Delayed deep learning for continuous-time dynamical systems

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Delayed deep learning for continuous-time dynamical systems

Master Thesis

Andreas Schlaginhaufen

January 29, 2021

Advisors: Prof. Dr. F. Dörfler, Prof. Dr. A. Krause, Philippe Wenk
Department of Computer Science, ETH Zürich

Abstract

Bridging the gap between deep learning and dynamical systems, neural ODEs are a promising approach to model continuous-time dynamical systems. Motivated by state augmentation in discrete-time models, we propose to extend the neural ODE framework to neural delay differential equations in order to naturally capture non-Markovian effects such as time delays or hysteresis, which are often encountered in real world applications. We demonstrate the superior performance of neural delay differential equations on the task of modelling a partially observed oscillator in comparison with augmented neural ODEs. Moreover, we showcase robustness to observation noise, generalization over time and initial conditions, and the expressive power on more complex dynamical systems. Furthermore, a result on universal approximation is provided and the connection to delay embeddings is discussed. In an exploratory part, we discuss deep learning approaches for stability analysis of time delay systems and propose to jointly learn a dynamics model and a Lyapunov-Razumikhin function via discretization of the Razumikhin condition. The applicability of this approach is demonstrated for the task of stabilizing an inverted pendulum with delayed feedback control.

Acknowledgements

First and foremost, I would like to thank Prof. Andreas Krause from the Learning & Adaptive Systems Group and Prof. Florian Dörfler from the Automatic Control Laboratory for co-supervising me during this project and for the many valuable inputs.

I would also like to express my special thanks to Philippe Wenk who was my supervisor at the Learning & Adaptive Systems Group. He was incredibly motivated and his ideas were a major contribution to this project. Furthermore, I really enjoyed the many long and insightful discussions during this time.

Moreover, I would like to thank the people from the Max Planck Institute for Intelligent Systems in Tübingen who supported me with literature references on control of time delayed systems and many other areas.

I would also like to thank Dr. Christopher V. Rackauckas from the Massachusetts Institute of Technology for his incredible support regarding questions about the implementation.

Last but not least, I would like to seize the opportunity to thank my friends and family for proof reading and supporting me during this time.

Contents

Chapter 1

Introduction

1.1 Motivation

Accurately predicting the behaviour of dynamical systems is a key challenge in robotics and many other areas. Besides classical physical modelling, the dynamics of such systems can also be learned directly from experimental data employing machine learning models. A promising continuous-time approach of the latter category are deep Neural Ordinary Differential Equations (NODE), where the underlying differential equations are modelled explicitly using a neural network. NODEs have been receiving vast attention in the machine learning community, as they are offering a continuous perspective on deep learning. However, the Markovian nature of Ordinary Differential Equations (ODE) becomes problematic in the presence of non-Markovian effects, such as time delays or hysteresis, as they are often encountered in partially observed systems in robotics. Whereas for discrete-time systems this problem can be tackled by state augmentation with past observations, it is not clear how to cope with it in continuous-time. The main goal of the first part of this thesis is to close this gap and to adapt the neural ODE framework in a way that we can naturally model the non-Markovian effects occurring in time delay systems.

In the second part, we employ machine learning methods to address stability of the resulting model and time delay systems in general. A wide-spread time-domain approach for stability analysis of non-linear ODEs relies on so-called Lyapunov functions. Since finding a Lyapunov function for a non-linear ODE turns out to be inherently difficult, neural network based approaches have become increasingly popular. The main idea here is to learn a Lyapunov function parametrized by a neural network. However, whereas a Lyapunov theory for time delayed systems exists, there are to the best of our knowledge presently no deep learning approaches in this area. We therefore attempt to extend those machine learning methods to the time delayed case.

1.2 Prior work

Neural ODEs are first proposed in [1]. Important extensions are augmented neural ODEs [2], where a neural ODE is learned in a higher dimensional state or regularization methods as described in [3]. Furthermore, the authors of [4] show that augmented neural ODEs are, similarly to discrete neural networks, universal function approximators and a true deep limit with continuous parameters is discussed in [5]. Moreover, different sensitivity methods are discussed in [6],[7]. For implementations an efficient Julia library with a full suit of differentiable ODE solvers is provided by [8] and in combination with a deep learning framework by [9].

Neural network based stability analysis is an active area of research and an increasing number of related publications exists. In [10] a neural network Lyapunov function is used to calculate a stable region of attraction for a fixed closed-loop system. A framework to jointly train feedback and Lyapunov function is proposed in [11]. Moreover, [12] suggests a method to jointly learn a dynamics model and a corresponding Lyapunov function which ensures the resulting dynamics to be stable.

1.3 Contribution

As a natural continuous-time analog of state augmentation with past observations in discrete-time systems we propose Neural Delay Differential Equations (NDDE). Here, we model the dynamics using delay differential equations rather than ordinary differential equations. On the one hand, this gives rise to a new continuous deep learning architecture, that combines ResNet- and DenseNet-like skip connections in an infinite-depth limit. On the other hand, it enables us to naturally model time delay systems.

In the first part of Chapter 4 we develop a complete training procedure including loss gradient calculations, mini-batching, and treatment of observation noise. For the loss gradients we provide an adjoint sensitivity method for continuous parameters and compare it to Automatic Differentiation (AD) through the DDE solver. Then, in Section 4.4 we provide theoretical results on the expressivity of NDDEs. In particular, we proof an analog of the universal approximation property of augmented neural ODEs for NDDEs and consider approximation of periodic solutions, as well as delay embeddings. In Chapter 5 we continue with the experimental evaluation of NDDEs. First of all, we showcase the vast performance of NDDEs in comparison to augmented neural ODEs on a partially observed oscillator. Furthermore, we provide experiments with observation noise, consider generalization over time and initial conditions, and showcase the expressive performance of NDDEs for more complex data. In Chapter 6 we explore the applicability of neural network based Lyapunov methods for time delay system. In particular, we extend the method described

in [12] to the time delayed case and propose another framework enforcing stability through a soft loss. In a last step, we showcase the applicability of the developed framework to the real world problem of stabilizing an inverted pendulum with delayed feedback control.

Chapter 2

Delay differential equations

The goal of this chapter is to introduce Delay Differential Equations (DDE) and to highlight some of their most important properties. First of all, DDEs are introduced as a special case of more general Functional Differential Equations (FDE). Numerical methods for general non-linear DDEs with constant delays are then discussed and finally an overview to the basic theory for the linear case is given.

2.1 Basics

Suppose $r \geq 0$ and consider the infinite-dimensional state space $\mathcal{C}([-r,0]) := \mathcal{C}([-r,0],\mathbb{R}^n)$ of continuous mappings from the interval $[-r,0]$ to $\mathbb{R}^n$. We equip $\mathbb{R}^n$ with the euclidean norm $\|\cdot\|_2$ and $\mathcal{C}([-r,0])$ with the supremum norm $\|\phi\|_\infty = \sup_{s\in[-r,0]} \|\phi(s)\|_2$. Further on, for $x \in \mathcal{C}([-r,T],\mathbb{R}^n)$ we make use of the notation $x_t(\cdot) := x(t+\cdot) \in \mathcal{C}([-r,0])$ for the infinite-dimensional state at time $t \in [0,T]$.

Given a function $f : \mathbb{R} \times \mathcal{C}([-r,0]) \to \mathbb{R}^n$ a retarded FDE is a relation,

$$\dot{x}(t) = f(t, x_t). \tag{2.1}$$

A very important sub-class of FDEs are retarded delay differential equations,

$$\dot{x}(t) = f(t, x(t), x(t-\tau_1), ..., x(t-\tau_K)), \tag{2.2}$$

where $\dot{x}(t)$ is only depending on discrete observations along $x_t \in \mathcal{C}([-r,0])$. For brevity we will sometimes make use of the notation,

$$\mathbf{x}_\tau(t) := (x(t), x(t-\tau_1), ..., x(t-\tau_K)),$$

or simply consider the retarded DDEs in (2.2) as special case of the retarded FDEs in (2.1).

2.2 Properties of solutions

In the following we consider the initial value problem

$$\begin{cases} \dot{x}(t) &= f(t, x_t) \\ x_{t_0} &= \psi. \end{cases} \tag{2.3}$$

Here, $\psi \in \mathcal{C}([-r, 0])$ is the initial condition and we usually refer to it as the initial history function. Similarly to the ODE case, existence and uniqueness of solutions to (2.3) follow if f is continuous in its first and locally Lipschitz continuous in its second argument [13]. We denote the solution at time t which is going through $(t_0, \psi) \in \mathbb{R} \times \mathcal{C}([-r, 0])$ by $x(t_0, \psi)(t)$ or $x_t(t_0, \psi)$ for the infinite-dimensional state.

For autonomous FDEs $\dot{x}(t) = f(x_t)$ the solution maps $\{\Phi(\cdot, t)\}_{t \geq 0}$ defined by,

$$\begin{aligned} \Phi(\cdot, t) : \mathcal{C}([-r, 0]) &\to \mathcal{C}([-r, 0]) \\ \psi &\mapsto \Phi(\psi, t) = x_t(\psi), \end{aligned}$$

build an additive semi-group. However, for FDEs and more specifically also for DDEs, $\Phi(\cdot, t)$ does not need to be invertible. This is in contrast to ODEs where the solution maps are homeomorphisms and therefore build an additive group. In practice this means that two solutions starting at different initial conditions may coincide after a finite amount of time and that backward continuation of solutions is not always well-defined [13].

2.3 Numerical methods

Most numerical DDE solvers are based on the method of steps. Assume $\tau_1 < \tau_2 < ... < \tau_K = r$ and we are given an initial history $x(t_0 + s) = \psi(s)$ for $s \in [-r, 0]$ and we would like to calculate the solution $x(t)$ on $[t_0, t_0 + \tau_1]$. Then for the segment $t \in [t_0, t_0 + \tau_1]$ the DDE reduces to the initial value problem,

$$\dot{x}(t) = f(t, x(t), \psi(t - t_0 - \tau_1), ..., \psi(t - t_0 - \tau_K)), \quad x(t_0) = \psi(0),$$

which can be integrated with an arbitrary numerical ODE solver. This step can be repeated on the interval $[t_0+\tau_1, t_0+2\tau_1]$ and so on. However, in contrast to ODEs we always need an interpolation of the current history $x_t \in \mathcal{C}([-r, 0])$. For more details on the implementation we refer to [14].

A difficulty of DDEs is that a $\mathcal{C}^n$ discontinuity of the solution at time t propagates along the trajectory and leads to $\mathcal{C}^{n+1}$ discontinuities at times $t+\tau_1, t+\tau_2, ..., t+\tau_K$. Such discontinuities usually occur at the transition between initial history and solution or can be introduced through a non-smooth DDE function. Since numerical ODE solvers usually require smoothness up to their order it is important to use smooth enough DDE functions or to keep track of any discontinuities that are introduced.

2.4 Linear DDEs with constant delays

Similar to ODEs a rich theory for linear delay differential equations exists. To start we consider the one-dimensional linear DDE,

$$\dot{x}(t) = -ax(t-\tau), \text{ with } a > 0. \tag{2.4}$$

We can easily check that for $a = \omega$ and $\tau = \pi/(2\omega)$,

$$x(t) = A\sin(\omega t - \varphi),$$

is a solution of (2.4). This oscillating behaviour is in stark contrast to one-dimensional ODEs, that can only have monotone solutions. Moreover, one can show that equation (2.4) has infinitely many solutions and that it is stable for $\tau < \pi/(2a)$ and unstable for $\tau > \pi/(2a)$ [15].

In the following, we provide a short overview on the theory for more general linear DDEs of the form,

$$\dot{x}(t) = A_0 x(t) + \sum_{k=1}^{K} A_k x(t-\tau_k) + f(t), \quad A_i \in \mathbb{R}^{n\times n}. \tag{2.5}$$

In close analogy to ODEs the roots of the characteristic polynomial,

$$\chi(s) := \det \Delta(s) = 0,\ s \in \mathbb{C},$$
$$\text{where } \Delta(s) := sI - A_0 - \sum_{k=1}^{K} A_k e^{-s\tau_k},$$

play a central role in solutions of (2.5). In particular, each root s_0 of $\chi(s)$ constitutes a solution of the homogeneous equation by $bt^m e^{s_0 t}$, where b is an eigenvector of $\Delta(s_0)$ and m an integer smaller than the multiplicity of the root s_0.

More generally defining the fundamental solution $X(t)$ as the solution of (2.5) with initial history,

$$h(t) = \begin{cases} \mathbf{1} & , t = 0 \\ 0 & , t < 0, \end{cases}$$

it can be shown that,

$$X(t) = \mathcal{L}^{-1}\left[\Delta(s)^{-1}\right],$$

where $\mathcal{L}^{-1}$ is the inverse Laplace transform [15]. Furthermore, for a general initial history $\phi(t)$ and some inhomogeneity $f(t)$ the solution is,

$$x(t) = X(t)\phi(0) + \sum_{k=1}^{K} \int_{-\tau_k}^{0} X(t-s-\tau_k) A_k \phi(s) ds + \int_0^t X(t-s) f(s) ds.$$

This looks very familiar to the formula for linear ODEs with constant coefficients. However, we only know that $X(t)$ is the inverse Laplace transform of $\Delta(s)^{-1}$. Using the residual value theorem it can be shown that,

$$X(t) = \sum_{j=1}^{l} p_j(t)e^{s_j t} + Y_\gamma(t), \quad Y_\gamma(t) \in o(e^{\gamma t}) \text{ as } t \to \infty,$$

where $s_1, ..., s_l$ are roots of $\chi(s)$ with $\mathrm{Re}(s_j) > \gamma$ and $p_j(t)$ are polynomials of degree smaller than the multiplicity of s_j [16]. Thus, similar as for ODEs, exponential stability to zero follows if all roots of the characteristic polynomial have negative real part. However, finding all of the infinitely many roots of the quasi-polynomial $\chi(s)$ turns out to be a difficult task and closed form expressions are typically not available for them. Nevertheless, it can be shown that there are always only finitely many roots in every right half-plane $\{s \,|\, \mathrm{Re(s)} > \gamma\}$ and that there is $\gamma < \infty$ such that all roots have real part smaller than γ. Furthermore, for equation (2.5) the characteristic roots decompose into root chains of retarded type [17], which means that $\mathrm{Re}(s) \to -\infty$ as $|s| \to \infty$. Along these lines a great deal of frequency domain stability criteria based on analysis of $\chi(s)$ exists (for an overview see [18]). However, since we will be mainly concerned about non-linear systems we will focus on Lyapunov-like time-domain criteria as discussed in Chapter 6.

Under certain conditions for the matrices A_k it can even be shown that the set of solutions of the form $\mathrm{span}\{t \mapsto p_j(t)e^{s_j t}\}_{j=1}^{\infty}$, where s_j are the roots of the characteristic polynomial, is dense in $\mathcal{C}([-r, 0])$[19]. This is in close analogy to partial differential equations where initial and boundary conditions can often be expanded in solutions of the equation. However, as noted above the difficulty in the analysis of DDEs is that we usually do not even have the characteristic roots s_j available in closed form.

Chapter 3

Neural differential equations

The aim of this chapter is to provide an overview on the basic ideas of Neural Ordinary Differential Equations (NODE).

3.1 Vanilla neural ODEs

3.1.1 Model architecture

Consider the state space $\mathcal{X} = \mathbb{R}^n$, parameters $\theta \in \mathbb{R}^p$ and a neural network $f(\cdot, \theta) : \mathbb{R}^n \to \mathbb{R}^n$. The key architectural idea of NODEs [1] is to learn the dynamics of an ordinary differential equation by means of a neural network,

$$\dot{x}(t) = f(x(t), \theta), \quad x(t) \in \mathcal{X}. \tag{3.1}$$

A main reason why NODEs have received tremendous attention recently is that they give rise to a continuous perspective on machine learning. In particular, the Euler discretization of (3.1) with unit stepsize,

$$x_{t+1} = x_t + f(x_t, \theta), \tag{3.2}$$

is exactly the transformation rule of the hidden state of a deep residual neural network (ResNets). So we can see NODEs as the continuous limit of a ResNet, although, as pointed out in [5], a true continuous limit involves time dependent parameters $\theta(t)$ for the transformation of the hidden state. While time dependent parameters lead to a non-linear optimal control problem, we will, at least for now, focus on constant parameters as we are interested in time-invariant dynamical systems.

To put it short, a neural ODE is a machine learning model that takes an initial state x_0 as input and integrates along the flow induced by the neural network $f(\cdot, \theta)$ to produce the output,

$$x(T) = \int_0^T f(x(t), \theta)dt + x_0 =: \Phi_T(x_0). \tag{3.3}$$

In order to compute (3.3) in practice we still need to discretize, but we would typically like to make use of higher order numerical solvers such as Runge-Kutta methods. Besides numerical integration, the key technical difficulty of NODEs is to calculate the gradients with respect to the parameters θ that are needed to perform gradient based optimization of a particular loss function. This will be quickly discussed in the next section and in more detail for the more general case of DDEs in Section 4.2.1.

3.1.2 Training

There is a wide range of possible applications for NODEs and accordingly many different training schemes for them. Similarly to deep neural networks, we can view NODEs as a black box model to perform supervised regression or classification tasks. For example for the use in image classification we may use a convolutional neural network for $f(\cdot, \theta)$, set $x = x_0$ and $y = x(T)$, and minimize a cross-entropy classification loss. In this case, we are usually only interested in the input-output-map of the model and can set $T = 1$. Another promising application for NODEs are continuous normalizing flows, where a continuous transformation of a probability distribution is learned.

For continuous-time dynamical systems modelling the setting is different. Given a data set of, possibly noisy, observations from a continuous dynamical system,

$$\{(t_0, x_0^{\text{data}}), ..., (t_N, x_N^{\text{data}})\}$$

we are interested in minimizing a mean-square loss. In the following we will always set $0 = t_0 < t_1 < ... < t_N = T$. The problem may be stated as the following optimization problem,

$$\begin{aligned} J(\theta) &= L(x(T)) + \int_0^T \ell(x(t))dt \\ \text{s.t.} \quad \dot{x}(t) &= f(x(t), \theta), \quad x(0) = x_0. \end{aligned}$$

Then the mean-square loss is recovered by,

$$\begin{aligned} J &= \sum_{k=1}^{N} \left\| x(t_k) - x_k^{\text{data}} \right\|_2^2 \\ &= \underbrace{\left\| x(T) - x_N^{\text{data}} \right\|_2^2}_{L(x(T))} + \int_0^T \underbrace{\sum_{k=1}^{N-1} \left\| x(t_k) - x_k^{\text{data}} \right\|_2^2 \delta(t - t_k)}_{\ell(x(t))} \, dt. \end{aligned} \tag{3.4}$$

Note, that we intentionally omit the loss for (t_0, x_0) since it usually vanishes and has always a zero gradient. An advantage of the introduction of a continuous loss functional is that it enables us to add continuous regularizers to $\ell(x(t))$.

For the calculation of the sensitivities $\frac{dJ}{d\theta}$ we can distinguish two fundamentally different methologies. On the one hand, *discretize-then-optimize* approaches first discretize the forward pass and then employ standard forward or backward-mode AD. In the case of neural ODEs this means that we directly apply AD to the numerical ODE solvers. Note that for Euler discretization this is equivalent to the training of a ResNet and for higher order numerical integrators this leads to alternative neural network architectures.

On the other hand, the *optimize-then-discretize* methods are first making use of continuous sensitivity methods and then integrate the forward or backward sensitivity equations by numerical solvers. It is important to note that these methods are not equivalent and each is having its own advantages and disadvantages as described in [6]. Since we are making use of differentiable ODE solvers, the discretize-then-optimize methods are usually working out of the box and are not fundamentally different from forward and backward mode AD of discrete neural networks. However, it will be instructive to remember the backprop formula for residual neural networks,

$$\begin{aligned} J &= L(x_T), \quad x_{k+1} = x_k + f(x_k, \theta) \\ \frac{dJ}{d\theta} &= \sum_{l=T}^{0} \underbrace{\frac{\partial L}{\partial x_T}\frac{dx_T}{dx_l}}_{\lambda_l^\top} \underbrace{\frac{\partial x_l}{\partial \theta}}_{f_\theta}, \quad \lambda_{t+1}^\top - \lambda_t^\top = -\lambda_{t+1}^\top \frac{\partial f(x_t, \theta)}{\partial x_t}. \end{aligned} \tag{3.5}$$

Here, we defined $\lambda_t = \left(\frac{dL}{dx_t}\right)^\top$, which will correspond to the adjoint state in the continuous case. The adjoint sensitivity method, as it is proposed in [1], is a backward optimize-then-discretize method and can be summarized as follows: If $\lambda(t)$ is a solution of the so-called adjoint final value problem,

$$\dot{\lambda}(t)^\top = -\ell'(x(t)) - \lambda(t)^\top \frac{\partial f(x(t), \theta)}{\partial x(t)}, \quad \lambda(T) = L'(x(T)), \tag{3.6}$$

then we have,

$$\frac{dJ}{d\theta} = \int_0^T \lambda(t)^\top \frac{\partial f(x(t), \theta)}{\partial \theta}\, dt. \tag{3.7}$$

When we omit ℓ this looks like a continuous limit of the backprop formula in (3.5). Furthermore, note that for the least square loss (3.4) we get,

$$\ell'(x(t)) = \sum_{k=1}^{N-1} 2\left(x(t_k) - x_k^{\text{data}}\right)\delta(t - t_k), \tag{3.8}$$

which is to be interpreted as jump discontinuities of the adjoint state $\lambda(t)$ at the observation times $\{t_k\}_{k=1}^{N-1}$. This is consistent with the fact that one can show that $\lambda(t) = \frac{dJ}{dx(t)}$ [20].

In practice, we may integrate (3.6) and (3.7) together with the ODE as the augmented final value problem,

$$\begin{cases} \dot{z}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{\lambda}(t) \\ \dot{b}(t) \end{pmatrix} = \begin{pmatrix} f(x(t), \theta) \\ -\ell'(x(t)) - \lambda(t)^\top \dfrac{\partial f(x(t), \theta)}{\partial x(t)} \\ \lambda(t)^\top \dfrac{\partial f(x(t), \theta)}{\partial \theta} \end{pmatrix} \\ z(T) = \begin{pmatrix} x(T) \\ L'(x(T)) \\ 0 \end{pmatrix}. \end{cases} \tag{3.9}$$

Here, we used $b(t) = \int_t^T \lambda(t)^\top \frac{\partial f(x(t),\theta)}{\partial \theta} dt$ and it follows $b(0) = dJ/d\theta$.

Similar to backpropagation in deep neural networks, the gradient formula in (3.9) is very memory efficient, since we do not need to store intermediate values from the forward pass. However, as it is discussed in [6] it can often lead to very imprecise gradient estimates due to numerically unstable backward integration of the ODE. Possible resolutions are to use an interpolation of $x(t)$ from the forward pass as proposed in [7] or to generate an interpolation of $x(t)$ and $\lambda(t)$ and integrate (3.7) by Gauss-Kronrod quadrature as proposed in [9]. Besides the adjoint sensitivity method we may also apply forward sensitivity methods. However, due to the large number of parameters we are confronted with, this is typically less efficient and is therefore not considered in the thesis.

3.2 Augmented neural ODEs

In contrast to deep residual neural networks, neural ODEs are not universal function approximators. Intuitively, the problem is that two solutions of an ODE can never cross in the state space. The authors of [2] therefore suggest to embed the problem into a larger state space $\mathcal{Z} \in \mathbb{R}^{n+m}$ and to train a NODE in $\mathcal{Z}$. Here we define $z(t) = (x(t), z_{\text{aug}}(t))^\top \in \mathcal{Z}$ and for the output we project back onto $\mathbb{R}^n$ by $y = z(T)_{1:n} = x(T)$. The resulting model is called Augmented Neural ODEs (ANODE).

Besides empirical advantages such as faster training and better generalization [2], the authors of [4] show that state augmentation brings back universal function approximation, which is summarized in the following theorem.

Theorem 3.1 ([4]) *For any homeomorphism $H : \mathcal{X} \to \mathcal{X}, \mathcal{X} \subset \mathbb{R}^n$ there exists a NODE in $\mathbb{R}^{2n}$ with input-output-map $\Phi_T : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$ and $T = 1$ such that,*

$$\Phi_T(\begin{pmatrix} x \\ 0 \end{pmatrix}) = \begin{pmatrix} H(x) \\ 0 \end{pmatrix}.$$

Besides the above advantages, ANODEs are also a natural extension to NODEs for modelling of partially observed dynamical systems. However, since it is only concerned about the input-output-map, Theorem 3.1 is not strong enough in this case and the initialization of the augmented state variables $z_{\text{aug},0}$ turns out to be crucial. Experimental results for this setting will be discussed in Section 5.1. Furthermore, an analog of Theorem 3.1 for neural delay differential equations will be provided in the next chapter.

Chapter 4

Neural delay differential equations

In this chapter we are introducing the main model of this project - Neural Delay Differential Equations (NDDEs). At first, we start with the model architecture and its motivation. Then we continue with the adjoint method and the training procedure. After that, we provide theoretical results about the expressivity of NDDEs and finally discuss details of the implementation.

4.1 Model architecture

The motivation of the model architecture is the following: We would like to develop a general method to learn continuous-time dynamical systems in the presence of non-Markovian effects without assuming in-depth knowledge of the system. In neural network based system identification for discrete-time dynamical systems, a standard approach is to augment the state space with past observations in order to lift the problem back into a Markovian setting [21]. A continuous time analog would lead us to a functional differential equation,

$$\dot{x}(t) = f(x_t), \quad \text{where } x_t = x(t+\cdot) \in \mathcal{C}([-r, 0]).$$

However, since obviously our neural network cannot represent the general non-linear functional f, we decide to use discrete samples along the infinite-dimensional history state x_t which brings us to the NDDE model,

$$\dot{x}(t) = f(x(t), x(t-\tau_1), ..., x(t-\tau_K), \theta). \tag{4.1}$$

This means that we are no longer learning an ordinary differential equation, but a retarded-type delay differential equation with constant delays. In this project we will always use commensurate delays $\tau_k = k \cdot \tau$ with $K\tau = r$.

In similar fashion as we can see the difference equation (3.2) as the Euler-discretization of a NODE, the Euler-discretization of (4.1) with stepsize τ and

commensurate delays $\tau_k = k \cdot \tau$ brings us back to the augmented difference equation,

$$x(t+\tau) = x(t) + \tau f(x(t), x(t-\tau), ..., x(t-K\cdot\tau), \theta).$$

Or from a continuous machine learning viewpoint we may see NDDEs as a continuous version of a neural network with combined Res- and DenseNet connections.

However, as opposed to the discrete-time case NDDEs come with some major complications: Although we are only using discrete lags, the NDDE belongs to the class of functional differential equations with an infinite-dimensional state space, as we need to specify our state $x_t(s) = x(t+s)$ on the whole interval $[-r, 0]$ to have well-defined dynamics. Since we assume the initial history to be continuous our state space is $\mathcal{X} = \mathcal{C}([-r, 0], \mathbb{R}^n)$ where $x(t) \in \mathbb{R}^n$. We will usually refer to $x(t)$ as the solution at time t and call x_t the current history or infinite-dimensional state. The practical difficulties are that we need a DDE solver that holds an interpolation of the current history and that discontinuities are propagated along the trajectory. Moreover, we cannot integrate backward in time, since backward continuation is in general not well-defined nor easily calculable. Further on, we should keep in mind that the NDDE has infinitely many solutions.

4.2 Loss gradient calculation

For the loss gradients we will first provide an adjoint method for continuous parameters in Section 4.2.1 and then explain how we may instead make use of AD through the DDE solver in Section 4.2.2.

4.2.1 Adjoint sensitivity method and optimal control

For the derivation of the adjoint method we extend the ideas of [5] to the DDE case and derive the gradient formulas for time-dependent parameters $\theta(t) \in \mathbb{R}^p$. Although we are only working with constant parameters, we find this interesting in its own right, since it gives us the equations for the true continuous limit of time dependent parameters as well as for gradient-based optimal control of a fixed NDDE model.

We will start with the optimal control-like problem of minimizing the loss functional,

$$\begin{aligned} &\tilde{J}(\theta(\cdot)) = L(x(T)) + \int_0^T \ell(t, x(t), x(t-\tau_1), ..., x(t-\tau_K), \theta(t))dt \\ &\text{s.t. } \dot{x}(t) = f(t, x(t), ..., x(t-\tau_K), \theta(t)), \quad x(0) = x_0. \end{aligned} \tag{4.2}$$

First of all, we derive the first variation of $\tilde{J}$ and then come back to the more realistic case where $\theta(\cdot) = \theta_\beta(\cdot)$ is parametrized through some finite-dimensional β and we are interested in the loss $J(\beta) = \tilde{J}(\theta_\beta(\cdot))$.

For the following we closely follow optimal control literature as in [20]. We denote by $\delta\tilde{J}(\theta;\eta) = \frac{d}{d\alpha}\big|_{\alpha=0}\tilde{J}(\theta(\cdot)+\alpha\eta(\cdot))$ the first variation due to a continuous and bounded perturbation $\eta(\cdot)$ of the parameters. Moreover, we define $v(t;\alpha) = \theta(t) + \alpha\eta(t)$ and let $y(t;\alpha)$ be the response associated to $v(t;\alpha)$ through (4.2). Now, the idea is to introduce the perturbed Lagrangian,

$$\begin{aligned}\mathcal{L}(v(\cdot;\alpha)) = L(y(T;\alpha)) &+ \int_0^T \ell(t, y(t;\alpha), ..., y(t-\tau_K;\alpha), v(t;\alpha)) \\ &- \int_0^T \lambda(t)^\top \underbrace{\left[\dot{y}(t;\alpha) - f(t, y(t;\alpha), ..., y(t-\tau_K;\alpha), v(t;\alpha))\right]}_{(\star)} dt.\end{aligned}$$

Since by construction $(\star) = 0$, $\forall\theta$ we can freely assign $\lambda(t)$ while $\mathcal{L}(\theta) = \tilde{J}(\theta)$ and $\delta\mathcal{L}(\theta;\eta) = \delta\tilde{J}(\theta;\eta)$. Note, that viewing the optimization problem as a problem in the function spaces $\mathcal{U}\times\mathcal{X}$, where $\theta\in\mathcal{U}$ and $x\in\mathcal{X}$, we may see λ as a Lagrange multiplier enforcing the constraint $\dot{x}(t) = f(t, \mathbf{x}_\tau(t), \theta(t))$ [22]. However, in our case we may simply view it as trick to derive the adjoint equation.

In the following we will drop the explicit dependencies whenever possible and make use of the notation $\tau_0 := 0$ and

$$\begin{aligned}\partial_k g &= \frac{\partial g(t, y(t-\tau_0;\alpha), y(t-\tau_1;\alpha), ..., y(t-\tau_K;\alpha), v(t;\alpha))}{\partial y(t-\tau_k;\alpha)} \\ \partial_\theta g &= \frac{\partial g(t, y(t-\tau_0;\alpha), y(t-\tau_1;\alpha), ..., y(t-\tau_K;\alpha), v(t;\alpha))}{\partial v(t;\alpha)}.\end{aligned}$$

Furthermore, we assume differentiability of $y(t;\alpha)$ (for a proof see [23]) and get,

$$\begin{aligned}\frac{d\tilde{J}(v(\cdot;\alpha))}{d\alpha} &= \frac{d\mathcal{L}(v(\cdot;\alpha))}{d\alpha} \overset{(i)}{=} \frac{dL}{d\alpha} - \frac{d}{d\alpha}\left[\lambda^\top y\Big|_0^T\right] + \frac{d}{d\alpha}\int_0^T \left[\ell + \dot{\lambda}^\top y + \lambda^\top f\right] dt \\ &\overset{(ii)}{=} \left[L'(y(T;\alpha)) - \lambda(T)^\top\right]\frac{dy(T;\alpha)}{d\alpha} + \int_0^T \left[\sum_{k=0}^K \partial_k\ell\frac{dy(t-\tau_k;\alpha)}{d\alpha}\right. \\ &\left. + \partial_\theta\ell\frac{dv(t;\alpha)}{d\alpha} + \dot{\lambda}^\top\frac{dy(t;\alpha)}{d\alpha} + \lambda^\top\left(\sum_{k=0}^K \partial_k f\frac{dy(t-\tau_k;\alpha)}{d\alpha} + \partial_\theta f\frac{dv(t;\alpha)}{d\alpha}\right)\right] dt \\ &\overset{(iii)}{=} \left[L'(y(T;\alpha)) - \lambda(T)^\top\right]\frac{dy(T;\alpha)}{d\alpha} + \sum_{k=0}^K\int_0^T \left[\partial_k\ell + \lambda^\top\partial_k f\right]\frac{dy(t-\tau_k;\alpha)}{d\alpha}dt \\ &+ \int_0^T \dot{\lambda}^\top\frac{dy(t;\alpha)}{d\alpha}dt + \int_0^T \left[\lambda^\top\partial_\theta f + \partial_\theta\ell\right]\eta(t)\,dt\end{aligned}$$

$$\overset{(iv)}{=} \left[L'(y(T;\alpha)) - \lambda(T)^\top\right] \frac{dy(T;\alpha)}{d\alpha} + \sum_{k=0}^{K} \int_{-\tau_k}^{T-\tau_k} \left[\partial_k \ell\Big|_{t'+\tau_k} + \lambda^\top \partial_k f\Big|_{t'+\tau_k}\right] \cdot$$
$$\cdot \frac{dy(t';\alpha)}{d\alpha} dt' + \int_0^T \dot{\lambda}^\top \frac{dy(t;\alpha)}{d\alpha} dt + \int_0^T \left[\lambda^\top \partial_\theta f + \partial_\theta \ell\right] \eta(t)\, dt$$

Here, in (i) we integrated by parts, in (ii) we used the chain rule and $dy(0;\alpha)/d\alpha = 0$, (iii) follows from rearranging terms and $dv(t;\alpha)/d\alpha = \eta(t)$, and in (iv) we applied the changes of variables $t' = t - \tau_k$. By noting that $dy(t;\alpha)/d\alpha = 0$ for $t \leq 0$, setting $\lambda(t) = 0$ for $t > T$, and taking the limit $\alpha \to 0$ we get,

$$\delta\tilde{J}(\theta;\eta) = \left[L'(y(T;\alpha)) - \lambda(T)^\top\right] \frac{dy(T;\alpha)}{d\alpha}\Big|_{\alpha=0} + \int_0^T \left[\lambda^\top \partial_\theta f + \partial_\theta \ell\right] \eta(t)\, dt$$
$$+ \int_0^T \left[\sum_{k=0}^{K} \left(\partial_k \ell\, \mathbb{1}_{[0,T]} + \lambda^\top \partial_k f\right)\Big|_{t+\tau_k} + \dot{\lambda}^\top\right] \frac{dy(t;\alpha)}{d\alpha}\Big|_{\alpha=0} dt.$$

Hence, if λ solves the adjoint final value problem,

$$\dot{\lambda}(t)^\top = -\sum_{k=0}^{K} \left(\partial_k \ell\, \mathbb{1}_{[0,T]} + \lambda^\top \partial_k f\right)\Big|_{t+\tau_k}, \quad \lambda(T)^\top = L'(x(T)),$$

then the first variation is given by,

$$\delta\tilde{J}(\theta;\eta) = \int_0^T \left[\lambda^\top \partial_\theta f + \partial_\theta \ell\right] \eta(t)\, dt. \tag{4.3}$$

Using (4.3) and some finite-dimensional representation of $\theta(\cdot)$,

$$\theta_\beta(t) = \sum_{n=1}^{N} \beta_n \theta_n(t), \quad \beta := (\beta_1, ..., \beta_N)^\top, \tag{4.4}$$

and defining $J(\beta) := \tilde{J}(\theta_\beta(\cdot))$ the loss gradients follow from a simple application of the chain rule,

$$\frac{d}{d\beta} J(\beta) = \delta\tilde{J}\left(\theta; \frac{d}{d\beta}\theta_\beta(\cdot)\right) = \int_0^T \left[\lambda^\top \partial_\theta f + \partial_\theta \ell\right] \left(\theta_1(t), ..., \theta_N(t)\right) dt. \tag{4.5}$$

Here, we used that $\delta J(\theta;\eta) = DJ(\theta)\eta$ is the differential of J applied to η. (4.5) provides us with a general gradient formula for the case when θ can be represented as in (4.4). θ may be a time-dependent parameter or an input in the setting of non-linear optimal control. More general non-linear optimal control problems with delayed inputs and constraints are treated in [23].

For the rest of this thesis, we will assume the parameters to be constant. In this case $p = N$ and $(\theta_1(t), ..., \theta_p(t)) = I_p$ is an identity matrix. Further on, we set $\theta := (\theta_1, ..., \theta_p)^\top := (\beta_1, ..., \beta_p)^\top$. Thus,

$$\frac{dJ}{d\theta} = \int_0^T \left[\lambda^\top \partial_\theta f + \partial_\theta \ell\right] dt.$$

Similar as for NODEs a discrete loss at time $\tilde{t} \in [0, T]$,

$$\ell(t, x(t), ..., x(t - \tau_K)) = g(x(t), ..., x(t - \tau_K))\delta(t - \tilde{t}), \tag{4.6}$$

leads to the adjoint jump discontinuities,

$$-\sum_{k=0}^{K} \partial_k \ell \, \mathbb{1}_{[0,T]}\Big|_{t+\tau_k} = -\sum_{k=0}^{T} \partial_k g(x(\tilde{t}), ..., x(\tilde{t} - \tau_K))\delta(t - (\tilde{t} - \tau_k)),$$

at the locations $t = \tilde{t} - \tau_k$.

Similar to the NODE adjoint method we may solve the adjoint and the loss gradients together in an augmented DDE. However, since backward integration is not possible, we will always use a local interpolation of the forward pass for $x(t)$. Furthermore, in order to get a delayed rather than an advanced adjoint equation we make the following change of variables,

$$s := T - t, \quad a(s) := \lambda(T - s).$$

This leads us to the augmented DDE problem,

$$\begin{cases} \dot{z}(s) = \begin{pmatrix} \dot{a}(s) \\ \dot{b}(s) \end{pmatrix} = \begin{pmatrix} \sum_{k=0}^{K} a(s - \tau_k)^\top \partial_k f\big|_{T-s+\tau_k} + \partial_k \ell \mathbb{1}_{[0,T]}\big|_{T-s+\tau_k} \\ a(s)^\top \partial_\theta f\big|_{T-s} + \partial_\theta \ell\big|_{T-s} \end{pmatrix} \\ z(0) = \begin{pmatrix} L'(x(T)) \\ 0 \end{pmatrix} \end{cases} \tag{4.7}$$

which can be solved with the method of steps. It then holds $dJ/d\theta = b(T)$.

A numerical difficulty with (4.7) is that for a discrete loss as in (4.6) a lot of jump discontinuities are introduced in the backward pass, which will propagate in the form of higher order discontinuities along the solution of the adjoint state. Furthermore, we also need to account for the discontinuities from the forward pass and restart the DDE solver accordingly. This makes the adjoint sensitivity method for DDEs substantially slower than for ODEs and it is usually a good idea to choose $\Delta t = t_{k+1} - t_k$ and τ such that they have a small common multiple. Due to those discontinuity issues the forward sensitivity method is, for a small number of parameters, reported to be more efficient in [24]. However, since in deep learning the parameters are usually high dimensional we did not consider this approach.

4.2.2 Continuous loss and AD through solver

When we want to calculate the loss gradients using automatic differentiation through the DDE solver we first need to rewrite the problem such that the continuous loss becomes a discrete end-point loss. For this purpose we define an additional state variable $G(t) := \int_0^t \ell(s, \mathbf{x}_\tau(s), \theta) ds$ and apply AD to the solution of the following augmented DDE problem,

$$\dot{z}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{G}(t) \end{pmatrix} = \begin{pmatrix} f(t, \mathbf{x}_\tau(t), \theta) \\ \ell(t, \mathbf{x}_\tau(t), \theta) \end{pmatrix}.$$

For NDDEs we found that in most cases this approach is computationally more efficient than the adjoint sensitivity method.

4.3 Mini-batching and observation noise

In practice, it is often inefficient, and for large data sets computationally infeasible, to train NDDEs along the full data belonging to a training trajectory. We will therefore make use of mini-batching. Furthermore, we will typically be faced with a data set $\{(t_0, y_0), ..., (t_N, y_N)\}$ of noisy observations $y_i = y_i^{\text{noise-free}} + \varepsilon_i$. Here $y_i^{\text{noise-free}}$ are the noise-free, possibly partial, observations of some dynamical system and $\varepsilon_i \sim \mathcal{N}(0, \sigma)$ is normally and independently distributed observation noise.

For mini-batching we follow a similar approach as usually employed for NODEs. Lets assume the observations $\{(t_0, y_0), ..., (t_N, y_N)\}$ all belong to a single trajectory and that on the interval $[t_0 - r, t_0)$ additional data exists for the initial history. We fix a $N^{\text{batch}} \leq N$ that we will call the batch time and some number $M^{\text{batch}} \in \mathbb{N}$ that we refer to as the batch size. Then, we uniformly sample M^{batch} starting times $\xi^k \sim \mathcal{U}(\{0, ..., N - N^{\text{batch}}\})$ to get a batch consisting of M^{batch} sub-trajectories of the form $\{(t_{\xi^k}, y_{\xi^k}), ..., (t_{\xi^k + N^{\text{batch}}}, y_{\xi^k + N^{\text{batch}}})\}_{k=1}^{M^{\text{batch}}}$. However, in contrast to NODEs we also need the corresponding initial histories ψ^k on the intervals $[t_{\xi^k} - r, t_{\xi^k}]$. For this we need an interpolation or in the presence of observation noise, a smoothed interpolation of the data on the in the intervals $[t_{\xi^k} - r, t_{\xi^k}]$. In the noise-free case polynomial interpolation methods may be employed and in the noisy case we can use non-linear regression methods such as kernelized regression, support vector regression, or Gaussian Processes (GP). In Section 5.2 we fit for this purpose a GP and use its mean prediction as the smoothed interpolation. For probabilistic models a possible approach would be to train an ensemble of models by sampling from this GP.

4.4 Expressivity

4.4.1 Universal approximation

As discussed in Proposition 3.1 ANODEs can learn any homeomorphism $H : \mathbb{R}^n \to \mathbb{R}^n$ when we augment the state space to $\mathbb{R}^{2n}$. In Proposition 4.1 we suggest an analog result for NDDEs.

Proposition 4.1 *Let $H : \mathbb{R}^n \to \mathbb{R}^n$ be some arbitrary mapping and consider a non-autonomous DDE of the form $\dot{x}(t) = f(t, x(t), x(t-\tau))$. Furthermore, let $x(t_0, \psi)$ be the solution starting at $(t_0, \psi) \in \mathbb{R} \times \mathcal{C}([-r, 0])$. Then for any H and $T > 0$ there exists an f such that,*

$$H(x_0) = x(0, \psi_{x_0})(T),$$

where $\psi_{x_0}(s) = x_0$ is a constant initial history.

Proof The idea is to make use of a very simple "straight line DDE",

$$\dot{x}(t) = \begin{cases} \dfrac{H(x(t-\tau)) - x(t-\tau)}{T} & , t < \tau \\ \dfrac{x(t) - x(t-\tau)}{\tau} & , t \geq \tau. \end{cases} \tag{4.8}$$

Now, for $t \in [0, \tau]$ we have $\dot{x}(t) = (H(x_0) - x_0)/T$ and therefore,

$$x(t) = t \cdot \frac{H(x_0) - x_0}{T} + x_0.$$

Furthermore, for $t \geq \tau$ the time derivative $\dot{x}(t)$ is simply a left-sided finite difference coefficient and the solution therefore remains on the same straight line as before. Thus (1) also holds for $t \in [\tau, T]$ and,

$$x(0, \psi_{x_0})(T) = T \cdot \frac{H(x_0) - x_0}{T} + x_0 = H(x_0),$$

as desired. □

This showcases that in contrast to ODEs, trajectories may easily cross in solutions of DDEs. Furthermore, we can represent (4.8) as an autonomous DDE in $\mathbb{R}^{n+1}$. Due to the universal approximation property of neural networks [25] it follows that the input-output-map as described in Proposition 4.1 of a non-autonomous NDDE in $\mathbb{R}^n$ or an autonomous NDDE in $\mathbb{R}^{n+1}$ is a universal function approximator for a single delay. Moreover, whenever we are allowed to choose either of T or τ we can set $\tau = T$ and do therefore not need an explicit time dependency. Alternatively, using a second time delay, e.g. $\tau/2$, we can get rid of the time dependency as follows. Setting,

$$\dot{x}(t) = \frac{H(x(t-\tau)) - x(t-\tau)}{T},$$

if $x(t) = x(t - \tau/2) = x(t - \tau)$ or if $x(t), x(t - \tau/2), x(t - \tau)$ are not on the same line and,

$$\dot{x}(t) = \frac{x(t) - x(t - \tau)}{\tau},$$

otherwise, leads to the same solutions as in Proposition 4.1 with $x(0, \psi_{x_0})(T) = H(x_0)$.

4.4.2 Fourier series representation

The result from the last section may be interesting when we view the model as a continuous deep learning model e.g. to classify images, but for our needs it is not strong enough since we are mainly interested in the trajectory in between $t = 0$ and $t = T$. In the following two sections we will attempt to investigate the representation capability along the trajectory.

In Section 2.4 we have seen that $A\sin(\omega t - \phi)$ is a solution of the linear DDE $\dot{x}(t) = -\omega x(t - \pi/(2\omega))$. Also there is at most one pair of purely imaginary roots of the characteristic polynomial and accordingly at most one frequency for solutions of the form $A\sin(\omega t - \phi)$. We may therefore ask whether we can represent more frequencies with more delays. Proposition 4.2 gives an upper bound on the number of delays K we need to represent a Fourier expansion up to a certain order L.

Proposition 4.2 *We can represent any truncated Fourier series expansion,*

$$x(t) = A_0 + \sum_{l=1}^{L} A_l cos(l\omega t + \phi_l),$$

as a solution of a linear DDE with $K = 2L$ commensurate delays,

$$\dot{x}(t) = a_0 + \sum_{k=1}^{K} a_k x(t - k \cdot \tau).$$

Where

$$\tau = \frac{2\pi}{\omega(L-1)} \quad and$$
$$(a_0, ..., a_K) = DFT^{-1}(0, i\omega, ...i\omega L, -i\omega L, ..., -i\omega).$$

Proof It is enough to show that $\{\pm il\omega\}_{l=0}^{L}$ are roots of the characteristic polynomial,

$$\chi(s) = s - a_0 - \sum_{k=1}^{K} a_k e^{-s\tau_k} = 0,$$

with τ and $\{a_k\}_k$ defined as in the statement. Defining $M := K + 1$ and $c_M := e^{-i2\pi/M}$ we can write this in matrix form as,

$$\underbrace{i\begin{pmatrix} 0 \\ \omega \\ 2\omega \\ \vdots \\ L\omega \\ -L\omega \\ \vdots \\ -\omega \end{pmatrix}}_{b\in i\mathbb{R}^M} = \underbrace{\begin{pmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & c_M & c_M^2 & ... & c_M^{M-1} \\ 1 & c_M^2 & c_M^4 & ... & c_M^{2(M-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & c_M^{M-1} & c_M^{2(M-1)} & ... & c_M^{(M-1)^2} \end{pmatrix}}_{C_M} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_K \end{pmatrix}}_{a\in\mathbb{R}^M}.$$

Here in the m-th row for $m > L+1$ the left hand side is $-i(M-m+1)\omega$ and we made use of,

$$e^{-(-i(M-m+1)\omega)\frac{2\pi k}{\omega M}} = e^{i(M-m+1)2\pi k/M} = e^{-i(m-1)2\pi k/M} = c_M^{(m-1)k}.$$

Inspecting the matrix C_M we readily see that this is the discrete fourier transform matrix (DFT) and that the coefficients vector a is the inverse DFT of the frequency vector b. Now, due to the conjugate symmetric structure of b the coefficients are real. □

4.4.3 Delay embeddings

Much more general results could be obtained by means of so-called delay embeddings. Assume we are given a dynamical system with $\mathcal{C}^2$ solution map,

$$\varphi_s : \mathbb{R}^n \to \mathbb{R}^n,\ x(t) \mapsto \varphi_s(x(t)) = x(t+s), \tag{4.9}$$

that is defined by a differential equation $\dot{x}(t) = g(x(t))$.

Furthermore, assume that $\mathcal{M} \subset \mathbb{R}^n$ is some submanifold that is invariant under φ_s and let,

$$h : \mathbb{R}^n \to \mathbb{R}, x(t) \mapsto y(t) = h(x(t)),$$

be some $\mathcal{C}^2$ observation map. Now, we are interested in the question whether we can retain information about the state $x(t)$ from time-series measurements of $y(t)$. Takens' theorem [26] provides us with conditions under which this can be answered positive. In particular lets define the delay coordinate map,

$$\begin{aligned} E : \mathcal{M} &\to \mathbb{R}^d, \\ x(t) \mapsto \mathbf{y}(t) &= \big(y(t),\quad y(t-T),\quad ...,\quad y(t-(d-1)T)\big) \\ &= \big(h(x(t)), h\circ\varphi_{-T}(x(t)), ..., h\circ\varphi_{-T}^{d-1}(x(t))\big), \end{aligned} \tag{4.10}$$

with sampling time T. Then the following theorem holds.

Theorem 4.3 (Takens[26]) *Let $\mathcal{M}$ be a compact manifold of dimension M and suppose we have a dynamical system defined by* (4.9) *that is confined on this manifold. Let $d > 2M$ and suppose the periodic points of φ_{-T} are finite in number, and φ_{-T} has distinct eigenvalues on any such periodic point. Then the observation maps h, for which the delay coordinate map* (4.10) *is an embedding, form an open and dense subset of $\mathcal{C}^2(\mathcal{M}, \mathbb{R})$.*

Loosely speaking the above theorem tells us that if we consider enough delays in (4.10) and choose T such that we do not hit to many periodic points, then for most observation maps h the delay coordinate map E is one-to-one on $\mathcal{M}$. Note that besides the condition on periodic points, the above theorem does not make a statement about the choice of T. However, in practice this turns out to be important and multiple, mostly empirical, criteria for this choice exist [27].

If E is invertible we have,

$$\begin{aligned}\dot{y}(t) &= \frac{d}{dt}h(x(t)) = h'(x(t))\dot{x}(t) = h'(x(t))g(x(t)) \\ &= h'(E^{-1}(\mathbf{y}(t)))g(E^{-1}(\mathbf{y}(t))) = f(\mathbf{y}(t)),\end{aligned}$$

which is a DDE in $y(t)$. Due to the universal approximation theorem for neural networks and provided that we know $y(t)$ on the interval $[t-(d-1)T, t]$, we should therefore be able to learn the dynamics of $y(t)$ by means of a NDDE.

Replacing M with the upper box-counting dimension Theorem 4.3 can be extended to chaotic attractors [28] and infinite-dimensional systems [29]. For future work, it would therefore be interesting to see whether we can experimentally verify this and predict the dynamics of a chaotic attractor by NDDEs. However, note that whereas the above theorems apply for periodic or chaotic attractors, they do not apply to dissipative systems as often encountered in practice.

4.5 Remarks on implementation

We close this chapter with a few remarks on our implementation of NDDEs. As already noted we make use of the Julia libraries described in [9] and [8]. Furthermore, for the DDE solver we usually use a method of steps method based on the *Tsit5* ODE solver, which is a 5/4 Runge-Kutta method with 4th order interpolant. For the gradient calculations, we did on the one hand implement the adjoint sensitivity method described in Section 4.2.1 and on the other hand use AD through the solver as provided by the above libraries. In our experiments automatic differentiation turns out to be more efficient for NDDEs. However, the performance of the adjoint method may be further optimized and it comes with the advantage that no differentiable DDE solvers are needed.

For the learning rates we found it to be very helpful to use a special version of cyclic learning rates. More specifically, we use repeated exponential decays that are decaying in amplitude over time.

For the activation functions of the neural network $f(\cdot, \theta)$ a wide-spread choice for NODEs is the hyperbolic tangent (tanh) activation. Rectified Linear Unit (ReLU) is usually avoided because it is not continuously differentiable. In our experiments the swish activation, which is a smoothed ReLU version proposed in [30], consistently outperformed tanh activations. Especially, in the presence of exponential decays or increases, tanh activations do not work well. The superior performance of swish is not surprising, as on the one hand, it is well-known that ReLU outperforms tanh on several deep learning tasks and on the other hand, swish reportedly outperformed ReLU on several classification tasks [30]. In the following experiments we are therefore relying on swish as activation function.

Chapter 5

NDDE Experiments

In this chapter we present our experimental results for NDDEs. In Section 5.1 we compare NODE, ANODE, and NDDE for the task of modelling a partially observed dynamical system. In Section 5.2 we will see how the NDDE behaves in the presence of observation noise and in Section 5.3 generalization is investigated. Lastly, the expressive power of the model is showcased for more complex data in Section 5.4.

5.1 Learning partially observed dynamics

In the following section, we compare the performance of NODE, ANODE and NDDE by means of their ability to learn the model of a partially observed oscillator. The experimental setup is as follows: Given an undamped harmonical oscillator with frequency $\omega = 1$,

$$\ddot{x}(t) + x(t) = 0, \quad x(0) = 1, \dot{x}(0) = 0, \tag{5.1}$$

we would like to train a neural network model of (5.1) based on observations of $x(t) = \cos(t)$ only. In particular our training data set $\{(t_0, y_0), ..., (t_N, y_N)\}$ consists of equally distributed sample times $t_k = k \cdot \Delta t$ and corresponding noise free observations $y_k = x(t_k)$.

For the neural network $f(\cdot, \theta)$ we choose to use a fully connected depth six architecture with hidden layer sizes $(32, 64, 128, 64, 32)$ and swish activation. The input and output layer sizes are chosen to match the respective model.

5.1.1 NODE

A first try to use an autonomous vanilla NODE model $\dot{x}(t) = f(x(t), \theta)$ fails completely. As depicted in Figure 5.1 we are only learning a monotone decay. This is obvious as even a non-linear one-dimensional ODE can only ever have monotone solutions $t \mapsto x(t)$. It is therefore evident that the vanilla

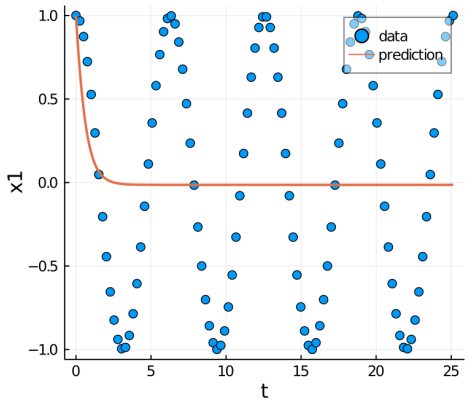


Figure 5.1: Solution predicted by NODE

NODE model is not expressive enough to model partially observed systems. From a statistical point of view, although the dynamics being deterministic, we may say that the continuous process $\{x(t)\}$ is non-Markovian, since the infinitesimal change depends not only on the current state but also on the past. However, autonomous ODEs are inherently Markovian as their evolution is completely predetermined by the current state. ANODEs and NDDEs can be seen as two distinct methods that lift the problem to a Markovian setting - ANODEs by allowing the state to include additional derivative information and NDDEs by augmenting the state with the past history.

5.1.2 ANODE

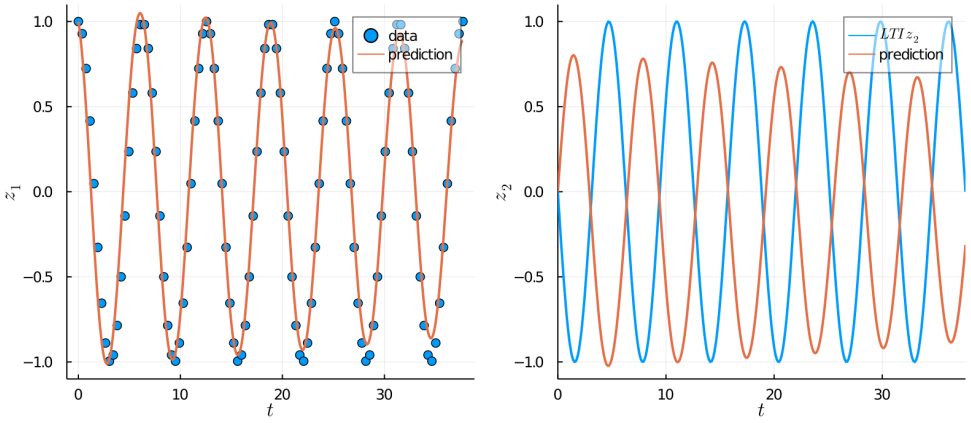
In the ANODE model as described in Section 3.2 we lift the problem from $\mathbb{R}$ to $\mathbb{R}^{1+m}$, train a NODE in that space, and then project back to $\mathbb{R}$. For the oscillator this is clearly very well justified, since we can rewrite (5.1) in state space form for Linear Time Invariant systems (LTI) as,

$$\begin{aligned} \dot{z}(t) &= \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} z(t), \quad z(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ y(t) &= \begin{pmatrix} 1 & 0 \end{pmatrix} z(t), \end{aligned} \tag{5.2}$$

with $z(t) = \begin{pmatrix} x(t) & \dot{x}(t) \end{pmatrix}^\top$. So indeed we are trying to learn a two-dimensional LTI system from the one-dimensional observations of $y(t)$. We will therefore choose $m = 1$ and try to learn (5.2) with a two-dimensional NODE $\dot{z}(t) = f(z(t), \theta)$.

Figure 5.2 shows the results for the case when we pass the model the true initial condition for the augmented state $z_2(0) = 0$. In this case, the model fits the training data well and the augmented state is roughly the negative

of $x(t)$. Considering that we may also use a state vector $\tilde{z} = \begin{pmatrix} x & -\dot{x} \end{pmatrix}^\top$ to rewrite (5.1), the negative sign in the augmented state is not surprising.

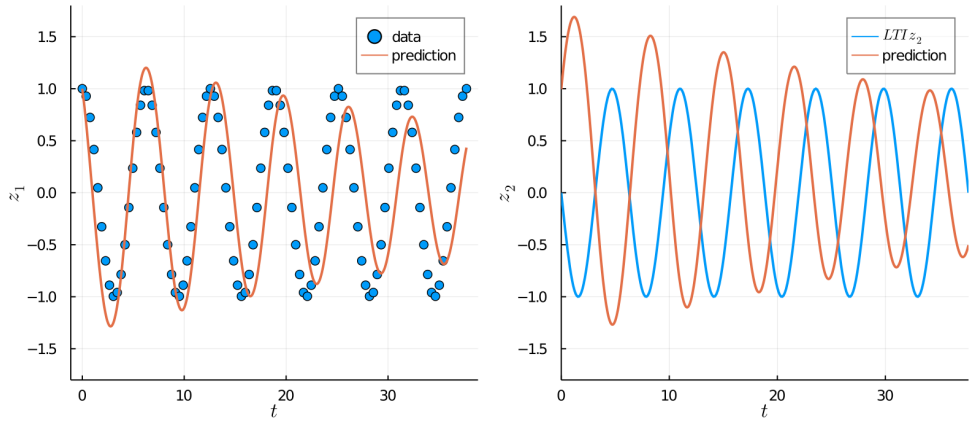


(a) Data and prediction (b) Augmented state

Figure 5.2: (a) shows the prediction for $x(t)$ and the training data. In (b) the predicted augmented state and $\dot{x}(t)$ are depicted. This is for the true initial condition of the augmented state.

For more general systems we will not always have $z_2(t) = \dot{z_1}(t)$ and it is therefore not clear how to initialize the augmented states of an ANODE. For this reason we conduct a second experiment where we randomly initialize the augmented initial condition according to $z_2(0) = z_{2,0} \sim \mathcal{N}(0, 1)$ and then try to learn it by gradient-based optimization. As we noted in Section 3.1.2, it holds $\frac{dJ}{dz_0} = \lambda(0)$, where λ is the adjoint state. We can therefore treat $z_{2,0}$ as a parameter and learn it together with the neural network parameters. The result is shown in Figure 5.3.

Clearly the initial condition of the augmented state is not learned correctly. Even in an alternating setting where we train only one of θ and $z_{2,0}$ for a few episodes while fixing the other, we get stuck in local minima. The result is that the exponential decay in the second variable translates to a decay in the prediction of $x(t)$ which is undesirable. In practice, if we do not know $z_{2,0}$ at all we might be even further off the true initial value through random initialization.



(a) Data and prediction (b) Augmented state

Figure 5.3: (a) shows the prediction for $x(t)$ and the training data. In (b) the predicted augmented state and $\dot{x}(t)$ are depicted. This is for the case where we randomly initialize $z_{2,0}$ and treat it as an optimizable parameter.

5.1.3 NDDE

For the NDDE $\dot{x}(t) = f(x(t), x(t-\tau_1), ..., x(t-\tau_K), \theta)$ the main question is which and how many delays we need to uniquely describe the infinitesimal change at a point $x(t)$ on the trajectory. In Section 2.4 we have seen that $x(t) = A\cos(t-\phi)$ is a solution of,

$$\dot{x}(t) = -x(t - \frac{\pi}{2}).$$

This means if we choose the delay perfectly, a single observation along the trajectory would be enough to predict $\dot{x}(t)$. However, this is a peculiarity of the harmonic oscillator and is due to the identity $\cos'(t) = -\cos(t - \pi/2)$. Since we do not want to hard code such system specific knowledge into our model we will at least need two observations along the trajectory. Therefore, we may fix some τ and see whether we can find a function $f : [0,1]^2 \to [0,1]$ such that,

$$f(\cos(t), \cos(t-\tau)) = -\sin(t) \quad , \forall t \in \mathbb{R}.$$

We readily see that if $\tau \neq n2\pi$ the points $\{(\cos(t), \cos(t-\tau))\}_{t\in[0,2\pi]}$ are all distinct and therefore f exists. A candidate for f could be constructed in the following way: If $t \in [2n\pi, (2n+1)\pi]$ we choose $f(x_1, x_2) = -\sin(\arccos(x_1))$ and otherwise $f(x_1, x_2) = \sin(\arccos(x_1))$. Additionally, since we are not allowed to use t, we need to learn a function $g(\cos(t), \cos(t-\tau)$ which tells us whether the angle t is lying in the first two or in the last two quadrants.

In order to experimentally verify the above considerations we simply choose $\tau = 1$. As depicted in Figure 5.4 the NDDE can easily fit the training data.

Note that theoretically the NDDE is also employing state augmentation from the one-dimensional state $x(t)$ to the infinite-dimensional state $x_t \in \mathcal{C}([-\tau, 0])$. Furthermore, our choice of f then restricts the infinitesimal change to only depend on $x_t(0)$ and $x_t(-\tau)$. But as opposed to ANODEs, the initialization is, when we are sampling fast enough, completely predetermined by the training data. This is clearly a great advantage, since initialization of augmented states was the main issue with ANODEs. However, the infinite-dimensionality also comes at the cost of more complicated analysis with infinitely many solutions. It is therefore particularly important to investigate generalization capabilities of NDDEs and whether they overfit to noisy observations in the training data. For the simple case of a harmonic oscillator these issues will be discussed in the following two sections.

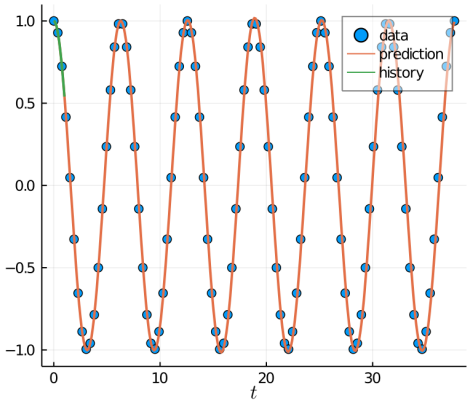


Figure 5.4: In this figure training data, NDDE prediction and the initial history are illustrated.

5.2 Noisy observations

In order to investigate the training behaviour under noisy observations we compare two NDDE configurations under two different noise settings. Once the model with a single delay $\tau = 1$ from Section 5.1.3 and once a NDDE with $K = 10$ equally spread delays $\tau_1 = 0.3, \tau_2 = 0.6, ..., \tau_{10} = 3$.

For the training we proceed as follows: First of all, we train a Gaussian Process (GP) with Radial Basis Function (RBF) kernel to generate the smoothed interpolation required for the initial history. Since the data set is small we can use a single GP for the whole trajectory, but for larger data sets we may simply train a GP for each initial history in a batch. In order to train the NDDE, we then use the mean of the GP as the smoothed interpolation of the initial histories. For the loss however we use the noisy data and not its

smoothed version.

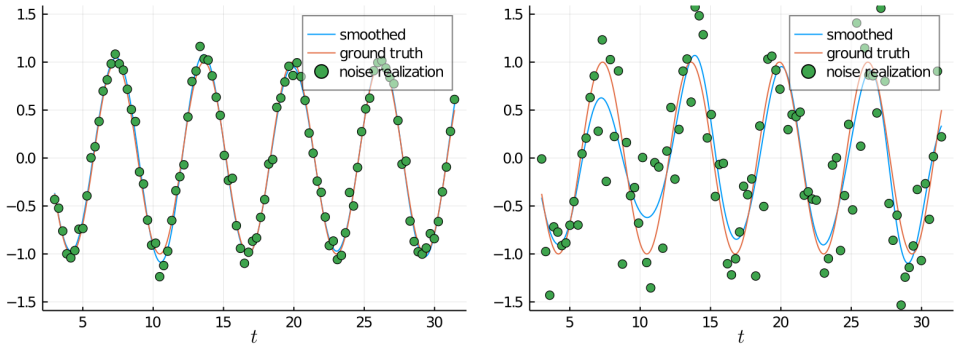
We compare the two models for two different noise settings. A low noise level with Gaussian noise of standard deviation $\sigma = 0.1$ and a high noise level of standard deviation $\sigma = 0.5$. For each setting 10 models are trained with independent noise realizations. In the Figures 5.5a and 5.5b a single data realization and its corresponding smoothed interpolation are compared to the ground truth. It is plain to see that, whereas the GP interpolation is almost exact in the low noise setting, it clearly deviates from the ground truth for high noise. However, since we are only using the smoothed interpolation for the initial history we may still hope for a better fit of the NDDE model itself.

Figures 5.5c-5.5f depict the median predictions for the different model and noise settings. The grey-shaded area indicates the 0.1- and 0.9-quantiles stemming from the 10 independently trained models. Whereas both the single and multiple delay model are able to fit the training data for $\sigma = 0.1$, only the multiple delay model is able to learn a good prediction in the high noise setting. Since we have clearly shown that the single delay model is expressive enough, this means that we are facing optimization problems and are getting stuck in local optima. This is in accordance with a general observation that we made during this thesis that more delays lead to better training performance. Moreover, the tighter confidence bounds and smaller Root-Mean-Square Error (RMSE) (see Table 5.1) also speak for the multiple delay model. However, more delays also come at the cost of a longer training time per iteration.

In the next section where we discuss extrapolation and generalization across initial conditions we will focus on the larger model with 10 delays.

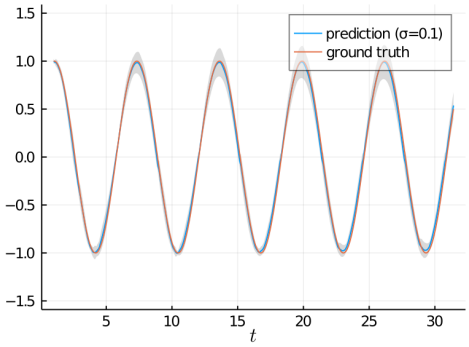
	$\sigma = 0.1$, $K = 1$	$\sigma = 0.1$, $K = 10$	$\sigma = 0.5$, $K = 1$	$\sigma = 0.5$, $K = 10$
episodes	100	40	100	40
data size	100	100	100	100
batch size	2	2	2	2
batch time	50	50	50	50
learning rates	5e-3-5e-5	5e-3-5e-5	5e-3-5e-5	5e-3-5e-5
avg. training time	38s	46s	35s	47s
avg. RMSE	0.1458	0.0832	0.8349	0.2439

Table 5.1: Training summary

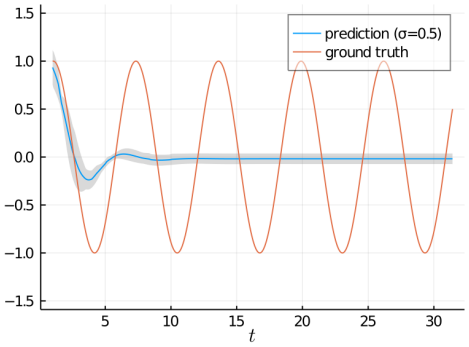


(a) Noise $\sigma = 0.1$

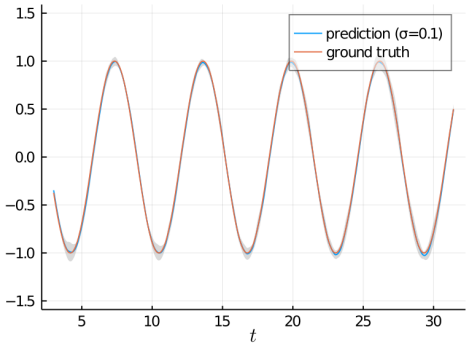
(b) Noise $\sigma = 0.5$



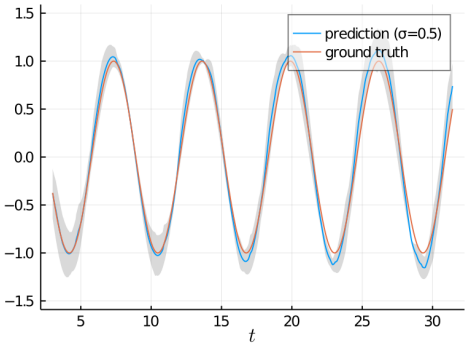
(c) $K = 1, \sigma = 0.1$



(d) $K = 1, \sigma = 0.5$



(e) $K = 10, \sigma = 0.1$



(f) $K = 10, \sigma = 0.5$

Figure 5.5: In (a) and (b) a single noise realization for noise levels $\sigma = 0.1, 0.5$ and the corresponding smoothed interpolation (mean of GP) are depicted. Figures (c)-(f) show the median prediction of 10 models trained with independent noise realizations. The grey area refers to 80%-confidence bounds.

5.3 Generalization

So far we have seen that an NDDE can achieve a low training error on a single solution of (5.1). This section discusses generalization over time and across different initial conditions. After training with a single trajectory we cannot expect the NDDE to be homogeneous in the sense,

$$f(c\mathbf{x}_\tau(t), \theta) = cf(\mathbf{x}_\tau(t), \theta),$$

and accordingly we cannot expect generalization across different initial conditions of (5.1). Therefore, our goal is to enforce this property through training data that includes multiple trajectories of different amplitude. In particular we compare a model trained with two, to a model trained with four training trajectories of the form $x(t) = A_i \cos(t - \varphi_i)$, where amplitude and phase are chosen according to Table 5.2.

	A_1	A_2	A_3	A_4	φ_1	φ_2	φ_3	φ_4
2 Trajectories	1	2	-	-	$-\pi/6$	0	-	-
4 Trajectories	1	2	0.75	2.5	$-\pi/6$	0	$-\pi$	$5\pi/3$

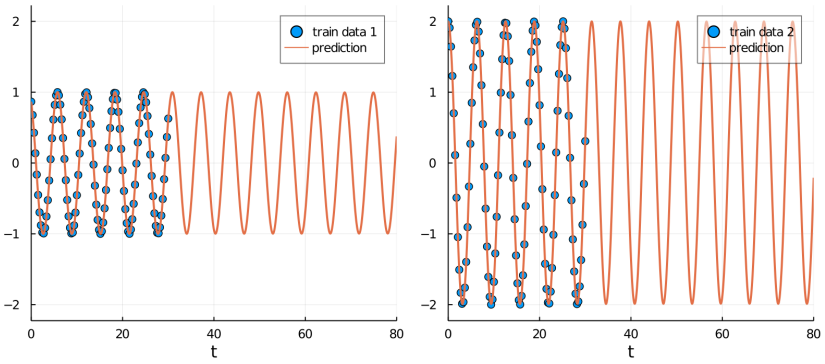
Table 5.2: Amplitudes and phases of training data

Moreover, we consider two test trajectories of amplitudes $B_1 = 1.5, B_2 = 0.5$ and phases $\vartheta_1 = -5\pi/6, \vartheta_2 = \pi/2$.

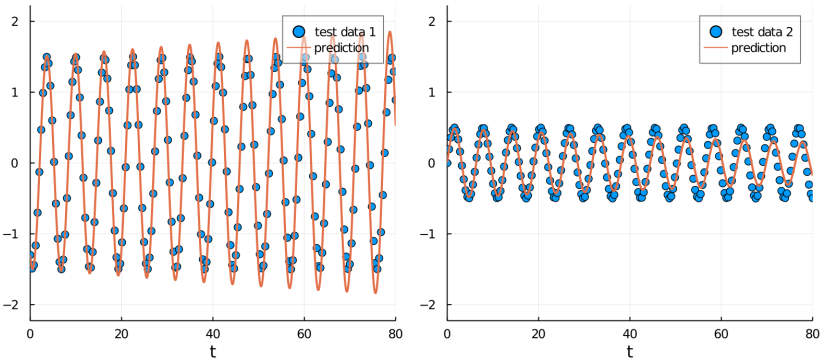
The results for the model which is trained with two trajectories are illustrated in Figure 5.6. From Figure 5.6a it is apparent that extrapolation of the training data over time works very well. This is to be expected due to periodicity of the data. The corresponding test predictions in Figure 5.6b still show an exponential increase on test data 1 and a decrease on test data 2. However, the test predictions of the model trained with four trajectories, as illustrated in Figure 5.7, indicate that this can be alleviated when we add two additional training trajectories. Not surprisingly also the test errors summarized in Table 5.3, show better generalization performance of the model trained on the larger data set.

	RMSE test data 1	RMSE test data 2
Model 2 Trajectories	0.3071	0.1608
Model 4 Trajectories	0.0619	0.0554

Table 5.3: Test errors



(a) Training data and extrapolation



(b) Test data and prediction

Figure 5.6: Extrapolation and generalization for two training trajectories. (a) shows the training data and extrapolation over time. In (b) test data and prediction are illustrated.

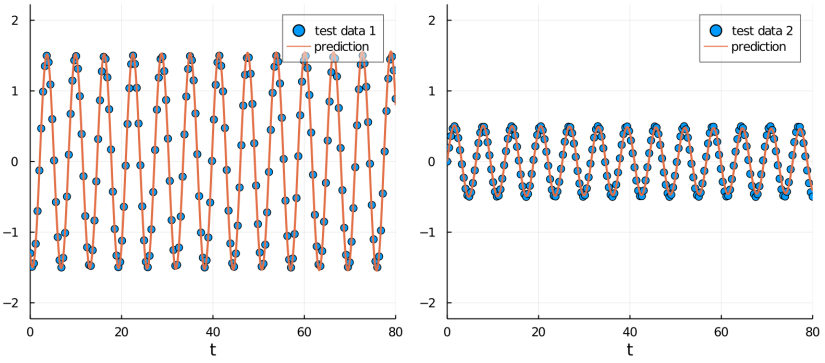


Figure 5.7: Test data for 4 training trajectories. We omit the corresponding training trajectories, as extrapolation is almost trivial in this case.

5.4 Expressivity

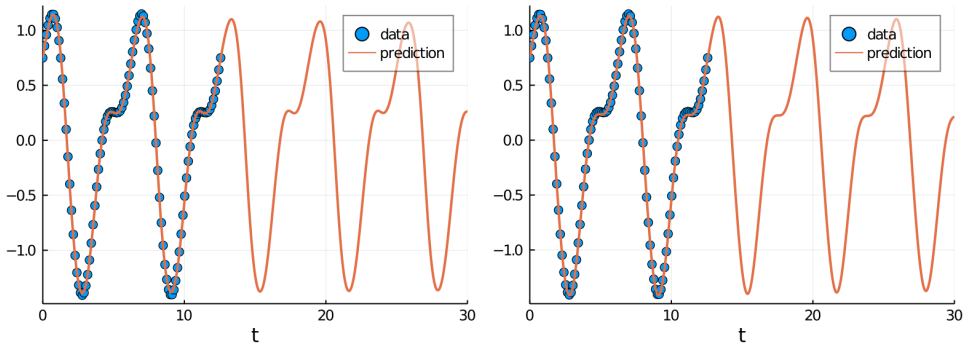
This section provides further examples that showcase the expressivity of NDDE. First of all, we experimentally investigate the result about Fourier series expansions from Section 4.4.2 and then quickly discuss the performance on a highly non-linear biological DDE model.

5.4.1 Fourier series

In Proposition 4.2 we showed that with $K = 2L$ delays we can represent solutions of the form,

$$x(t) = A_0 + \sum_{l=1}^{L} A_l \cos(l\omega t + \varphi_l), \text{ with } \tau = \frac{2\pi}{\omega(L-1)},$$

as solutions of a linear DDE. In the last section we have already seen that for $L = 1$ a single delay rather than two is enough. In the following we experimentally investigate the case $L = 2$. We choose $\omega = 1, A_0 = 0, A_1 = 1, A_2 = 0.5, \varphi_1 = 0, \varphi_2 = 2\pi/3$. According to Proposition 4.2 we choose the four delays $\{\tau_k := k\pi/2\}_{k=1}^{4}$ and compare the resulting NDDE to a model trained with the two delays $\{\pi/2, \pi\}$ only. Moreover, in order to investigate the influence of τ we consider a two and a four delay model with integer delays. The results and training parameters are summarized in Table 5.4. Furthermore, Figure 5.8 illustrates the predictions for the first two models. For the predictions of the latter two models we refer to Appendix A.1.



(a) $\tau = \pi/2$, $K = 4$ (b) $\tau = \pi/2$, $K = 2$

Figure 5.8: Training data and prediction for fourier series with two non-zero coefficients.

We clearly see that we do not need the four delays suggested by Proposition 4.2. This is not really surprising as Proposition 4.2 is only giving us sufficient but not necessary conditions to be able to fit our training data. Moreover,

	$\tau = \pi/2$, $K = 4$	$\tau = \pi/2$, $K = 2$	$\tau = 1$, $K = 4$	$\tau = 1$, $K = 2$
episodes	200	1600	400	2000
data size	100	100	100	100
batch size	2	2	2	2
batch time	50	50	50	50
learning rates	5e-3-5e-5	5e-3-5e-5	5e-3-5e-5	5e-3-5e-5
training time	1min 42s	12min 38s	3min 3s	15min 36s
RMSE	0.0190	0.0744	0.0229	0.1355

Table 5.4: Model and training summary

in contrast to Proposition 4.2 our dynamics are not constrained to be linear. However, considering the training times that were needed to fit the training data with a low RMSE, we clearly see that the two four delay models are easier to train. Further on, we also see slight advantages for the delays proposed by Proposition 4.2. On the one hand, a reason for this might be that, at least in the four delay case, we only need to learn a linear function. On the other hand, it is a well-known issue from delay embeddings that the choice of the delays is crucial in practice [27].

5.4.2 Mackey-Glass dynamics

For a last experiment we consider the Mackey-Glass equation - a highly non-linear time delay model from biology. The equation is given by,

$$\dot{x}(t) = \frac{\beta x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t),$$

and is used to model blood cell production. Further on, it is reported to exhibit a wide-range of different periodic and chaotic behavior depending on the choice of parameters [31]. For our choice $\gamma = 1, \beta = 2, \tau = 2, n = 9.65$ its dynamics are chaotic. To model the dynamics we use a NDDE model with 10 delays $\tau_1 = 0.4, ..., \tau_{10} = 4$ and train over 5000 episodes. The resulting prediction is illustrated in Figure 5.9 and a summary of the training parameters is provided in Table 5.5. We see that we can accurately fit the training data and also the extrapolation is not too far off the ground truth. This is remarkable as we are training the model on a relatively short time interval and our NDDE is heavily overparametrized in terms of delays.

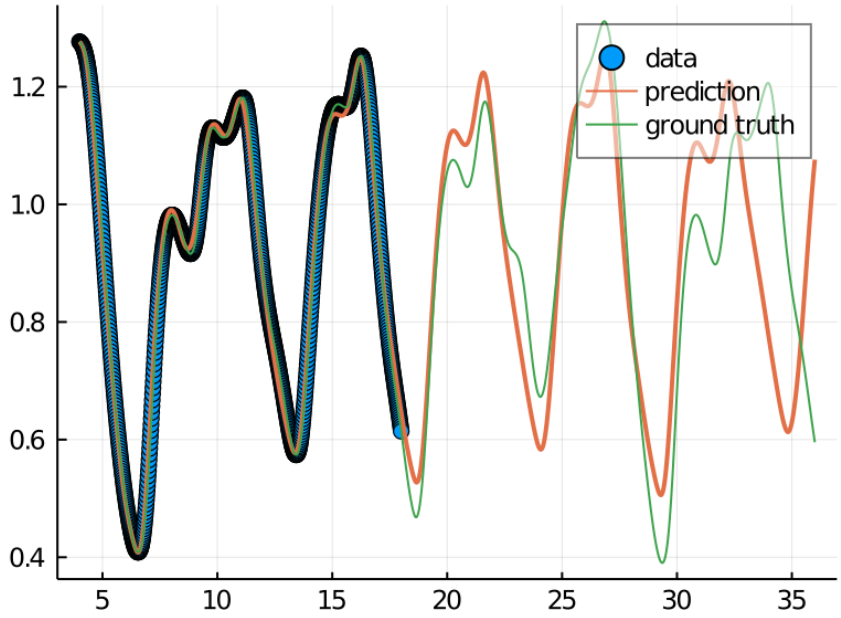


Figure 5.9: Mackey-Glass training data and corresponding NDDE prediction are illustrated. Moreover, the extrapolation of the ground truth is depicted. Note that this system exhibits chaotic behavior.

episodes	5000
data size	1000
batch size	16
batch time	100
learning rates	1e-3-1e-6
training time	14min 58s
RMSE	0.0161

Table 5.5: Training summary

5.5 Conclusion

In summary, we have shown that NDDEs outperform ANODEs on the task of modelling a partially observed harmonical oscillator. As the main reason for this we identify the initialization of augmented states, which is problematic for ANODEs. Further on, we have seen that despite their expressivity NDDEs do not tend to overfit to observation noise and that generalization over initial conditions can be achieved by training on multiple trajectories. Finally, we demonstrated expressive power of NDDEs on more complex periodic data and also for a chaotic non-linear time delay system.

Chapter 6

Learning stable dynamics

In real world applications, it is often crucial to learn models that are stable with respect to some equilibrium point and we almost always want to avoid exploding behaviour of solutions. On the one hand, we may see this as a chance to further restrict the hypothesis space of NDDEs without assuming in-depth knowledge about the true dynamics of the system at hand. On the other hand, especially in the context of control engineering and reinforcement learning, an unstable dynamics model may cause physical harm to the agent or its environment. Therefore, suitable stability guarantees would not only introduce an additional inductive bias, but are often critical for real world applications.

For local stability analysis of non-linear DDEs, a possibility is to linearize the DDE function around the equilibrium and employ frequency-domain stability criteria for linear DDEs. Another wide-spread approach to stability of non-linear differential equations is time-domain analysis by means of Lyapunov functions [32]. This comes with the major advantage that stability can be proven globally and that a safe region of attraction can be computed. However, finding a suitable Lyapunov function for non-linear systems turns out to be a hard task. Recently, neural networks have proven to be a promising tool for tackling this problem. In [10] neural network Lyapunov functions are used to calculate a stable region of attraction for a fixed feedback control of a known discrete-time system. In [11] both a Lyapunov function and a linear feedback control are learned for continuous-time systems in order to establish provable stability guarantees for the closed loop system. Furthermore, the authors of [12] suggest to learn the dynamics model together with a Lyapunov function in order to ensure the learned dynamics to be stable. The goal of this chapter is to explore the applicability of neural network Lyapunov functions to time delay systems. First, we will attempt to extend the method of [12] to time delay systems. Secondly, we will employ a method similar as presented in [11] to learn a stabilizing control of delayed feedback systems and show that

it is able to stabilize a delayed inverted pendulum. However, for this second method we do, at least for now, not attempt to be provable stable, as the infinite-dimensionality of the state space makes this problem more difficult.

In the following section, we quickly present the Lyapunov theorems for time delay systems.

6.1 Time domain stability for DDEs

In time-domain stability analysis of time delay systems we usually consider the general FDE,

$$\begin{cases} \dot{x}(t) = f(t, x_t) \\ x_{t_0} = \psi, \quad \psi \in \mathcal{C}([-r, 0]). \end{cases} \tag{6.1}$$

First of all, we start with the more common notion of uniform asymptotic stability of (6.1) and the corresponding stability theorems. Then we proceed with exponential stability.

Definition 6.1 ([15]) *The trivial solution $x(t) \equiv 0$ of (6.1) is*

- *uniformly stable if $\forall t_0 \in \mathbb{R}$ and $\forall \varepsilon > 0$, there exists a $\delta = \delta(\varepsilon)$ such that $\|\psi\|_\infty < \delta$ implies $\|x(t)\|_2 < \varepsilon$ for all $t \geq t_0$.*
- *uniformly asymptotically stable if it is uniformly stable and there exists a $\delta_a > 0$ such that for any $\eta > 0$ there is a $T(\delta_a, \eta)$ such that $\|\psi\|_\infty < \delta_a$ implies $\|x(t)\|_2 < \eta$ for all $t \geq t_0 + T(\delta, \eta)$ and $t_0 \in \mathbb{R}$.*
- *globally uniformly asymptotically stable if δ_a can be arbitrary large.*

For stability of time delay systems two main direct Lyapunov methods exists. On the one hand, the method of Lyapunov functionals due to Krasovskii[33] which is the natural extension of Lyapunov's method to an infinite-dimensional state space. The goal is to find a positive-definite functional,

$$V : \mathbb{R} \times \mathcal{C}([-\tau, 0]) \to \mathbb{R}_+, (t, \psi) \mapsto V(t, \psi),$$

such that the upper right derivative along the trajectory,

$$\dot{V}(t, \psi) = \limsup_{h \to 0^+} \frac{1}{h} \left[V(t + h, x_{t+h}(t, \psi)) - V(t, \psi) \right],$$

is non-negative. Here, $x_\xi(t, \psi)$ indicates the solution at time $\xi \geq t$ going trough the point (t, ψ). The theorem is stated as follows:

Theorem 6.2 (Krasovksii[33]) *Suppose $u, v, w : \mathbb{R}_+ \to \mathbb{R}_+$ are continuous nondecreasing functions, $u(s), v(s), w(s) > 0$ for $s > 0$, and $u(0) = v(0) = w(0) = 0$. Then the trivial solution of 6.1 is uniformly asymptotically stable*

if there is a continuous functional $V : \mathbb{R} \times \mathcal{C}([-\tau, 0]) \to \mathbb{R}_+$*, which is positive definite,*

$$u(\|\psi(0)\|_2) \leq V(t, \psi) \leq v(\|\psi\|_\infty),$$

such that,

$$\dot{V}(t, \psi) \leq -w(\|\psi(0)\|_2).$$

If in addition $\lim_{s\to\infty} u(s) = \infty$ *then it is globally uniformly asymptotically stable.*

Similarly to Lyapunov's method for ODEs also criteria for exponential stability based on Lyaponv-Krasovskii functionals exist.

On the other hand, we can in principle also make use of positive-definite Lyapunov functions as in the ODE case. However, the condition of a negative derivative of V along the trajectory will for most systems be much too restrictive to proof stability (e.g. consider stable oscillations). The idea of Razumikhin-type theorems [34] is to require a negative derivative at time t only when we are about to leave the sublevel set $V^{\leq\eta}$ of $\eta := \sup_{s\in[-r,0]} V(t + s, x(t + s))$. The following standard extension of Razumikhin's original theorem treats uniform asymptotic stability and is due to Hale [35].

Theorem 6.3 (Razumikhin [35]) *Suppose* $u, v, w : \mathbb{R}_+ \to \mathbb{R}_+$ *are continuous nondecreasing functions,* v *is strictly increasing,* $u(s), v(s), w(s) > 0$ *for* $s > 0$*, and* $u(0) = v(0) = w(0) = 0$*. Furthermore, let* $q(s) > s$ *be a continuous nondecreasing function. The trivial solution is uniformly asymptotically stable if there is* $V : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}_+$ *such that,*

(i) $u(\|x\|_2) \leq V(t, x) \leq v(\|x\|_2)$

(ii) $\dot{V}(t, x(t)) \leq -w(\|x(t)\|_2)$
whenever $V(t + s, x(t + s)) \leq q(V(t, x(t)))) \quad \forall s \in [-r, 0].$

If in addition $\lim_{s\to\infty} u(s) = \infty$ *then it is globally uniformly asymptotically stable.*

Here, V is a differentiable function and the derivative along the solution is,

$$\dot{V}(t, x(t)) = \frac{d}{dt}V(t, x(t)) = \frac{\partial V(t, x(t))}{\partial t} + \frac{\partial V(t, x(t))}{\partial x} f(t, x_t).$$

Since we are often also interested in the rate of decay, we continue with the definition of exponential stability for time delay systems.

Definition 6.4 *The time delay system* (6.1) *is said to be exponentially stable if there exist* $M > 0, \gamma > 0$ *such that for any initial history* $x_{t_0} = \psi$ *it holds,*

$$\|x(t)\|_2 \leq e^{-\gamma(t-t_0)} M \|\psi\|_\infty \quad \text{for } t \geq t_0.$$

In the following we follow [36] which treats exponential stability for the more general case of impulsive delay systems and present their result for the special case of no impulses.

Theorem 6.5 ([36]) *Assume there exists a differentiable function* $V : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}_+$ *and constants* $p > 0, q > 1, c_1 > 0, c_2 > 0, \alpha > 0$ *such that the following hold:*

(i) $c_1 \|x\|_2^p \leq V(t,x) \leq c_2 \|x\|_2^p$

(ii) $\dot{V}(t, x(t)) \leq -\alpha V(t, x(t))$
whenever $V(t+s, x(t+s)) \leq qV(t, x(t)) \quad \forall s \in [-r, 0]$.

Then system (6.1) *is exponentially stable with decay rate* $\gamma = \min(\alpha, \frac{\log q}{r})/p$.

This theorems follows from [36] by assuming the absence of impulses. However, its proof is considerably more complicated than the analog for ODEs or Lyapunov-Krasovksii functionals.

Note that the above theorem establishes sufficient, but not necessary conditions for exponential stability. The problem is that often it is not powerful enough to check the Razumikhin condition,

$$qV(t, x(t)) - V(t+s, x(t+s)) \geq 0$$

only on the interval $s \in [-r, 0]$, but we should take into account more of the past observations of $V(t, x(t))$. Sometimes it is therefore helpful to reinterpret problem (6.1) as one in the state space $\mathcal{C}([-\tilde{r}, 0])$ with some $\tilde{r} > r$ and to apply Theorem 6.5 to that problem. Centred around this idea necessary and sufficient Razumikhin-type conditions for discrete-time delay systems are stated in [37] and for exponential stability of continuous-time systems with quadratic Lyapunov functions and slightly different definitions in [38]. In the following we will treat r therefore as a hyperparameter that has to be chosen for the respective problem at hand. Furthermore, since we are interested in autonomous dynamical systems we can omit the explicit time-dependency in V. Moreover, in order to allow quadratic Lyapunov functions we will choose $p = 2$.

6.2 Neural network Lyapunov-Razumikhin function

In this section, we discuss an approach to represent a Lyapunov-Razumikhin function as a neural network. The only difference of Theorem 6.5 compared to its ODE counterpart is that the decay condition,

$$\dot{V}(x(t)) \leq -\alpha V(x(t)),$$

is not always necessary. In particular, as we choose $p = 2$, the Lyapunov-Razumikhin function V should still satisfy,

$$c_1 \|x\|_2^2 \leq V(t,x) \leq c_2 \|x\|_2^2, \tag{6.2}$$

and must not have any local optima except 0. For the construction of V we therefore closely follow [12] and parametrize V via an input-convex neural network (ICNN)[39].

The ICNN g is a neural network defined as follows:

$$\begin{aligned} z_1 &= \sigma_0(W_0 x + b_0) \\ z_{i+1} &= \sigma_i(U_i z_i + W_i x + b_i), \quad i = 1, ..., L-1 \\ g(x) &= z_L \end{aligned}$$

Where the entries of U_i are constrained to be non-negative weights and σ_i are arbitrary convex, monotonically non-decreasing activation functions. This construction insures $x \mapsto g(x)$ to be convex [39] and any convex function can be approximated by such neural networks [40]. As we shall see we will additionally require σ_i to have slope no greater than one in order to satisfy the upper bound in (6.2).

To ensure strict convexity and to make sure that the global minimum is at $x = 0$ the following top layer is chosen,

$$V(x) = \sigma_L(g(x) - g(0)) + \varepsilon \left\|x\right\|_2^2,$$

where σ_L is positive, convex, and non-decreasing function with $\sigma_L(0) = 0$ and ε is a small constant. Since DDE solvers usually require a lot of smoothness and as we want to ensure continuity of the loss derivatives, our choice of σ_L slightly differs from [12]. We use a twice, instead of only once, continuously differentiable smoothed ReLU version defined as follows,

$$\sigma_L(x) = \begin{cases} 0 & , x \leq 0 \\ \dfrac{x^3}{d^2} - \dfrac{x^4}{(2d)^3} & , 0 \leq x \leq d \\ x - \dfrac{d}{2} & , x > d. \end{cases} \tag{6.3}$$

If required also higher order polynomials could be used to ensure continuity of higher order derivatives. This construction ensures that $V(x) = \mathcal{O}(\|x\|_2^2)$ as $x \to 0$ and also $V(x) = \mathcal{O}(\|x\|_2^2)$ as $\|x\|_2 \to \infty$. We can therefore always find constants c_1, c_2 such that condition (6.2) is satisfied.

6.3 Discretization of Razumikhin condition

In order to verify condition (ii) of Theorem 6.5 we need to first check whether there is a $s \in [-r, 0]$ such that,

$$V(x(t+s)) > qV(x(t)), \tag{6.4}$$

and only if there is none then the decay condition must hold. In practice we need to discretize and we therefore only check (6.4) at discrete time steps $s = -\tau_k = -k \cdot \tau$. While this is still sufficient to proof stability, it is possible that we miss a $s \in [-r, 0]$ where (6.4) holds and therefore enforce the decay condition too often.

However, Proposition 6.6 tells us that if the discretized Razumikhin condition holds, τ is small enough, and the solution has not been decaying too fast before, then the continuous condition holds for some $\tilde{q} > q$. Furthermore, $\tilde{q}$ converges quadratically to q as $\tau \to 0$.

Proposition 6.6 *Let K and τ be such that $r_V := K\tau \geq r$ and let $x(\cdot)$ be a solution of $\dot{x}(t) = f(x_t)$ passing through $x_{t_0} \in \mathcal{C}([-r, 0])$. Let $\rho := \max_{s \in [t_0 - r_V, t_0]} \|x(s)\|_2 < \infty$ and assume f to be L_f-Lipschitz with respect to $\|\cdot\|_\infty$. Moreover, let $V : \mathbb{R}^n \to \mathbb{R}_+$ be convex with,*

$$c_1 \|x\|_2^2 \leq V(x) \leq c_2 \|x\|_2^2 ,$$

f and V both be differentiable, and assume that there is a function $g : \mathbb{R}_+ \to \mathbb{R}_+$ with $g(s) \geq 1$ such that along the trajectory it holds for any t and $c > 0$,

$$\max_{s \in [t - r_V, t]} \|x(s)\|_2 \leq g(c) \max_{s \in [t - r_V + c, t + c]} \|x(s)\|_2 . \tag{6.5}$$

Now if,

$$V(x(t_0 - k\tau)) \leq qV(x(t_0)) \quad , \forall k \in \{0, 1, ..., K\}, \tag{6.6}$$

and $8c_1 > (4c_2 - c_1)L_f^2 g(2r)\tau^2$, then it holds for any $s \in [-r_V, 0]$,

$$\begin{aligned} V(x(t_0 + s)) &\leq \tilde{q}(\tau) V(x(t_0)), \\ \text{with } \tilde{q}(\tau) &= \frac{q}{1 - (4c_2 - c_1)L_f^2 g(2r)\tau^2/(8c_1)} = q + \mathcal{O}(\tau^2). \end{aligned} \tag{6.7}$$

Proof We start with bounding the deviation of $x(\cdot)$ from the linear interpolation between the observation points $\{(t_k := t_0 - k\tau, x_k := x(t_0 - k\tau))\}_{k=1}^K$. Lets denote the linear interpolation as $\tilde{x}(\cdot)$. Then by Rolle's Theorem [41] we get the following standard upper bound on the interpolation error $e(t) := \tilde{x}(t) - x(t)$,

$$\|e(t)\|_2 =\leq \frac{\tau^2}{8} \max_{s \in [t_0 - r_V, t_0]} \|\ddot{x}(s)\|_2 , \quad \forall t \in [t_0 - r_V, t_0]. \tag{6.8}$$

Using the Lipschitz continuity of f we get for the operator norm of the differential $|||Df||| \leq L_f$ and $\|f(x_t)\|_2 \leq L_f \|x_t\|_\infty$. Here, in the second inequality we used $f(0) = 0$. Therefore, applying the chain rule and using the above

inequalities (6.8) simplifies to,

$$\begin{aligned}\|e(t)\|_2 =& \leq \frac{\tau^2}{8} \max_{s\in[t_0-r_V,t_0]} \left\| \frac{d}{dt} f(x_t)\big|_{t=s} \right\|_2 = \frac{\tau^2}{8} \max_{s\in[t_0-r_V,t_0]} |||Df(x_s)||| \cdot \|\dot{x}_s\|_\infty \\ &\leq \frac{\tau^2}{8} L_f \max_{s\in[t_0-r_V,t_0]} \max_{\xi\in[s-r,s]} \|f(x_\xi)\|_2 \leq \frac{\tau^2}{8} L_f \max_{s\in[t_0-r_V-r,t_0]} L_f \|x_s\|_\infty \\ &= \frac{L_f^2\tau^2}{8} \max_{s\in[t_0-r_V-2r,t_0]} \|x(s)\|_2 \leq \frac{L_f^2 g(2r)\tau^2}{8} \max_{s\in[t_0-r_V,t_0]} \|x(s)\|_2 \\ &= \frac{L_f^2 g(2r)\rho\tau^2}{8}, \end{aligned} \tag{6.9}$$

for all $t \in [t_0 - r_V, t_0]$.

Now, we proceed with the derivation of (6.7) and assume that (6.6) holds. Further on, we make use of the following claim, which we will prove later.

Claim : $\|\nabla V(x)\|_2 \leq M\rho,\ \forall x \in B_\rho(0, \|\cdot\|_2)$ with $M := (4c_2 - c_1)$ □

In particular, this means that V is $(M\rho)$-Lipschitz in $B_\rho(0, \|\cdot\|_2)$. It then holds for any $s \in [-r, 0]$,

$$\begin{aligned} V(x(t_0+s)) = V(\tilde{x}(t_0+s) + e(t_0+s)) &\overset{(i)}{\leq} V(\tilde{x}(t_0+s)) + M\rho \|e(t_0+s)\|_2 \\ &\overset{(ii)}{\leq} V(\beta x_k + (1-\beta)x_{k+1}) + \frac{ML_f^2 g(2r)\rho^2\tau^2}{8} \\ &\overset{(iii)}{\leq} \beta V(x_k) + (1-\beta)V(x_{k+1}) + \frac{ML_f^2 g(2r)\rho^2\tau^2}{8} \\ &\overset{(iv)}{\leq} qV(x(t_0)) + \frac{ML_f^2 g(2r)\rho^2\tau^2}{8} \end{aligned}$$

Here, in (i) we used that V is Lipschitz and (ii) follows from (6.9) and the fact that $\tilde{x}(t+s)$ is a convex combination of two neighbouring data points. In (iii) we used convexity of V and (iv) follows from the discretized Razumikhin condition (6.6).

In order to continue we define $x_m := \operatorname{argmax}_{s\in[t_0-r_V,t_0]} \|x(s)\|_2$ and $x^* := \operatorname{argmax}_{s\in[t_0-r_V,t_0]} V(x(s))$. It then holds $V(x(t_0+s) \leq V(x^*)$ and,

$$V(x^*) \leq qV(x(t_0)) + \frac{ML_f^2 g(2r)V(x_m)\tau^2}{8c_1} \leq qV(x(t_0)) + \frac{ML_f^2 g(2r)V(x^*)\tau^2}{8c_1}.$$

Therefore, if $8c_1 > ML_f^2 g(2r)\tau^2$, then for all $s \in [t_0 - r, t_0]$,

$$V(x(t_0+s)) \leq V(x^*) \leq \frac{q}{1 - ML_f^2 g(2r)\tau^2/(8c_1)} V(x(t_0)).$$

Noting that $1/(1-a\xi^2) = 1 + a\xi^2 + \mathcal{O}(\xi^4)$ as $\xi \to 0$ it follows that,

$$V(x(t_0+s)) \leq \tilde{q}(\tau)(V(x(t_0)) = q + \mathcal{O}(\tau^2).$$

It only remains to proof the claim. For this purpose consider $x \in B_\rho(0, \|\cdot\|_2)$ and $h \in \mathbb{R}^n$ with $\|h\|_2 = 1$. We then have,

$$c_2 \|x+\rho h\|_2^2 \overset{(i)}{\geq} V(x+\rho h) \overset{(ii)}{\geq} V(x) + \rho \nabla V(x)^\top h \overset{(iii)}{\geq} c_1 \|x\|_2^2 + \rho \nabla V(x)^\top h.$$

Here, in (i) and (iii) we used the definition of V and (ii) follows from convexity of V. Rearranging terms and using the Cauchy–Schwarz inequality we arrive at,

$$\begin{aligned}
\nabla V(x)^\top h &\leq \frac{1}{\rho}\left(c_2 \|x+\rho h\|_2^2 - c_1 \|x\|_2^2\right) \\
&= \frac{1}{\rho}\left((c_2 - c_1)\|x\|_2^2 + c_2\rho^2 \|h\|_2^2 + 2\rho c_2 h^\top x\right) \\
&\leq (4c_2 - c_1)\rho,
\end{aligned}$$

and since h was an arbitrary element of the unit sphere it holds $\|\nabla V(x)\|_2 \leq (4c_2 - c_1)\rho$. □

The assumption (6.5) in Proposition 6.6 was needed to bound the linear interpolation error proportional to $\rho := \max_{s\in[t_0-r_V,t_0]} \|x(s)\|_2$. It tells us that the solution is not allowed to decay too fast in the norm $\max_{s\in[t-r_V,t]} \|x(s)\|_2$. For exponentially decaying oscillations of the form,

$$x(t) = e^{-\gamma t}(a + b\cos(2\pi t/T)),$$

it is well-justified if we choose $r_V \geq T$, since,

$$\begin{aligned}
\max_{s\in[t-r_V,t]} |x(s)| &\leq e^{-\gamma(t-r_V)}(|a|+|b|) \\
e^{-\gamma(t+c)}(|a|+|b|) &\leq \max_{s\in[t-r_V+c,t+c]} |x(s)| \\
\Rightarrow \max_{s\in[t-r_V,t]} |x(s)| &\leq g(c) \max_{s\in[t-r_V+c,t+c]} |x(s)| \\
&\text{with } g(c) = e^{\gamma(c+r_V)}.
\end{aligned}$$

Moreover, the choice $r_V \geq T$ is anyways a good idea, as it ensures that a local maximum of V is contained in the interval where we check the Razumikhin condition.

Combined with Theorem 6.5, Proposition 6.6 tells us that discretization of the Razumikhin condition introduces some conservatism by demanding a higher rate of decay which can however be controlled through the choice of τ.

6.4 Hard stability enforcement

In the last section we have seen how we may construct a neural network Lyapunov-Razumikhin function and how we may discretize the Razumikhin condition. In the rest of this chapter we will discuss different training methods for V and how we can take advantage of V to learn stable dynamics models. In [12] a projection based approach is suggested to jointly learn an ODE dynamics model and a Lyapunov function where the dynamics is guaranteed to be stable.

Specifically, let $\hat{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a "nominal" dynamics function and $V : \mathbb{R}^n \to \mathbb{R}$ a neural network Lyapunov function candidate constructed as discussed in Section 6.2. Then the dynamics,

$$\begin{aligned} f(x) &= \text{Proj}\left(\hat{f}(x), \{f : \nabla V(x)^\top f \leq -\alpha V(x)\}\right) \\ &= \hat{f}(x) - \nabla V(x) \frac{\text{ReLU}\left(\nabla V(x)^\top \hat{f}(x) + \alpha V(x)\right)}{\|\nabla V(x)\|_2^2}, \end{aligned}$$

are guaranteed to be stable. The idea is that we simply project $\hat{f}(x)$ onto the stable half plane $\nabla V(x)^\top f \leq -\alpha V(x)$. Thus, $f(x)$ always satisfies the Lyapunov decay condition and is accordingly exponentially stable.

Making use of the discretized Razumikhin condition we may adapt this to the DDE case in the following way: Since the decay condition only needs to hold if the Razumikhin condition is satisfied, we may extend this to,

$$f(\mathbf{x}_\tau(t)) = \hat{f}(\mathbf{x}_\tau(t)) - \nabla V(x(t)) \frac{\text{ReLU}\left(\nabla V(x(t))^\top \hat{f}(\mathbf{x}_\tau(t)) + \alpha V(x(t))\right)}{\|\nabla V(x(t))\|_2^2} R(\mathbf{x}_\tau(t))$$

$$\text{where } R(\mathbf{x}_\tau(t)) := \prod_{k=1}^{K} \Theta\left(qV(x(t)) - V(x(t - k\tau))\right). \tag{6.10}$$

Here, $\Theta(t) := \mathbb{1}_{\{t \geq 0\}}(t)$ is the heaviside step function. This means we only enforce the decay condition through projection if the discretized Razumikhin condition is true. However, a major problem of this setup is that the resulting DDE function f is no-longer continuous due to the step functions. Furthermore, for DDE solvers we would usually like to have as smooth dynamics as possible, since discontinuities are propagated along the trajectory. Therefore, we may consider the following smoothed $\mathcal{C}^2$ version of the step function,

$$\Theta_d(t) = \begin{cases} 0 & , t \leq -d \\ 6(t+d)^5 - 15(t+d)^4 + 10(t+d)^3 & , -d < t < 0 \\ 1 & , t \geq 0. \end{cases} \tag{6.11}$$

However, while we still project whenever we have to, this comes with the disadvantage that we sometimes alter the value of $\hat{f}(\mathbf{x}_\tau(t))$ when not necessary. Furthermore, we cannot simply replace ReLU by a smoothed version, as we need to ensure that the projection term is not exploding as $x \to 0$.

In first experiments, this approach turned out to be problematic for two reasons. On the one hand, training as well as inference of trained model is computationally very demanding due to the Lyapunov neural network that is incorporated into the forward pass. On the other hand, the projections are introducing $\mathcal{C}^1$ discontinuities which pose difficulties for the DDE solver especially in the backward pass.

6.5 Soft stability enforcement

As we noted in the last section, an obvious drawback of a projection based approach is that the Lyapunov neural network V is also present during inference time which renders the model slow and memory consuming. Furthermore, we still have discontinuous derivatives due to the ReLU. A possible way to resolve this is to incorporate V into the loss function rather than directly into the dynamics. Since we want condition (ii) of Theorem 6.5 to hold we propose for each $\mathbf{x}_\tau(t)$ a loss,

$$\ell(\mathbf{x}_\tau(t)) = \mathrm{ReLU}\left(\nabla V(x(t))^\top f(\mathbf{x}_\tau(t)) + \alpha V(x(t))\right) R(\mathbf{x}_\tau(t)), \tag{6.12}$$

with $R(\mathbf{x}_\tau(t))$ as defined in (6.10). Based on this we propose two schemes to jointly train V and f. On the one hand, we introduce a continuous loss to minimize (6.12) along trajectories and on the other hand we suggest to simply minimize (6.12) on a data set of possibly uncorrelated data points $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$.

6.5.1 Training along trajectory

Since condition (ii) in Theorem 6.5 does only need to hold along those $x_t \in \mathcal{C}([-r, 0])$, that belong to a solution of $\dot{x}(t) = f(x_t)$, it is reasonable to minimize (6.12) along trajectories. We therefore introduce the loss,

$$\begin{aligned} J(\theta, \phi) &= \int_0^T \ell_{\theta,\phi}(\mathbf{x}_\tau(t))dt \\ &= \int_0^T \sigma_d\left(\nabla V_\phi(x(t))^\top f(\mathbf{x}_\tau(t), \theta) + \alpha V_\phi(x(t))\right) R_\phi(\mathbf{x}_\tau(t))dt \\ \text{with } R_\phi(\mathbf{x}_\tau(t)) &:= \prod_{k=1}^{K} \Theta_d\left(qV_\phi(x(t)) - V_\phi(x(t - k\tau))\right). \end{aligned} \tag{6.13}$$

Note, that we again make use of the smoothed step function (6.11). Furthermore, for σ_d we propose to use a shifted version of the smoothed ReLU (6.3),

$$\sigma_d(x) = \begin{cases} 0 & , x \leq -d \\ \frac{(x+d)^3}{d^2} - \frac{(x+d)^4}{(2d)^3} & , -d \leq x \leq 0 \\ x + \frac{d}{2} & , x > 0. \end{cases}$$

Introducing this smoothed version of (6.12) turned out to be crucial for reliable computation of the loss gradients. We suspect the reason for this to lie in the discontinuities which are introduced in the backward pass through the non-smooth heaviside and step function.

The loss (6.13) can either be used to train $f(\cdot, \theta)$ and V_ϕ jointly or to train one of them while fixing the other one. Furthermore, we may also add additional loss terms such as a discrete mean square loss as discussed for NDDEs.

6.5.2 Uncorrelated training

For the uncorrelated training we minimize the loss (6.12) for a given training data set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$. There is no need to smooth the ReLU and the heaviside, as no DDE solver is involved. The data set may consist out of samples from DDE solutions or out of completely uncorrelated data points $\mathbf{x}^{(i)} \in \mathbb{R}^{n(K+1)}$. The later is more conservative, since, with a large enough data set, we enforce condition (ii) to hold not only along DDE solutions $x_t \in \mathcal{C}([-r, 0])$, but for all $\psi \in \mathcal{C}([-r, 0])$. However, in the context of control Lyapunov-Razumikhin functions the same approach is suggested in [42].

An interesting question which we leave open for future work is whether we can achieve provable stability guarantees everywhere in a bounded set $B_\rho(0, \|\cdot\|_\infty) \subset \mathcal{C}([-r, 0])$ when our data set consists of data points that are sampled close enough.

Before we proceed to the experimental results for these soft stability enforcement methods, we quickly discuss ideas to learn a discretized version of a Lyapunov-Krasovksii functional.

6.6 Learning Lyapunov Krasovksii-functionals

A general form of a quadratic Lyapunov-Krasovskii is,

$$V(\psi) = \psi^\top(0) P \psi(0) + 2\psi^\top(0) \int_{-r}^{0} Q(\xi)\psi(\xi) d\xi + \int_{-r}^{0}\int_{-r}^{0} \psi^\top(\xi) R(\xi, \eta)\psi(\eta) d\eta d\xi + \int_{-r}^{0} \psi^\top(\xi) S(\xi)\psi(\xi) d\xi,$$

where $P \in \mathbb{R}^{n\times n}$ is symmetric and $Q(\xi), R(\xi,\eta), S(\xi) \in \mathbb{R}^{n\times n}$ are continuous. A discretization method based on piecewise linear Q, R, S is treated in [18].

Another very simple discrete version of a Lyapunov-Krasovksii functional which fits easier into a neural network based approach would be,

$$V(\psi) = \tilde{V}(\psi(0), \psi(-\tau_1), ..., \psi(-\tau_K)).$$

The upper right derivative is then simplifying to,

$$\dot{V}(x_t) = \partial_0 \tilde{V} f + \sum_{i=1}^{K} \partial_i \tilde{V} \dot{x}(t-\tau_i).$$

For $\tilde{V}$ we could use the same neural network construction as for the Lyapunov-Razumikhin function just with multiple inputs. However, in contrast to the Lyapunov-Razumikhin approach where we only discretized the Razumikhin condition, here we need to discretize the functional itself. This might be harder to justify. An other advantage of Lyapunov-Razumikhin functions is that they immediately yield invariant sets in the space $\mathbb{R}^n$ rather than $C([-r, 0])$. However, especially for the hard stability enforcement method an advantage of this approach would be that no step functions are needed that introduce discontinuities.

6.7 Stabilization with delayed feedback

To demonstrate the applicability of soft stability enforcement as discussed in Section 6.5, we consider the problem of stabilization of an inverted pendulum with delayed feedback. The dynamics of an inverted pendulum with fixed pivot point and delayed input are given by the following differential equation,

$$\dot{x}(t) = \begin{pmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{pmatrix} = \begin{pmatrix} \dot{\varphi}(t) \\ \ddot{\varphi}(t) \end{pmatrix} = \begin{pmatrix} x_2(t) \\ \frac{g}{l}\sin(x_1(t)) - \frac{b}{ml^2}x_2(t) + \frac{1}{ml^2}u(t-\tau) \end{pmatrix}. \tag{6.14}$$

Here, $\varphi(t)$ is the angle between the current and the fully upright position. g indicates the acceleration of gravity, l and m the length and mass of the pendulum, and b is the friction coefficient. Furthermore, $u(t)$ is the torque which is applied at the pivot point. We are interested in stabilizing the unstable equilibrium point $x = (0, 0)^\top$ through the delayed linear feedback,

$$u(t-\tau) = \pi(x(t-\tau)) = k_1 x_1(t-\tau) + k_2 x_2(t-\tau).$$

6.7.1 Related work

In the case of no delay, we could linearize (6.14) around $(0, 0)$ and stabilize the system by a Linear Quadratic Regulator (LQR). Moreover, for very small

delays the LQR feedback remains stable up to some critical delay. For the inverted pendelum this is analyzed in [43]. Beyond that critical delay we can take advantage of the fact that the closed loop system is a DDE which can be analyzed by DDE methods. The authors of [44] linearize the DDE and choose the feedback such that the roots of the characteristic polynomial all have negative real parts. Using this method the system can be stabilized in cases where LQR feedback is failing. However, since we are linearizing the system around $(0, 0)$ we can only make local statements about stability. In contrast to this approach we will make use of the Lyapunov-Razumikhin-based soft stability enforcement methods discussed in Section 6.5.

Note that for large delays also DDE methods fail and we would have to make use of infinite-dimensional feedback of the form $u(t) = \pi(x(t), u_t)$, where $u_t \in \mathcal{C}([-r, 0])$ is the history of past inputs. The so-called predictor-feedback methods belong into this category. As discussed in [45] this leads to ODE-PDE cascades for which the standard Razumikhin or Krasovskii methods are not applicable. Moreover, in contrast to a DDE based approach those predictor-feedback methods can usually not deal with state delays in the open loop system.

6.7.2 Training setup

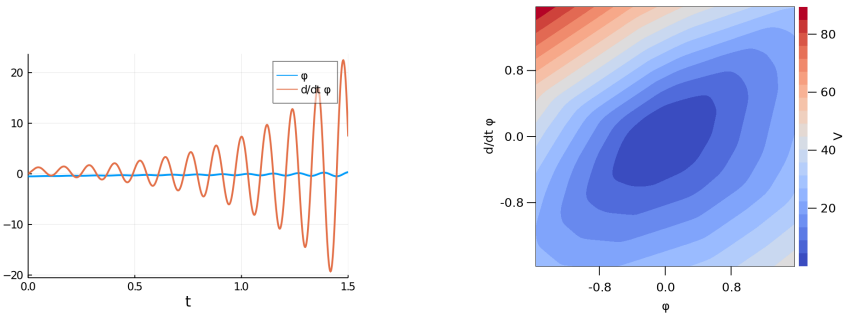
In the following, we compare the two soft stability enforcement methods discussed in Sections 6.5.1 and 6.5.2 by means of their ability to stabilize system (6.14). Besides the neural network Lyapunov function, we also consider a quadratic Lyapunov function of the form,

$$V(x) = x^\top P x + \varepsilon \|x\|_2^2 .$$

Here, we choose $P := W^\top W$ for some weight matrix $W \in \mathbb{R}^{n \times n}$ to ensure positive semi-definiteness of P. This in total leads to four different configurations of loss and Lyapunov function.

Our goal is to jointly learn a Lyapunov-Razumikhin function V_ϕ and the linear feedback policy $K := (k_1, k_2)$. As reported in [11], it is difficult to jointly learn a neural network Lyapunov function and a stabilizing feedback from random initialization of parameters. Similar to [11], we therefore initialize the feedback with LQR parameters and the parameters of the neural network Lyapunov function randomly. A realization of such a random initialization of V_ϕ is illustrated in Figure 6.1b. For the quadratic Lyapuonv function we start with $P = I$.

In Table 6.1 the pendulum parameters are summarized. Calculating the corresponding LQR feedback yields $K \approx (1.749,\ 1.031)$. Furthermore, we use the loss parameters listed in Table 6.2 and the Lyapunov function parameters from Table 6.3. Note, that τ_V, K_V correspond to the number and distance



(a) LQR feedback (b) Initial NN V

Figure 6.1: (a) shows the unstable LQR feedback and (b) illustrates a realization of the randomly initialized neural network Lyapunov function.

between delays for which the Razumikhin condition is checked and d_L, d_V are the smoothing parameters for the loss and the Lyapunov function activation. Moreover, according to Theorem 6.5 we choose $\alpha = \log(q)/r_V$.

l	m	b	g	τ
0.3	0.2	0	9.81	0.03

Table 6.1: Pendulum parameters

	τ_V	K_V	r_V	α	q	T	d_L	data size	batch size
along trajectory	0.04	5	0.2	0.1	1.0202	1.5	0.5	-	-
uncorrelated	0.04	5	0.2	0.1	1.0202	-	-	64000	64

Table 6.2: Loss parameters and training parameters

	hidden layers	ε	d_V
neural network	(16,16)	0.001	0.1
quadratic	-	0.001	-

Table 6.3: Lyapunov function parameters

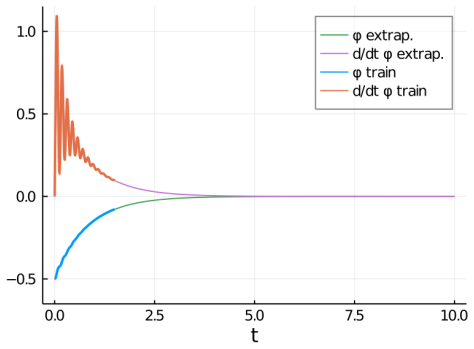
For the training along trajectories we choose $T = 1.5$ and decide to train with a single trajectory originating in $(\varphi, \dot{\varphi}) = (-0.5, 0)$. Furthermore, we choose the initial history such that the state follows the pendulum law (6.14) with zero input for $t < 0$. As depicted in Figure 6.1a the LQR feedback is unstable

in this case. As our usual choice of Runge-Kutta method for the method of steps, *Tsit5*, was leading to inaccurate gradient estimates in this case, we choose to use a *RK4* solver with residual control.

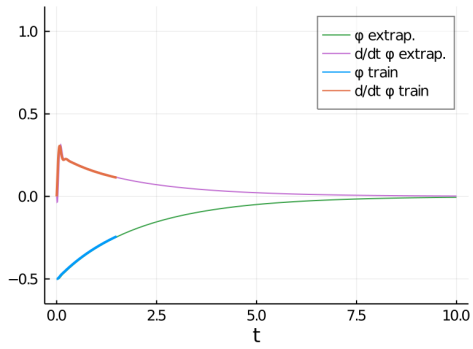
For the uncorrelated training we generate a data set of 64'000 independently drawn samples $\mathbf{x} \in \mathbb{R}^{2(K_V+2)}$ with $\mathbf{x}_i \sim \mathcal{U}([-\pi/2, \pi/2])$. Moreover, we use a batch size of 64 and train over 3 episodes.

6.7.3 Results

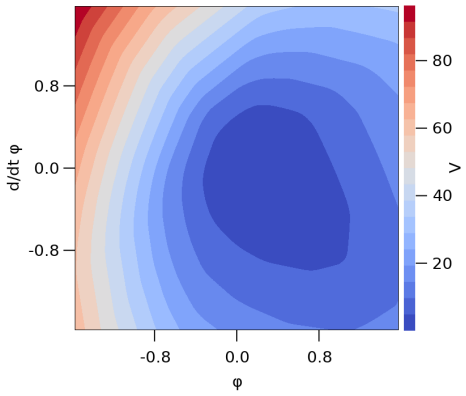
In the following we compare the resulting feedback and Lyapunov functions for the four different training settings. We make use of the following abbreviations: Training along trajectory with neural network Lyapunov function (TTNN) and quadratic Lyapunov function (TTQ). Moreover, uncorrelated training with neural network Lyapunov function (UTNN) and with quadratic Lyapunov function (UTQ).



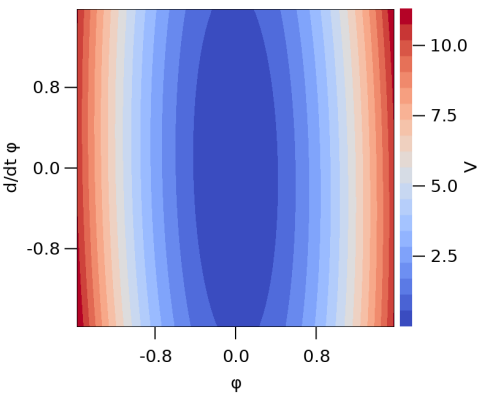
(a) Prediction TTNN



(b) Prediction TTQ



(c) V TTNN



(d) V TTQ

Figure 6.2: (a) and (b) illustrate the resulting feedback for TTNN and TTQ. The trained Lyapunov function candidates are illustrated in (c) and (d).

Figure 6.2 illustrates the resulting feedback and V after training. It is evident that both TTNN and TTQ lead to a stabilizing feedback along the training trajectory. However, the Lyapunov function candidates V differ considerably. Whereas V is symmetric for TTQ, this is not the case for TTNN. This is to be expected, as on the one hand, the quadratic Lyapunov function is symmetric by construction and on the other hand the training trajectory is fully contained within the upper-left quadrant and there are accordingly no incentives for the neural network Lyapunov function to look symmetric. However, as the problem itself is symmetric and we only train along a single training trajectory, we would expect bad generalization performance of TTNN in terms of the continuous loss (6.13). In order to investigate this, we look at generalization from 8 initial conditions distributed over a circle according to $x(0) = (\cos(2\pi l/8), \sin(2\pi l/8))_{l=1}^{8}$. Where for the initial histories we again assume zero inputs before $t = 0$. The resulting average test losses are listed in Table 6.4 together with the training losses. Here, we also listed the losses without smoothing, since smoothing has only been necessary for the gradient calculations. We see that only the neural network achieves zero training loss. We suppose this is due to the additional flexibility compared to the quadratic Lyapunov function. However, once we get rid of the smoothed ReLU and step function also TTQ shows zero loss on the training trajectory. Moreover, we clearly see that, as expected, TTNN is not generalizing to new initial conditions.

	train loss	train loss - no smoothing	test loss - no smoothing
TTNN	0	0	3.1502
TTQ	0.0196	0	0

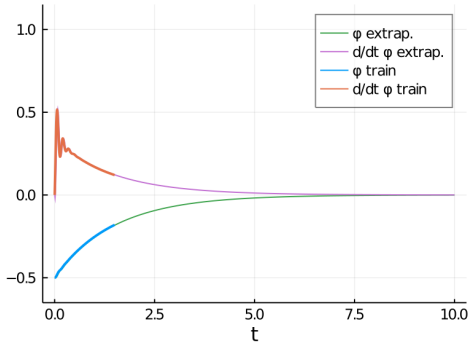
Table 6.4: Train and test loss for training along a single trajectory.

In the following, we proceed with the results for uncorrelated training. In this setting we expect a more symmetric V for UTNN compared to TTNN, as our training data is more uniformly distributed across the phase space. Comparing the resulting V illustrated in Figure 6.3c to Figure 6.2c, we see that this is indeed the case. Moreover, although we did not train along any trajectory here, the feedback is stable along the training trajectory of TTNN and TTQ. From Table 6.5 we see that for both models the training loss is not yet zero on all training samples. One possible explanation for this could be that we are being too conservative, since we enforce condition (ii) of Theorem 6.5 for all $\mathbf{x}$ and not only for those $\mathbf{x}_\tau(t)$ belonging to a solution in the sense $\mathbf{x}_\tau(t) = (x(t), x(t-\tau), x(t-\tau_V), ..., x(t-K_V\tau_V))$. Another very likely possibility is that we should consider to train the model over more than 3 episodes. This is supported by the fact that the neural network Lyapunov function shows a higher training error compared to the quadratic version. However,

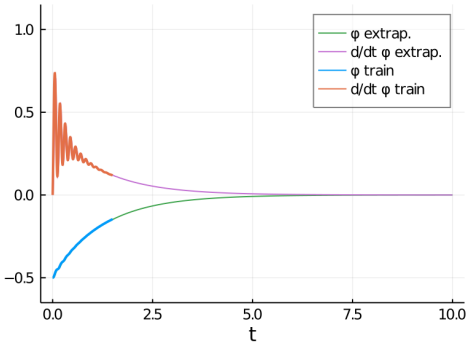
when we evaluate the models on the 8 test trajectories described above, then both UTNN and UTQ achieve zero test loss (see also Table 6.5).

	Avg. uncorrelated train loss	test loss - no smoothing
UTNN	0.1464	0
UTQ	2.28e-10	0

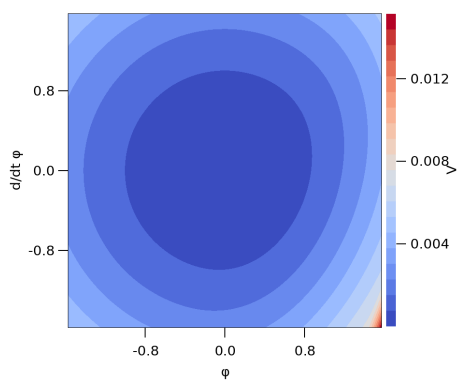
Table 6.5: Train and test loss for uncorrelated training.



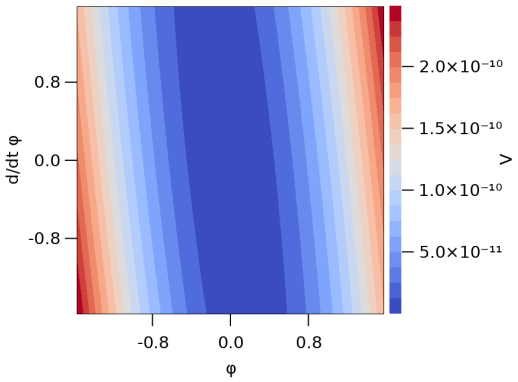
(a) Prediction UTNN



(b) Prediction UTQ



(c) V UTNN



(d) V UTQ

Figure 6.3: (a) and (b) illustrate the resulting feedback for UTNN and UTQ. The trained Lyapunov function candidates are illustrated in (c) and (d).

6.8 Conclusion

To conclude, we discussed several approaches to extend neural network based Lyapunov methods to a time-delayed setting. First of all, we adapted the neural network Lyapunov function from [12] to our needs and suggest to use it as a Lyapunov-Razumikhin function. Furthermore, Proposition 6.6 shows that, we do not loose a lot when we discretize the Razumikhin condition. We then explored hard and soft stability enforcement methods and shortly discussed a possible approach to learn a discretized Lyapunov-Krasovskii function. Finally, the applicability of the soft stability enforcement methods is showcased for the problem of stabilizing an inverted pendulum with delayed feedback control.

Chapter 7

Summary and future work

7.1 Summary

In the course of this thesis we develop neural DDEs as an extension of neural ODEs that can model non-Markovian effects. Only very recently and thus completely independently from our work a similar model architecture has been proposed in [46]. We derive formulas for the loss gradients based on continuous sensitivity analysis and experimentally compare the performance to backward-mode automatic differentiation through the DDE solvers. In contrast to the non-delayed case, we found the automatic differentiation to be more efficient than the adjoint sensitivity method. Similarly to augmented neural ODEs we show that neural DDEs are universal function approximators. Moreover, we proof that truncated Fourier series can be represented as solutions of linear DDEs and note the connection to delay embeddings of general attractors of dynamical systems.

In a first series of experiments with a partially observed oscillator we show that the initialization of augmented states in augmented neural ODEs is a major problem for modelling partially observed systems and that neural DDEs greatly outperform augmented neural ODEs on this task. Moreover, we show that despite their vast expressivity neural DDEs do not overfit to noisy observations and that they generalize over time and also across initial conditions if the respective diversity is provided by the training data. Further on, we showcase the expressive power of NDDEs on more complex data.

In a second step, exploratory results on neural network based stability analysis of time delay systems are provided. In particular, we focus on approaches to learn neural network Lyapunov-Razumikhin functions. On the one hand, we adapt a projection based method from ODEs, where the neural network is directly incorporated into the forward pass. However, additional discontinuities as well as a much more complex forward pass due to the Lyapunov neural network turn out to be problematic. On the other hand, we consider

soft stability enforcement, where we incorporate the Lyapunov neural network into a loss function. This comes with the advantages of a faster forward pass and of more explicit training of the Lyapunov function. While this framework may be applied to enforce stability of a neural DDE, we showcase its applicability to the problem of stabilizing an inverted pendulum with delayed linear feedback.

7.2 Future work

Even though we have proven the expressive power of neural DDEs and their applicability to model partially observed dynamics of a harmonical oscillator, experiments on more complex, higher dimensional data should be performed. Here, a next step would be to apply the model to the real world robot data set provided by [47]. Moreover, it would be interesting to further investigate the connection to delay embeddings and to experimentally evaluate the applicability of neural DDEs to predict the dynamics of a chaotic attractor from partial observations. Further on, criteria for the choice of delays in delay embeddings such as average mutual information [48] may be used to choose the delays of the neural DDE.

Following the ideas we presented for machine learning based stability analysis of time delay systems a lot of work remains to be done. Specifically, it would be very interesting to see whether we can get provable stability guarantees for uncorrelated training of the neural network Lyapunov-Razumikhin function. Alongside this one could also investigate the applicability of a falsifier-based approach as it is employed in [11]. Another possibility would be to consider Razumikhin's theorem for time-discrete delay systems, as it is suggested in [37], which would simplify analysis considerably. Moreover, one could try more approaches for soft stability enforcement, such as a discrete observation loss along trajectories or for uncorrelated training we could incorporate additional knowledge of solutions, such as Lipschitz continuity, into the data set. Furthermore, also here a next step would be to consider stabilization of more complex systems or to combine the approach with neural DDEs.

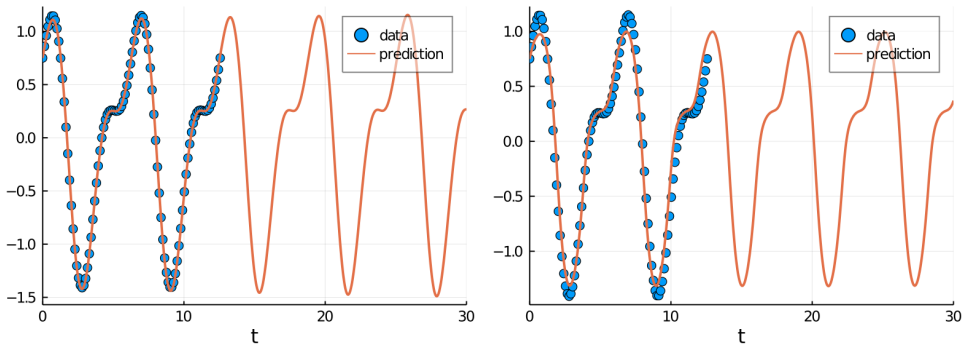
Yet another interesting direction would be optimal or model predictive control of an NDDE model. On the one hand, we have already derived gradient formulas for non-linear optimal control which could easily be extended to delayed inputs. On the other hand, we may linearize the model and make use of the vast amount of literature on optimal control of linear time delay systems.

Appendix A

Appendix

A.1 Fourier series experiments

Here we provide the figures for the two models with integer delays from Section 5.4.1. It is plain to see that also those models can fit the data, although we see that in the two delay case the fit is a bit worse than for the other models.



(a) $\tau = 1$, $K = 4$ (b) $\tau = 1$, $K = 2$

Figure A.1: Training data and prediction for fourier series with two non-zero coefficients.

Bibliography

[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," June 2018.

[2] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural odes," Apr. 2019.

[3] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman, "How to train your neural ode: the world of jacobian and kinetic regularization," Feb. 2020.

[4] H. Zhang, X. Gao, J. Unterman, and T. Arodz, "Approximation capabilities of neural odes and invertible residual networks," July 2019.

[5] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama, "Dissecting neural odes," Feb. 2020.

[6] A. Gholami, K. Keutzer, and G. Biros, "Anode: Unconditionally accurate memory-efficient gradients for neural odes," Feb. 2019.

[7] T. Daulbaev, A. Katrutsa, L. Markeeva, J. Gusak, A. Cichocki, and I. Oseledets, "Interpolation technique to speed up gradients propagation in neural odes," Mar. 2020.

[8] C. Rackauckas and Q. Nie, "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia," *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[9] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, "Universal differential equations for scientific machine learning," Jan. 2020.

[10] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," Aug. 2018.

[11] Y.-C. Chang, N. Roohi, and S. Gao, "Neural lyapunov control," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, pp. 3245–3254, Curran Associates, Inc., 2019.

[12] G. Manek and J. Z. Kolter, "Learning stable deep dynamics models," Jan. 2020.

[13] J. K. Hale and S. M. V. Lunel, *Introduction to Functional Differential Equations*. Springer New York, 1993.

[14] H. ZivariPiran and W. H. Enright, "An efficient unified approach for the numerical solution of delay differential equations," *Numerical Algorithms*, vol. 53, pp. 397–417, sep 2009.

[15] E. Fridman, *Introduction to time-delay systems : analysis and control*. Cham: Birkhauser, 2014.

[16] Niculescu, *Advances in Time-Delay Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[17] R. Bellman, *Differential-difference equations*. New York: Academic Press, 1963.

[18] K. V. L. C. J. Gu, Keqin, *Stability of Time-Delay Systems*. Boston, MA: Birkhauser Boston Imprint Birkhauser, 2003.

[19] O. Diekmann, *Delay Equations : Functional-, Complex-, and Nonlinear Analysis*. New York, NY: Springer New York, 1995.

[20] D. Liberzon, *Calculus of Variations and Optimal Control Theory*. Princeton University Press, Dec. 2011.

[21] S. CHEN, S. A. BILLINGS, and P. M. GRANT, "Non-linear system identification using neural networks," *International Journal of Control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[22] D. G. Luenberger, *Optimization by Vector Space Methods*. Wiley-Interscience, Jan. 1997.

[23] K. L. Teo, *A unified computational approach to optimal control problems*. Harlow, Essex, England New York: Longman Scientific and Technical Wiley, 1991.

[24] J. Calver and W. Enright, "Numerical methods for computing sensitivities for ODEs and DDEs," *Numerical Algorithms*, vol. 74, pp. 1101–1117, sep 2016.

[25] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, dec 1989.

[26] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980* (D. Rand and L.-S. Young, eds.), (Berlin, Heidelberg), pp. 366–381, Springer Berlin Heidelberg, 1981.

[27] I. P. Neil Dhir, Adam R Kosiorek, "Bayesian delay embeddings for dynamical systems," *NIPS Time Series Workshop 2017*, 2017.

[28] T. Sauer, J. A. Yorke, and M. Casdagli, "Embedology," *Journal of Statistical Physics*, vol. 65, pp. 579–616, nov 1991.

[29] J. C. Robinson, "A topological delay embedding theorem for infinite-dimensional dynamical systems," *Nonlinearity*, vol. 18, pp. 2135–2143, jul 2005.

[30] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," Oct. 2017.

[31] H. Kantz, *Nonlinear time series analysis*. Cambridge: Cambridge University Press, 2003.

[32] A. M. LYAPUNOV, "The general problem of the stability of motion," *International Journal of Control*, vol. 55, pp. 531–534, mar 1992.

[33] A. E. Scheidegger, "Stability of motion, by n. n. krasovskii. translated from the (1959) russian ed. by j. l. brenner. stanford university press, 1963. $6. u.s.," *Canadian Mathematical Bulletin*, vol. 7, pp. 151–151, mar 1964.

[34] B. Razumikhin, "On the stability of systems with a delay (russian)," *Prikladnaya Matematika i Mekhanika, vol. 20, pp. 500-512*, 1956.

[35] J. Hale, *Functional differential equations*. New York: Springer-Verlag, 1971.

[36] B. Liu, X. Liu, K. L. Teo, and Q. Wang, "Razumikhin-type theorems on exponential stability of impulsive delay systems," *IMA Journal of Applied Mathematics*, vol. 71, pp. 47–61, 02 2006.

[37] R. H. Gielen, M. Lazar, and S. V. Rakovic, "Necessary and sufficient razumikhin-type conditions for stability of delay difference equations," *IEEE Transactions on Automatic Control*, vol. 58, pp. 2637–2642, oct 2013.

[38] S. E. Grossman and J. A. Yorke, "Asymptotic behavior and exponential stability criteria for differential delay equations," *Journal of Differential Equations*, vol. 12, pp. 236–255, sep 1972.

[39] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," *CoRR*, vol. abs/1609.07152, 2016.

[40] Y. Chen, Y. Shi, and B. Zhang, "Optimal control via neural networks: A convex approach," May 2018.

[41] G. M. Phillips, *Interpolation and Approximation by Polynomials*. Springer New York, 2003.

[42] M. Jankovic, "Control lyapunov-razumikhin functions and robust stabilization of time delay systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 7, pp. 1048–1060, 2001.

[43] M. Landry, S. A. Campbell, K. Morris, and C. O. Aguilar, "Dynamics of an inverted pendulum with delayed feedback control," *SIAM Journal on Applied Dynamical Systems*, vol. 4, pp. 333–351, jan 2005.

[44] J. Milton, J. L. Cabrera, T. Ohira, S. Tajima, Y. Tonosaki, C. W. Eurich, and S. A. Campbell, "The time-delayed inverted pendulum: Implications for human balance control," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, p. 026110, jun 2009.

[45] M. Krstic, *Delay Compensation for Nonlinear, Adaptive, and PDE Systems*. Birkhäuser Boston, 2009.

[46] Q. Zhu, Y. Guo, and W. Lin, "Neural delay differential equations," in *International Conference on Learning Representations*, 2021.

[47] D. Agudelo-Espana, A. Zadaianchuk, P. Wenk, A. Garg, J. Akpo, F. Grimminger, J. Viereck, M. Naveau, L. Righetti, G. Martius, A. Krause, B. Scholkopf, S. Bauer, and M. Wuthrich, "A real-robot dataset for assessing transferability of learned dynamics models," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, may 2020.

[48] S. Wallot and D. Mønster, "Calculation of average mutual information (AMI) and false-nearest neighbors (FNN) for the estimation of embedding parameters of multidimensional time series in matlab," *Frontiers in Psychology*, vol. 9, sep 2018.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Delayed deep learning for continuous-time dynamical systems

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Schlaginhaufen	Andreas

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Frauenfeld, 29.01.2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.