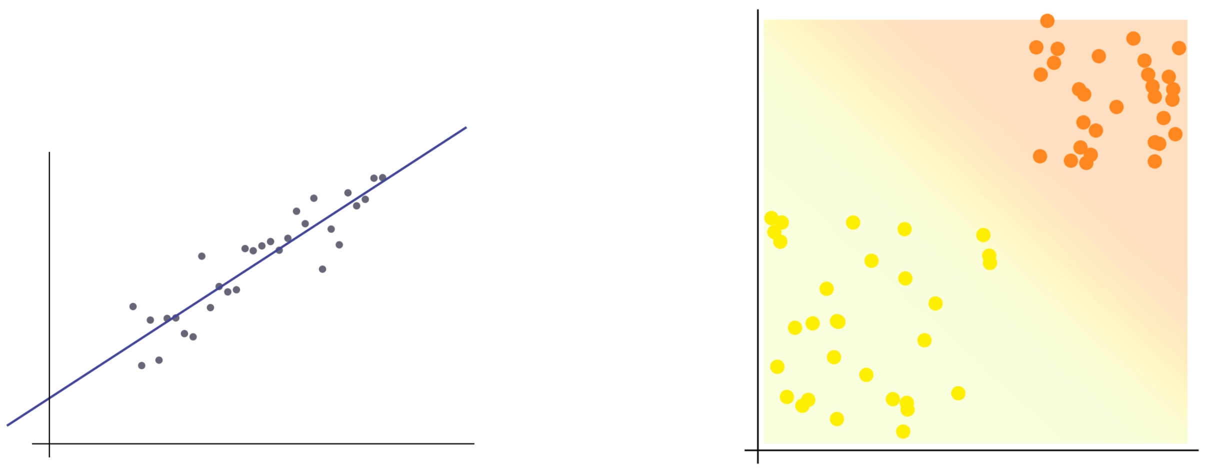
To utilize the found probability to make a prediction about the class, we can set a threshold value $t \in (0, 1)$, such that:

$$\hat{y} = \begin{cases} 0, & \text{if } f_\theta(x) < t \\ 1, & \text{else} \end{cases} \tag{3}$$

This approach allows to non-linearly map input data onto a probability for simple classification tasks. However, since sigmoid functions are monotonic, this approach still only works on linearly dependent variables, finding a cutoff point on a regressed spectrum. For non linearly dependent variables we therefore must extend this approach.

2.1.2. Multi Layer Perceptron

The approach for separating two classes by means of a linear separator was first introduced by Frank Rosenblatt in 1958 as the *Perceptron* [11]. The term stemmed from the inspiration he drew for his approach from the human brain. Every human neuron has input connections and output connections. Any neuron then fires once a threshold of weighted input signals is reached. The perceptron was modeled to replicate this behavior, the nonlinearity in the shape of a sigmoid function was only added later, thereby also making the function continuous.


(a) Linear Regression

(b) Logistic Regression

Figure 3. (a) Linear regression finds the line of best fit for the given data. (b) Logistic regression on the other hand produces a continuous separator between two linearly separable classes.

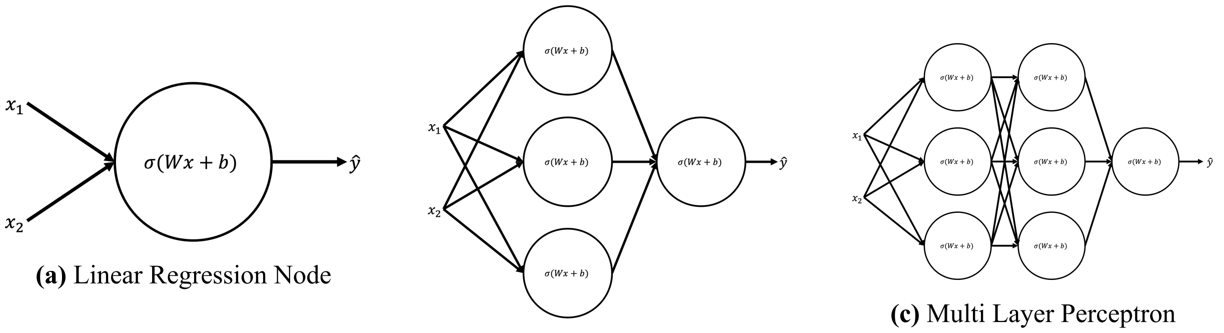
As stated in the preceding section, logistic regression (as the continuous extension to the perceptron) is not capable of modeling any non-linear separation of classes. Therefore, the human brain is further used as inspiration to form a network of multiple interconnected perceptrons, arranged in multiple layers, and therefore called a Multi Layer Perceptron (MLP). In order to be able to do arithmetic operations in such a network, commonly logistic regression nodes are used instead of discrete perceptrons. Every single one of these nodes is also called a *neuron*, while the entire network is therefore referred to as an Artificial Neural Network (ANN) [10].

Each neuron computes its activation a as before:

$$a(x) = \sigma(Wx + b) \tag{4}$$

The nodes are arranged in layers, starting from the input layer, which take just the multidimensional input to the network, then the hidden layers, where each neuron from layer i receives the activations from every neuron in layer $i-1$ and sends its activation to each neuron in layer $i+1$. The neurons in the first hidden layer simply receive the network input as their individual inputs. Finally, the output layer contains as many neurons as there are desired dimensions in the output. To compute the output to any input, the data is passed from layer to layer in a forward pass through the network. The computations can be decomposed into a series of matrix multiplications, additions and the application of the activation function [10].

While the original approaches used logistic activation functions (such as the sigmoid), many different activation functions have been proposed in literature to this day [12], with ReLU being the most commonly used due to its simplicity and computational efficiency. The choice of activation function for the output neurons is based on the desired shape of the output of the network. For classification tasks, logistic functions are commonly used as they map into a bounded interval which may be interpreted as class label probabilities. On the other hand, the identity might be useful for regression tasks. See Figure 5 for an illustration of different activation



(a) Linear Regression Node

(b) One-layer ANN

(c) Multi Layer Perceptron

Figure 4. The derivation of a Multi Layer Perceptron or deep neural network from logistic regression.

functions proposed in literature.

An ANN is a non-linear function $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ parameterized by its weights and biases θ. Given a lot of input-output pair data, the parameters can be learnt to accurately model the desired dependency (cf. Sec. 2.1.3). It has been shown that ANNs are universal function approximators, given that the network has enough neurons [7, 13].

2.1.3. Backpropagation

Finding values for the weights and biases of the network such that f_θ accurately models the desired dependency (i.e., $f_\theta \approx f_{true}$) is referred to as *training the network*. For this purpose, noisy training examples (x, y) (i.e., $y \approx \hat{y}(x)$, but $y \neq \hat{y}(x)$) are collected and passed through the network one by one. We can now define a loss function L that captures how much the predicted value $f_\theta(x)$ differs from the observed y. A common choice for L in the context of binary classification is the log-loss:

$$L(\hat{y}, y) = -(y\, ln(\hat{y}) + (1-y)\, ln(1-\hat{y})) \tag{5}$$

$$L_\theta(\hat{y}, f_\theta(x)) = -(f_\theta(x)\, ln(\hat{y}) + (1-f_\theta(x))\, ln(1-\hat{y})) \tag{6}$$

While this captures the loss of one training example, L can be subsumed to the cost function C which captures the loss over all m training examples:

$$C(\theta) = \frac{1}{m} \sum_{i=1}^{m} L_\theta(\hat{y}^{(i)}, f_\theta(x^{(i)})) \tag{7}$$

The goal is to minimize this function, i.e., get the network to produce predictions as close as possible to the desired outputs. For very simple predictors, such as a single perceptron, this optimization problem might be solvable analytically. However in bigger neural networks, this becomes infeasible. The function though is differentiable which is why we can take the following approach to find the best values for the weights and biases of the network. We compute the gradients with regard to W and b and then take small iterative steps (by updating W and b) into the direction of minimization. This approach is known as *gradient descent* and is predominantly used for ANN training. Therefore, every cost function used must be differentiable [14].

In deep neural networks, gradients can only be computed iteratively from "right to left". Gradients are computed for the weights and biases in every layer only with regard to the output of the

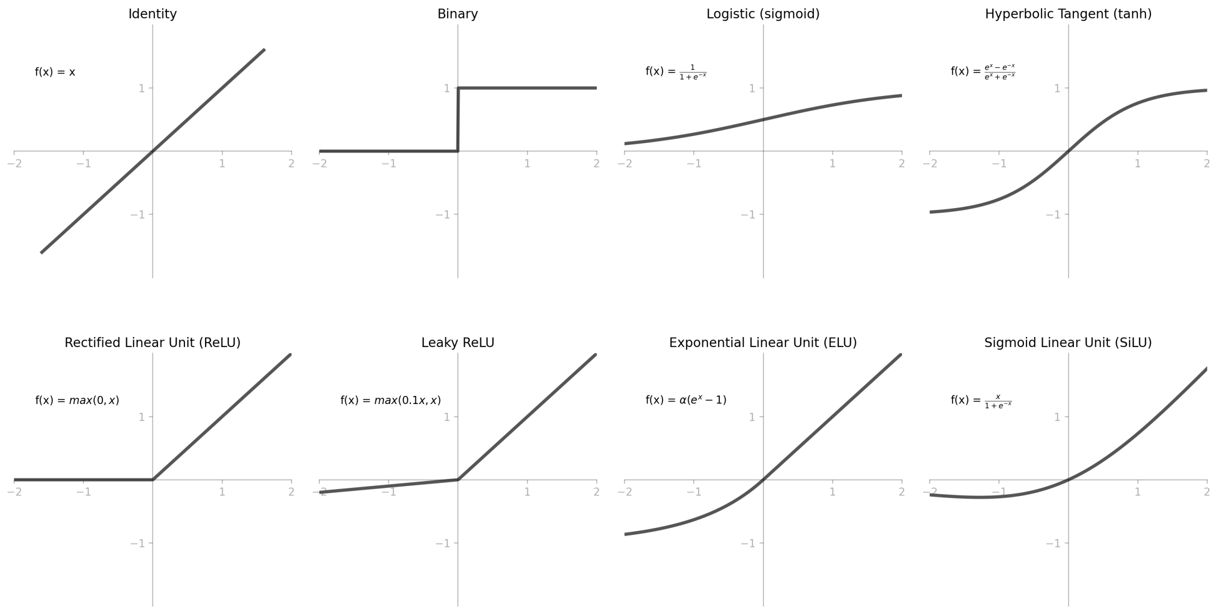


Figure 5. Different activation functions used.

neurons in the previous layer. Hence, we do a *backward pass* (or *backpropagation*) through the network after having computed the predictions and cost function in the *forward pass*.
After training, only the forward pass is required to make predictions for any given input.

2.2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are one of the major sub-branches of ANNs that are capable of handling sequence data and temporal relationships. This section introduces the idea, the most common types of RNNs and a variation that is fundamental to the core of this thesis.

2.2.1. Motivation

Sequence data, such as time series, consist of ordered pairs of observations (see Def. 1.1). They are intrinsically different to static observations – which are captured at a specific point in time, yet are independent of the time when they have been captured – in that they entail temporal dependencies. Several further difficulties arise (see Sec. 1.2), however this section outlines the classical approaches and therefore assumes regular sampling and full observation.

When processing sequence data, MLPs are incapable of learning temporal relationships since they have no memory. All they can do is output a prediction given data from one timestamp, without taking the previous timestamps into account. One possibility to solve this problem is to input the entire sequence all at once into the network. This however limits the model to fixed-size time series. RNNs solve this problem by introducing recurrent connections into neurons. At every observation timestamp t, the neurons receive as input not only the observation $x^{<t>}$ itself, but also the activations $a^{<t-1>}$ of the previous timestep. It then computes its activation as follows:

$$a^{<t>} = \sigma(W_{rec}a^{<t-1>} + W_{in}x^{<t>} + b_a) \tag{8}$$

The prediction $\hat{y}$ at every timestep can then be computed as follows:

$$\hat{y}^{<t>} = \sigma(W_{out}a^{<t>} + b_{out}) \tag{9}$$

Sometimes, the observation x_t is concatenated to the activation:

$$\hat{y}^{<t>} = \sigma(W_{out}[a^{<t>}|x^{<t>}] + b_{out}) \tag{10}$$

Where $[\cdot|\cdot]$ denotes vector concatenation. The activations a_t which are computed at each timestep and then passed as input to the next timestep are also commonly referred to as the *hidden* or *latent* state h_t of the model.
The parameters θ of such a Recurrent Neural Network now consist of the recurrent weights W_{rec}, the input weights W_{in}, the output weights W_{out} as well as the biases b_a and b_{out}:

$$\theta = [W_{in}, W_{rec}, W_{out}, b_a, b_{out}]$$

Training can still be achieved using gradient descent by utilizing backpropagation through time [15, 16, 17].

Short-term relationships can now be captured by such a model, however it still is unable to model long-term relationships due to vanishing gradient problems [18]. Several different architectures for extending one layer of neurons by clever gating techniques have therefore been proposed in literature. The two most widely used cell types are Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). They are briefly presented in the ensuing sections. Echo State Networks (ESNs) are yet another extension to RNNs and were designed to counteract the enormous computational cost of deep RNN training. They are outlined in Section 2.2.4.

2.2.2. Long-Short Term Memory Units

The Long-Short Term Memory cell was proposed by Hochreiter and Schmidhuber in 1997 [18] and was a major step forward in the research of RNN training. Their design includes several gates – an update, forget, and output gate – and two distinct hidden states. Thereby the model is able to capture long-term dependencies by learning what information to keep or discard at every timestep.
An LSTM is goverened by the following equations:

$$\begin{aligned}
\text{Candidate Memory:} \quad & \tilde{c}^{<t>} = tanh(W_{ch}\, h^{<t-1>} + W_{cx}\, x^{<t>} + b_c) && (11)\\
\text{Update Gate:} \quad & \Gamma_u = \sigma(W_{uh}\, h^{<t-1>} + W_{ux}\, x^{<t>} + b_u) && (12)\\
\text{Forget Gate:} \quad & \Gamma_f = \sigma(W_{fh}\, h^{<t-1>} + W_{fx}\, x^{<t>} + b_f) && (13)\\
\text{Output Gate:} \quad & \Gamma_o = \sigma(W_{oh}\, h^{<t-1>} + W_{ox}\, x^{<t>} + b_o) && (14)\\
\text{Cell State:} \quad & c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} && (15)\\
\text{Hidden State:} \quad & h^{<t>} = \Gamma_o * tanh(c^{<t>}) && (16)\\
\text{Prediction:} \quad & \hat{y}^{<t>} = \sigma(W_{yh}\, h^{<t>} + b_y) && (17)
\end{aligned}$$

with σ the sigmoid function, $\cdot$ the dot product and $*$ the element-wise multiplication. Learnable parameters now are:

$$\theta = [W_{ca}, W_{cx}, W_{ua}, W_{ux}, W_{fa}, W_{fx}, W_{oa}, W_{ox}, W_{ya}, b_c, b_u, b_f, b_o, b_y]$$

The model has become more complex and the amount of learnable parameters increased significantly, at the same time the model's expressiveness also increases, rendering it more powerful than a simple RNN [19].

2.2.3. Gated Recurrent Units

Gated Recurrent Units were introduced by Cho et al. in 2014 [5] and are a simplification of LSTM cells. GRU cells have fewer gates, parameters, and only keep one hidden state. It has been shown that GRUs are almost as expressive as LSTMs and sometimes even outperform LSTMs in certain tasks [20, 21]. They are now the preferred unit cell architecture in RNNs.
The equations governing a GRU are as follows:

$$\begin{aligned}
\text{Candidate Memory:} \quad & \tilde{h}^{<t>} = tanh(W_{ha}\,(\Gamma_r * h^{<t-1>}) + W_{hx}\,x^{<t>} + b_c) && (18)\\
\text{Update Gate:} \quad & \Gamma_u = \sigma(W_{uh}\,h^{<t-1>} + W_{ux}\,x^{<t>} + b_u) && (19)\\
\text{Reset Gate:} \quad & \Gamma_r = \sigma(W_{rh}\,h^{<t-1>} + W_{rx}\,x^{<t>} + b_r) && (20)\\
\text{Hidden State:} \quad & h^{<t>} = \Gamma_u * \tilde{h}^{<t>} + (1 - \Gamma_u) * h^{<t-1>} && (21)\\
\text{Prediction:} \quad & \hat{y}^{<t>} = \sigma(W_{yh}\,h^{<t>} + b_y) && (22)
\end{aligned}$$

with σ the sigmoid function, $\cdot$ the dot product and $*$ the element-wise multiplication. Learnable parameters now are:

$$\theta = [W_{ha}, W_{hx}, W_{uh}, W_{ux}, W_{rh}, W_{rx}, W_{yh}, b_c, b_u, b_r, b_y]$$

The amount of trainable parameters has significantly decreased with regard to LSTM units rendering this cell more computationally efficient. However, it also has lost expressiveness as it is a specialization of LSTM cells.

2.2.4. Echo State Networks

While RNNs and especially their extensions, LSTMs and GRUs, are powerful techniques to build sequence processing models, they still suffer from a major drawback. As computationally expensive as all deep neural networks are, so are RNNs. In the early 2000s, when computational power was still sparse, multiple researchers therefore proposed similar ideas to tackle this problem by – instead of optimizing – randomly initializing the recurrent and input weight matrices W_{rec} and W_{in} in a clever fashion, increasing the size of W_{rec} to a huge number, and then only training the readout weights W_{out}. This approach was independently published by Jaeger in 2001 [22], as *Echo State Networks*, and by Maas et al. in 2002 [23], as *Liquid State Machines*. The concept has become jointly known as *Reservoir Computing*. The only conceptual difference is that the latter uses discrete neuron outputs, while ESNs utilize continuous activation functions. We therefore focus on the former and use the term Echo State Networks throughout this work.

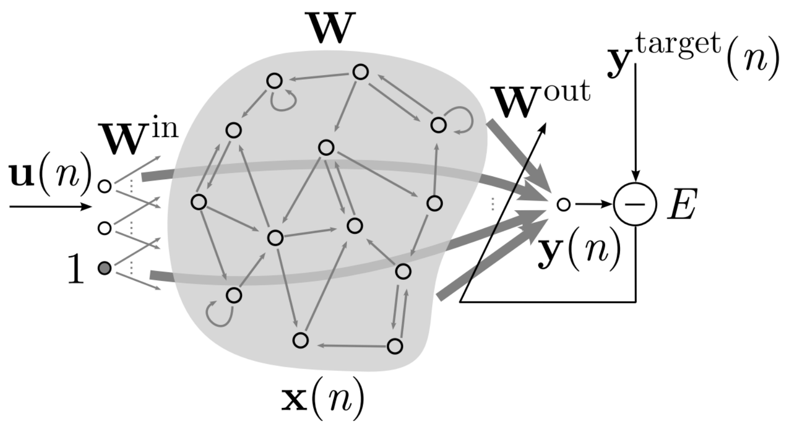


Figure 6. A schematic view of an ESN. Only the connections drawn bold are trained. Figure from [24]

The motivation stems from the fact that having a big reservoir of randomly connected hidden neurons serves as a non-linear expansion of the observed input data and is therefore able to capture the dynamics of the system under consideration. In essence, ESNs define a random non-linear expansion of a sequence of observations onto a fixed-length hidden space, from which a readout then can be trained. Usually, simple linear regression is chosen for this. This enables efficient one-shot learning [24, 25]. However, since a huge reservoir is needed to capture enough information for a linear readout, ESNs trade off memory space for expressivity.
More than two decades have passed since the first introduction of reservoir computing approaches. At that time, ESNs had a decisive advantage over fully trained deep Recurrent Neural Networks in that they did not need computational power, but lots of memory space, which was already available then. Today, RNNs of the same size as an ESN can be fully trained, thereby increasing their expressiveness. However, the growing societal consciousness about energy consumption has revived the justification and need for approaches that are energy efficient, such as ESNs.

The basic approach is as follows (this paragraph is adapted from[2] [24]). Just like RNNs, vanilla ESNs are also designed to handle only fully observed and regular time series. We therefore consider a time series TS with a fixed interval t', i.e., $t_i - t_{i-1} = t'$ for all $0 < i < n$, and $x^{<t_i>} \in \mathbb{R}^{d_x}$. We then initialize the model by defining the two random weight matrices using the hidden state size d_h (a hyperparameter of the model):

- The input weight matrix $W_{in} \in \mathbb{R}^{d_h \times (1+d_x)}$ is randomly sampled from a uniform distribution. The scaling can be tuned as a hyperparameter of the network. The additional column is to encode the bias directly inside of the weight matrix. For that purpose, prior to computing the dot product between W_{in} and an observation, the constant 1 is concatenated to the observation, yielding valid dimensions: $[1|x^{<t_i>}] \in \mathbb{R}^{(1+d_x)\times 1}$.
- The recurrent weight matrix $W_{rec} \in \mathbb{R}^{d_h \times d_h}$ is also randomly sampled from a uniform distribution. During hidden state updates, this matrix is then multiplied by the hidden state $h^{<t_i>} \in \mathbb{R}^{d_h \times 1}$.

[2]However, notice the different notations used.

The scaling of this matrix requires special attention and is of essential importance for the ESN to yield good results. For this purpose the matrix is scaled in such a way that its spectral radius (another hyperparameter) is of a certain size (usually close to 1). It has been shown that this ensures that the reservoir incrementally discards information from previous timesteps which is essential for the hidden state not to explode in magnitude in response to earlier observations. This is known as the *echo state property* [24]. The proper scaling requires a computation of the eigenvalues of the matrix which is one of only two computational bottlenecks of ESNs.

To then train the readout, we must first collect the hidden states from every timestep of the training data by computing $h^{<t_i>} \in \mathbb{R}^{d_h}$:

$$\widetilde{h}^{<t_i>} = tanh(\ W_{in}\ [1|x^{<t_i>}] + W_{rec}\ h^{<t_{i-1}>}) \tag{23}$$

$$h^{<t_i>} = (1-\alpha)\ h^{<t_{i-1}>} + \alpha\ \widetilde{h}^{<t_i>} \tag{24}$$

where $tanh(\cdot)$ is element-wise applied, $[\cdot|\cdot]$ denotes vector concatenation, $\widetilde{h}^{<t_i>}$ is the update for the hidden state neuron activation $h^{<t_i>}$. The leaking rate α (another hyperparameter) determines how much information to keep from timestep to timestep, essentially defining the memory duration. It is used to update the hidden state as a weighted average of the previous hidden state $h^{<t_i>}$ and the update $\widetilde{h}^{<t_i>}$. A common choice is $\alpha = 0.3$. Different activations functions than $tanh$ are sometimes used [24].
The linear readout $\hat{y}^{<t_i>} \in \mathbb{R}^{(1+d_x+d_h)\times 1}$ for timestep i is then defined as:

$$\hat{y}^{<t_i>} = W_{out}\ [1|x^{<t_i>}|h^{<t_i>}] \tag{25}$$

The additional 1 on top of the vector again serves as the bias. The readout takes the observation $x^{<t_i>}$ as well as the current hidden state $h^{<t_i>}$ into account.
Notice that the readout at every timestep is independent of both observations and hidden states from different timesteps. By horizontally concatenating $[1|x^{<t_i>}|h^{<t_i>}] = H \in \mathbb{R}^{(1+d_x+d_h)\times n}$ across all timesteps, we can therefore define the one-shot linear regression readout problem as follows:

$$Y = W_{out}\ H \tag{26}$$

where $Y \in \mathbb{R}^{d_y \times n}$ are the horizontally concatenated observations. A common choice to solve this for W_{out} is ridge regression, which minimizes the mean squared error [24].

Notice, that this approach also seamlessly allows batching of multiple time series:

- Horizontally concatenate all observations across the m time series at timestep i to form $x'^{<t_i>} \in \mathbb{R}^{d_x \times m}$.
- The hidden state $h^{<t_i>}$ will now have dimensions $\mathbb{R}^{d_h \times m}$.
- Now, instead of concatenating vectors to form the matrices H and Y, horizontally concatenate the matrices across time series to get $H \in \mathbb{R}^{(1+d_x+d_h)\times(m*n)}$ and $Y \in \mathbb{R}^{d_y \times (m*n)}$.

With this, datasets consisting of multiple time series can be modeled by an ESN.
There is one remaining caveat. The hidden state of the previous timestep is not available at step t_1.

Therefore, $h^{<t_0>}$ is commonly initialized to 0^{d_h}. It takes a few steps for the network to properly capture the currently modeled time series, therefore the first few steps per series, referred to as the network *warmup*, are usually discarded during training.

2.3. Neural Differential Equations

Deep learning research has evolved from simple MLPs to complex composite networks to solve specific tasks efficiently and powerfully. The two most prominent and well-researched branches are Convolutional Neural Networks for image processing and Recurrent Neural Networks for sequence processing which can both be seen as having brought a major leap to the field [26]. Over the recent years, several big new proposals of network architectures have been made, such as Residual Networks [27] or Transformers [28]. The common denominator of all such approaches is being based on a hidden state propagation through discrete layers. Therefore, the introduction of Neural Ordinary Differential Equations (Neural ODEs), which utilize a continuous hidden state, brought another big leap forward to the field as recently as in 2018 [1]. The applications of Neural ODEs with regard to time series data are especially exciting and will be discussed in more detail in Section 3.5.
The present section merely introduces the conceptual architecture as well as the mathematical background of Neural Differential Equations.

2.3.1. Motivation – Differential Equations

Generally speaking, differential equations are equations that contain a dependent and an independent variable in addition to one or more derivatives of the dependent variable. The canonical example differential equation is the following:

$$\frac{dy}{dt} = ky$$

Here, the derivative of y with regard to t is a function of y itself. A solution to this differential equation is the function $y(t)$ that satisfies the condition above. In the case of such a simple equation, the solution can be obtained analytically. Here, the solution happens to be $y(t) = ce^{kt}$ for every arbitrary constants $c, k \in \mathbb{R}$.

Differential equations are usually interpreted to represent dynamical systems evolving with time. Therefore, the symbol t is chosen to denote the independent variable instead of x.

Since the equation contains y and its first-order derivative, it is called a differential equation of first order. Higher order differential equations, where the function would contain higher order derivatives, are also possible, yet much harder to solve.

The equation above contains only one derivative with regard to one independent variable. It is therefore called an Ordinary Differential Equation (ODE). In contrast, if the equation contained multiple derivatives with regard to multiple independent variables, it would be called a Partial Differential Equation (PDE). PDEs are generally much harder to solve. Throughout this work we will only deal with Ordinary Differential Equations of first order.

If an ODE is arranged such that the derivative of y is a function of y itself (i.e., $\frac{dy}{dt}(t, y) = \cdots$), then the right-hand side $\cdots$ can be interpreted as a representation of the vector field of all

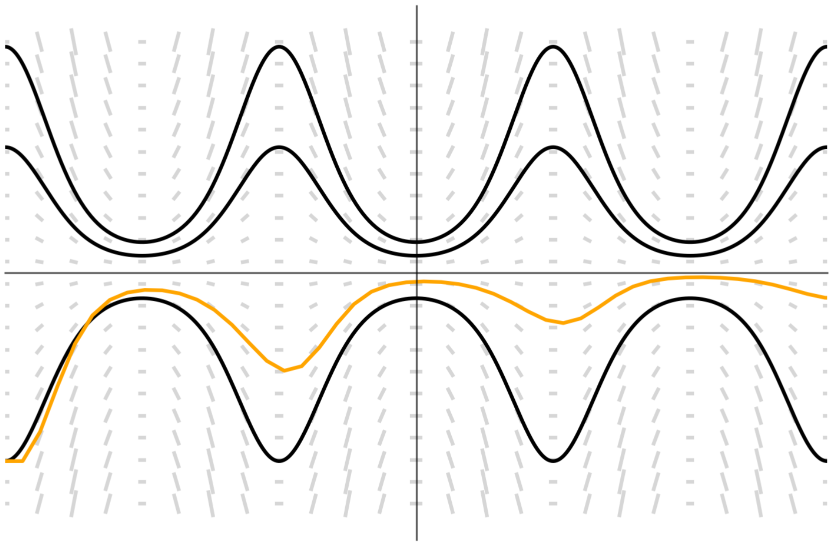


Figure 7. The slope field of the ODE $\frac{dy}{dt} = y\ sin(t)$ and three solutions in black. The orange trajectory shows an Euler estimation with constant step size. Notice the accumulating error.

possible solutions to this differential equation. Each line in that vector field depicts the slope of the function at that coordinate. It directly follows from this that a differential equation is deterministic and solely dependent on its slope field, as well as an initial condition (i.e., given $\frac{dy}{dt}(t, y) = \cdots$ and an initial condition (t_0, y_0), the solution is deterministically defined by following the slope field starting from (t_0, y_0)). See Figure 7 for an illustration.

2.3.2. Differential Equation Solvers

While simple ODEs such as the one above are easy to solve analytically, it becomes infeasible quite fast with more complex equations. Consider the following for instance:

$$(\frac{dy}{dt})^5 + 3c = yc$$

It is unclear whether it is possible to solve such a differential equation. We therefore need to fall back to using approximate solving techniques.

As mentioned before, a solution to a differential equation can be found by starting at some coordinates and then following the slope field since we know the gradients at every point (given by the differential equation itself).

The naive approach is to take fixed step sizes in the direction of the gradient (see Fig. 7). Small step sizes are essential here, due to the error accumulating over time. This approach is known as the *Euler estimation*:

$$y(t + \Delta t) \approx y(t) + \Delta t * \frac{dy}{dt}(t, y) \tag{27}$$

with Δt the fixed step size. This approach can be seen as time discretization of the dynamical system represented by $y(t)$.

ODEs are well researched for over one hundred years [29, 30, 31]. More robust solvers adapt their step size to the magnitude and the rate of change of the gradient.

2.3.3. Neural Ordinary Differential Equations

Neural ODEs were popularized by Chen et al. in 2018 [1], although similar research was done independently by other authors [32, 33, 34] and has already been done in the 1980s and 90s [35, 36, 37, 38, 39, 40].

Consider a residual neural network architecture where the hidden state of the network is propagated using skip connections [27]:

$$h^{<t_i>} = h^{<t_{i-1}>} + f_\theta(h^{<t_{i-1}>}) \tag{28}$$

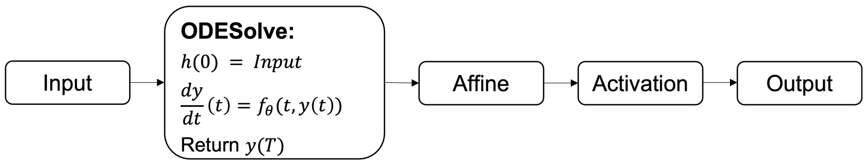
where $h^{<t_i>}$ is the hidden state at depth t_i and f_θ is a layer of a neural network. Notice that t now represents the depth of the network as it can be interpreted as a dynamical system evolving with time, hence the propagation from one layer to the next. This can be interpreted as the Euler discretization of some continuous hidden state dynamics with constant step size $\Delta t = 1$ (see Eq. 27). By adding more layers, thereby making the step size smaller and smaller, we reach a continuous-depth, and continuously evolving hidden state, in the limit. The gradients, hence the vector field, of the hidden state is then determined by f_θ. We thereby arrive at the definition of a Neural Ordinary Differential Equation according to Kidger [2]:

Definition 2.1 (Neural ODE)**.** "A *Neural [Ordinary] Differential Equation* is a [ordinary] differential equation using a neural network to parameterize the vector field". It is defined by the input to the network which is also the initial condition of the ODE $x = h(0)$ and neural network f_θ parameterizing the vector field of the ODE: $\frac{dh}{dt}(t) = f_\theta(t, h(t))$.

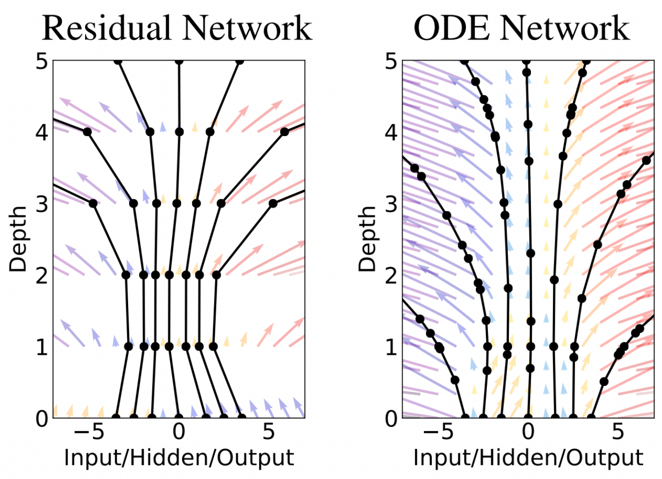
The function $h(t)$ is the solution of the ODE and determines the propagation of the hidden state. The ODE is then evolved starting at the initial condition $h(0)$ (the input to the network) until some fixed "depth" T. The then obtained hidden state can be read out using linear regression or any type of feed-forward neural network architecture. The input x can also be first non-linearly expanded by some network before feeding that into the Neural ODE [2].
Notice that t here does not denote time but rather the depth of the network, although it is unclear whether depth is even the right term to use here, since the notion of layers has disappeared in this architecture.

Figure 8b shows a standard residual neural network (ResNet) on the left and a Neural ODE on the right. Notice how the ResNet evaluates the hidden state only on the discrete depth levels of the network, whereas the Neural ODE on the right allows hidden state evaluations at any network depth.

The Neural ODE architecture formulates the computation graph in Figure 8a. The ODESolve block consists of a series of Euler discretization steps (or other solvers), which can be backpropagated through one by one even with existing automatic differentiation tools provided by most modern deep learning frameworks. However this poses high memory costs and extra accumulating error. Therefore, the authors of [1] proposed to use the well-known *adjoint sensitivity method* [41] in order to compute the gradients necessary for training the network. This approach allows a tradeoff between speed and error and poses only constant memory costs as it uses a second ODE to solve backwards in time.



(a) Adapted from [2]



(b) Source: [1]

Figure 8. (a) The computation graph of a simple Neural ODE. (b) Illustration of the discrete-depth hidden state propagation in residual neural networks (left) and the continuous-depth hidden state propagation in Neural ODEs (right).

The original Neural ODE architecture [1] was proposed as the continuous limit of feed forward ANNs and not on sequence data models. However, the potential of a continuously evolving hidden state for sequence models is apparent and was already followed up on by Rubanova [6], inter alia. One issue preventing Neural ODEs to be applied to time series directly is the fact that the hidden state evolution is deterministically determined solely by the initial condition. To overcome this drawback, Kidger showed that yet another type of differential equations – Neural Controlled Differential Equations (CDEs) – are the continuous time limit of RNNs [42]. We will dive deeper into this approach in Section 3.5.

2.4. Summary

The present chapter roughly outlines the required theoretical background necessary for the remainder of this work. It is meant to be an overview of different relevant topics, it was not written to be a detailed and comprehensive tutorial of these topics. The reader may refer to [1, 4, 9, 10, 14, 19, 22, 24, 25, 28] for deeper insights.

3. A Taxonomy of Universal Time Series Analysis

After having grasped the problem statement and relevant theoretical background, the now following chapter provides the first contribution of this thesis by outlining and categorizing most of the proposed approaches in modern literature. While classic problem specific approaches have existed for a very long time (cf. [43, 44, 45, 46]), they were mostly over-engineered or not generalizable. Standard domain-independent deep learning blueprint architectures for sequence processing to this day only exist in the form of discrete-time RNNs. This blueprint has only recently been generalized to the continuous-time domain [42] and will with high probability become the new blueprint approach. However, this approach was only conceptually derived and has yet to be proven to be effective on real world data domains.
This chapter gives an overview over the many different types of approaches to deal with sequence processing, taking multiple objectives into account. Special attention is given to the three fundamental problems which were outlined in Section 1.2. Every discussed method is evaluated with regard to these challenges. The goal of this chapter is to serve as a reference for many different approaches, their capabilities, advantages, and disadvantages.

These are the aspects which the analysis of the respective methods will center around (cf. also [2, 47]):

Different lengths. Two sequence observations in the real world are hardly ever of the exact same length. For a method to be universally applicable it is therefore essential that it can handle datasets that contain sequences of different lengths without discarding information.

Irregular Sampling. For much the same reason as above, real world data usually comes with irregular temporal intervals in between observations. While simple methods can be used to pre-process datasets by artificially re-sampling them into regular intervals, each of these methods discards some information along the way (see Sec. 3.1 for a discussion). It is therefore desirable for a model to naturally "live" in continuous.

Missing Values. This is distinct from the concept of irregular sampling. Some models may still assume a constant time interval in between observations (i.e., it expects values at each timestep), however are still able to represent missing values. True universally applicable methods however can be used by only feeding them the actual observations as they come in, instead of having to input tokens for missingness when no data is observed.

Partial Observation. As stated in Section 1.2, partial observation directly follows from irregular sampling, when observing dimensions separately. For the same reasons, incoming data therefore must be allowed to provide new inputs only to a subset of the dimensions under consideration.

Informative Missingness. This refers to the insight that the fact that an observation is missing, or a gap between two observations is longer or shorter than usual, carries information on its own. Not every model that can handle partial observations and irregular sampling also is able to learn from this concept.

Discrete or Continuous Hidden State. Since the hidden state of a model is an encoding of the information it deems to be relevant at the time in question, it is desirable that the hidden state is continuous in its entirety. This allows readout sampling at every point in time and also updates to the hidden state without regard for regular temporal intervals. Since the state of every enclosed and observed system is considered to be encodable into a fixed dimensional space, continuity only follows from this assumption.

Burden on Training. Does the proposal extend another approach in such a way, that training becomes significantly more time or energy consuming? Or does it even make training more efficient?

Some methods may appear in multiple categories. In this case the discussion on the models was done with regard to their current categorization.

3.1. Basic Methods

Before considering actual model-based approaches, a rational approach to tackle several of the aforementioned problems is to properly pre-process the sequence prior to feeding it into a standard discrete model incapable of handling universal time series observed in irregular intervals. Usually, such approaches yield a high loss of information and are therefore undesirable in general. They trade off information loss versus simplicity.

Methods in this section are mostly not proposed by certain literature, they rather the rational choice decisions for how to approach the given problems using well-known models.

3.1.1. Discretization Methods

When using models that were designed for regular time intervals, one could simply fit the input data into the model one after the other, without regarding the differing intervals in between observations. However, this is highly undesirable for multiple reasons. First, a lot of information is discarded (that's what informative missingness means). And second, each model is based on an assumption about the temporal evolution of observations. An RNN for instance is based on the assumption that observations are evenly spaced in time. Given three observations at times $0s, 10s, 100s$ and feeding them sequentially into the network, the network would give the same results as if the observations have been recorded at times $0s, 10s, 20s$, although the desired results can be vastly different.

The classical approach for still using a plain RNN with irregularly sampled data is therefore to pre-process the data in such a way, that the resulting sequence is evenly spaced in time again. This is achieved by binning multiple observations within a certain time span by averaging them for instance. The result is a time series that has regularly spaced observations. However, the operation of averaging multiple observations also discards information. Furthermore, when there are really big gaps in the series, then the bin intervals have to be at least as big as half of the biggest gap. The larger the bin size, the more information is lost since one has to average over more observations. For these reasons, this approach can only be used under certain circumstances, where there are no big gaps in the time series, and every interval in between observations is so small that the binning does not discard relevant pieces of information (i.e., the standard deviation of the binned observations are small, see [1] e.g.). In that case, binning can even help to speed up computation times.

3.1.2. Imputation and Interpolation

Missing data (i.e., given a regular time interval, observations are made only at a subset of all possible timesteps) can be replaced with imputed or interpolated values prior to feeding the data to a model. The actual data the model then receives is regularly spaced in time and can therefore be processed by a huge variety of models designed for this purpose. *Interpolation* refers to the technique of filling in missing data by taking only the last and next observation into account. The simplest form is linear interpolation, where the linear function (a straight line) connecting the two adjacent observations is used to re-sample at missing timesteps. *Imputation* on the other hand refers to generating the missing data based on all the values of the entire sequence. A simple example is to just fill in the missing spots with the overall average of all observations across time.

Notice that this technique can in principle be applied to every irregularly samples sequence, by setting the time interval to the shortest gap between two observations of the series. However, this often leads to the majority of the inputs to be the token for missing data, which comes at a greater computational cost. It is therefore advisable to apply interpolation or imputation only to sequences which are regularly sampled, with just a few data points missing at some timesteps.

An important distinction has to be made in terms of the different types of approaches for filling in missing data. One can either choose to interpolate or impute based on *a priori* knowledge about the data domain under consideration. Che et al. [47] for instance note that in medical datasets, each observed variable tends to fall back to a certain default value during long intervals of missing data. Based on this observation, it is rational to fill in missing data points with this known default value, or at least with the overall average of the observed data. Such approaches are possible without looking at the training data to select a function for imputing. On the other hand, when no *a priori* knowledge is available, the imputation function may be learnt from the training data [47, 48, 49, 50]. The result of both types of methods however is the same: A regularly spaced time series which can be fed into any model.

Since such methods rely on prior pre-processing before feeding data into a model, the notion of a continuous hidden state is not considered here.

It is furthermore necessary to distinguish based on the randomness of the missing data, as [51] noticed, so as to not introduce a bias when imputing data into these missing spots:

- *Missing Data Completely at Random (MCAR)*: Here, missing data occurs completely at random and does not depend on other missing or known data.
- *Missing at Random (MAR)*: In this case, the probability of an occurrence of a missing datapoint may be dependent on other known values, but independent of other missing values. In other words, the missing data may be imputed based on other attributes or timesteps in the series.

- *Not Missing at Random (NMAR)* This occurs, when the probability of missing data is dependent on the actual value of the missing data. In such a case, imputation based on other values may introduce a bias and become unreliable.

While any imputation method may be used for MCAR, more caution is required when working with MAR data. Imputation may be completely misleading when dealing with NMAR data.

Basic imputation methods that do not require training as summarized by [47] are:

- Filling in the arithmetic mean over all observations in the series.
- Filling in the arithmetic mean of the two adjacent observations.
- Filling in the last-observed data.
- Linearly interpolating based on the two adjacent data points.
- Interpolating by means of a cubic spline.
- etc.

Approaches that do not rely on any *a priori* knowledge and need to be trained on the training data are usually more sophisticated. The most relevant approaches proposed in literature are therefore collected and will be dedicated a paragraph each for discussion. The ordering is chronological of the publication dates.

K-Nearest-Neighbor (Batista et al. [52])

The authors of this approach propose to utilize k-nearest neighbor search for imputing missing values in any multivariate time series. Given a suitable similarity measure between sequences, missing values are imputed by retrieving the k nearest neighbors to the sequence, according to the similarity measure in use, and then using the retrieved sequence to extract the necessary information for imputation. This can be done for instance by averaging over the retrieved sequences. It is also possible to retrieve only short chunks of high similarity and use only the direct vicinity of the missing data point.

As the main advantages of this approach, the authors state that the method is suitable for both discretely and continuously valued attributes (as only the similarity measure has to be defined on these domains). They also note that this approach is training-free. No training is required, as the data itself is used as a “lazy model” [52]. It is therefore also possible to add more data later, as this increases the model expressivity as it becomes more likely that new sequences find other sequences of high similarity which already are stored in the dataset.

However, as with all case-based approaches, the main drawback is that in order to perform the k-nearest neighbor search, similarities have to be computed with each sequence in the dataset. This can become expensive quite quickly. To mitigate this, it is possible to pre-process the dataset in a way that clusters sequences of high similarity together (such that only one similarity computation is done thereafter) or prunes sequences to remove parts of less relevance [3].

The authors also note that their approach is very universal and has to be adapted to the specific domain where it is deployed on. The approach therefore works on data which contains missing values according to MCAR, the most universal unbiased case.

Irregular sampling, missing values and partial observation can all be handled by this approach, depending on the similarity measure chosen. The authors unfortunately do not go into detail on this issue.

MICE – Multiple Imputation by Chained Equations (Azur et al. [53])

This paper serves as a meta study on the approach of chained equations for time series imputation. They show that this method can be used for MAR data, making it slightly more restrictive. The term *chained equations* refers to the fact that the imputation process is done in multiple steps:

1. First, basic imputation is performed by, for instance, inserting the mean value of each respective attribute over the entire sequence into all the spots where data is missing. The authors denote this as setting "place holders" [53] into these spots.
2. Then, one attribute is set back to *missing*, resulting in only one attribute having missing values.
3. Regression is performed by setting the attribute that has missing values as the dependent variable, and all other attributes as the independent variable. That way, the missing data points can be predicted based on the other variables. Notice, that here a noise component must be introduced as well.
4. Now the missing spots are replaced by the values obtained via the regression model. When performing regression thereafter for the other attributes, these newly imputed values will also be used for this.
5. Steps 2 to 4 are repeatedly executed for each attribute which contains missing values. At the end, each attribute has been imputed with missing values based on regression using the other attributes as independent variables.

This cycle is repeated for a pre-determined number of times, with the goal of the values converging towards the end.

Furthermore, the entire process is done on multiple copies of the original dataset, yielding multiple datasets with slightly differently imputed values in each of them in the end. Inference shall then be conducted on all these copies, and the results aggregated afterwards.

This method is mostly proposed for non-sequential data, however the ideas are useful in the context of time series models as well.

Random Forests (Stekhoven & Bühlmann [54])

This method seeks to propose an approach that is applicable to arbitrarily valued and multivariate datasets. Even mixed-type datasets containing both categorical and continuous data can be handled by their model, called *missForest*. It furthermore works without assumptions about the stochastic data distribution underlying the dataset.

They propose to train a random forest on the observed data, and then iteratively use this random forest to make predictions on the missing data. This allows a parameter-free model that can be quickly deployed. They show that this approach is more powerful than using k nearest neighbors or chained equations.

Principle Component Analysis (Josse & Husson [55])

Principle Component Analysis (PCA) is a staple of multivariate statistics for a long time. This method aims to identify a low-dimensional representation of the data being observed. The individual dimensions of this found representation represent the principle components of the dataset.

The authors propose to iteratively perform PCA until convergence. Each step using the found principle components for imputing the missing data.

PCA is a highly manually engineered methods and requires deep knowledge of the domain where it is applied to in order to get reasonable results. The choice of the number of dimensions is only one issue.

Gaussian Processes (Rybicki & Press [43], Alvarez & Lawrence [56], Li & Marlin [57] Futoma et al. [58], Soleimani, Hensman & Saria [59])

Now this type of approach has been proposed by many researchers as it is a simple yet effective way to model data frequency separately from the actual data. Sometimes, this method is used for imputation [58] while others use the frequency estimation as additional input to a model without imputing data [6].

The individual proposed approaches are discussed in greater detail in Section 3.4. Here, we will only give a short overview of the idea with regard to imputation.

The underlying assumption that must be applicable to the data at hand is that the variables must be sampled from an assumed normal distribution. When this is the case, gaussian process methods are applicable.
With this assumption, it is then possible to create a regressor based on gaussian processes that will provide a probability distribution for every regressed value. It thereby inherently models uncertainty about the missing values and allows for sampling from a probability distribution to generate these missing values. Such methods can naturally handle partially observed and irregularly sampled sequences of differing lengths. Furthermore, this approach also allows for sampling new observations from previously completely unobserved timesteps.

While the data itself can be modeled using a gaussian process, it is also possible to model just the frequency of observations. This allows for models to learn from this frequency, which allows inference and the notion of informative missingness to be taken into account by a model.

GRU-D (Che et al. [47])

This approach is very popular among those which are willing to modify the model architecture of an RNN. It is therefore discussed in detail in Section 3.2.
However, it is still mentioned here since the approach only works given some underlying assumptions on the data. In this approach, a GRU cell is modified in such a way, that the hidden state gradually decays over time until a new observation comes in. The authors designed this approach for application on medical datasets which is a well-known domain. They pose the reasonable assumption that after a certain variable has not been observed for a long time, the influence of this variable on the current state should be minimal. They therefore also let the input variables decay over time towards some default value. This is a type of imputation that can only be applied when certain assumptions about the underlying data hold.

BRITS (Cao et al. [48])

The authors of this approach seek to propose a solution for imputing data without making any assumption about the distributions or inter-dependencies of the underlying dataset. Instead, the propose to use a bidirectional Recurrent Neural Network to learn these dependencies on its own. By using a bidirectional network, information is taken into account from both the ensuing and preceding data of missing observations. An RNN can naturally produce the outputs at the data points of missing spots. Furthermore, the authors showed that the same network can be used for imputation and inference at the same time. This is achieved by feeding observed values into the network at steps where data is available, while feeding predicted data back into the network when no actual data is observed.

This approach however does not allow for irregularly sampled or partially observed data directly. The first issue is overcome by the authors by introducing a decay term that lets the hidden state inside the network decay over time. The latter problem is not explicitly stated in the paper. However, this may also be overcome by combining the observed with the predicted values and feeding a complete set of data back into the network.

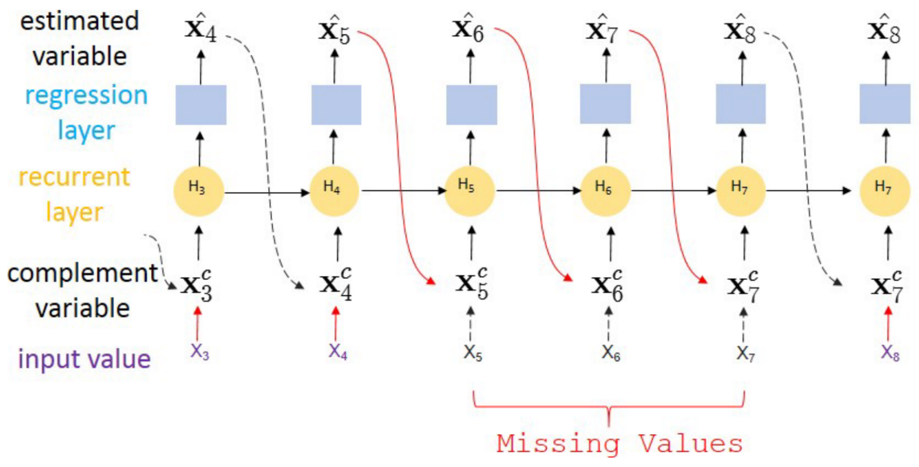


Figure 9. The proposed method by Cao et al. Figure from [48]

Generative Adversarial Networks (Luo et al. [50])

Here, the authors propose to use a Generative Adversarial Network (GAN) to not only predict the missing values, but to generate the entire sequence. For this purpose, a GAN is trained such that it will generate sequences based on a sample from a latent space that consists of random noise. This model therefore learns a distribution over all possible sequences in this domain. Arbitrary intervals are possible to be generated by this GAN since it uses an adapted GRU cell for this purpose.

Given a sequence to impute, random noise is generated in such a way that the resuling generated sequence has high similarity with the observed one. Since the generated sequence will be fully observed and regular, missing data points can be imputed in that way. The then constructed fully observed series can be fed into any normal network architecture.

Overall, this approach is certainly an overkill in most situations as it poses very high additional costs.

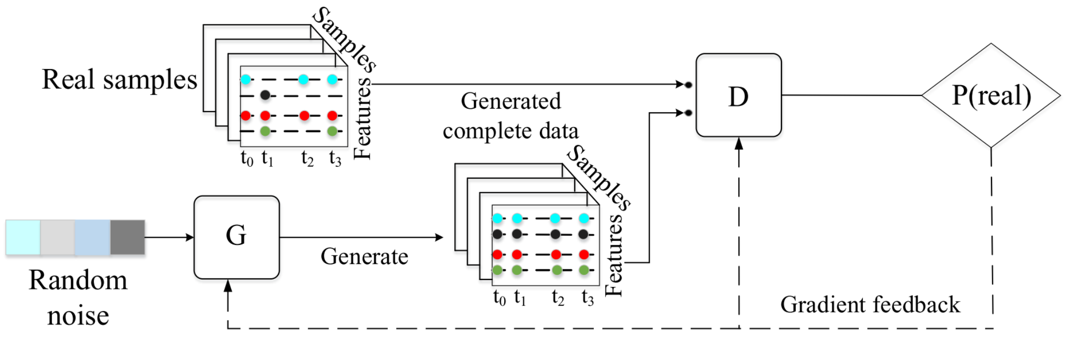


Figure 3: The structure of the proposed model.

Figure 10. The proposed method by Luo et al. Figure from [50]

Interpolation Neural Network (Shukla & Marlin [49])

The authors of this approach use a manually engineered interpolation network with only sparse learnable parameters in order to interpolate data and produce an intensity function that provides the data frequency at every point. By combining high and low pass interpolation approaches, it is possible to construct a stable interpolation prediction network.

The interpolated data together with the frequency map is then fed into any type of standard neural network as it is regularly sampled and partially observed.

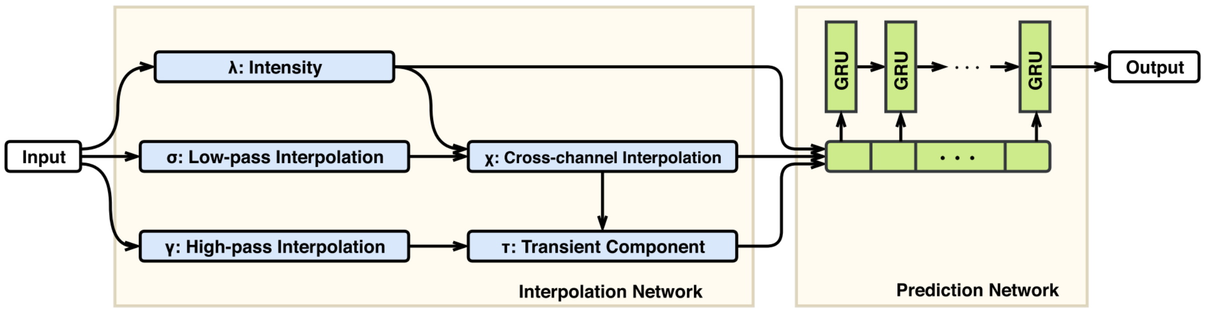


Figure 11. The interpolation network. Figure from [49]

3.1.3. Additional Input

Neural networks are arbitrary function estimators and as such can model in theory every mapping from input to output data. They learn to extract the relevant information from the incoming data to produce as accurate results as possible. It is therefore reasonable to simply feed additional information to the network when series are not sampled regularly. For instance, the interval Δt can be concatenated to the observed data and fed to any model. In the case of missing data or partial observation, an observation mask can be created and also concatenated to the observation and then fed to the network. Thereby the model receives all information necessary to entirely describe the observed time series. It can learn to interpret the observational mask as a means to decide which information from the other input it deems to be relevant.

In literature, two distinct approaches can be found. The naive approach is, as described, to not alter any model architecture and simply concatenate the observed data, the time interval, as well as the observational mask and feed everything to the model [60, 61, 62]. In many cases however, model convergation can be significantly enhanced by altering the architecture in such a way to take advantage of the different kinds of inputs. Several such approaches of the latter category will be described in this section.

Let us first formally define how to construct an observational mask as proposed by Kidger [2].

Definition 3.1 (Observational Mask). Consider a time series $TS = ((t_1, x^{<t_1>}), ..., (t_n, x^{<t_n>}))$ according to definition 1.1. An *Observational Mask* is then constituted by a function ψ:

$$\psi : \mathbb{R} \cup \{\varnothing\} \to \{0, 1\}$$

$$z \mapsto \begin{cases} 0, & \text{if } z = \varnothing \\ 1, & \text{if } z \in \mathbb{R} \end{cases}$$

that replaces the missing value token $\varnothing$ with 0 and every other value by 1.

The data itself is then also mapped onto real values by simply replacing the missing value token $\varnothing$ with 0:

$$\phi : \mathbb{R} \cup \{\varnothing\} \to \mathbb{R}$$

$$z \mapsto \begin{cases} 0, & \text{if } z = \varnothing \\ z, & \text{if } z \in \mathbb{R} \end{cases}$$

Together with the observational mask, the model can then learn that a 0 does only sometimes mean that the data was observed with value 0, while it sometimes means that no data was observed at all. It can then draw different conclusions from this information [61].

The methods reviewed here are (in chronological order of the data of publication):

Intensity Functions (Du et al. [63], Rubanova, Chen & Duvenaud [6])

Instead of simply concatenating more information to the input of a model in use, the idea of intensity functions is to continuously model the frequency of observation at any given point. [6] use simple poisson point processes, which model the likelihood of an observation at any given point, and feed this information to the model addtitionally. They showed empirically that this information enhances model performance in principle by applying it to a toy dataset. In their real word physionet dataset however, they noticed that this additional information yielded no performance gain.

[63] on the other hand propose an extension to a RNN architecture that not only outputs predictions about the observed data, but also models the poisson process likelihood of a next observation coming in.

Since such models have access to the current observational rate (frequency of observations), they are able to extract information from the fact that data is either missing or unusually frequently observed.

Time Embeddings (Sousa, Pereira & Soares [64])

Instead of just feeding Δt or t_i to a network in addition to the observed data, the the authors propose to embed this information into a latent space containing information also on the frequency of the observed data.

3.2. RNN Cell Adaptation Methods

After looking at approaches that modify steps *prior* to feeding the data into the model, i.e., modifying the data before feeding it to a standard model, we will now look at approaches that propose to alter RNN model architectures to properly take advantage of the additional information available. Still, the model has to be fed this additional information, which may be done with sophisticated techniques as discussed in the preceding section.

In the context of RNNs, it is vital to reiterate the fundamental assumption of such networks: They keep a hidden state which is updated upon receiving new data input. This hidden state is a representation, or encoding, of the observed system, with regard to the objected inference, off of which predictions can be made. In standard RNNs this hidden state remains constant – or undefined – in-between observations. Most of the proposals therefore change aspects about how

Table 1: Comparison of the presented basic methods. “(pc)” denotes a partially continuous hidden state. Most of the approaches serve the pipeline prior to feeding data into the model and therefore pose no implications for the hidden state of any ensuingly adopted model (–).

Method	**Different Lenghts**	**Irregular Sampling**	**Missing Values**	**Partial Observation**	**Informative Missingness**	**Hidden State**	**Costs**
KNN [52]	✓	✓	✓	✓	×	–	high
MICE [53]	✓	✓	✓	✓	×	–	low
Random Forests [54]	✓	✓	✓	✓	×	–	low
PCA [55]	✓	✓	✓	✓	×	–	low
Gaussian Proccesses							
[43, 56, 57, 58, 59]	✓	✓	✓	✓	(✓)	–	low
GRU-D [47]	✓	✓	✓	✓	✓	(pc)	low
BRITS [48]	✓	✓	✓	✓	×	(pc)	low
GAN [48]	✓	✓	✓	✓	×	–	high
Interpolation							
Network [49]	✓	✓	✓	✓	✓	–	low
Intensity							
Functions [63, 6]	✓	✓	✓	✓	✓	–	low
Time Embeddings [64]	✓	✓	✓	–	✓	–	low

the hidden state evolves and how it is updated upon an observation coming in.

The approaches discussed here in chronological order are as follows:

Phased LSTM (Neil, Pfeiffer & Liu [65])

This model is an extension to the standard LSTM cell in the context of Recurrent Neural Networks. They introduce a new time gate into the architecture (Fig. 12b) that controls when updates to the hidden and cell state are allowed. It models an oscillation between *open* and *closed* states, thereby forcing updates only in certain intervals. The oscillation's parameters are learnt during training.

This approach was motivated by sensor data collections that produce data with very short time gaps yet still irregularly. It is most probably not suited for applications that include very long gaps in the sequence.

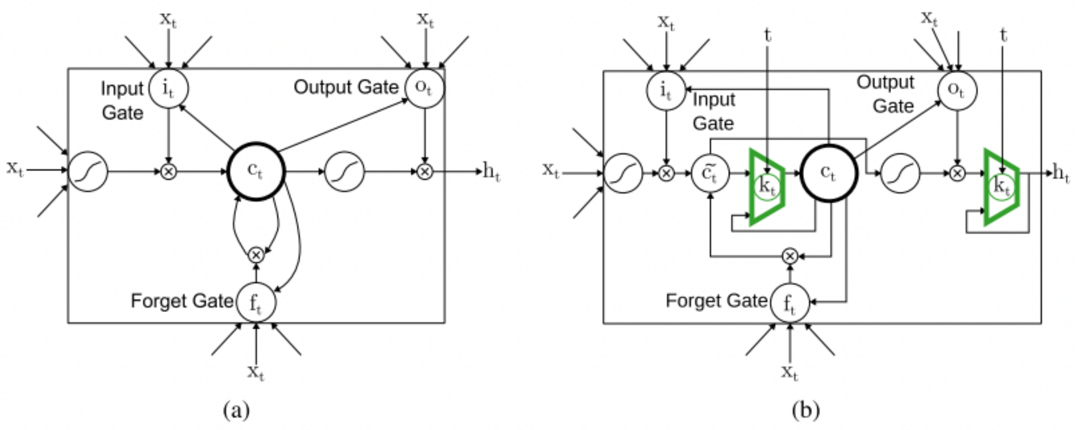


Figure 12. Illustration of a standard LSTM (a) and the Phased LSTM (b). Figure from [65]

CT-GRU (Mozer, Kazakov & Lindsey [66])

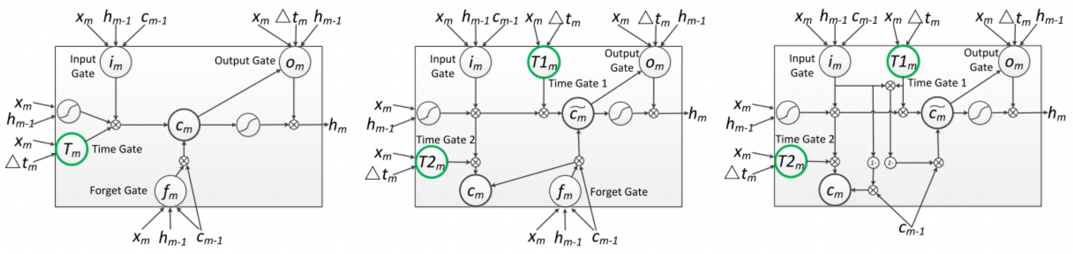
Here, instead of an LSTM, a GRU cell is adapted to be applied to the continuous time domain. They therefore replace the gating approach with a scaling approach. The idea is to scale stored information in a way that is dependent on the length of the gap since the last data has come in. Essentially, the information kept inside the cell exponentially decays over time.

1. *Determine retrieval scale and weighting*	$\ln \boldsymbol{\tau}_k^R \leftarrow \boldsymbol{W}^R\boldsymbol{x}_k + \boldsymbol{U}^R\boldsymbol{h}_{k-1} + \boldsymbol{b}^R$ $\boldsymbol{r}_{ki} \leftarrow \mathrm{softmax}_i\left(-(\ln \boldsymbol{\tau}_k^R - \ln \tilde{\tau}_i)^2\right)$
2. *Detect relevant event signals*	$\boldsymbol{q}_k \leftarrow \tanh(\boldsymbol{W}^Q\boldsymbol{x}_k + \boldsymbol{U}^Q(\sum_i \boldsymbol{r}_{ki} \circ \hat{\boldsymbol{h}}_{k-1,i}) + \boldsymbol{b}^Q)$
3. *Determine storage scale and weighting*	$\ln \boldsymbol{\tau}_k^S \leftarrow \boldsymbol{W}^S\boldsymbol{x}_k + \boldsymbol{U}^S\boldsymbol{h}_{k-1} + \boldsymbol{b}^S$ $\boldsymbol{s}_{ki} \leftarrow \mathrm{softmax}_i\left(-(\ln \boldsymbol{\tau}_k^S - \ln \tilde{\tau}_i)^2\right)$
4. *Update multiscale state*	$\hat{\boldsymbol{h}}_{ki} \leftarrow \left[(1 - \boldsymbol{s}_{ki}) \circ \hat{\boldsymbol{h}}_{k-1,i} + \boldsymbol{s}_{ki} \circ \boldsymbol{q}_k\right] e^{-\Delta t_k/\tilde{\tau}_i}$
5. *Combine time scales*	$\boldsymbol{h}_k \leftarrow \sum_i \hat{\boldsymbol{h}}_{ki}$

Figure 13. The governing update equations of a CT-GRU. Figure from [66]

Time-LSTM (Zhu et al. [67])

This approach adapts again the architecture of an LSTM cell. Three distinct types are proposed. Each time, one or several new time gates are added, which are modeled in a similar way to the standard LSTM gates, and then included into the update equations. This way, the model can learn to extract gating information from Δt, hence learn depending on the time gap, how much and how to update the hidden and cell states.



(a) Time-LSTM 1 (b) Time-LSTM 2 (c) Time-LSTM 3

Figure 14. An illustration of the different types of extensions to the LSTM architecture as proposed under this approach. Figure from [67]

Time-Aware LSTM (Baytas et al. [68])

This extension to the LSTM architecture introduces a single additional gate that receives Δt at each timestep. This allows the cell state to be updated based on this gating operation. The model is thereby capable to better represent short and long term dependencies in the irregular data domain.

GRU-D (Che et al. [47])

This approach was directly motivated by the medical domain. It was therefore developed with certain assumptions that are applicable to medical data, but not necessarily to other domain. The two basic assumptions the authors based their architecture on, are:

- The influence of a variable that has not been observed for a very long time should gradually

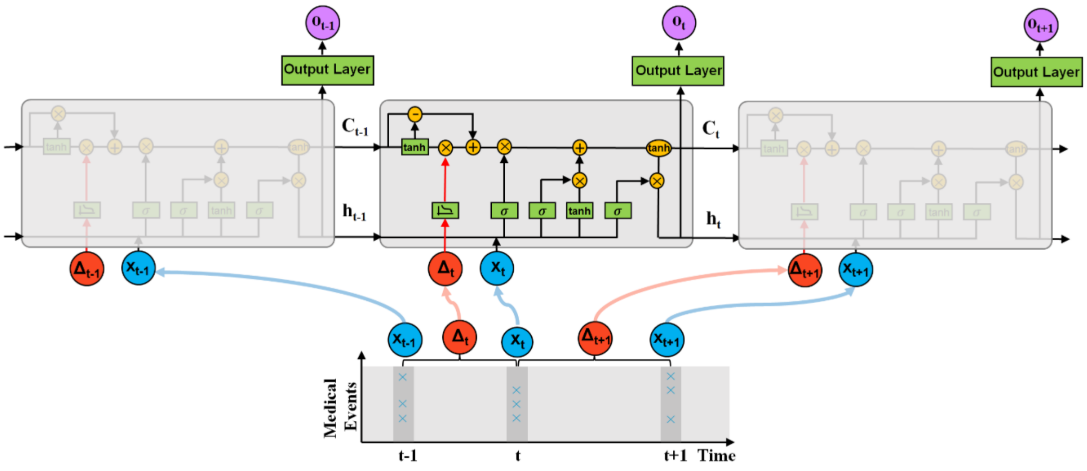


Figure 15. An illustration of the architecture of the T-LSTM cell. Figure from [68]

decay.

- A variable that has not been observed for a very long time will tend to fall back to some default value.

Based on these assumptions, the authors propose the following decay term:

$$\gamma^{<t_i>} = e^{-max\{0, W_\gamma \Delta^{<t_i>} + b_\gamma\}}$$

Notice that this term contains two learnable parameters W_γ and b_γ. As depicted in Figure 16, such decay terms are inserted both for the hidden state, as well as the input value. It is thereby possible to draw conclusions based on the fact that data has been missing for a long time.

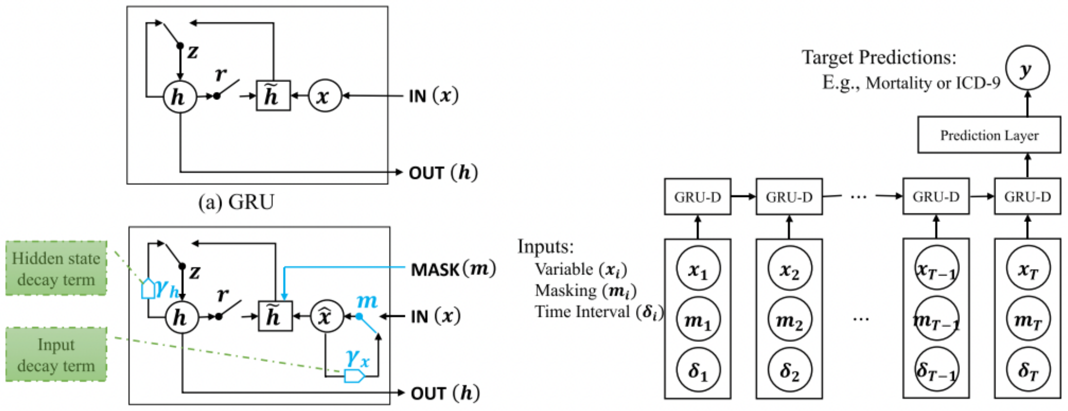


Figure 16. Top left shows a normal GRU cell, whereas bottom left shows the proposed GRU-D cell which introduces two decay terms. This cell gets as input the complete set of information as discussed in the beginning of this section. Figure from [47]

ODE-RNN (Rubanova, Chen & Duvenaud [6])

We now look at the first approach that proposes a continuously evolving hidden state in between observations. The authors propose to have a standard GRU cell, however, instead of having

constant hidden states in between observations, they let the hidden state evolve according to a Neural ODE. They show, that this is a generalization of the CT-GRU cell [66].
After the ith observation, the hidden state evolves continuously according to this Neural ODE from t_i to t_{i+1} until a new observation comes in, at which point the hidden state is updated based on standard GRU update equations. However, instead of using the hidden state from the last timestep, the evolved hidden state at time t_{i+1} is used instead. This way, the model only implicitly depends on Δt.

The hidden state is here evolved continuously in between observation, but updated discretely upon a new observation, resulting in jumps.

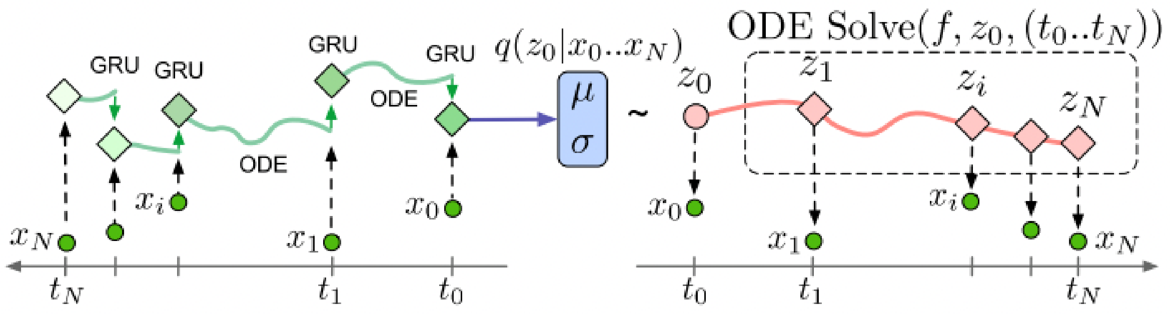


Figure 17. The left shows an illustration of an ODE-RNN. Figure from [6]

GRU-ODE-Bayes (De Brouwer et al. [69])

This approach was published independently from [6] and is very similar. It also utilizes a GRU cell to build a Recurrent Neural Network for time series inference, by also using a Neural ODE to evolve the hidden state in between observations. Updates to the hidden state are done upon an incoming new observations in a discrete bayesian fashion.

ODE-GRU-D (Habiba & Pearlmutter [70])

This approach was motivated by the effectiveness of the GRU-D cell [47] and the applicability of Neural ODEs to continuous time domains to leverage the notion of informative missingness. It is yet another similar work to [6] and [69] as it proposes to let the hidden state evolve continuously according to a Neural ODE. The difference being that this approach is an extension to the GRU-D cell.

TA-GRU (Lukoševičius & Uselis [71])

The authors here propose to update Equation 22 governing the hidden state update of a GRU cell as follows:

$$h^{<t>} = (\Delta t\, \Gamma_u) * \tilde{h}^{<t>} + (1 - \Delta t\, \Gamma_u) * h^{<t-1>}$$

By adding this to the update equation without making further changes, the model does not increase

Table 2: Comparison of the presented RNN-adaptation methods. *(d)* denotes a discrete hidden state, *(pc)* denotes a partially continuous hidden state (mostly continuous in-between observations, but updated discretely upon an observation).

Method	**Different Lenghts**	**Irregular Sampling**	**Missing Values**	**Partial Observation**	**Informative Missingness**	**Hidden State**	**Costs**
Phased LSTM [65]	✓	✓	✓	×	×	(d)	low
CT-GRU [66]	✓	✓	✓	×	×	(d)	low
Time-LSTM [67]	✓	✓	✓	×	×	(d)	low
T-LSTM [68]	✓	✓	✓	×	×	(d)	low
GRU-D [47]	✓	✓	✓	✓	✓	(pc)	low
ODE-RNN [6]	✓	✓	✓	×	✓	(pc)	low
GRU-ODE-Bayes [69]	✓	✓	✓	×	✓	(pc)	high
ODE-GRU-D [70]	✓	✓	✓	✓	✓	(pc)	high
TA-GRU [71]	✓	✓	✓	×	×	(d)	low

the size of its parameters, thereby making this a very lightweight approach. It allows the update equation to adapt to the time gap of the observation.

3.3. ESN-based Methods

Echo State Networks are also a staple in sequence processing, however, just as RNNs, are built to work with regularly spaced and equally observed data. One could also take the naive approach here an simply feed more information to the network, however, in the case of ESNs it is even more difficult to get good results since no relationships are learnt, the information is simply expanded into the reservoir.

Several authors therefore proposed various techniques to extend the basic architecture of ESNs in order to be applicable in more general circumstances when working with time series data.

In chronological order, these methods were deemed most relevant:

Leaky Integrator Neurons (Jaeger et al. [72], Lukoševičius et al. [73])

The notion of leaky integrator neurons has found its way into standard ESN definitions for a long time. The standard way of updating the hidden state of an ESN (Eq. 24) includes a leaking rate α out of the box. This leaking rate specifies the speed of the network. A higher leaking rate leads to the reservoir evolving more quickly when new data comes in. Another way to think about the leaking rate is that it specifies how much information of the previous hidden state leaks into the

Table 3: Comparison of the presented ESN-adaptation methods. *(d)* denotes a discrete hidden state.

Method	**Different Lenghts**	**Irregular Sampling**	**Missing Values**	**Partial Observation**	**Informative Missingness**	**Hidden State**	**Costs**
Leaky Integrator Neurons [72]	✓	✓	✓	×	×	(d)	low
CT-ESN [75]	✓	✓	✓	×	×	(pc)	low
TA-ESN [75]	✓	✓	✓	×	×	(d)	low

updated hidden state.

The authors of [72] take this one step further, by not setting a constant speed of the network, but rather setting this value based on the most recent time gap, thereby adapting the speed according to the incoming data. Equation 24 to update the hidden state is adapted to:

$$h^{<t_i>} = (1 - \alpha\gamma(t_i))\, h^{<t_{i-1}>} + \alpha\gamma(t_i)\, \widetilde{h}^{<t_i>} \tag{29}$$

with $\gamma(t_i) = ||x^{<t_i>} - x^{<T_{i-1}>}||$.

Continuous-Time ESN (Anantharaman et al. [74])

The authors of this paper demonstrate a Neural ODE where all layers of the defining neural network are fixed at random, and only the linear readout is trained. This can be seen as a continuous time counterpart of Echo State Networks.

However, here, the hidden state is still discretely updated when a new observation comes in. Furthermore, stable empirical evaluation is missing. Overall, the paper leaves several questions unanswered and can therefore not be considered to be a baseline approach.

TA-ESN (Lukoševičius & Uselis [71])

The approach here is overall very similar to the leaky integrator neurons approach from [72], albeit they take the time gap Δt directly:

$$h^{<t_i>} = (1 - \alpha\Delta t)\, h^{<t_{i-1}>} + \alpha\Delta t\, \widetilde{h}^{<t_i>} \tag{30}$$

However, the main contribution here is the proposal of a time gap scaling function that serves the purpose of not letting the term $(1 - \alpha\Delta t) < 0$. The function is defined as: $f(\Delta t_n) = 1 - e^{-\Delta t_n}$.

3.4. Stochastic Methods

The methods in this section are repetitive. All of them serve to impute data prior to feeding it into the network. Still, it was decided to dedicate a section to these approaches to give special consideration to stochastic approaches. They were briefly mentioned in Section 3.1.2.

The approach here is more or less the same in every proposed paper [43, 57, 58, 59] going back decades. The authors of [43] from 1992 do note, interestingly, that the methods discussed in their paper should have been pointed out in earlier decades, which, however, was not possible, since the available computing power had just reached a level where such approaches had become feasible.

Gaussian processes are used to interpolate time series data. The result of such a process is a probability distribution over all possible interpolation functions from which data points can be sampled at arbitrary intervals. Figure 18 shows the result of such an operation.

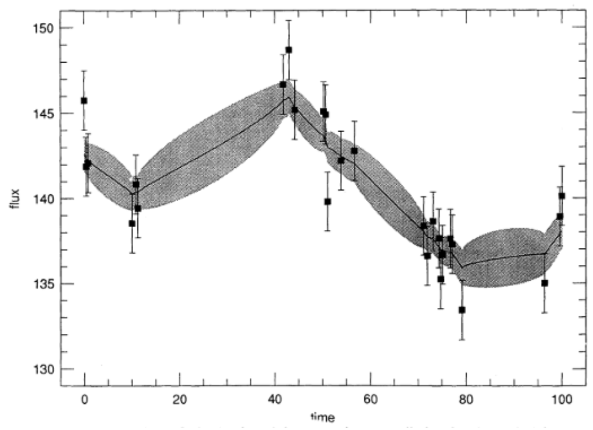


Figure 18. The result of a gaussian process for interpolation.

3.5. Neural Differential Equations

Neural ODEs were introduced in Section 2.3.3 as popularized by [1]. The potential of a continuous hidden state for time series analysis is apparent and was already briefly mentioned in the original paper [1]. Since then, many researchers went to work and extended or applied the idea of Neural ODEs to sequence data processing in order to cope with irregularly sampled and partially observed data. Some approaches already are quite sophisticated, however, conclusive empirical evidence is still rare. Patrick Kidger can be seen as being the most influential author in the field, aside from the original authors of the Neural ODE paper, as he generalized the approach to a mathematical framework [2]. He therefore appears as an author in many of the following works.

Latent ODEs (Chen et al. [1], Rubanova, Chen & Duvenaud [6])

This method is a sequence to sequence model implemented as a variational autoencoder. An ODE-RNN is is used for encoding, with the special property that it runs backwards in time, while a plain Neural ODE is used for decoding. The authors motivate this approach as opposed to using the ODE-RNN directly as an autoregressive model, mainly by interpretability of the hidden state. Furthermore, latent variable models are able to model uncertainty via the probability distribution that is encoded. They also argue, that latent variable models perform better in domain where data is sparsely sampled.

The ODE-RNN (Fig. 19 left) is, architecture-wise, a normal RNN with any arbitrary cell architecture. The only difference is that the hidden state in between observations evolves according to a Neural ODE. This means that in the update equations of RNN cells, the previous hidden state $h^{<t_{i-1}>}$ has to be replaced by $ODESolve(f_\theta, h^{<t_{i-1}>}, t_{i-1}, t_i)$, the solution of an ODE, given the initial condition $h^{<t_{i-1}>}$, computed in the interval (t_{i-1}, t_i) according to the vector field parameterized by the neural network f_θ. This introduces a completely new neural network that has to learn the trajectories of the hidden states.

The final hidden state of the the ODE-RNN is then fed into yet another feed-forward neural network that outputs two values specifying mean and variance of the latent state. This is used as the approximate posterior at time t_i. A sample from this latent state is then generated and subsequently used as the initial condition for another Neural ODE which reconstructs the time series starting at t_0. This second reconstructing ODE is completely deterministic.

The authors showed that this approach is very powerful for interpolation and extrapolation, while they did not provide conclusive empirical evidence that the hidden state as a representation for classification tasks provides a leap with regard to other approaches.

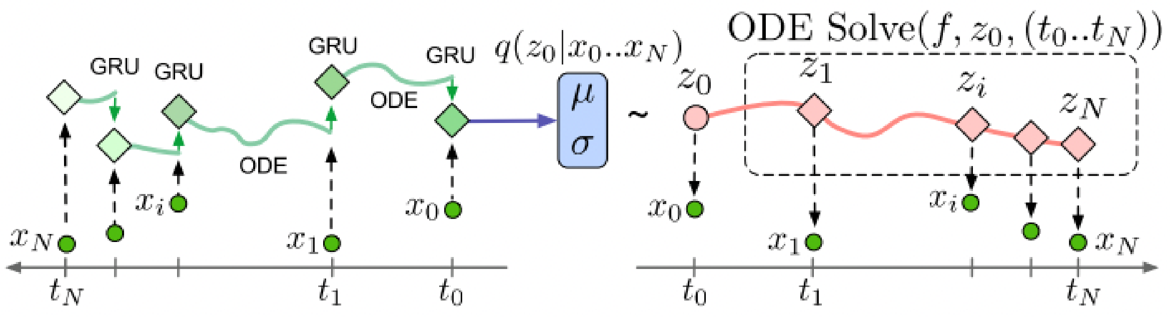


Figure 19. Illustration of a latent ode. Figure from [6]

GRU-ODE-Bayes (De Brouwer et al. [69])

Already described in Section 3.2.

Neural Controlled Differential Equations (Kidger et al. [42], Kidger [2])

As described in this work's introductory remarks, a perfect temporal model would have a continuous hidden state, even upon new observations coming in. All approaches discussed until now either have completely discrete hidden states (stay constant in between observations), or partially continuous hidden states (continuous in between observations, but discretely updated upon a new observation coming in). Discrete updates to continuously evolving hidden states are necessary due to ODEs being completely determined by their initial condition and thereafter evolve deterministically. Controlled Differential Equations (CDEs) are well understood in mathematics and extend ODEs in such a way that the ODE can adapt its trajectory according to a second function which may not be completely available at the beginning. Kidger et al. applied the same idea that underlies Neural ODEs to CDEs in order to obtain Neural CDEs. This CDE

can be thaught of as a ODE that is driven by a second function. In the context of Neural CDEs this second function is proposed to be an interpolated version of the observations. The authors note that this interpolation (for which they use cubic splines) is just a necessary means of feeding the observations to the model, which requires a continuous representation of the input data, but does not constitute a pruning of information that other interpolation methods often entail.
A Neural Controlled Differential Equation is the solution to the following Riemann-Stieltjes integral, as defined in [2]:

$$h(t) = h(0) + \int_0^t f_{\theta_0}(h(s))dx(s) \tag{31}$$

where $h(0) = f_{\theta_1}(x(0))$ the input data embedded via some neural network $f_{\theta_1} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_h}$, $f_{\theta_0} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h \times d_x}$ and $x : \mathbb{R} \to \mathbb{R}^{d_x}$ the interpolated input data.

They show that this model can be seen as the continuous time limit of RNNs and that it can naturally handle irregularly sampled and partially observed data by interpolating each channel individually.

Figure 20 depicts on the left an approach where the hidden states are updated by discrete jumps upon observing new data. Neural CDEs are shown on the right, where the observations are embedded into continuous path space (by cubic spline interpolation for instance) which then drives the Neural CDE's continuous hidden state.

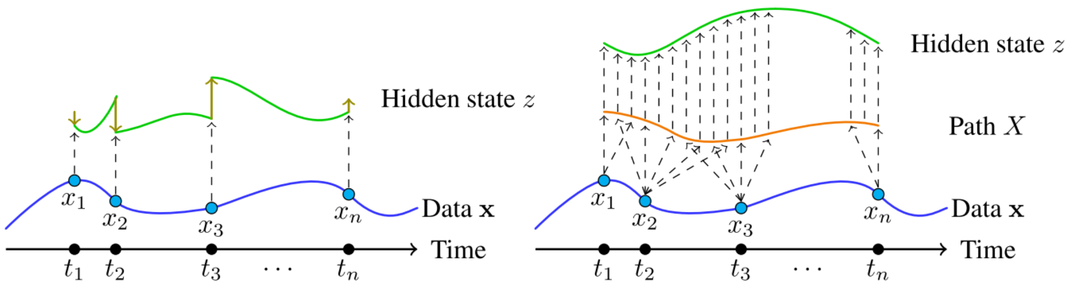


Figure 20. Illustration of Neural Controlled Differential Equations. Figure from [42]

Neural Rough Differential Equations (Morrill et al. [76])

This approach is an extension to Neural CDEs motivated by extremely long time series and their long-term dependencies. Here, instead of one continuous interpolated function driving the ODE, instead multiple data points are subsumed via their *log-signature* over short intervals which. It is these summaries over intervals that is then used to drive the hidden state. The term used for this approach stems from the mathematical background of rough path theory.

Neural Stochastic Differential Equations (Tzen & Raginsky [77], Li et al. [78], Hodgkinson et al. [79], Kidger et al. [80], Kidger [2])

This type of Neural Differential Equation is yet another extension to Neural RDEs. Here, brownian motion is introduced in addition to the input signal so that some statistical uncertainty

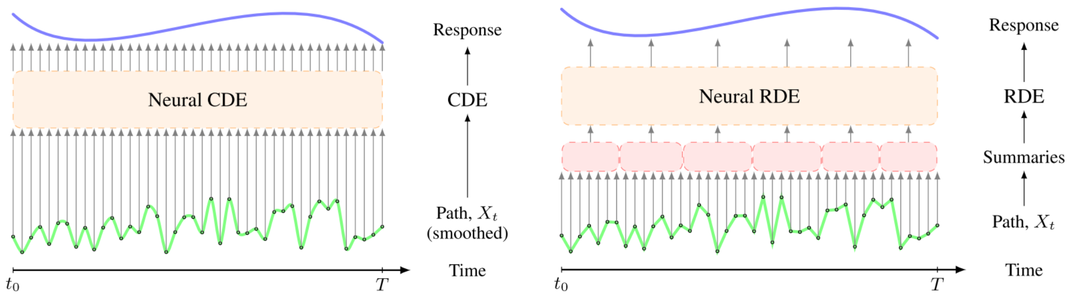


Figure 21. Illustration of Neural Rough Differential Equations. Figure from [76]

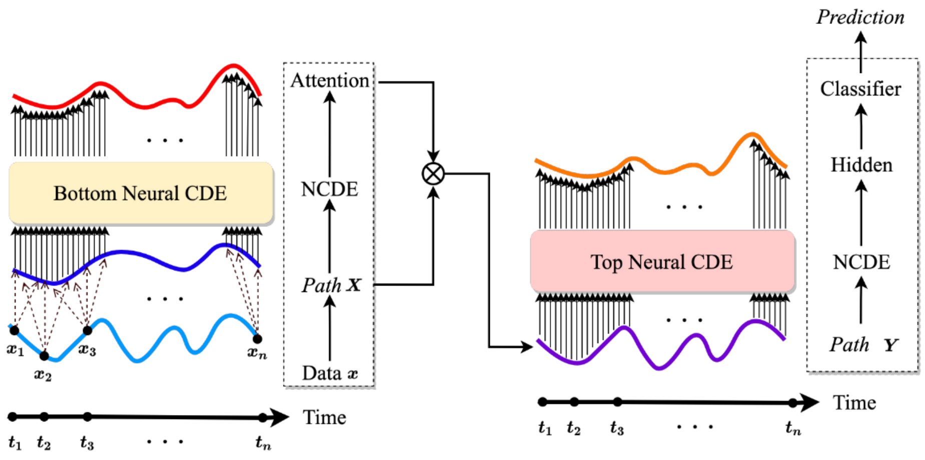


Figure 22. Illustration of a standard Neural CDE (a) and the proposed Attentive Neural CDE. Figure from [81]

can be represented. [2] notes that a hierarchy can be constructed of generalizations starting from Neural ODEs, to Neural CDEs, to Neural RDEs and finally to Neural SDEs.

ODE-GRU-D (Habiba & Pearlmutter [70])

Already described in Section 3.2.

Attentive Neural Controlled Differential Equations (Jhin et al. [81])

This interesting paper proposes to use two separate Neural CDEs for modeling attention-inspired behavior. One Neural CDE is driven by the input signal directly and produces a path that serves as an attention mechanism, which is then in turn used to drive the second Neural CDE.

Neural Flows (Biloš et al. [82])

Here, the authors proposed the idea to directly model the hidden states trajectory by a neural network, instead of having the neural network model the vector field which then requires several calls to an ODE solver for a forward pass. In their approach, the solution trajectories are directly approximated, yielding less computational costs and favorable performance in theory.

Table 4: Comparison of the presented methods that rely on NDEs. *(pc)* denotes a partially continuous hidden state, *(c)* denotes a continuous hidden state.

Method	**Different Lenghts**	**Irregular Sampling**	**Missing Values**	**Partial Observation**	**Informative Missingness**	**Hidden State**	**Costs**
Latent ODEs [1, 6]	✓	✓	✓	×	✓	(pc)	high
GRU-ODE-Bayes [69]	✓	✓	✓	×	✓	(pc)	high
NCDEs [42, 2]	✓	✓	✓	✓	✓	(c)	high
NRDEs [76]	✓	✓	✓	✓	✓	(c)	high
NSDEs [77, 78, 79, 80, 2]	✓	✓	✓	✓	✓	(c)	high
ODE-GRU-D [70]	✓	✓	✓	✓	✓	(pc)	high
Att. NCDEs [81]	✓	✓	✓	✓	✓	(c)	high
Neural Flows [82]	✓	✓	✓	×	×	(c)	high
EXIT [83]	✓	✓	✓	✓	✓	(c)	high

EXIT (Jhin et al. [83])

The authors of this paper were motivated by the issue they saw in having an interpolated input data path used as the driving path for a Neural CDE. They approached this in a similar way as [81], as they also propose to use two stacked Neural CDEs, where the first one would produce a path that is then used to drive the second one. Here, however, the generated trajectory of the first Neural CDE is directly applied to the second one, while in [81], the result is first multiplied by the original interpolated input data, in order to get a path corrected for attention.

3.6. Summary

This section covered many different ways to approach universal time series processing. First, simple methods were discussed that pre-process the data in such a way that a normal black-box model meant for regular time series can be used. This was either done by imputing missing values, or by appending more information, such as the elapsed time since the last observation, to the data. Next, methods were discussed that alter parts of the cell architecture of various RNN models. Of the simpler approaches, GRU-D [47] is widely seen as state of the art, while newer and more complex models, such as the ODE-RNN, that use ODEs to evolve a hidden state in between observations, are more powerful and more expressive. Afterwards, various ESN based approaches were discussed. This area is still sparsely researched in terms of continuously evolving hidden states. After looking briefly at stochastic methods, the chapter concluded by discussing the newest state of the art approaches, namely Neural Differential Equations.

4. Methods and Evaluation

The preceding chapter discussed different types of methods proposed in popular literature that allow, to certain degrees, the processing of arbitrarily sampled sequential data. When assessing these presented methods with regard to the notion of a "perfect model" (i.e., being modeled around a continuous hidden state, see Fig. 2), only Neural Controlled Differential Equations and extensions thereof (such as Neural Stochastic Differential Equations) comply to this notion. They keep an internal hidden state, that is continuously influenced by incoming data and can therefore also continuously be used for readouts.

When work on the present thesis started, the concepts of Neural Differential Equations have just been proposed and have quite quickly popularized due to their unique properties. It was only later after having started to work on this thesis, that Neural Controlled Differential Equations have been proposed and were established as the state of the art approach for time series processing with irregular sampling, partial observations and missing data. For these reasons, the focus of this work shifted after having started to work on it.

The first section of this chapter is therefore a pre-focus-shift matter, that was then abolished. It was decided to nevertheless include the partial results in this thesis as to resemble the work that went into it, as well as to draw conclusions from the empirical experiments.

Section two then presents the main contribution of this thesis. The Neural CDE approach was extended and combined with the field of Echo State Networks to create a novel continuous time ESN that is invariant to the sampling and observational rate of the data which it processes. Different experiments on toy and real world datasets have been conducted to evaluate the proposed method. The results are presented in this section.

4.1. TA-ESN

Echo State Networks have long been established as a viable alternative to fully trained Recurrent Neural Networks. They are especially powerful on smaller datasets whose characteristics can be quite easily extracted from a random expansion of the observed data space, which is the underlying principle of such reservoir computing approaches. Just as with standard RNNs, problems arise when the incoming data is not sampled in regular intervals, or only partially observed.

The overall fundamental challenges when dealing with such real world sequence datasets was outlined in Section 1.2. Let us iterate over each issue one by one, looking at them from the perspective of RNNs and especially ESNs.

Datasets containing sequences of varying lengths is unproblematic for any recurrent network based approach. Since a hidden state is iteratively computed over each timestep, one can stop at any timestep for different sequences in the dataset to train with, which enables this notion to be present in the data. Furthermore, while parallelization is desirable for efficient computing times, this is only possible across sequences, but not across timesteps, rendering the lengths of

the sequences indifferent in terms of parallelization potentials.
As stated multiple times before, basic recurrent network approaches are based on the assumption that incoming data is evenly spaced in time. Section 4 presents different methods for how an RNN cell can be modified, such that the hidden state follows certain trajectories in between observations, which renders it possible to handle incoming data with differing temporal intervals. The rules governing these hidden state trajectories have to either be manually engineered based on the data domain the model is applied to, or have to be learnt from the data. The latter case, while more universal and therefore more powerful, introduces additional learnable parameters into the model, making training less efficient. Furthermore, all these approaches rely on a discrete jump of the hidden state upon a new observation coming in. A readout of the hidden state right before it is updated by new data is therefore completely reliant and determined by the preceding hidden state, even though the new data might carry information of the state of the system even from before that observation was made. Instead, the new incoming data should therefore continuously influence the hidden state in both temporal directions. Neural Controlled Differential Equations pose a solution to this issue – as discussed before – and will be looked at in much greater detail in Section 4.2.

While RNNs can handle irregular sampling reasonably well by introducing an evolving hidden state when no data is observed, the same idea is unfortunately not applicable to ESNs. Since the hidden state of an ESN is updated solely by a matrix multiplication between a fixed and randomly generated recurrent weight matrix on the one hand and the observed data on the other hand, an evolving hidden state would not be conditioned on any data and therefore evolve completely at random, discarding most of the information that has just been fed to the network by the observed data. For these reasons, continuously evolving hidden states have not been proposed for ESNs in literature and are unlikely to emerge in the future.

Since ESNs themselves are discretizations of an ODE, the step size taken when a new observation comes in can be chosen depending on the time gap between the current and lastly observed data. This is precisely what TA-ESNs were designed for [71]. They condition their step size in the hidden space on the length of the time gap prior to the current observation. This of course is a very crude way to traversing the hidden space, still taking zero steps in between observations. A solution to this problem are continuous time ESNs which take smaller steps through a latent space and condition every step on ensuing and preceding observations. They are presented later in Section 4.2. To summarize, irregular sampling can be handled by TA-ESNs, although with limitations.

Partially observing data is another one of the fundamental challenges in terms of sequence processing. This poses a major challenge on all recurrent network approaches, as hidden state updates depend on every incoming dimension (reiterate that neurons work by taking a weighted sum across dimensions and then applying a non-linear function to the resulting sum). There is no way to architecturally condition state updates only on single dimensions. If this would be done, the resulting model would be unable to represent dependencies across dimensions and instead would be as powerful as having multiple models that take only single dimensions as their input. This issue is usually approached by one of two methods. The data is either imputed prior

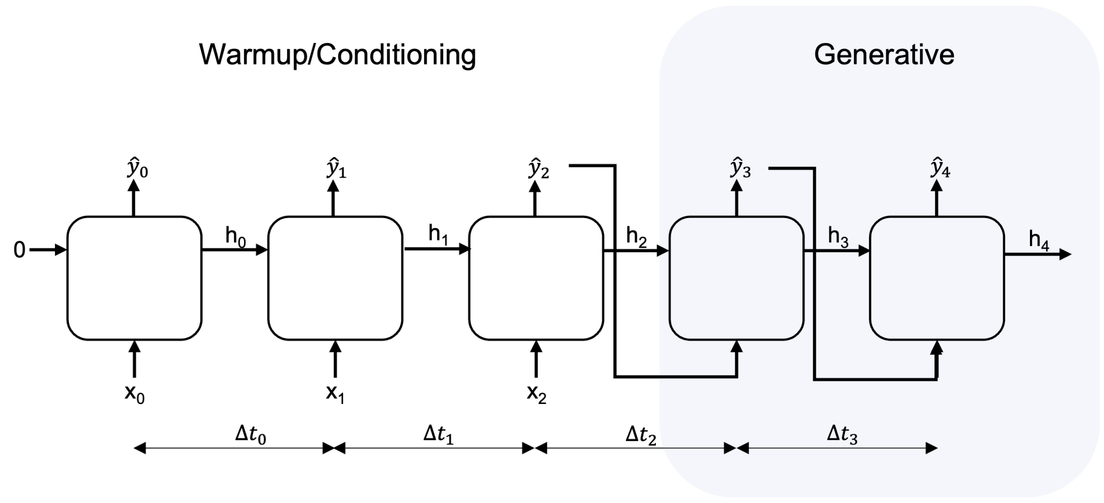


Figure 23. An RNN or ESN unrolled in time.

to feeding it to the model, or an observational mask is concatenated to the data. Again, Neural CDEs pose a natural way of handling this, which will only be discussed later. Data imputation works for any kind of model, on RNNs as well as ESNs, is however unfavorable as it modifies the data and assumes knowledge about the missing spots. It furthermore removes the possibility to draw conclusions from the notion of informative missingness. Observational masks on the other hand are only manageable by RNNs. The reason is that the model has to make a distinction between observed data and the meta information contained in the observational mask. In a standard RNN, this distinction can be learnt, while in an ESN the information contained in the observational mask is lost in the latent space and mixed with the actual observed data. It is very difficult to impossible for the readout to be able to extract this kind of information.

The last challenge of dealing with sequential data that is discussed here is the notion of informative missingness. Much the same arguments apply here as with partial observations. A meta information about the data can be concatenated to the network input where a standard RNN can learn to extract the required information. To an ESN on the other hand, this additional information is indistinguishable from the the actual observed data and therefore hard to utilize efficiently.

As outlined in the preceding paragraphs, much of the capabilities that RNNs can learn are almost impossible to achieve for ESNs with reasonable costs. The following experiments reaffirm the hypotheses made by applying TA-ESNs on different tasks and data. They show, that TA-ESNs are incapable of extracting meta informational structures from the latent dimensions without further manual architectural feature engineering.

All experiments were carried out using TA-ESNs as proposed by [71]. They have been extended to the domain of partial observations in addition to irregular sampling.

4.1.1. Extrapolation

Recurrent Neural Networks, including ESNs, can serve as great autoregressive models, by feeding the predicted next observation back into the network to create a generative autoregressive model. This works fine while the timesteps taken in between observations are constant, it however falls

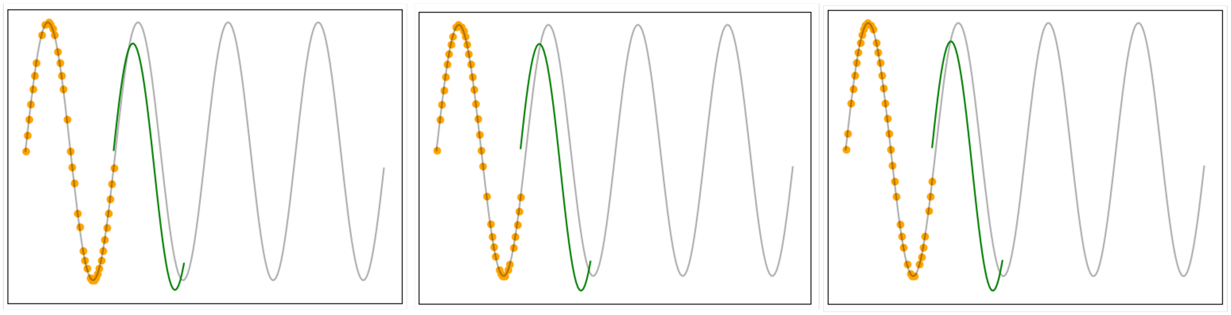


Figure 24. Some results from simple autoregressive approaches conditioned on irregularly sampled data. Orange dots denote the irregular data that the model was conditioned on, green is the generated trajectory, and grey is the ground truth.

apart as soon as the timesteps become irregular. Figure 23 shows any type of RNN schematic unrolled in time. Notice that during generative mode, the prediction $\hat{y}_i$ at time t_i is fed as input back into the network at time t_{i+1}. However, the prediction $\hat{y}_i$ is made without knowledge about the following time gap Δt_i. This time gap however is crucial since the data evolves with time. Notice further that during conditioning, the predictions are also made without knowledge about the ensuing time gap. When the intervals are constant, the network can do autoregressive generative inference since the predictions are invariant to the time gaps, as they never change. Therefore the network can implicitly assume constant timesteps and make accurate predictions. This crucial bit of information however is unavailable to the network when the intervals are variable.

Experiment

Figure 24 shows some outcomes of a TA-ESN applied to a simple extrapolation task, conditioned on irregularly sampled data. Notice that the model picks up the trajectory and produces a smooth path after the conditioning phase. This is due to the time intervals being constant after conditioning. However, due to the conditioning on irregular intervals, the readout right after the conditioning is slightly off, as the hidden state fails in accurately representing where the trajectory is currently at. This leads to small jumps and the failure of picking up the true amplitude and frequency of the since curve.

This experiment was conducted on a small dataset of randomly generated sine curves which were randomly shifted horizontally. Training and conditioning have been made on irregularly sampled data points.

4.1.2. Classification

A classification task on sequence data is different to a generative task in that no output from the model is fed back into the model. Furthermore, it is usually only the final hidden state that is obtained after traversing through the sequence that is utilized for the actual classification task. Therefore, TA-ESN models should be very capable of encoding irregularly sampled datasets into such a reasonable format, that a linear readout can be trained for the classification task. This is again normal ESN behavior, with the additional capability that the data may be irregularly sampled.

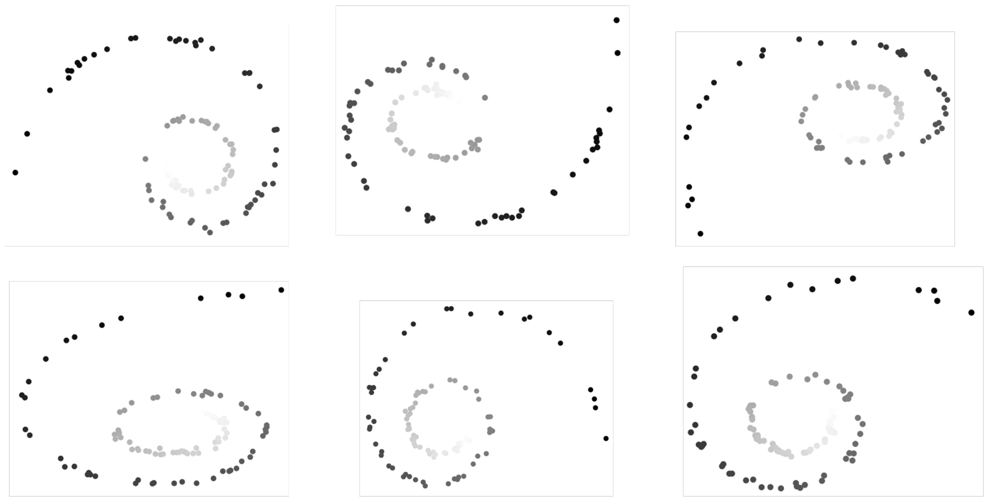


Figure 25. A sample of the toy dataset containing of clockwise (upper row) and anti-clockwise (bottom row) spirals. The lightness of the dots depicts the progress of time.

Experiment

To illustrate and test these capabilities, a simple binary classification task was carried out on a two dimensional toy dataset. The dataset contains 256 spirals, half of them going clockwise and half of them going anti-clockwise (see Fig. 25 for example spirals). Each spiral was generated by sampling 100 datapoints randomly and in potential irregular intervals. Random gaussian noise was added to each sampling point. A simple TA-ESN (spectral radius of 1.25, leaking rate of 0.3, 256 hidden units) was then trained on half the dataset and tested on the other half.

Table 6 shows the results on different dataset configurations. The 100 data points were either regularly sampled (denoted by a $\times$in the first column), or randomly subsampled from 1000 or 10000 (denoted by 10 and $100n$ in the first column, respectively) regularly spaced points. As long as the dataset is fully observed, even widely irregular data can be handled fairly well by the network (see first three rows). However, as soon as partial observation is introduced into the dataset, the model becomes weak. In further experiment runs, a random number of datapoints was discarded in individual dimensions, leading to partial observation. Since TA-ESNs have no way of representing missing data, a 0 was passed in those spots. In this case, some points may only have one dimension observed, while the other is 0. Now RNNs may struggle with this setup as well, which is why they are usually passed an observational mask of the data as well. The network can then learn which input is now a real observation and which is not. However, in the case of ESNs, as already stated, this information is dispensed into the reservoir and becomes hardly distinguishable from actual data. The results of the experiments where an observational mask was passed to the TA-ESN underscore this hypothesis. The accuracy is close to a random guess classifier when little datapoints were observed and the observational mask was passed to the network.

Table 5: Results of applying a TA-ESN on the classification task of the spirals. “10n” in the first column indicates that the samples were subsampled from 10 times the amount of regularly spaced points. The values in the second column denote the fraction of observations that are kept.

Irregular Sampling	**Partial Observation**	**Observational Mask**	**Accuracy**
×	×	×	1.000
10n	×	×	0.961
100n	×	×	0.959
×	0.9	×	0.911
10n	0.9	×	0.923
10n	0.9	✓	0.834
×	0.5	×	0.754
10n	0.5	×	0.671
10n	0.5	✓	0.589
×	0.2	×	0.620
10n	0.2	×	0.580
10n	0.2	✓	0.537

4.1.3. Encoder

This is a brief comment only. During experimentation, it have tried to replace the ODE-RNN encoder of a Latent ODE [6] with a TA-ESN. The architecture is as follows. The irregularly sampled dataset is passed to the TA-ESN which encodes the sequences into a state of its reservoir. This reservoir state is then passed into a small MLP to create the latent space of the variational autoencoder. This latent space is subsequently sampled from and passed into an ODE that aims to reconstruct the original series. It is thereby possible to perform both interpolation and extrapolation on the encoded series.

The original version utilizes an ODE-RNN which is a type of fully trained Recurrent Neural Network with its hidden state evolving according to another learnt Neural ODE. During experimentation, I was unable to reconstruct sequences to a satisfying degree using a TA-ESN as the encoder part of the model. While all the information necessary to recreate the series is in fact encoded in its reservoir, it is really hard to extract this information from this random state even for a small trained MLP. This shows that a classification task can indeed be carried out on relatively cheaply generated reservoirs, while the information necessary to recreate the entire sequence is difficult to extract from the random reservoir.

4.1.4. Summary

This section discussed various aspects about Echo State Networks and their shortcomings. Different experiments were carried out. It was empirically evaluated that ESNs can be utilized for classification tasks, even on irregularly sampled data, as long as the data is fully observed. The random reservoir encoding of information has mixed the input data, thereby also making no distinction between actual observations and meta data, such as observational masks. It is therefore almost impossible to create a version of ESNs that can handle partial observations in a natural way. Furthermore, it was shown why ESNs cannot properly extrapolate sequences in an autoregressive manner when they have been trained or conditioned on irregularly sampled data.

4.2. CDE-ESN

Now to the main contribution of this thesis. While working on the present thesis over the past two years, a new method for handling sequential data has been established as the new state of the art approach. It is naturally able to handle every challenge that was discussed in the introductory remarks and in the preceding section. Neural Controlled Differential Equations can deal with sequences of different lengths, that are partially observed and contain information in the missingness of data itself. Kidger [2] describes Neural CDEs as continuous time RNNs. They are powerful tools on arbitrarily sampled time series with only one drawback. Neural CDEs are, as most sophisticated deep learning approaches, computationally expensive and slow. On the other hand, ESNs are not as powerful, yet computationally cheap and very fast. The drawbacks of ESNs in terms of universal time series processing were outlined in the preceding section. For these reasons, the focus will shift now to constructing a methodology that takes the best of both worlds.

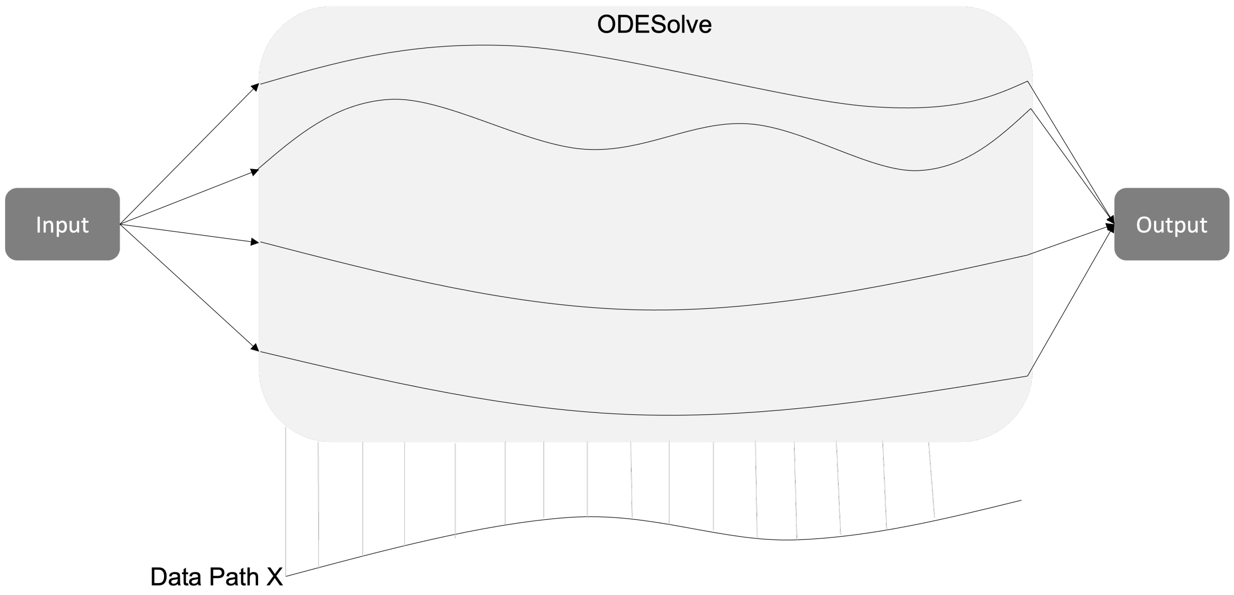


Figure 26. Illustration of a neural CDE.

Let us reiterate how and why Neural Controlled Differential Equations work in detail. The central equation governing a Neural CDE according to [2] is as follows. The function $h : [0, T] \to \mathbb{R}^{d_h}$ solves the *controlled differential equation* driven by x, if the following equations are satisfied:

$$h(0) = f_{\theta_{in}}(x^{<t_0>}), \quad h(t) = h(0) + \int_0^t f_{\theta_r}(h(s))\, dx(s) \quad \text{for } t \in (0, T] \tag{32}$$

where $T > 0$ is the temporal length of the sequence, $d_x, d_h \in \mathbb{R}$ are input and hidden dimensionality, respectively, $x : [0, T] \to \mathbb{R}^{d_x}$ is a continuous bounded function, and $f_{\theta_{in}} : \mathbb{R}^{d_x} \to R^{d_h}$ and $f_{\theta_r} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h \times d_x}$ are neural networks. Let further $f_{\theta_o} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_y}$ be another neural network with the desired output dimensionality d_y. All this together describes the entirety of what Neural Controlled Differential Equations are.

Figure 27 depicts the (very) abstract computation graph that underlies any Neural CDE. In order to compute one forwards pass, the input data is passed through $f_{\theta_{in}}$, which usually expands the dimensionality into the hidden space. Then, starting at this value, the latent state evolves according to the differential equation above, whose vector field is in turn parameterized by f_{θ_r}. Hence, the CDE is solved (or approximated) in the interval $[0, T]$, with its initial condition being $h(0) = f_{\theta_{in}}(x^{<t_0>})$. The final evolved hidden state $h(T)$ is then in turn passed through the function f_{θ_o} to obtain the output prediction (in case of a sequence to sequence task, multiple intermediate values $h(t)$ are used instead).

The decisive difference to Neural ODEs is the fact that the hidden state trajectory is *not* determined entirely by its initial condition. Instead, the continuous function $x : [0, T] \to \mathbb{R}^{d_x}$ is said to control the differential equation and hence the hidden state trajectory. The expression '$\int_0^t f_{\theta_r}(h(s))dx(s)$' is known as a Rieman-Stieltjes integral, with '$f_{\theta_r}(h(s))dx(s)$' being a matrix-vector multiplication [2].

The only piece of the puzzle that is missing is how to obtain the continuous path x that controls the

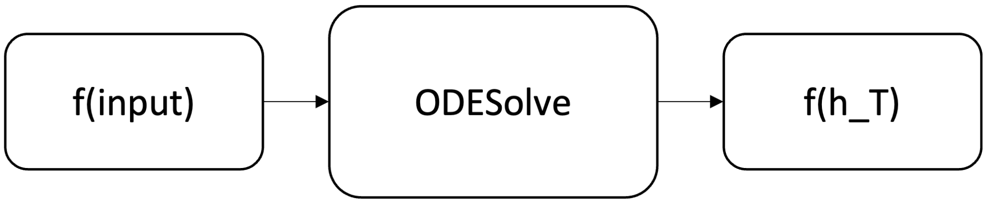


Figure 27. Illustration of the computation graph of a neural CDE.

CDE. Kidger proposes to use any form of interpolation on the discrete input data, and specifically suggests to use cubic hermetic spline interpolation. It must be stressed here, that this interpolation is *not* a pre-processing step on the data to impute missing values and thereby discarding relevant information. Instead, this interpolation is necessary for the hidden state to be continuously driven by the observed data. It is this interpolation scheme that allows the processing of arbitrarily sampled and partially observed data since every input channel will be interpolated to create a continuous driving path for the CDE [2].

To round thing up, it is suggested to concatenate the timestep, as well as an observational mask to the input data and feed this concatenated data to $f_{\theta_{in}}$. The neural network can then in turn learn what information is relevant, which information is now fed into the network at all, and at which rate new data is coming in. This allows the model to represent the notion of informative missingness as well. The former bit of concatenating the timestamp to the observed data is especially important, since CDEs posses a reparameterization invariance [2], which means that the CDE is indifferent to the speed at the controlling path data arrives. It is therefore a necessity to include the timestamps as a channel into the input data.

The similarities between Neural Controlled Differential Equations and Recurrent Neural Networks are striking, and indeed, Neural CDEs can be seen as the continuous time limit of RNNs. Some initial data from the start of the series is expanded into the hidden state of the network by some non-linear transformation. Upon new data coming in, this hidden state is then updated depending on the current value of the hidden state, as well as the new data coming in at that timestep. At the end of the sequence, the final value of the hidden state is traversed through another non-linear transformation to read out some desired information. When discretizing Equation 32 according to Euler discretization of differential equations, we obtain:

$$h(t_i + \Delta t) = h(t_i) + k\ f_{\theta_r}(h(t_i))\ \frac{dx}{dt}(t_i) = h(t_i) + k\ f_\theta(h, x, t_i) \tag{33}$$

which indeed is a discretized update operation that governs an Recurrent Neural Network.

Since Echo State Networks are just a type of Recurrent Neural Networks, why can they not also be seen as a discretization of a Neural CDE? Indeed they can. The function f_{θ_r} may as well be excluded from learning and instead be sampled from a random distribution over non-linear functions that parameterize the vector field of the CDE. The same property as for normal discrete ESNs shall be applicable here as well. First and foremost this concerns the echo state property, which ensures, when obliged by the network, that the hidden state does gradually forget about information that was fed to it a longer time ago. Since any ODE in the end will be solved by a numerical solver that discretizes the vector space and traverses it in discrete steps, the same rules can be applied to the continuous-time counterparts of ESNs.

Let a CDE-ESN be the continuous-time version of an Echo State Network, defined via a Neural Controlled Differential Equation with the following deviations:

- $f_{\theta_{in}} : \mathbb{R}^{d_x} \to R^{d_h}, x^{<t_i>} \to W_{in}\, x^{<t_i>}$
  with $W_{in} \in \mathbb{R}^{d_h \times d_x}$ randomly sampled from a normal distribution, just as the usual input matrix used for ESNs.
- $f_{\theta_r} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h \times d_x}, h \to \begin{bmatrix} W_{rec,1}\ h \\ \vdots \\ W_{rec,d_x}\ h \end{bmatrix}$ where $W_{rec} = \begin{bmatrix} W_{rec,1} \\ \vdots \\ W_{rec,d_x} \end{bmatrix} \in \mathbb{R}^{(d_x\ d_h) \times d_h}$

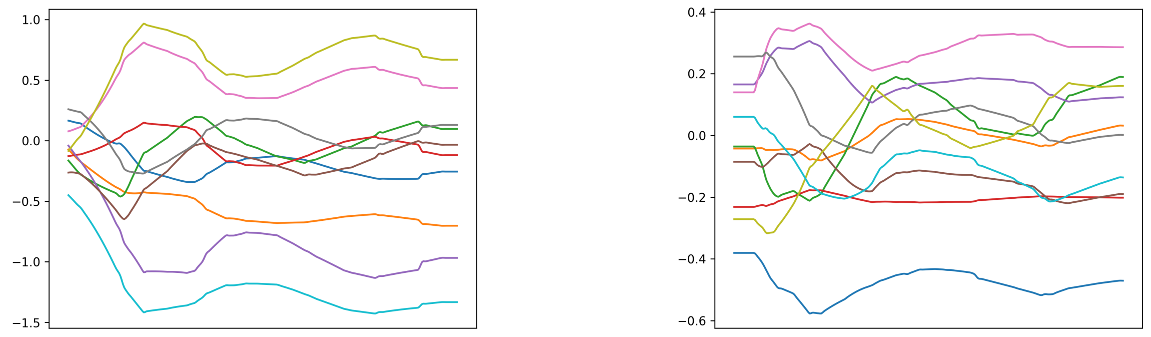
  with each $W_{rec,j}$ sampled from the normal distribution and pre-specified spectral radius. This corresponds to having d_x recurrent weight matrices in the discrete case of standard Echo State Networks.
- The readout function f_{θ_o} is left untouched and can be chosen to either be a simple linear readout or a small feed-forward neural network. The parameters of this function are the only values that are being optimized during training.

Both W_{in} and W_{rec} are randomly generated, and then kept fixed. W_{rec} is the determining factor for the vector field of the CDE and therefore the hidden state trajectory.
This formulation gives a continuous-time Echo State Network with a continuous hidden state in its continuous reservoir. This hidden state can be continuously utilized for readouts.

Notice, that unlike as argued in the preceding section, the hidden state here does not evolve completely at random in between observations, as it is conditioned on the continuous interpolation of the input data. It is therefore continuously influenced by that data and does also continuously encode the observed data in its reservoir.

When compared to standard ESNs, this formulation entails decisive advantages. As mentioned before, a continuous hidden state means that it can be continuously, at any arbitrary time, be read out from. Furthermore, the interpolation scheme required by Neural CDEs allows for natural support of irregularly sampled and partially observed data.

Notice also, that for standard, fully trained Neural CDEs, the computation graph (Fig. 27) has to be fully traversed back during backpropagation in order to create all gradients. This is especially costly to do through the ODESolve method, which consists of many computations corresponding to each step through the vector field. Using the adjoint method, this can be enhanced to a certain degree [2, 1]. However, in the proposed CDE-ESN, only the readout is trained, which means that the costly backpropagation through the ODE solver is omitted. This renders this approach faster then fully trained Neural CDEs on at least an order of magnitude. This of course follows from much the same advantages that ESNs have over RNNs.



(a) Clockwise **(b)** Anticlockwise

Figure 28. Samples from the hidden state trajectories through the latent space (the continuous reservoir) in the CDE-ESN. Sampled from a clockwise (a) and an anti-clockwise spiral (b). These were constructed by an ODE solver

There are, however, also drawbacks in this approach. With regard to fully trained Neural CDEs, CDE-ESNs are a lot less expressive. It requires a huge reservoir to encode all the information needed for complex tasks, whereas standard Neural CDEs can just learn certain dependencies. It is furthermore difficult, to extract separate pieces from information from the reservoir. This is important in the context of informative missingness. As already stated, Neural CDEs are usually used by concatenating the timestamps and observational masks to the input data, and then feeding everything to the network together. However, while fully trained networks can learn to distinguish the different types of information, ESN based approaches treat every input channel in the same way. This leads to the information about observational frequency and timestamps to be lost in the reservoir due to it being "mixed" with the other pieces of information. It is then really difficult for a readout to extract the relevant bits of information again, such that informational missingness can be represented. This is confirmed empirically by the experiments conducted.

To summarize, the proposed CDE-ESN is a novel approach for continuous-time Echo State Networks. In comparison with TA-ESNs [71], CDE-ESNs are additionally capable of handling partial observations naturally, which TA-ESNs are incapable of doing. At the same time however, it is very difficult, if not impossible, for CDE-ESNs to be able to utilize the notion of informative missingness, as observational masks are rather hurtful than beneficial for such networks.

The posed hypotheses will now be empirically evaluated.

Experiment – Synthetic Dataset

First, the proposed method will be evaluated on a synthetic dataset of spirals, as depicted in Figure 25. Each spiral is represented by a two dimensional time series of length 100. Random gaussian noise was added to each individual trajectory. The train and test datasets contain 128 sequences each, half of which are clockwise spirals, the other half anti-clockwise spirals, respectively. A binary classifier was built based on that. Table 6 shows the results. Each method was trained 10 times with newly generated datasets each time.

In addition to the two baseline approaches, different variations of CDE-ESNs have been tested. The following methods have been applied:

- *TA-ESN*: The Time Adaptive Echo State Network [71]. First baseline approach.
- *Neural CDE*: Neural Controlled Differential Equation [2]. Second baseline.
- *CDE-ESN*: The proposed method of this thesis (novel). A logistic regression classifier was trained as the readout.
- *CDE-ESN'*: A CDE-ESN, but with a small MLP trained as the readout.
- *CDE-ESN**: A CDE-ESN, but an additional channel containing the timestamps was concatenated to the input.
- *CDE-ESN***: A CDE-ESN, but both timestamps and observational mask were concatenated to the input as additional channels.

The experiments were then carried out on three differently sampled datasets:

1. *Regularly Sampled*
   The 100 spiral datapoints are evenly spaced in time.
2. *Irregularly Sampled*
   The 100 spiral datapoints were randomly subsampled from a set of 10000 evenly spaced timestamps.
3. *Partially Observed*
   The 100 spiral datapoints were randomly subsampled from a set of 10000 evenly spaced timestamps, and then 50% of the datapoints were removed. This led to some datapoints only having observations in either its x or y dimension.

The results confirm the hypotheses from before. CDE-ESNs are robust and very quick models, requiring only little trainable parameters, as long as the reservoir can do a good job of linearly separating the different classes. The plain CDE-ESN with just 32 parameters to train for its readout performed well across all scenarios. The only model that performed better was the fully trained Neural CDE, as expected. This however comes at a much higher computational cost, as the training time is higher on several order of magnitudes.

Furthermore, it is also evident that CDE-ESNs outperform TA-ESNs when data is only partially observed. The TA-ESN was given the observational mask in addition to the raw input data (where missing values were replaced by a 0), but it still performed only slightly better than a random guesser. Even a significant increase in its hidden units could not improve the performance. It is expected that the hidden state has to be made so large for the performance to improve, that the computational cost would then be even higher as compared to a fully trained network. This tradeoff is therefore most probably a loose-loose.

Lastly, feeding more meta information to an CDE-ESN does not help, but rather hurts its performance. This of course is undesirable in situations, where missing data carries a lot of information on itself, where CDE-ESNs are therefore not recommended to be applied to.

Table 6: Results of applying a TA-ESN on the classification task of the spirals. “10n” in the first column indicates that the samples were subsampled from 10 times the amount of regularly spaced points. The values in the second column denote the fraction of observations that are kept.

	Accuracy	**Training Time [s]**	**#Parameters**
Regularly Sampled			
TA-ESN	1.0 ± 0.000	0.063 ± 0.021	128
CDE-ESN	1.0 ± 0.000	0.001 ± 0.000	32
CDE-ESN*	1.0 ± 0.000	15.396 ± 0.812	1057
NCDE	1.0 ± 0.000	48.654 ± 5.963	3297
Irregularly Sampled			
TA-ESN	0.948 ± 0.074	0.062 ± 0.021	128
CDE-ESN	1.0 ± 0.000	0.001 ± 0.000	32
CDE-ESN’	1.0 ± 0.000	18.637 ± 0.394	1057
NCDE	1.0 ± 0.000	84.197 ± 6.077	4289
Partially Observed			
TA-ESN	0.577 ± 0.056	0.073 ± 0.041	128
TA-ESN	0.571 ± 0.052	0.330 ± 0.055	1024
CDE-ESN	0.980 ± 0.020	0.006 ± 0.000	32
CDE-ESN	0.997 ± 0.005	0.006 ± 0.000	64
CDE-ESN*	0.818 ± 0.053	0.002 ± 0.000	32
CDE-ESN*	0.951 ± 0.054	0.005 ± 0.000	256
CDE-ESN**	0.570 ± 0.047	0.003 ± 0.000	32
CDE-ESN**	0.638 ± 0.072	0.007 ± 0.000	256
CDE-ESN**	0.836 ± 0.027	0.017 ± 0.002	1024
CDE-ESN’	0.912 ± 0.087	11.243 ± 0.184	1057
NCDE	1.0 ± 0.000	148.822 ± 13.444	6364

Experiment – Gesture Dataset

This experiment was conducted on a real world univariate gesture dataset[3] [84]. It consists of sensor data of eight different gestures. Here, 2388 samples were used for training, and 1194 for testing. The same methodology as in [71] was applied by randomly subsampling 10% of the sequences irregularly. The CDE-ESN model was trained using logistic regression and a hidden state size of 4096. The results are compared to other approaches in Table 7.

[3]The UWaveGestureAll dataset is available here: `https://timeseriesclassification.com`

The results show that the model produces state of the art performance on a real world, irregularly sampled large size dataset. It also again shows its computational speed.

Table 7: Results on the UWaveGestureAll dataset. Results as reported in [71].

	Accuracy	**Training Time [s]**	**#Parameters**
LSTM	0.794 ± 0.016	1.445×10^2	41 608
LSTMT	0.882 ± 0.009	1.500×10^2	42 008
GRU-F	0.855	4.840×10^2	50 952
GRU-D	0.860	6.810×10^2	
GRU-HD	0.860	6.900×10^2	
GP-GRU	0.948	8.365×10^4	
IPred-GRU	0.935	1.463×10^3	51 721
TAGRU	0.876 ± 0.012	2.410×10^2	31 408
GRU	0.773 ± 0.009	1.579×10^2	31 408
GRUT	0.861 ± 0.007	1.733×10^2	31 708
ESNT	0.567 ± 0.265	1.101×10^1	4016
TAESN	0.921 ± 0.006	1.277×10^1	4016
CDE-ESN[4]	0.871 ± 0.026	3.610×10^1	4096

Experiment – Physionet

Finally, some experiments have also been conducted on data from the MIMIC database [85], which contains highly irregularly sampled and partially observed medical data from intensive care patients recorded over the period of their ICU stay. Extensive experiments have been conducted on the Physionet Challenge 2012 dataset, where several thousand ICU stays had to be labeled with in-hospital mortality. Now since this task probably requires a model very sensitive to the notion of informative missingness, the proposed CDE-ESN performed poor across all experiment setups. It was still decided to mention this experience, as it reaffirms the hypothesis that CDE-ESNs are incapable of being deployed to tasks with high meta informational values, such as informative missingness.

5. Discussion

The core of this thesis was the investigation of sequence data processing methods, focussing on the universal case of irregularly sampled and partially observed time series data. The implications of this include awareness about the notion of informative missingness, which denotes the fact that the very missingness of some data might on its own carry informational value with it.

Literature on the afore-mentioned issues is widely available, however most of the proposed approaches are highly over-engineered manual architectural adaptations to existing methods. The first contribution of this work was therefore a thorough, yet not comprehensive, analysis of the most popular methods proposed in literature with regard to processing sequential data that might be irregularly sampled, partially observed, have missing data, be of different lengths or include the notion of informative missingness.

For this purpose, a taxonomy was created, which the methods discussed were sorted by. The list started with basic methods, such as imputation, interpolation and other data pre-processing techniques. Such approaches however usually discard a lot of information and are therefore unfavorable. They do, however, allow the adoption of arbitrary models, as the data is pre-processed to be regularly sampled and fully observed. The second type of approaches was deemed to be adaptations to some RNN cell architecture. Either the current timesteps are accepted as additional input and subsequently and adequately processed within the cell, or the hidden state's trajectory in between observations was altered. Those methods took one step closer towards the goal of having a universal model with a continuous hidden state. Afterwards, some ESN based methods were discussed, with none of them having a continuous hidden state across observations. Gaussian estimators were briefly discussed next. The taxonomy closed with an overview over ODE based methods, which became popular only a few years ago and have since taken the field by storm. Especially Neural Controlled Differential Equations were given special consideration, as they have established themselves as the de facto state of the art approach for universal time series processing.

The recently proposed TA-ESN approach was ensuingly analyzed and empirically evaluated. TA-ESNs are great for irregularly sampled, yet fully observed time series. They inherit the common advantages and drawbacks of ESNs, such as their computational efficiency, but also the relative limit in expressivity. On the other hand, Neural CDEs provide a means to process any irregularly sampled and partially observed dataset in a continuous fashion. This powerful state of the art approach has only one drawback, its computational cost required to train and run it. For these reasons, the main contribution of the present thesis is the proposal of CDE-ESNs, a novel method of creating a continuous reservoir based on the ideas of Neural CDEs, with the computational advantages of ESNs, essentially taking the best of both worlds.

CDE-ESNs are computationally cheap, while allowing to handle irregularly sampled and partially observed data in a natural fashion. Their only limitation is the incapability of processing meta data as additional input. Therefore, data whose observational frequency itself carries a lot of relevant information, cannot be processed by utilizing CDE-ESNs to a satisfactory degree. In such cases, fully trained Neural CDEs are recommended instead.

The theoretical properties of CDE-ESNs were subsequently empirically confirmed. CDE-ESNs can be seen as continuous-time Echo State Networks for universal sequence processing, with the exception of meta data processing. However, it may be argued that this is as best as ESNs can do, as meta data processing requires the logical separation of data from meta data, which has to be learnt by a model. Since neither the input weights, nor the recurrent weights of ESNs are learnt, the information from both the actual data and the meta data is dispersed into the reservoir, from which it can only be extracted again with enormous costs. The costs necessary to achieve this are probably higher compared to training a standard Neural CDE on the same task in the first place.

To conclude, CDE-ESNs can be used whenever the processing of a dataset does not require knowledge about meta data. Future work might include examining different ways of creating the non-linear continuous reservoir, as well as applying the approach on a bigger scale for further empirical evaluation.

6. Conclusions

1. Most methods found in literature for universal time series processing are highly over-engineered and/or domain specific.

2. The current deep learning state of the art approach is Neural Controlled Differential Equations [2], which are able to process both irregularly sampled and partially observed sequences. They can also take informative missingness into consideration. However, this method also poses a high computational cost.

3. In this thesis, CDE-ESNs have been proposed as a novel method to create continuous time Echo State Networks. This is done by randomly setting the weights of the function governing the slope field of a Neural CDE, and then only training the readout.

4. There is a hierarchy between different ESN approaches. Vanilla ESNs cannot handle irregular sampling, nor partial observations. TA-ESNs [71] *are* able to process irregularly sampled time series, yet cannot process partial observations. CDE-ESNs (from this thesis) *can* process both irregularly sampled and partially observed time series data.

5. The only shortcoming that CDE-ESNs have is that they cannot represent the notion of informative missingness at reasonable costs. However, this may not even be possible at all in Echo State Networks (see Discussion).

6. Compared to ESNs, CDE-ESNs add the capability of handling partial observations, plus the continuous hidden state. Compared to Neural CDEs, CDE-ESN are less expressive, yet faster by several order of magnitudes. Overall, CDE-ESNs can be applied to problems where meta data knowledge is not required.

List of references

1. CHEN, Ricky T Q; Yulia RUBANOVA; Jesse BETTENCOURT; David K DUVENAUD. Neural ordinary differential equations. In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.
2. KIDGER, Patrick. On Neural Differential Equations. *arXiv preprint arXiv:2202.02435*. 2022. Available from arXiv: `2202.02435`.
3. SCHAKE, Erik; Lisa GRUMBACH; Ralph BERGMANN. A Time-Series Similarity Measure for Case-Based Deviation Management to Support Flexible Workflow Execution. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2020, vol. 12311 LNAI, pp. 33–48. ISBN 9783030583415. ISSN 16113349. Available from DOI: `10.1007/978-3-030-58342-2_3/FIGURES/5`.
4. CHUNG, Junyoung; Caglar GULCEHRE; Kyunghyun CHO. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. [N.d.]. Available from arXiv: `1412.3555v1`.
5. CHO, Kyunghyun et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*. 2014.
6. RUBANOVA, Yulia; Ricky T.Q. CHEN; David DUVENAUD. Latent ODEs for irregularly-sampled time series. *Advances in Neural Information Processing Systems*. 2019, vol. 32. ISSN 10495258. Available from arXiv: `1907.03907`.
7. HORNIK, Kurt; Maxwell STINCHCOMBE; Halbert WHITE. Multilayer feedforward networks are universal approximators. *Neural Networks*. 1989, vol. 2, no. 5, pp. 359–366. ISSN 0893-6080. Available from DOI: `10.1016/0893-6080(89)90020-8`.
8. MINSKY, Marvin; Seymour PAPERT. Perceptrons. 1969.
9. NG, Andrew. *Deep Learning Specialization*. 2021. Available also from: `https://www.coursera.org/specializations/deep-learning`.
10. GOODFELLOW, Ian; Yoshua BENGIO; Aaron COURVILLE. *Deep learning*. MIT press, 2016.
11. ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*. 1958, vol. 65, no. 6, p. 386.
12. SHARMA, Sagar; Simone SHARMA; Anidhya ATHAIYA. Activation functions in neural networks. *towards data science*. 2017, vol. 6, no. 12, pp. 310–316.
13. KIDGER, Patrick; Terry LYONS; Jacob ABERNETHY; Shivani AGARWAL. Universal Approximation with Deep Narrow Networks. *Proceedings of Machine Learning Research*. 2020, vol. 125, pp. 1–22.
14. SCHMIDHUBER, Jürgen. Deep learning in neural networks: An overview. *Neural networks*. 2015, vol. 61, pp. 85–117.
15. WERBOS, Paul J. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*. 1988, vol. 1, no. 4, pp. 339–356.
16. ROBINSON, A J; Frank FALLSIDE. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering Cambridge, 1987.

17. MOZER, Michael C. A focused backpropagation algorithm for temporal. *Backpropagation: Theory, architectures, and applications*. 1995, vol. 137.
18. HOCHREITER, Sepp; Jürgen SCHMIDHUBER. Long short-term memory. *Neural computation*. 1997, vol. 9, no. 8, pp. 1735–1780.
19. SHERSTINSKY, Alex. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*. 2020, vol. 404, no. March, pp. 1–43. ISSN 01672789. Available from DOI: `10.1016/j.physd.2019.132306`.
20. RAVANELLI, Mirco; Philemon BRAKEL; Maurizio OMOLOGO; Yoshua BENGIO. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*. 2018, vol. 2, no. 2, pp. 92–102.
21. SU, Yuanhang; C-C Jay KUO. On extended long short-term memory and dependent bidirectional recurrent neural network. *Neurocomputing*. 2019, vol. 356, pp. 151–161.
22. JAEGER, Herbert. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*. 2001, vol. 148, no. 34, p. 13.
23. MAASS, Wolfgang; Thomas NATSCHLÄGER; Henry MARKRAM. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*. 2002, vol. 14, no. 11, pp. 2531–2560.
24. LUKOŠEVIČIUS, Mantas. A practical guide to applying echo state networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2012, vol. 7700 LECTU, pp. 659–686. ISBN 9783642352881. ISSN 16113349. Available from DOI: `10.1007/978-3-642-35289-8_36`.
25. LUKOŠEVIČIUS, Mantas; Herbert JAEGER. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*. 2009, vol. 3, no. 3, pp. 127–149.
26. LECUN, Yann; Yoshua BENGIO; Geoffrey HINTON. Deep learning. *nature*. 2015, vol. 521, no. 7553, pp. 436–444.
27. HE, Kaiming; Xiangyu ZHANG; Shaoqing REN; Jian SUN. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
28. VASWANI, Ashish et al. Attention is all you need. *Advances in neural information processing systems*. 2017, vol. 30.
29. RUNGE, Carl. Über die numerische Auflösung von Differentialgleichungen. *Mathematische Annalen*. 1895, vol. 46, no. 2, pp. 167–178.
30. KUTTA, Wilhelm. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.* 1901, vol. 46, pp. 435–453.
31. WANNER, Gerhard; Ernst HAIRER. *Solving ordinary differential equations II*. Springer Berlin Heidelberg, 1996.
32. WEINAN E. A Proposal on Machine Learning via Dynamical Systems. *Commun. Math. Stat.* 2017, vol. 5, pp. 1–11. Available from DOI: `10.1007/s40304-017-0103-z`.
33. HABER, Eldad; Lars RUTHOTTO. Stable architectures for deep neural networks. 2017. Available from DOI: `10.1088/1361-6420/aa9a90`.

34. LU, Yiping; Aoxiao ZHONG; Quanzheng LI; Bin DONG. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In: *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*. 2018.
35. RICO-MARTINEZ, Ramiro; Ioannis G. KEVREKIDIS. Continuous time modeling of nonlinear systems: a neural network-based approach. *1993 IEEE International Conference on Neural Networks*. 1993, pp. 1522–1525. ISBN 0780312007. Available from DOI: `10.1109/ICNN.1993.298782`.
36. RICO-MARTINEZ, R.; J. S. ANDERSON; I. G. KEVREKIDIS. Continuous-time nonlinear signal processing: A neural network based approach for gray box identification. *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*. 1994, pp. 596–605. Available from DOI: `10.1109/NNSP.1994.366006`.
37. CHU, S. Reynold; Rahmat SHOURESHI. A neural network approach for identification of continuous-time nonlinear dynamic systems. *Proceedings of the American Control Conference*. 1991, vol. 1, pp. 1–5. ISBN 0879425652. ISSN 07431619. Available from DOI: `10.23919/ACC.1991.4791308`.
38. LECUN, Yann. A theoretical framework for back-propagation. In: *Proceedings of the 1988 connectionist models summer school*. 1988. Available from DOI: `10.2307/j.ctv7cjw41.5`.
39. PINEDA, Fernando J. GENERALIZATION OF BACKPROPAGATION TO RECURRENT AND HIGH-ORDER NETWORKS. In: 1987, p. 15.
40. ALMEIDA, Luis B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In: *Artificial neural networks: concept learning*. 1990, pp. 102–111.
41. PONTRYAGIN, Lev Semenovich. *Mathematical theory of optimal processes*. CRC press, 1987.
42. KIDGER, Patrick; James MORRILL; James FOSTER; Terry LYONS. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*. 2020, vol. 2020-Decem, no. 1. ISSN 10495258. Available from arXiv: `2005.08926`.
43. RYBICKI, George B.; William H. PRESS. Interpolation, realization, and reconstruction of noisy, irregularly sampled data. *The Astrophysical Journal*. 1992, vol. 398, pp. 169–176.
44. BOS, Robert; Stijn DE WAELE; Piet M.T. BROERSEN. Autoregressive spectral estimation by application of the Burg algorithm to irregularly sampled data. *IEEE Transactions on Instrumentation and Measurement*. 2002, vol. 51, no. 6, pp. 1289–1294. ISSN 00189456. Available from DOI: `10.1109/TIM.2002.808031`.
45. SAKOE, Hiroaki; Seibi CHIBA. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*. 1978, vol. 26, no. 1, pp. 43–49.
46. SMITH, Temple F; Michael S WATERMAN, et al. Identification of common molecular subsequences. *Journal of molecular biology*. 1981, vol. 147, no. 1, pp. 195–197.

47. CHE, Zhengping et al. Recurrent Neural Networks for Multivariate Time Series with Missing Values. *Scientific Reports*. 2018, vol. 8, no. 1. ISSN 20452322. Available from DOI: `10.1038/s41598-018-24271-9`.
48. CAO, Wei et al. BRITS: Bidirectional recurrent imputation for time series. In: *Advances in Neural Information Processing Systems*. 2018, vol. 2018-Decem, pp. 6775–6785. ISSN 10495258. Available from arXiv: `1805.10572`.
49. SHUKLA, Satya Narayan; Benjamin M. MARLIN. Interpolation-Prediction Networks for Irregularly Sampled Time Series. 2019. Available from arXiv: `1909.07782`.
50. LUO, Yonghong; Xiangrui CAI; Ying ZHANG; Jun XU, et al. Multivariate time series imputation with generative adversarial networks. In: *Advances in Neural Information Processing Systems*. 2018, pp. 1596–1607.
51. LITTLE, Roderick JA; Donald B RUBIN. *Statistical analysis with missing data*. John Wiley & Sons, 2019.
52. BATISTA, Gustavo Enrique; Maria-Carolina MONARD; Gustavo E A P A BATISTA; Maria Carolina MONARD. A Study of K-Nearest Neighbour as an Imputation Method. On the definition of fuzzy classification systems using the genetic paradigm View project PhD Thesis-Combining Symbolic Classifiers Using Evaluation Metrics of Knowledge Rules and Genetic Algorithms. 2002. Available also from: `https://www.researchgate.net/publication/220981745`.
53. AZUR, Melissa J.; Elizabeth A. STUART; Constantine FRANGAKIS; Philip J. LEAF. Multiple imputation by chained equations: what is it and how does it work? *International Journal of Methods in Psychiatric Research*. 2011, vol. 20, no. 1, pp. 40–49. ISSN 1557-0657. Available from DOI: `10.1002/MPR.329`.
54. STEKHOVEN, Daniel J.; Peter BÜHLMANN. MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*. 2012, vol. 28, no. 1, pp. 112–118. ISSN 1367-4803. Available from DOI: `10.1093/BIOINFORMATICS/BTR597`.
55. JOSSE, Julie; François HUSSON. Handling missing values in exploratory multivariate data analysis methods. *Journal de la Société Française de Statistique*. 2012, vol. 153, no. 2.
56. ÁLVAREZ, Mauricio A; Neil D LAWRENCE. Computationally efficient convolved multiple output Gaussian processes. *The Journal of Machine Learning Research*. 2011, vol. 12, pp. 1459–1500.
57. LI, Steven Cheng Xian; Benjamin MARLIN. A scalable end-to-end Gaussian process adapter for irregularly sampled time series classification. In: *Advances in Neural Information Processing Systems*. 2016, pp. 1812–1820. ISSN 10495258. Available from arXiv: `1606.04443`.
58. FUTOMA, Joseph; Sanjay HARIHARAN; Katherine HELLER. Learning to detect sepsis with a multitask Gaussian process RNN classifier. *arXiv preprint arXiv:1706.04152*. 2017.
59. SOLEIMANI, Hossein; James HENSMAN; Suchi SARIA. Scalable Joint Models for Reliable Uncertainty-Aware Event Prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2018, vol. 40, no. 8, pp. 1948–1963. ISSN 01628828. Available from DOI: `10.1109/TPAMI.2017.2742504`.
60. CHOI, Edward et al. Doctor ai: Predicting clinical events via recurrent neural networks. In: *Machine Learning for Healthcare Conference*. 2016, pp. 301–318.

61. LIPTON, Zachary C; David C KALE; Randall WETZEL, et al. Modeling missing data in clinical time series with rnns. *Machine Learning for Healthcare*. 2016, vol. 56.
62. LI, Yang; Samy BENGIO; Nan DU. Time-Dependent Representation for Neural Event Sequence Prediction. *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*. 2017. Available from DOI: 10.48550/arxiv.1708.00065.
63. DU, Nan et al. Recurrent marked temporal point processes: Embedding event history to vector. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 1555–1564.
64. SOUSA, Rafael T.; Lucas A. PEREIRA; Anderson S. SOARES. Improving Irregularly Sampled Time Series Learning with Dense Descriptors of Time. 2020, pp. 1–7. Available from arXiv: 2003.09291.
65. NEIL, Daniel; Michael PFEIFFER; Shih-Chii LIU. Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. 2016. Available from arXiv: 1610.09513.
66. MOZER, Michael C.; Denis KAZAKOV; Robert V. LINDSEY. Discrete Event, Continuous Time RNNs. 2017. Available from arXiv: 1710.04110.
67. ZHU, Yu et al. *What to Do Next: Modeling User Behaviors by Time-LSTM*. 2017. Tech. rep.
68. BAYTAS, Inci M et al. Patient subtyping via time-aware lstm networks. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 65–74.
69. BROUWER, Edward de; Jaak SIMM; Adam ARANY; Yves MOREAU. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. *Advances in Neural Information Processing Systems*. 2019, vol. 32, no. NeurIPS. ISSN 10495258. Available from arXiv: 1905.12374.
70. HABIBA, Mansura; Barak A. PEARLMUTTER. Neural ODEs for Informative Missingess in Multivariate Time Series. *2020 31st Irish Signals and Systems Conference, ISSC 2020*. 2020. ISBN 9781728194189. Available from DOI: 10.1109/ISSC49989.2020.9180216.
71. LUKOŠEVIČIUS, Mantas; Arnas USELIS. Time-Adaptive Recurrent Neural Networks. *arXiv preprint arXiv:2204.05192*. 2022.
72. JAEGER, Herbert; Mantas LUKOŠEVIČIUS; Dan POPOVICI; Udo SIEWERT. Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks*. 2007, vol. 20, no. 3, pp. 335–352. ISSN 08936080. Available from DOI: 10.1016/J.NEUNET.2007.04.016.
73. LUKOŠEVICIUS, Mantas et al. Time warping invariant echo state networks. *International University Bremen, Technical Report*. 2006.
74. ANANTHARAMAN, Ranjan et al. Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks. *CEUR Workshop Proceedings*. 2020, vol. 2964. ISSN 16130073. Available from DOI: 10.48550/arxiv.2010.04004.
75. ANANTHARAMAN, Ranjan et al. Stably Accelerating Stiff Quantitative Systems Pharmacology Models: Continuous-Time Echo State Networks as Implicit Machine Learning. *bioRxiv*. 2021, p. 2021.10.10.463808. Available from DOI: 10.1101/2021.10.10.463808.

76. MORRILL, James et al. Neural Rough Differential Equations for Long Time Series. 2020. Available from arXiv: 2009.08295.
77. TZEN, Belinda; Maxim RAGINSKY. Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit. 2019, pp. 1–18. Available from arXiv: 1905.09883.
78. LI, Zhongliang; Zhixue ZHENG; Rachid OUTBIB. Adaptive prognostic of fuel cells by implementing ensemble echo state networks in time-varying model space. *IEEE Transactions on Industrial Electronics*. 2020, vol. 67, no. 1, pp. 379–389. ISSN 15579948. Available from DOI: 10.1109/TIE.2019.2893827.
79. HODGKINSON, Liam; Chris VAN DER HEIDE; Fred ROOSTA; Michael W MAHONEY. Stochastic Normalizing Flows. 2020. Available from arXiv: 2002.09547v2.
80. KIDGER, Patrick et al. Neural SDEs as Infinite-Dimensional GANs. 2021.
81. JHIN, Sheo Yon et al. Attentive Neural Controlled Differential Equations for Time-series Classification and Forecasting. *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2021, vol. 2021-Decem, pp. 250–259. ISBN 9781665423984. ISSN 15504786. Available from DOI: 10.1109/ICDM51629.2021.00035.
82. BILOŠ, Marin et al. Neural Flows: Efficient Alternative to Neural ODEs. 2021, no. NeurIPS, pp. 1–13. Available from arXiv: 2110.13040.
83. JHIN, Sheo Yon et al. EXIT: Extrapolation and Interpolation-based Neural Controlled Differential Equations for Time-series Classification and Forecasting. 2022, p. 11. ISBN 9781450390965. Available from DOI: 10.1145/3485447.3512030.
84. LIU, Jiayang; Lin ZHONG; Jehan WICKRAMASURIYA; Venu VASUDEVAN. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*. 2009, vol. 5, no. 6, pp. 657–675.
85. JOHNSON, Alistair E.W. et al. MIMIC-III, a freely accessible critical care database. *Scientific Data*. 2016, vol. 3. ISSN 20524463. Available from DOI: 10.1038/sdata.2016.35.