

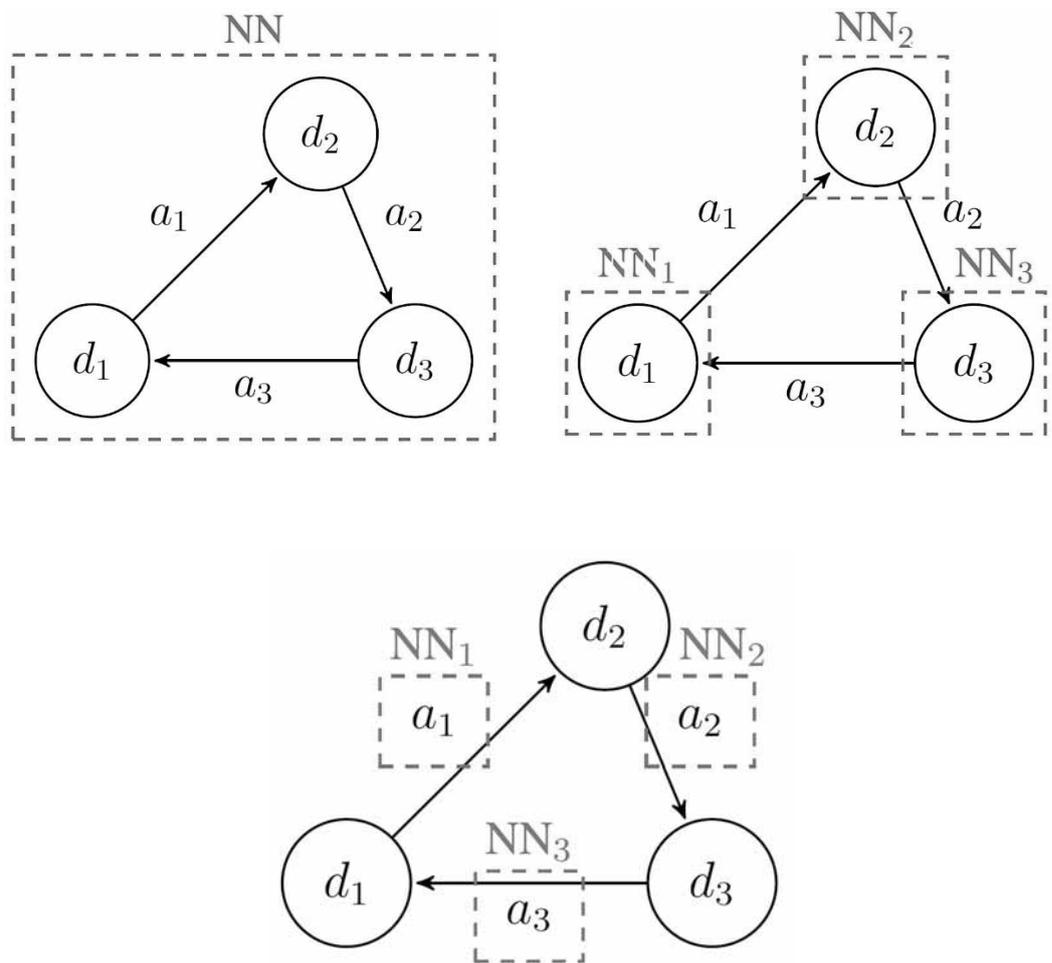


FBS  
34

Fortschrittsberichte Simulation  
Advances in Simulation

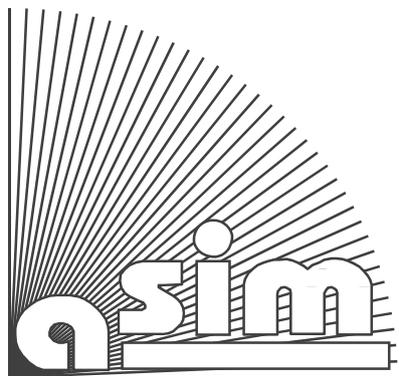
# A Framework Including Artificial Neural Networks in Modelling Hybrid Dynamical Systems

*Stefanie Nadine Winkler*



ISBN print 978-3-903311-12-1 ISBN ebook 978-3-903347-34-2 DOI 10.11128/fbs.34







# *FBS Fortschrittsberichte Simulation*

## *Advances in Simulation*

Herausgegeben von ASIM - Arbeitsgemeinschaft Simulation, Fachausschuss der GI – Gesellschaft für Informatik - im Fachbereich ILW – Informatik in den Lebenswissenschaften  
Published by ASIM – German Simulation Society, Section of GI – German Society for Informatics – in Division ILW – Informatics in Life Sciences

### *ASIM FBS 34*

Stefanie Nadine Winkler

**A Framework Including Artificial  
Neural Networks in Modelling  
Hybrid Dynamical Systems**

# FBS - Fortschrittsberichte Simulation / Advances in Simulation

Published on behalf of **ASIM** – German Simulation Society, → [www.asim-gi.org](http://www.asim-gi.org)

ASIM is a section of of **GI** – German Society for Informatics, → [www.gi.de](http://www.gi.de), in division **ILW** – Informatics in the Life Sciences, → [fb-ilw.gi.de](http://fb-ilw.gi.de)

Herausgegeben von **ASIM** - Arbeitsgemeinschaft Simulation, → [www.asim-gi.org](http://www.asim-gi.org)

ASIM ist ein Fachausschuss der **GI** - Gesellschaft für Informatik, → [www.gi.de](http://www.gi.de), im Fachbereich **ILW** – Informatik in den Lebenswissenschaften, → [fb-ilw.gi.de](http://fb-ilw.gi.de)

## Series Editors

*Prof. Dr.-Ing. Th. Pawletta (ASIM), HS Wismar, Thorsten.pawletta@hs-wismar.de*

*Prof. Dr. D. Murray-Smith (EUROSIM / ASIM), Univ. Glasgow,*

*David.Murray-Smith@glasgow.ac.uk*

*Prof. Dr. F. Breitenecker (ARGESIM / ASIM), TU Wien, Felix.Breitenecker@tuwien.ac.at*

**Title:** A Framework Including Artificial Neural Networks in  
Modelling Hybrid Dynamical Systems

**Author:** Stefanie Nadine Winkler, [stefanie.winkler@tuwien.ac.at](mailto:stefanie.winkler@tuwien.ac.at)

**FBS Volume:** 34

**Type:** Dissertation

**Supervisor / Reviewer:**

Gašper Mušič, Univ. Ljubljana - Fac. ET, Ljubljana, Slovenia

Thorsten Pawletta, Univ. Applied Sciences, Wismar, Germany

Felix Breitenecker, TU Wien, Austria

**ISBN print:** 978-3-903311-12-1, TU-Verlag, Vienna, 2020;

Print-on-Demand, [www.tuverlag.at](http://www.tuverlag.at)

**ISBN ebook:** 978-3-903347-34-2, ARGESIM Publisher Vienna, 2020;

[www.argesim.org](http://www.argesim.org)

**DOI:** 10.11128/fbs.34

**Pages:** 112 + vi



TECHNISCHE  
UNIVERSITÄT  
WIEN

## DISSERTATION

# A Framework Including Artificial Neural Networks in Modelling Hybrid Dynamical Systems

Ausgeführt zum Zwecke der Erlangung des akademischen Grades einer  
Doktorin der technischen Wissenschaften unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker

und

Dipl.-Ing. Dipl.-Ing. Dr.techn. Andreas Körner

E101 - Institut für Analysis und Scientific Computing

eingereicht an der Technischen Universität Wien

Fakultät für Mathematik und Geoinformation

von

Stefanie Nadine Winkler

0625659

Gernotgasse 5/10

1150 Wien

Wien, Januar 2020



## Kurzfassung

Ziel der Arbeit ist unterschiedliche Einsatzszenarien von neuronalen Feed-Forward Netzwerken in der hybriden Modellierung zu untersuchen. Im Speziellen konzentriert sich die Arbeit auf hybride dynamische Systeme. Dabei handelt es sich um kontinuierliche Vorgänge, welche durch diskrete Ereignisse Änderung im Verhalten aufzeigen. Dementsprechend basiert die Modellierung solcher Systeme auf einer Kombination aus diskreten und kontinuierlichen Methoden. Derzeitige Standards der hybriden Modellbildung werden überblicksmäßig zusammengefasst.

In der Softwareentwicklung kommen vermehrt Machine Learning Algorithmen zur Anwendung. Ein signifikanter Anteil der dabei verwendeten Methoden sind künstliche neuronale Netze, welche vor allem im Bereich der Bild- und Datenverarbeitung eingesetzt werden. Der grundlegende Aufbau sowie unterschiedliche gängige Strukturen neuronaler Netze werden eingeführt.

Aufbauend auf die Gebiete Machine Learning und hybride dynamische Systeme, wird ein Framework vorgestellt, welches es ermöglicht, einzelne Elemente hybrider Modelle durch neuronale Netzwerke zu ersetzen. Drei unterschiedliche Anwendungsszenarien werden dabei untersucht: Die Approximation des dynamischen Verhaltens, die Vorhersage der diskreten Prozesse sowie der Ersatz des gesamten hybriden Systems durch neuronale Netze.

Das Framework wird eingesetzt, um die drei definierten Szenarien anhand von zwei Beispielen auf Machbarkeit zu untersuchen. Es wird sich zeigen, dass die Approximation einzelner lokaler dynamischer Prozesse mittels speziellen neuronalen Netzwerken möglich ist. Auch eine Approximation des gesamten hybriden Systems durch ein neuronales Netz kann unter speziellen Voraussetzungen zum Ziel führen. Für eine gesamte Zeitreihe des Systemzustandes werden bereits etablierte Methoden mit Hilfe des Frameworks charakterisiert. Im Bezug auf diskrete Ereignisse in hybriden Systemen kann gezeigt werden, dass sich neuronale Netze für die Klassifikation des Systemzustandes, hinsichtlich des Auftretens eines Events, nicht eignen. Abseits von Anwendungen in der Modellierung hybrider Systeme kann das Framework eingesetzt werden, um Feed-Forward Netzwerke zu charakterisieren.

## Abstract

The aim of this thesis is to investigate different application scenarios of feed-forward networks in hybrid modelling. The term *hybrid* describes the approach to combine different methods. In particular, this work focuses on hybrid dynamical systems. These describe continuous processes, which experience changes in behaviour by discrete events. Accordingly, the modelling of such systems is based on a combination of discrete and continuous methods. Current standards of hybrid modelling are summarised.

Machine learning algorithms in software development are increasingly used. A significant amount of methods used in this context are artificial neural networks. Such networks are mainly used in the field of image and data processing, but in recent years they have been increasingly applied in modelling and simulation of physical systems. The basic design as well as common structures of neural networks are introduced.

Based on these two areas a framework is presented, which allows to replace certain elements of hybrid models by neural networks. Three different application scenarios will be examined: The approximation of the local dynamic behaviour, the prediction of the discrete processes and the replacement of an entire hybrid system applying neural networks. The defined formalism standardises the use of feed-forward networks in hybrid modelling and enables an analysis of different network structures.

The framework is used to examine the feasibility of the three introduced scenarios on the basis of two examples. An evaluation of the results suggests that the approximation of individual local dynamic processes is feasible. Specific assumptions enable a successful approximation of the basic behaviour of the hybrid system. For a complete time series of the system's state, already established processes can be characterised using the introduced framework. With regard to the discrete events in hybrid systems, it is shown that neural networks are not suitable to classify the system states with regard to the occurrence of an event.

Apart from applications in the modelling of hybrid systems, the framework can be used to characterize feed-forward networks. The modular design of the structure makes it possible to replace individual elements to describe different network structures. An interdisciplinary exchange can thus be supported.

## Acknowledgement

*I have no special talents.  
I am only passionately curious.*

Albert Einstein

The following acknowledgements are written in my native language.

Zuallererst möchte ich mich herzlich bei meinem Betreuer Felix Breiteneker bedanken. Er hat mich vor vielen Jahren in seine Forschungsgruppe aufgenommen und mich seither auf meinem akademischen Werdegang begleitet. Seine Unterstützung und Anregungen haben einen großen Teil zu meiner wissenschaftlichen aber auch persönlichen Entwicklung beigetragen. Diverse außeruniversitäre Aktivitäten und kulinarische Entdeckungen haben meinen Horizont zusätzlich erweitert.

Im selbigen Maße möchte ich mich bei Andreas Körner, meinem Mitbetreuer und Arbeitskollegen, bedanken. Ohne seine Mithilfe und Freundschaft wäre ich möglicherweise nicht geblieben um das Doktorat abzuschließen. Vorallem aber im letzten Dissertationsjahr bin ich für seine Unterstützung und seinen Beistand sehr dankbar.

Besonderen Dank möchte ich Martin und Lisi zukommen lassen, die mir bei fehlender Motivation und Intuition zur Seite gestanden sind. Danke für die hilfreichen Tipps, Anregungen und Gesprächstherapien. Vielen Dank an Franziska, Ruth und Irene, die sich einerseits mit meinen eigenwilligen Formulierungen als auch mit mir als Person über Jahre auseinandergesetzt haben und für den einen oder anderen Lachkrampf gesorgt haben. Astrid, dir danke ich für unsere Mittagspausen-Gespräche und die immer wieder neuen Perspektiven.

Meinen Kolleginnen und Kollegen aus dem Universitätsalltag und ARGESIM Kreis möchte ich hiermit ebenfalls meinen Dank aussprechen. Ich habe die studentische Zeit sehr genossen. Meinem Freundeskreis, ja hier gibt es wohl leichte Überschneidungen, bin ich für die jahrelange Treue und unterhaltsamen Stunden dankbar.

Meinen Eltern danke ich dafür, dass sie mir ein Studium ermöglicht haben. Ihre Unterstützung, ihr Zuspruch und manchmal lästige Nachfrage, war meist förderlich. Meinem Bruder danke ich für sein stetiges Interesse und die entspannten und aufmunternden Lunchbreaks. Eure zukünftige Dr. Winkler :)

Aber vor allem ein herzliches Dankeschön an Matthias, meinem Freund und Partner in Crime. Danke, dass du mich immer unterstützt und meine Launen und Quirx immer noch erträgst. Du hast das Talent mich wieder zurück auf den Boden der Tatsachen zu bringen und meinen Fokus auf das Wesentliche zulenken. Ohne deine Unterstützung hätte ich es nicht geschafft. DANKE!

*Es war sehr schön.  
Es hat mich sehr gefreut.*

Franz Joseph I.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Modelling and Simulation . . . . .	2
1.2	Related Work . . . . .	9
1.3	Research Objective and Structure of the Thesis . . . . .	11
<b>2</b>	<b>Hybrid Dynamical Systems</b>	<b>15</b>
2.1	State Space Representation . . . . .	16
2.2	Modelling of Hybrid Dynamical Systems . . . . .	19
2.3	Simulation of Hybrid Dynamical Systems . . . . .	27
<b>3</b>	<b>Artificial Neural Networks</b>	<b>33</b>
3.1	Machine Learning Overview . . . . .	33
3.2	Structure of Artificial Neural Networks . . . . .	34
3.3	Training of Neural Networks . . . . .	43
<b>4</b>	<b>Definition of the Modelling Framework</b>	<b>49</b>
4.1	Framework Overview . . . . .	49
4.2	Artificial Hybrid Model . . . . .	51
4.3	Artificial Hybrid Dynamics . . . . .	53
4.4	Artificial Hybrid Events . . . . .	55

<b>5 Framework Application</b>	<b>59</b>
5.1 Definition of Application Examples . . . . .	59
5.2 Artificial Hybrid Model Application . . . . .	65
5.3 Artificial Hybrid Dynamics Application . . . . .	74
5.4 Artificial Hybrid Event Application . . . . .	83
<b>6 Discussion</b>	<b>87</b>
<b>7 Conclusion</b>	<b>91</b>
<b>A Appendix</b>	<b>93</b>
A.1 Switched Affine Systems . . . . .	93
A.2 Mixed Logical Dynamical System . . . . .	95
A.3 Framework Application - Neural Network . . . . .	96
<b>List of Figures</b>	<b>99</b>
<b>List of Tables</b>	<b>101</b>
<b>Bibliography</b>	<b>103</b>

# 1

## Introduction

---

Hybrid systems are used in versatile fields of application and embrace a wide range of different techniques and methods to solve complex problems. These systems are commonly known from system control (Seatzu et al., 2006), the automotive industry (Lian et al., 2017; Lu et al., 2016) as well as logistics and manufacturing technology (Choy et al., 2008). Models of these hybrid systems are descriptions combining particular modelling approaches (Antsaklis, 2000). In particular, this work focuses on hybrid dynamical systems. These describe continuous processes, which experience changes in behaviour by discrete events. Accordingly, the modelling of such systems is based on a combination of discrete and continuous methods.

In the process of modelling and simulation of hybrid systems, the physical structure of the underlying model is often defined but the exact parameter values might be unknown. In some cases even the mathematical structure of the model can be unidentified. These problems are some of the reasons why more and more data-based approaches, such as machine learning, are taken into account. In fact, data-based approaches are already used in logistics, legal decision processes (Christin et al., 2015) and politics (Maschewski and Nosthoff, 2018). Machine learning approaches, such as artificial neural networks and reinforcement learning become increasingly popular even for modelling physical and mechanical systems (Breen et al., 2019; Narayanan and Jagannathan, 2018). This thesis investigates the interaction possibilities and application scenarios of neural networks within the modelling framework for hybrid dynamical systems.

This chapter gives an overview of fundamental elements of modelling and simulation to define a common terminology and outlines the scope of the study. The structure as well as the research objective of this thesis are presented and relevant research contributions are discussed.

## 1.1 Modelling and Simulation

Modelling and simulation are integral elements of today's research and development. Most technical solutions consist of expensive components and rare materials. Hence, experiments and analysis of prototypes can get very expensive. It is common practice to investigate and simulate the developed construction with computer experiments first, before actually building it. This procedure reduces unnecessary expenditures and enables optimised products.

Between 1920 and 1950 analogous techniques dominated the field of modelling and simulation. A significant change took place when Ragazzini et al. (1947) demonstrated that simulations could be done electronically. By 1967 more than 23 different programs for digital computers were available and the first graphical environment appeared, representing models in block structure. In 1980s matrix environments were established and provided additional modelling tools. Since then, the availability of significant computing resources as well as software and knowledge to use it effectively are constantly increasing (Birta and Arbez, 2013). Nowadays modelling and simulation are important and well-established aspects of engineering and science.

### 1.1.1 Simulation Circle

The basic concept of modelling and simulation describes the process of transforming an observation or phenomenon into an executable application. In the technical field this phenomenon is often referred to as a system. In Fritzson (2004) a definition of a system is given as follows:

**Definition 1.1.1** (System). *A system is an object or a collection of objects whose properties we want to study.*

The motivation to study such systems in details arises from the desire and often the requirement to understand, develop or even optimise the observed phenomenon. In order to analyse or replicate the system's behaviour, a problem definition has to be formulated. This definition enables the development of a corresponding model description. The transformation from the problem definition to the model often requires certain simplifications. Either certain aspects of the system's process are unknown or the system is too complex to be realised completely. A general definition of a model is given by Minsky (1965).

**Definition 1.1.2 (Model).** *To an observer  $B$  an object  $A^*$  is a model of an object  $A$  to the extent that  $B$  can use  $A^*$  to answer questions of interest about  $A$ .*

This definition presents a general idea of a model and is applicable for phenomena of any discipline. In Velten (2009) a more specified version of the concept, the mathematical model, is introduced.

**Definition 1.1.3 (Mathematical Model).** *A mathematical model is a triplet  $(S, Q, M)$  where  $S$  is a system,  $Q$  is a question related to  $S$ , and  $M$  is a set of mathematical statements,  $M = \{ \sum_1, \sum_2, \dots, \sum_n \}$ , which can be used to answer  $Q$ .*

The mathematical model outlines a formal definition of a structure involving mathematical statements. It enables a description of the system but does not necessarily enable computer experiments.

Therefore the mathematical model represents the base for the development of an executable implementation of the system, the simulation model. Due to the fact that simulation models are an idealised and often simplified imitation of the real-world system, verification and validation is required. After the first computer-based experimental investigations, so-called simulations (Siegfried, 2014), the simulation results are used to verify whether the executable model reflects the conceptual model description. In other words the *verification* process evaluates whether changing input values and parameters leads to expected simulation outcomes. The model *validation* assesses whether the simulation model reflects the real-world system and can be applied to answer posed questions about the system. In Figure 1.1 the described procedure is illustrated.

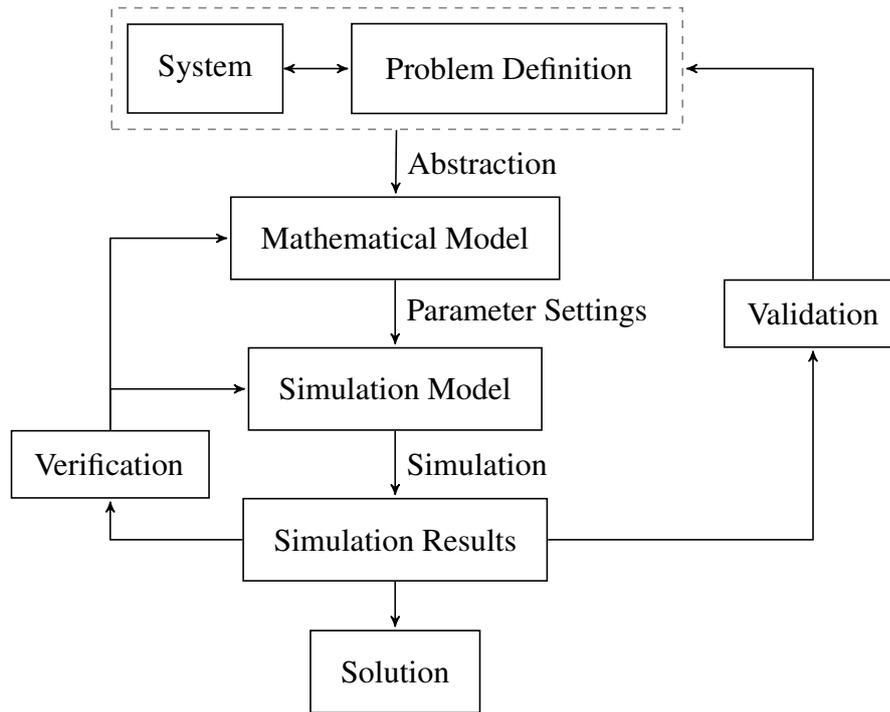


Figure 1.1: The modelling and simulation process, motivated by Sargent et al. (2016).

In the following different model characteristics are discussed, based on the work of Velten (2009).

### 1.1.2 Model Characterisation

Most systems in applied mathematics and engineering represent observable processes, resulting in some kind of measurable output. If there is no further information and investigation of the system, the according model can only be based on the given input and produced output data. An uncertainty about the internal processes of the system remains. The resulting model is referred to as *black box model*. Common examples are empirical, statistical and data-driven models. In contrast, models based on a-priori knowledge about the system are known as *white box* and *first principle models*. Here, the mathematical description of the system, including physical laws and constraints, is identified. In most applications, hypotheses about the internal processes are available to some extent. Hence, models based on partial information are called *grey box* mod-

els. Dependent on the extent of the a-priori knowledge different shades of grey can be defined.

In addition to the classification of white, grey and black box modelling, various characterisations of mathematical models can be distinguished.

- *Static & Dynamic*: Static modelling is a time independent view of a system presenting its structure including class and object diagrams. A dynamical model is a behavioural representation of a system. Dynamical modelling consists of activities, interactions and state changes over time.
- *Stochastic & Deterministic*: A deterministic model contains no randomised processes. Using the same input values and parameters, the deterministic model results in the same output. In stochastic models, the process is at least partially randomised. There is always an uncertainty in the exact status of the modelled variables.
- *Continuous & Discrete*: In a time continuous model, the entity of interest changes continuously with time. In contrast, time discrete models describe entities only at certain points in time with no information in-between these points. Considering a process that is dependent on time and space, its model can be time continuous and spatial discrete if the state variables are defined at any point in time but only accept integer values. In case of a time and space continuous model
- *Linear & Nonlinear*: In linear models the considered variables are combined applying linear operations, e.g. addition, subtraction and scalar-multiplication. Nonlinear models involve multiplications and other transcendental functions such as sine and cosine.
- *SISO & MISO*: These abbreviations are mostly used in telecommunication applications defining special input and output structures of a model. SISO stands for single input single output models whereas MISO refers to multiple input single output. In this manner, it is possible to abbreviate multiple input multiple output as MIMO. Additionally to applications in telecommunication, these definitions can be applied to characterise the structure of any mathematical model.

### 1.1.3 Modelling Approaches

White box models consist of mathematical equations and conditions. Focusing on dynamical systems, different types of equations can be used to model the dynamical behaviour. A difference equation for example describes changes in the quantity as a discrete sequence of states. Hence, difference equations are applied if the resulting model is time discrete. In case of a model for continuous dynamics, differential equations are applicable. The dynamical behaviour of such problems is based on ordinary differential equations (ODE), differential algebraic equations (DAE) or partial differential equations (PDE). Modelling complex systems composed of a large number of heterogeneous, interacting components, often requires model descriptions consisting of more than one equation (Siegfried, 2014). Hence, a mathematical model can comprise a system of differential equations or a combination of different equation structures including algebraic equations (AE). In Chapter 2 basic concepts of white box modelling approaches are discussed in more detail.

Black box models aim to find an approximation of the unknown relation between given input and output values. Elementary statistical methods can be used to analyse the data and detect certain properties, whereas regression models provide a mathematical description of an input-output system and thereby enable computing an output for a given input value (Velten, 2009). Hence, regression models can be used for interpolation and prediction purposes.

Based on Sjöberg et al. (1995) and Juditsky et al. (1995), the mathematical description of a regression model, the regression function, can be defined.

**Definition 1.1.4** (Regression Function). *Let  $(X, Y)$  be a pair of random variables with values in  $X \in \mathbb{R}^d$ ,  $d \in \mathbb{N}$  and  $Y \in \mathbb{R}$ , respectively. A function  $f: X \rightarrow Y$  is said to be the regression function of  $Y$  on  $X$  if*

$$Y = f(X) + e,$$

where  $e$  is zero mean and independent of  $X$ . For  $n \geq 1$ ,  $\hat{f}_n$  shall denote an estimator of  $f$  based on the random sample  $O_i^n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$  from the distribution of  $(X, Y)$ .

Based on Velten (2009), different mathematical structures for the regression function can be defined.

- *Linear Regression:* A simple linear regression model is given as linear function

$$\hat{f}_n(X) = aX + b, \quad X, a, b, \in \mathbb{R}.$$

The parameters  $a$  and  $b$  are so-called regression parameters and  $f$  describes the relation of the given data points

$$(X_1, Y_1), \dots, (X_n, Y_n), n \in \mathbb{N}.$$

A regression model is linear with respect to the regression parameter. Hence, a generalised form of the linear regression function is given as

$$\hat{f}_n(X) = a_0 + a_1g_1(X) + \dots + a_s g_s(X), \quad s \in \mathbb{N},$$

where  $g_s$  are arbitrary linear and nonlinear functions a like.

- *Multiple Linear Regression:* In contrast to the linear regression model, the regression function of the multiple linear regression depends on several independent variables  $X_1, X_2, \dots, X_n, n \in \mathbb{N}$ .
- *Nonlinear Regression:* The nonlinear regression function depends nonlinearly on the regression coefficients  $a_i$ . Hence, the estimation of the output  $y$  can be given as

$$\hat{f}_n(X) = g(X, a).$$

- *Parametric & Non-parametric Regression:* In non-parametric regressions, the predictor does not take a predetermined form, but is derived from the information given by  $X$  and  $Y$ . Parametric regression functions have a fixed number of unknown parameters.
- *Random & Deterministic Design:* Whereas a random design assumes the variables  $X_i$  to be random, independent and identically distributed, the deterministic design suggests the input variables  $X_i$  to be nonrandom. A simple example for the latter would be a regular grid.

A regression model based on observation data  $Y_i$ , can be written as

$$Y_i = f(\Phi_i) + e_i, \quad i \in \{1, 2, \dots, N\},$$

where  $i$  is the sample index.  $\Phi_i$  consists not only of system inputs  $U_i$  but also on past system output values  $Y_j$

$$\Phi_i = (Y_{i-1}, \dots, Y_{i-m}, U_i, \dots, U_{i-p}).$$

Such models is called *nonlinear auto-regression exogenous model* (NARX). Considering only past output values  $\Phi_i = (Y_{i-1}, \dots, Y_{i-d})$ , the model is called *non-parametric auto-regression* (NAR) of dimension  $d$ .

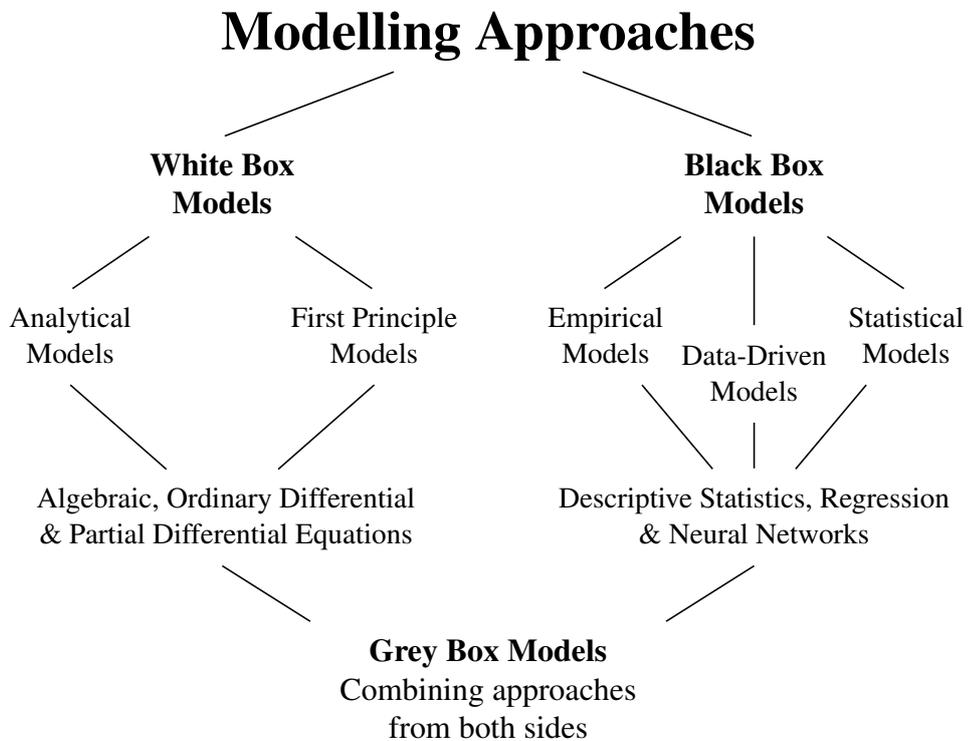


Figure 1.2: An overview of the different modelling approaches within white and black box categories.

The quality and quantity of the sample data  $\{(X_i, Y_i) : i \in \mathbb{N}\}$  affect the accuracy of the estimation. For the assessment of that accuracy a cost function can be introduced to evaluate the discrepancy between data and estimation. There are various nonlinear

approaches to estimate the regression function  $f$ . The different regression models, as introduced above, require theoretical reasoning or certain hypotheses about the structure of the regression function. In some cases, no a-priori knowledge is available to conjecture the structure for the regression function. This is where (artificial) neural networks (ANN) come into play. An illustrative summary of the discussed modelling approaches is given in Figure 1.2.

## 1.2 Related Work

There are various well-established modelling approaches, each characterisable with attributes of Section 1.1.2. Cellular automata for example, are applicable for time continuous and state discrete processes, whereas ordinary differential equations are used to describe time and state continuous systems. For the modelling of hybrid systems a combination of different modelling approaches is required. The combination of the methods system dynamics, agent-based and discrete events, is referred to as *Multi-method modelling* (Borshchev, 2014; Glock et al., 2015). In contrast, hybrid dynamical systems are a combination of time continuous modelling approaches based on ordinary differential equations and differential algebraic equations and discrete processes.

A well-known formalism for hybrid discrete time event systems, so-called DEVS (Discrete Event System Specification), is based on the work of Zeigler et al. (2000). Since its first introduction the formalism has been extended to meet growing demands. It includes not only combinations of discrete but also continuous model descriptions (Deatcu and Pawletta, 2012) and is constantly evolving (Preyser et al., 2016, 2019). Nevertheless, these enhancements are mainly focussing on discrete event systems.

One of the first formal definitions of hybrid dynamical systems is given in Henzinger (1996). Subsequently, various formulations, focussing either on the mathematical modelling or the corresponding simulation process of hybrid dynamical systems, have been introduced and discussed in literature (Antsaklis, 2000; Kwiatkowski et al., 2003; Krüger et al., 2012; Mehlhase, 2015; Körner et al., 2018).

In regard to the implementation of hybrid dynamical models, *Co-Simulation*, as discussed in Hafner and Popper (2017) and Huynh (2016) is not applicable. It investigates tools and requirements to coordinate simulation processes of coupled subsystems. However, the structural description of the considered hybrid dynamical system (HDS) consists of decoupled discrete and dynamical processes. Hence, regular simulation techniques are sufficient.

An overview of other existing simulation methods and tools is given by Carloni et al. (2006). HYSDEL, an abbreviation for hybrid system description language, provides not only a formal structure to describe and implement hybrid systems, but also offers a corresponding simulation environment, see Torrisi and Bemporad (2004). The embedded compiler translates the given HYSDEL problem into a mixed logical dynamical system (MLD) by reformulating constraints and logical conditions into equations and inequalities (Bemporad and Morari, 1999; Williams, 2013). Afterwards the different system descriptions are formulated using piecewise affine systems (PWA) (Paoletti et al., 2010) also called switched affine systems (SAS) (Seatzu et al., 2006; Zhu and Antsaklis, 2013). Especially in control theory PWA systems and SAS are often applied to formulate hybrid control problems (Potočník et al., 2004; Bemporad et al., 2009). HYSDEL 3.0 uses well-established toolboxes such as MPT (Herceg et al., 2013) and YALMIP (Lofberg, 2004) to improve performance and handling of hybrid systems (Kvasnica, 2008).

A black box modelling approach for hybrid dynamical systems is included in the so-called hybrid identification toolbox (HIT). Based on measured input and output data the integrated algorithm creates an approximation of the underlying SAS structure and clusters the data to determine the domain for each submodel (Ferrari-Trecate and al., 2003; Cauty et al., 2012). HIT is mainly based on regression algorithms. However, Hornik et al. (1989) have proven that neural networks with one hidden layer are feasible to approximate any continuous function.

There are various papers discussing combinations of white box modelling and neural networks, often referred to as hybrid neural networks (Psichogios and Ungar, 1992; Laursen et al., 2007; Yang et al., 2008; Sen et al., 2011; Cheng et al., 2012; Lu et al., 2016). In Lin and Unbehauen (1992) neural networks are used to approximate a PWA

system. Števek and Kozák (2011) and Števek et al. (2012b) introduce a method based on neural networks with orthogonal activation functions. In Martius and Lampert (2016) and Sahoo et al. (2018) a special network structure, the so-called equation learner is introduced, to examine extrapolation capabilities of neural networks.

### 1.3 Research Objective and Structure of the Thesis

The aim of this thesis is to investigate how artificial neural network concepts can support the modelling of hybrid dynamical systems. In this context, a framework is introduced to include neural networks into the hybrid automaton structure. Based on the hybrid modelling elements introduced by Henzinger (1996); Zeigler et al. (2000); Kwiatkowski et al. (2003); Krüger et al. (2012); Mehlhase (2015); Körner et al. (2018) new framework components will be defined. This framework enables a structured embedding of neural networks in hybrid models and facilitates a standardised analysis of the formulated interactions. The feasibility of neural networks in the modelling of hybrid dynamical models will be investigated. Therefore, different neural network structures found in literature (Lagaris et al., 1998; Martius and Lampert, 2016; Chen et al., 2018; Bar-Sinai et al., 2019) will be applied. Additionally the applicability of the introduced framework will be tested for existing hybrid dynamical model approaches such as (Števek et al., 2012b). In addition, a generalised structure characterising feed-forward networks is established.

In Chapter 2, the methodology of hybrid systems in general, and hybrid dynamical systems in particular, are discussed. Based on previous work, the formal definition of a hybrid dynamical system and a characterisation of the different elements of a hybrid dynamical automaton are given. Three possible definitions of the involved discrete events are presented.

In Chapter 3, the different elements of artificial neural networks are described. The basic structure as well as common learning methods are explained. Based on the given definition of activation functions, current network structures are introduced and suitable applications are discussed.

Chapter 4 introduces the framework applications for including neural networks in the modelling of hybrid dynamical systems, based on the definitions given in Chapter 2. Motivated by the structure of the hybrid automaton, the framework defines three different application scenarios by replacing particular elements of the hybrid model with artificial neural networks.

The first application case formulates the replacement of the entire hybrid system. In contrast to existing methods, the aim is to investigate the extrapolation capabilities of the network approximation, predicting not only the next time step but also the behaviour of the system for scenarios excluded from training. The second application defines the replacement of the dynamical processes of the hybrid automaton. If the dynamics of at least one local subsystem are unknown, the framework enables a standardised model description to include a network approach to approximate the local system behaviour. It facilitates the integration of various feed-forward network structures. The different structures are compared and the requirements of the dataset for the training procedure of the networks are analysed. The last scenario discusses a partial replacement of the discrete process of the system. Based on a classification of the system output, a network application is evaluated.

In Chapter 5, two hybrid dynamical systems are chosen to apply the framework. Regarding the mathematical model description, these examples represent two different classes of hybrid systems. The feasibility of the three defined scenarios for both examples is examined. The properties of each case as well as the limitations of the structure are discussed. Different neural network structures are applied to investigate advantages and disadvantages with respect to the different application scenarios.

For the training and validation process of the neural networks, different datasets are used. Depending on the size and quality of the used training data the accuracy of the approximations is evaluated. The approximation potential of neural networks is well-established, whereas the research regarding extrapolation capabilities of neural networks is ongoing (Martius and Lampert, 2016). Hence, the case study partially investigates prediction and extrapolation abilities of the applied networks.

General implications and specifications, such as stating the example classes for which the introduced framework is particularly suitable, are given in Chapter 6. The simulation outcomes are compared with common modelling approaches to evaluate feasibility and accuracy of the neural network approach. Implied advantages and disadvantages for practical applications are discussed. In Chapter 7, a summary of the study is given and possible extensions of the framework as well as future research objectives are suggested.



# 2

## Hybrid Dynamical Systems

---

Modelling and simulation is a concept used in different disciplines for various applications. Therefore, some of the terminologies are not uniquely defined, e.g. *hybrid models*, and are applicable in different situations. On the one hand, hybrid models are a combination of different modelling methods to benefit from the advantages of each applied modelling approach. This offers various possibilities to model complex systems. On the other hand, a combination of different model descriptions can be required due to a mixture of continuous and discrete processes in the system. The latter are so-called hybrid dynamical systems (HDS). Such systems are mostly modelled using first principle models, where the mathematical description can be given explicitly. In hybrid dynamical systems, the dynamical behaviour is described either by ordinary differential equations (ODE) or differential algebraic equations (DAE). The discrete process of the system defines the moment when the system description changes from one dynamical submodel to the next and is referred to as event.

In the following basic mathematical concepts for modelling and simulation of dynamical systems are summarised. Secondly, established modelling approaches for hybrid dynamical systems, including the hybrid dynamical automaton, are introduced. Possible structural changes as well as common event formulations are discussed. Following that, an overview of existing simulation techniques for hybrid dynamical systems is given.

## 2.1 State Space Representation

Originating from disciplines such as engineering, computer science and mathematics, different modelling approaches for dynamical processes have been developed. The state space representation is a frequently used mathematical model for physical systems. First-order differential equations are used to describe the relation of input, output and state variables. It is a suitable and compact concept to model and analyse systems with multiple inputs and outputs (Miková et al., 2016).

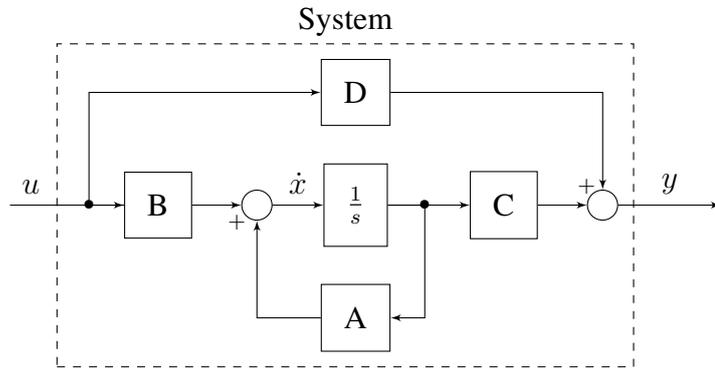


Figure 2.1: A block diagram of the linear time-invariant state space representation.

**Definition 2.1.1** (Linear State Space Representation). *The linear state space representation of a dynamical system is given in the form*

$$\begin{cases} \dot{x}(t) = A(t)x(t) + B(t)u(t), \\ y(t) = C(t)x(t) + D(t)u(t), \end{cases} \quad (2.1)$$

where  $x(t) \in \mathbb{R}^n$  is the state vector and  $y(t) \in \mathbb{R}^q$  the output vector. Vector  $u(t) \in \mathbb{R}^p$  denotes the input (control) vector,  $A(t) \in \mathbb{R}^{n \times n}$  the state matrix,  $B(t) \in \mathbb{R}^{n \times p}$  the input matrix,  $C(t) \in \mathbb{R}^{q \times n}$  the output matrix and  $D(t) \in \mathbb{R}^{q \times p}$  symbols the feed-forward matrix.

The system is called *time-variant* if the matrices  $A, B, C, D$  are time-dependent, otherwise it is a *time-invariant* representation. The state space representation describes the system's output and the change of the states depending on the state vector and the given input. The matrix  $A$  describes the connection of all internal states, whereby  $B$  states

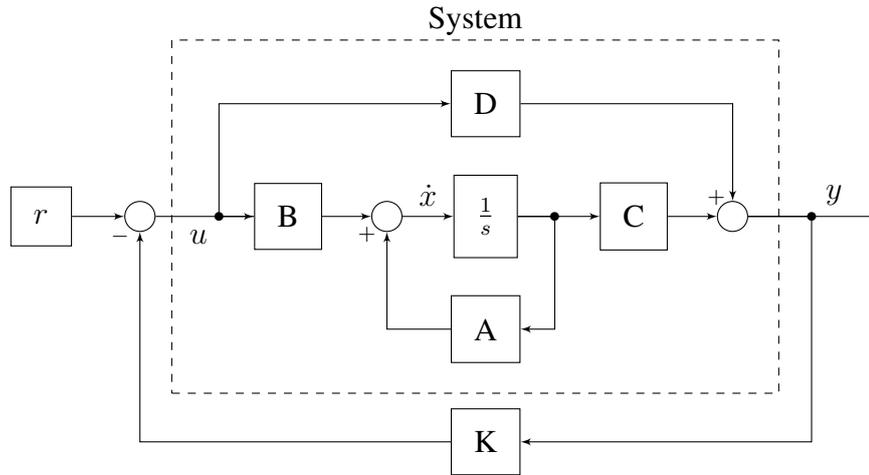


Figure 2.2: A block diagram of the extended LTI including reference signal  $r$  and feedback loop  $K$ .

how the input effects the state variables. Matrix  $C$  defines the relation of the state variables in the output and  $D$  enables the input to be part of the output equation. In Figure 2.1 the linear time-invariant state space (LTI) is illustrated as block diagram. Considering time-discrete signals instead of continuous states and inputs, a discrete formulation of the state space representation can be given as

$$\begin{cases} x_{k+1} = A_k x_k + B_k u_k, \\ y_k = C_k x_k + D_k u_k. \end{cases} \quad (2.2)$$

Analogous to the continuous case, the matrices  $A$ ,  $B$ ,  $C$  and  $D$  can be constant, describing a time-invariant case. In case the matrices depend on the sample index  $k \in \mathbb{N}$  the corresponding time-variant system representation is given.

Especially in engineering, the control of dynamical systems is an essential aspect of modelling and simulation. The continuous state space model given in equation (2.1) can be extended by adding a feedback loop  $K$ . This can be done by substituting the input signal with  $u(t) = Ky(t)$ , which results in the following representation

$$\begin{cases} \dot{x}(t) = Ax(t) + BKy(t), \\ y(t) = Cx(t) + DKy(t). \end{cases} \quad (2.3)$$

The feedback loop enables the output to influence the system's behaviour, but it does not facilitate controlling the system behaviour. An additional element, a reference signal  $r$ , has to be introduced, resulting in  $u(t) = r - Ky(t)$ . In Figure 2.2 a block diagram of the extended model including the feedback loop and the reference signal is illustrated.

The corresponding mathematical representation is given as

$$\begin{cases} \dot{x}(t) = Ax(t) + B(r - Ky(t)), \\ y(t) = Cx(t) + D(r - Ky(t)). \end{cases} \quad (2.4)$$

An obvious restriction of the described models is the linearity of the mathematical description. A state space model for nonlinear systems is defined as

$$\begin{cases} \dot{x}(t) = f(t, x(t), u(t)), \\ y(t) = h(t, x(t), u(t)), \end{cases} \quad (2.5)$$

where  $f$  denotes the change of the system over time and  $h$  defines the output of the system. In the modelling process of complex systems required constraints can be described by additional algebraic equations and variables. To comply with the additional requirements, these constraints have to be included into the model description using differential-algebraic equations (DAEs).

**Definition 2.1.2** (Implicit Differential Algebraic Equation). *Given the state vector  $x \in \mathbb{R}^n$  and its time-derivative  $\dot{x}$  an implicit differential algebraic equation is defined by*

$$F: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^q \times \mathbb{R} \rightarrow \mathbb{R}^m, \quad F(x, \dot{x}, z, t) = 0, \quad (2.6)$$

where  $z$  denotes the algebraic variable vector. If  $F$  is continuously differentiable and the derivative of  $F$  with respect to  $\dot{x}$  is regular, the implicit function theorem is applicable and equation (2.6) can be transformed into an explicit differential equation

$$\dot{x} = f(x, z, t).$$

If  $F_{\dot{x}}$  is singular, equation (2.6) is a real differential algebraic equation.

Hybrid dynamical systems usually contain numerous dynamical processes. Instead of only one dynamical description, each local behaviour requires its own description.

Hence, an index  $d \in D$  for labelling the local system descriptions is required. Those dynamical behaviours can be described using the state space model for nonlinear systems in (2.5) by including differential algebraic equations.

**Definition 2.1.3** (Semi-Explicit Differential Algebraic Equation System). *Given the state vector  $x$ , the time-derivative  $\dot{x}$ , the input  $u$  and a vector containing external variables  $w$ , the semi-explicit differential algebraic equation system can be defined. At each location  $d \in D$  a set of differential and algebraic equations is given as*

$$\begin{cases} \dot{x}_d = f_d(x_d, u_d, w_d, t), \\ 0 = g_d(x_d, u_d, w_d, t), \\ y_d = h_d(x_d, u_d, w_d, t), \end{cases} \quad (2.7)$$

where  $f_d$  denotes the change of the state over time,  $g_d$  describes the algebraic constraints and  $h_d$  defines the output of the system.

An analytical solution for systems as described in definition 2.1.3 does often not exist. An alternative approach is the approximation of the system behaviour using piecewise affine models (PWA). These models are received by partitioning the state-input domain into a finite number of convex polyhedral regions and associating each region with an affine submodel (Paoletti et al., 2010; Potočnik et al., 2004).

**Definition 2.1.4** (Piece-Wise Affine System). *Piece-wise affine systems are defined as*

$$\begin{aligned} x_{k+1} &= A_i x_k + B_i u_k + f_i, \\ y_k &= C_i x_k + D_i u_k + g_i, \end{aligned} \quad \text{for } (x_k, u_k)^T \in X_i, \quad i \in I \quad (2.8)$$

at time  $k \in \mathbb{N}_0$ , where  $x_k \in \mathbb{R}^{n_i}$  denotes the state vector,  $u_k \in \mathbb{R}^{q_i}$  the input vector and  $y \in \mathbb{R}^{p_i}$  the output vector. The variables  $A_i, B_i, f_i, C_i, D_i, g_i$  are matrices of suitable size. The domain space is partitioned into convex polyhedrons  $X_i$  described by sets of linear inequalities and index  $i \in I$  labels the according affine submodel.

## 2.2 Modelling of Hybrid Dynamical Systems

In applied mathematics and computer science the creation of modelling standards is an important aspect. The advantage of a modelling frameworks is that they can be applied

not only to one unique problem description but to an entire class of problems. Based in related work the structural and graphical framework for hybrid dynamical systems is defined. Characterisations of different event formulations and transitions are given.

### 2.2.1 Discrete Event & Differential Equation System Simulations

The formalism defined by Zeigler (1976), called Discrete Event System Simulation (DEVS), focuses on the simulation and realisation of discrete event systems. This formalism covers mostly applications like event scheduling, activity scanning and process interaction. An event processor can be used as an implementation tool for such models (Solcany, 2008). The DEVS description has been extended to include Discrete Time (DTSS) as well as Differential Equation System Simulations (DESS) (Zeigler et al., 2000). Discrete time simulations can be described by event lists or difference equations. In case of DTSS the simulation includes recursive algorithms whereas DESS are described by differential equations realised with numerical integrators. In order to apply the formalism to more complex systems an extension has been established, namely DEV&DESS, combining discrete and continuous model dynamics. This concept is used to describe hybrid systems and enables a standardised framework to connect and reuse model parts realising discrete and continuous behaviours. According to the DEVS formalism, a septuple containing necessary elements to describe a discrete event system is given (Vangheluwe, 2000; Deatcu and Pawletta, 2012) by

$$\text{DEV} := (X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, ta).$$

The elements are defined as follows.

- $X$  is the set of input event values.
- $Y$  is the set of output event values.
- $S$  is the set of state values.
- $\delta_{\text{int}}: X \rightarrow S$  is the internal state transition function.
- $\delta_{\text{ext}}: S \rightarrow S$  is the external state transition function.
- $\lambda: S \rightarrow Y$  is the output function.
- $ta$  is the time advance function.

In order to define elements for DEV&DESS, discrete and continuous input as well as output sets are necessary. In addition, two different output functions, for the discrete as well as the continuous process are required. A continuous event condition function  $C$  and the state transition function  $\delta_{\text{state}}$  are responsible to detect state events and perform the transition. Based on Preyser et al. (2016) and Winkler et al. (2017) a characterization for hybrid systems in the DEV&DESS formalism can be stated as

$$\text{DEV\&DESS} := (X, Y, S, f, \lambda_c, \delta_{\text{state}}, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda_d, C_{\text{int}}, ta)$$

where

- $X = X_c \times X_d$  is the set of inputs,
- $Y = Y_c \times Y_d$  is the set of outputs,
- $S = S_c \times S_d$  is the set of state values,
- $f$  is the rate of change (given by an ODE),
- $\lambda_c: S \rightarrow Y$  is the continuous output function,
- $\delta_{\text{state}}: S \rightarrow S$  is the external state transition function,
- $\lambda_d: S \rightarrow Y$  is the discrete output function,
- $C_{\text{int}}: Q \times 0, 1 : X_c \rightarrow \{\text{true}, \text{false}\}$  is the state event condition function and
- $\delta_{\text{int}}, \delta_{\text{ext}}$  and  $ta$  are defined as in DEV.

In addition to the elements defined above, it is possible to create coupled components containing several atomic subcomponents.

In Mehlhase (2015) a similar approach is used to introducing a framework for simulation models of variable structure systems. The basic elements of this formalism are listed in Table 2.1. If more than one variable is required to describe the system, *VARNAMESSET* is used and *VALUE* is substituted by *STATE*, describing the value of all variables, including constants  $v$  fulfilling  $\dot{v} = 0$ . In case of more than one equation characterising the component's behaviour, *EQS* replaces *EQ*. The additional element *CONNECT* describes the relation between different interfaces. The combination of all these elements forms a reusable component of the model description.

<i>VARNAME</i>	name of the variable
<i>VALUE</i>	value of the variable
<i>TIME</i>	time
<i>EQ</i>	an equation: ODE, DAE, etc.
<i>CONNECT</i>	the transition description
<i>SOLVER</i>	the considered solver for the component
<i>ID</i>	the component identifier

Table 2.1: Basic Elements for simulation models of hybrid systems.

Both formalisms are primarily used to support the development of simulation models, independent of any specific simulation environment. The next section focuses specifically on the mathematical modelling approaches.

### 2.2.2 Hybrid Dynamical Automaton

The complex structure of hybrid systems is commonly illustrated using an automaton (Körner et al., 2018; Carloni et al., 2006). Automata are often applied to depict abstract machines as well as theoretical concepts in computer science, such as combinational logic. It supports structuring mathematical tasks and illustrating finite state machines (Bemporad and Morari, 1999). It shows current states, update rules as well as transition conditions. By expanding the description possibilities of the update rules, the automata concept can be applied to model hybrid dynamical systems, as shown in Figure 2.3.

In terms of layout, an automaton is an ambiguous description. It just characterises the basic structure of the model in a compact way. The nodes of the automaton describe different local dynamics whereas the connecting lines, called edges, define transition conditions. If fulfilled, the transition from one location (node) to the next is initiated.

Focusing on modelling of hybrid dynamical systems, a mathematical definition of the hybrid automaton is given (Henzinger, 1996; Carloni et al., 2006; Körner, 2015).

**Definition 2.2.1** (Hybrid Dynamical Automaton). *A hybrid dynamical automaton  $A_h$  is defined as the septuple*

$$A_h = (D; X; B; W; E; \text{Inv}; \text{Act}),$$

where

- $D$  is the finite set of discrete states.
- $X$  is the continuous state space containing all continuous states  $x \in X \subseteq \mathbb{R}^n$ .
- $B$  is a finite set of symbols.
- $W \subseteq \mathbb{R}^q$  stands for the continuous communication space and contains each continuous external variable  $w \in W$ .
- $E$  is a finite set of events. Every event  $e \in E$  is defined by  $e = (d; b; G_{d,d'}; J_{d,d'}; d')$ , labelled  $b \in B$  and depicted as an edge interconnecting different locations.  $G_{d,d'}$  symbolises a finite convex subset of  $X$ , the so called guard region.  $J_{d,d'}$  defines the jump relation for the transition from location  $d$  to  $d'$  on a subset of  $X \times X$  with  $d, d' \in D$ .
- $\text{Inv}: D \rightarrow \mathcal{P}(X)$  is called the location invariant, where  $\mathcal{P}(X)$  is the power set of  $X$ . At location  $d \in D$ , the corresponding state  $x$  must satisfy  $x \in \text{Inv}(d)$ .
- $\text{Act}: D \rightarrow \mathcal{F} : d \mapsto F_d$  maps each location  $d \in D$  to a set of differential algebraic equations  $F_d := (f_d, g_d, h_d)$ , defined as

$$F_d \begin{cases} \dot{x}_d = f_d(x_d, u_d, w_d, t), \\ 0 = g_d(x_d, u_d, w_d, t), \\ y_d = h_d(x_d, u_d, w_d, t). \end{cases} \quad (2.9)$$

Function  $f_d$  describes the state change with respect to time,  $g_d$  denotes the algebraic constraints and  $y_d$  defines the output. The variable  $x$  denotes the state vector,  $u$  the input and  $w$  containing external variables including system parameters  $p$ . The set  $\mathcal{F}$  contains the triplets of every DAE.

The dimension of a hybrid dynamical automaton is defined as the dimension  $n$  of the continuous state space  $X \subseteq \mathbb{R}^n$ .

**Remark.** Each node in Figure 2.3 represents the local dynamical behaviour in the form of a semi-explicit DAE, as given in equation (2.7). The dynamics can also be given as an implicit equation  $F(\dot{x}, x, u, w, t)$ , combining the description of the time-continuous change and the algebraic constraints. If it is sufficient to describe the system using ODE without any additional algebraic constraints, then the equation  $g_d$  in (2.9) can be omitted. The guard region  $G$  is restricted to convex sets, since it enables exact state localisations.

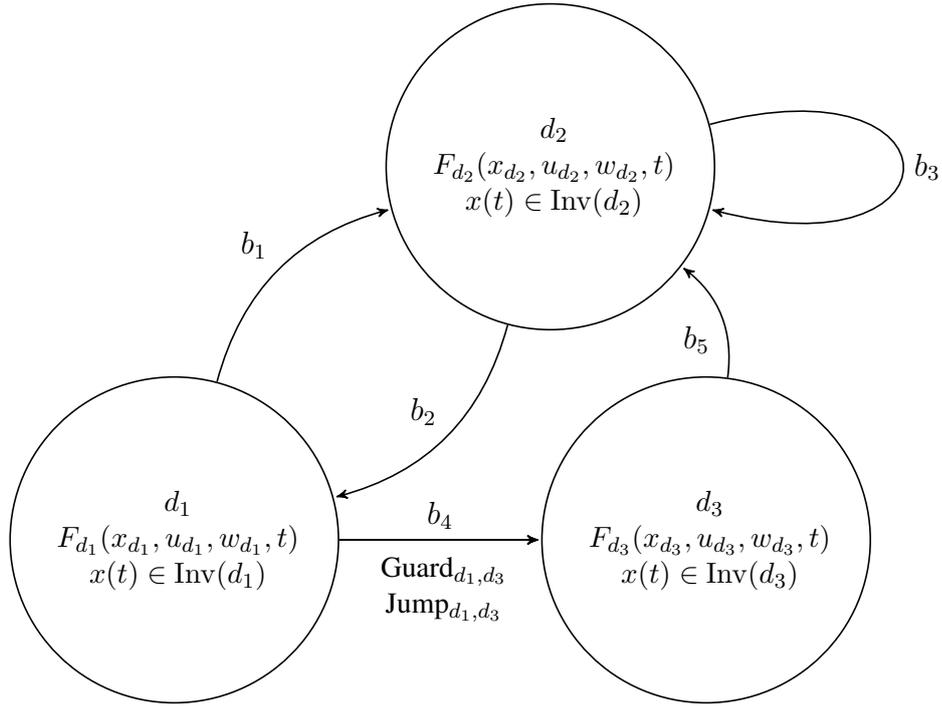


Figure 2.3: Conceptual structure of a hybrid automaton based on the work of Körner (2015).

### 2.2.3 Event Characterisation in Hybrid Systems

In hybrid dynamical systems an *event* defines the interruption of the continuous process and initiates changes in parameters, variables or entire descriptions. Two different types of events, the autonomous and the controlled event, can be distinguished. The latter can only be induced from outside the system, for example an input signal triggering the event. An autonomous event happens without any external influence. The requirements enabling a transition are a consequence of the system description. In this contribution purely autonomous events are considered.

Based on the characteristics of the events another distinction can be made. A *time event* occurs at a specific predefined point in time  $t_e \in \mathbb{R}^+$  and is part of the model description. When the system reaches  $t = t_e$ , the event  $e = (d; b; G_{d,d'}; J_{d,d'}; d')$  is initiated and the system change is executed. For the implementation, a logical variable or a simple if-condition is sufficient to verify the inequality  $t \geq t_e$  and initiate the time event. In case

of a *state event*, the state vector  $x \in \mathbb{R}^n$  has to fulfil predefined requirements. Thus, the event time  $t_e$  depends on the state value itself.

In Henzinger (1996) events are characterised using a jump condition, an event function, a transition relation and special pre- and post-jump regions. In Kwiatkowski et al. (2003) an autonomous jump set and a jump transition map are sufficient to define the event. In the latter the events are triggered if the state variables reach their predefined thresholds. Comparing these two descriptions with definition 2.2.1, two consistent elements can be found, a special event region and a transition map. The approaches differ with regard to the definition of the initiation process of the event. Hence, three different event actuators can be distinguished:

- (a) defining specific *thresholds*  $\Delta x$  for the state values  $x$  as shown in Kwiatkowski et al. (2003),
- (b) formulating an *event function*  $e$ , describing the required relation of the state variables, where  $e(x) = 0$  initiates the event and
- (c) defining the *guard region*  $G$ , which is a convex subset of the domain.

In Figure 2.4 the different event conditions are depicted. The complexity of the hybrid system and the level of simplification determine which formulation is applicable.

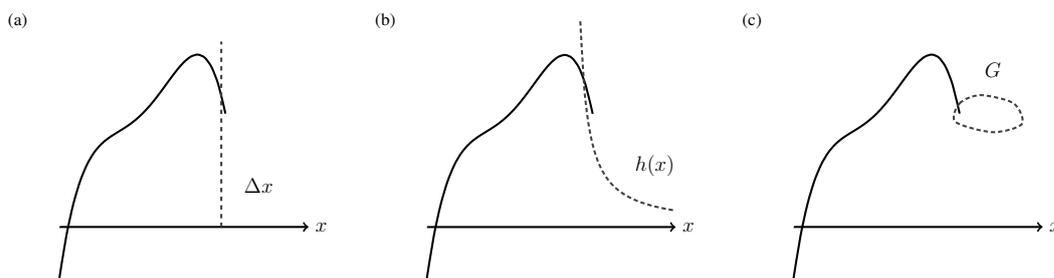


Figure 2.4: The different event constraints are illustrated, depicting the threshold  $\Delta x$  in (a), an event function  $h(x)$  in (b) and the guard region  $G$  in (c).

**Remark.** *The guard region can be seen as generalisation of the threshold and the event function, whereas the threshold represents a special case of the event function. Considering a two-dimensional hybrid system, the thresholds of a state variable form a rectangular area, which can then be defined as guard region. If an event function  $h$  is defined, the set of state values fulfilling  $e(x) = 0$  represents the corresponding guard region.*

Each node of the hybrid automata contains an individual model description, e.g. differential-algebraic equations as defined by (2.7). The state and time events, respectively, enable the transition from one node to the next. A characterisation of the different possible changes during the event, transitioning from one subsystem to the next, is formulated based on Körner (2015).

- *Value Changes:* On the one hand this category describes changes in the parameter vector  $\mathcal{T}_p$  with  $p_l \mapsto p_k$  as well as possible modifications in the external values  $\mathcal{T}_w$  with  $w_l \mapsto w_k$ . On the other hand, value changes also refer to adaptations in the state vector  $\mathcal{T}_x$  which is defined by the jump relation  $J_{l,l'}$  as

$$\mathcal{T}_x : x \mapsto J_{l,l'}(x).$$

- *Structural Changes:* There are different possibilities for structural changes. One option are modifications in the definition of the time-derivative  $\mathcal{T}_{\dot{x}}$ , given as  $f_l \mapsto f_k$ . A structural change of the algebraic equation of the DAE, denoted as  $\mathcal{T}_g$  with  $g_l \mapsto g_k$ , and variations in the output vector  $\mathcal{T}_y$ , defined as  $y_l \mapsto y_k$ , are possible.
- *Model Changes:* Considering all the different changes from above, an overall switch in the system description is defined as  $\mathcal{T}_M : \mathcal{M}_l \rightarrow \mathcal{M}_k$ , redefining every component of the DAE, including the time-derivative equation, the algebraic equation and the output vector as well as the initial states, outputs, parameters and the external variables.

An arbitrary combination within the different categories can be formalised using function compositions of the different transitions. A detailed characterization of the different combinations can be found in Körner (2015).

## 2.3 Simulation of Hybrid Dynamical Systems

Hybrid approaches are common practice to find a suitable problem solution and benefit from combining advantages of different modelling methods. Especially in control engineering various hybrid dynamical systems can be found. Hence, different tools to support the simulation process of hybrid models, have been developed.

In the following, a selection of simulation tools is introduced. A broader overview can be found in Carloni et al. (2006).

### 2.3.1 Hybrid System Description Language – HYSDEL

HYSDEL is an open MATLAB toolbox which supports modelling as well as simulating hybrid systems. It was established in 2004 at the University of Zurich (Torrise and Bemporad, 2004). On the one hand, HYSDEL is a descriptive modelling language, similar to DEV&DESS, and contains various modelling elements, as listed in Table 2.2. On the other hand, HYSDEL enables the simulation of hybrid systems in a feasible

<i>INTERFACE</i>	declaration of all system variables
<i>IMPLEMENTATION</i>	definition of the relation between the variables
<i>AUX</i>	declaration of auxiliary variables
<i>CONTINUOUS</i>	state update equation for real variables
<i>AUTOMATA</i>	state update equation for boolean variables
<i>LINEAR</i>	definition of additional variables using affine expressions
<i>LOGIC</i>	logical relations between boolean variables
<i>AD</i>	describing the binary variables and the event generator
<i>MUST</i>	constraints on input, state and output variables
<i>OUTPUT</i>	output variables for the overall MLD system

Table 2.2: Modelling components for HDS using HYSDEL, see Torrise and Bemporad (2004).

and efficient way. After initialising the listed components according to the system description using the HYSDEL language, the HYSDEL compiler automatically creates an accessible MATLAB function. The formulated constraints and logical conditions are transformed into equations and inequalities forming an MLD, see (A.5). Following that, a polyhedral partition of the input-state domain is created to find suitable sub-domains

for each local piece-wise affine system (PWA), as defined in (2.8). The final step of the algorithm creates a MATLAB function which can be executed directly in MATLAB.

HYSDEL is based on a scalar description and was originally developed in the programming language C. The revised version, HYSDEL 3.0, overcomes weaknesses like the scalar based description and the inability to reuse submodels (Bemporad et al., 2009; Kvasnica, 2008). Additionally, the lengthy compiling process has been improved by a 'Yet Another LMI Parser' (YALMIP) module, which automatically detects the problem class and selects a suitable solver based on its analysis (Lofberg, 2004). HYSDEL 3.0 expects a problem description in HYSDEL language, transforms it into an xml-file using an xml parser to enable the YALMIP module. It results in an MLD system, which is transformed into a PWA, similar to the original algorithm. Figure 2.5 illustrates this procedure. In Kvasnica et al. (2011) an alternative approach for approximating the involved

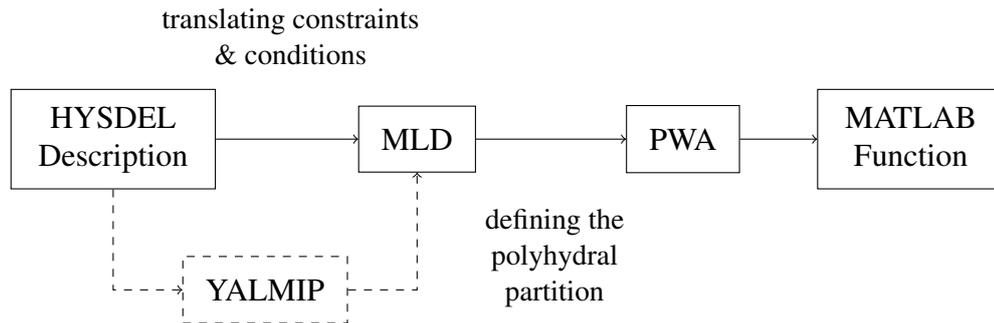


Figure 2.5: The flow chart describes the simulation steps of the HYSDEL compiler, including the improvements (dashed lines) made in HYSDEL 3.0.

PWA systems is given. The approximation is retained by solving nonlinear programming problems resulting in an efficient and computational tractable algorithm. Under certain assumption the PWA approximation of multivariable functions narrows down to solving a series of one-dimensional approximations. A toolbox is introduced to export the obtained optimal PWA approximations into the HYSDEL language. This enables a mathematical problem formulation and hence control design.

### 2.3.2 Multi-Parametric Toolbox – MPT

MPT is a MATLAB Toolbox especially developed to cope with optimisation tasks in the field of model predictive control. The improved version MPT 3.0 is based on the YALMIP modelling language and supports solving problems in computational geometry, see Herceg et al. (2013). In the original version of MPT a single entity consisted of multiple algorithms. In MPT 3.0 the structure follows a hierarchy, derived from object-oriented programming, to enable better maintenance. A wide range of test cases underlines the numerical reliability of the toolbox which shows superior performance compared with the previous version. This toolbox focuses on solving linear and quadratic optimisation problems including a linear-complementarity problem (LCP) solver and a parametric LCP solver. Its interface supports modelling autonomous and controlled LTI systems as well as PWA and MLD systems, e.g. HYSDEL language.

In many practical cases, the mathematical description of the nonlinearity is not available. Alternatively, the given analytical form misses its numerical parameters. In such cases one option is to construct the PWA approximation manually using the HYSDEL language Bemporad and Morari (1999). Another is to use measured input-output data to extract the PWA characteristics (Roll et al., 2004; Ferrari-Trecate et al., 2001; Paoletti et al., 2010) as described in the following.

### 2.3.3 Hybrid Identification Toolbox – HIT

Considering a MISO system, where only input-output data pairs are available, the hybrid identification toolbox (HIT) is applicable. It generalises a classical AutoRegressive eXogenous (ARX) approach to determine a discrete-time hybrid system in PWA form. Applying a mixture of clustering, pattern recognition and linear identification techniques, PWA systems can be identified (Ferrari-Trecate et al., 2005). The different steps of the algorithm are summarised in the following and can be found in detail in Ferrari-Trecate and al (2003).

1. Given the data  $(u(k), y(k))$  a regression matrix is defined and noise is added, leading to  $(x(k), y(k))$ .

2. The resulting data points are used for clustering. For each data point  $(x(k), y(k))$  the closest  $c - 1$  data points are collected into a subset  $C_j$ . The value  $c$  has a high impact on the quality of the resulting clusters. For every subset  $C_j$  a weighted least squares regression is applied to determine the local data model  $\theta_j$ .
3. A feature vector, including information about coefficients and localisation of the submodels  $\theta_j$ , is defined to distinguish models with the same coefficient but different spatial constraints.
4. These feature vectors  $\zeta_j$  are clustered with a K-means-like algorithm to minimise the given cost functional, resulting in a refined classification of the original data.
5. A weighted least squares algorithm is applied to approximate the parameters of the PieceWise AutoRegressive eXogenous (PWARX) model.
6. Finally a region for each subsystem is determined, using a combination of linear support vector machines, multi-category pattern recognition strategies and robust linear programming ideas.

Canty et al. (2012) criticises the algorithm to fail in accurately identifying model parameters. Due to its sensitivity to the magnitude of noise in the given data, the weakest element of the algorithm is the least squares approximation (PWA-LS) of the submodel parameters in step 5. The idea they suggest is to replace the PWA-LS with a PWA-OE, which means substituting the past output data in the error criterion with past model output data. The Levenberg-Marquardt-Fletcher gradient based algorithm is applied to optimise the output error model, improving the experimental results significantly.

Two crucial disadvantages of approaches like this are mentioned in Szűcs et al. (2012). Due to solving high-dimensional optimization problems the algorithms are very time consuming. Additionally a well-defined PWA approximation depends on the determination of a proper partition of the domain. The latter can not overlap nor leave undefined holes, but to cover the whole space of parameters of interest. To overcome these difficulties another method can be applied.

### 2.3.4 PWA OAF Identification Toolbox

In Altin et al. (2018) a framework for hybrid model predictive control for hybrid systems is introduced. Based on this framework, Števek and Kozák (2011) developed an identification toolbox to approximate piece-wise affine approximations of the system. A generalised Fourier series based on orthogonal polynomials is used as approximation approach. By applying neural networks with orthogonal activation functions (OAF) this black-box modelling approach identifies nonlinear models.

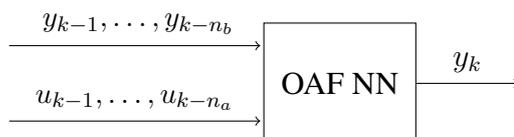


Figure 2.6: PWA approximation structure with OAF neural networks.

For this toolbox linear approximations of the Chebyshev polynomials are used as orthogonal function (Kozák and Števek, 2011; Števek et al., 2012b,a). The applied neural network consists of one hidden layer and the input values of the network have a predefined order which determines the number of past input  $n_a$  and output data points  $n_b$ , as depicted in Figure 2.6. The network applies the Chebyshev polynomials to each input, resulting in a weighted sum predicting the system output  $y_k$ . Due to the symmetrical behaviour of the Chebyshev polynomials it is possible to reduce the number of linearisation points by half and therefore decrease the number of changing points of the PWA. The accuracy of the approximation depends on the trade-off between the number of linearisation points and the complexity of the PWA models. The transition into the PWA state space enables applying existing tools such as Multi-Parametric Toolbox (MPT) (Kvasnica et al., 2004).

A similar approach is discussed in Szűcs et al. (2012). Based on the work of Kvasnica et al. (2011) a toolbox called AUTOPROX is developed. It applies a two-stage optimization-based technique to derive PWA approximations of static nonlinearities obtained from measured data. Moreover, it solves a binary optimization problem to reduce the complexity of the resulting approximation function. The resulting PWA approximations can be exported to HYSDEL for simulations.



# 3

## Artificial Neural Networks

---

Neural networks and machine learning are a central part of modern technology. Various software packages in computer science and engineering apply machine learning methods. In system identification problems for example, neural networks are used to approximate suitable model structures. This chapter gives an overview of machine learning in general and discusses artificial neural networks in detail. Hereby, the different elements of neural networks as well as the necessary training process are described.

### 3.1 Machine Learning Overview

Machine learning is a broad and continuously growing field of research. Its applications play an essential role in everyday items, starting with individual dictionaries on smart phones and browser search adjustments through to individual advertisements as well as content suggestions on streaming websites. Due to current computer performances as well as the possibility to gather and save immense amounts of data, many researchers and developers in various fields of applications focus their research intensively on machine learning approaches. These methods benefit from the fact that an exact mathematical model of the underlying process is not required. Instead, it is sufficient to provide adequate data for the training process.

An overview of machine learning is depicted in Figure 3.1. It currently comprises three main areas of application: reinforcement learning, classical learning and deep learning.

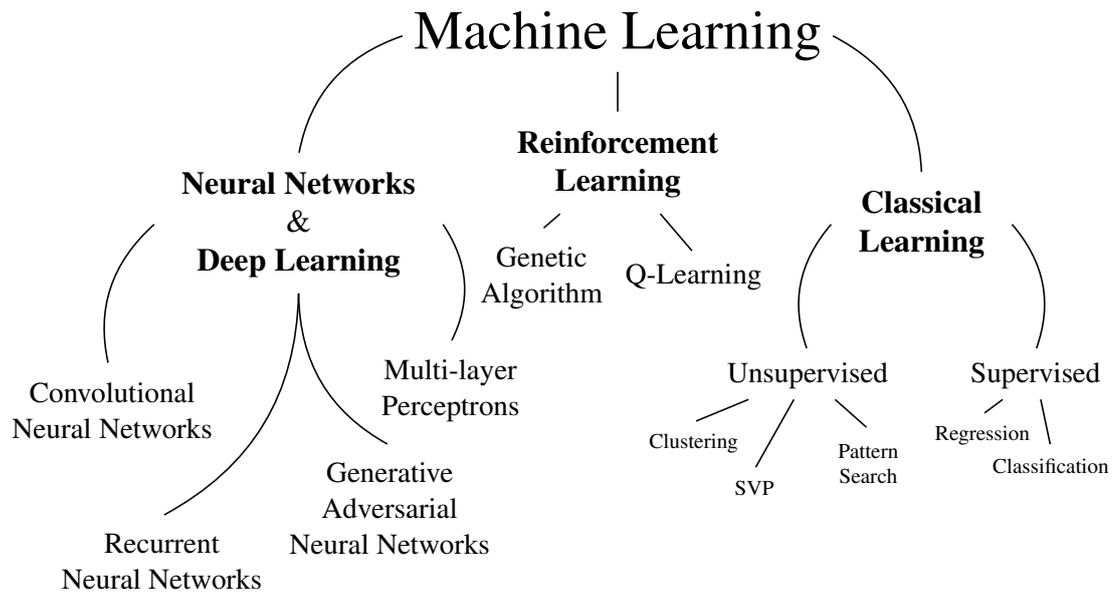


Figure 3.1: An overview of the different areas of machine learning based on Rodríguez (2019).

Reinforcement learning has shown amazing results in various applications in control engineering, see Breen et al. (2019) and Nagendra et al. (2017). Classical learning methods are mainly used for classification tasks, pattern recognition and clustering. The origin of most of these methods lies in statistics. Deep learning approaches have various fields of application, including prediction, decision making and time series approximation (Sahoo et al., 2018).

The scope of this work focuses on the application possibilities of artificial neural networks, specifically multi-layer perceptrons.

## 3.2 Structure of Artificial Neural Networks

In the early 1950s the first neural network was introduced and consisted of so-called perceptrons (McCulloch and Pitts, 1943). The task of a perceptron is to test incoming signals against a predefined threshold, whereas exceeding this value results in 1 and otherwise in 0. Hence, perceptron networks enable implementations of logical functions. The downside of the threshold is the sensitivity in regard to changes in the input

signal. Small differences in the input can result in the opposite output. To decrease the sensitivity, current neural networks apply smooth functions instead of thresholds.

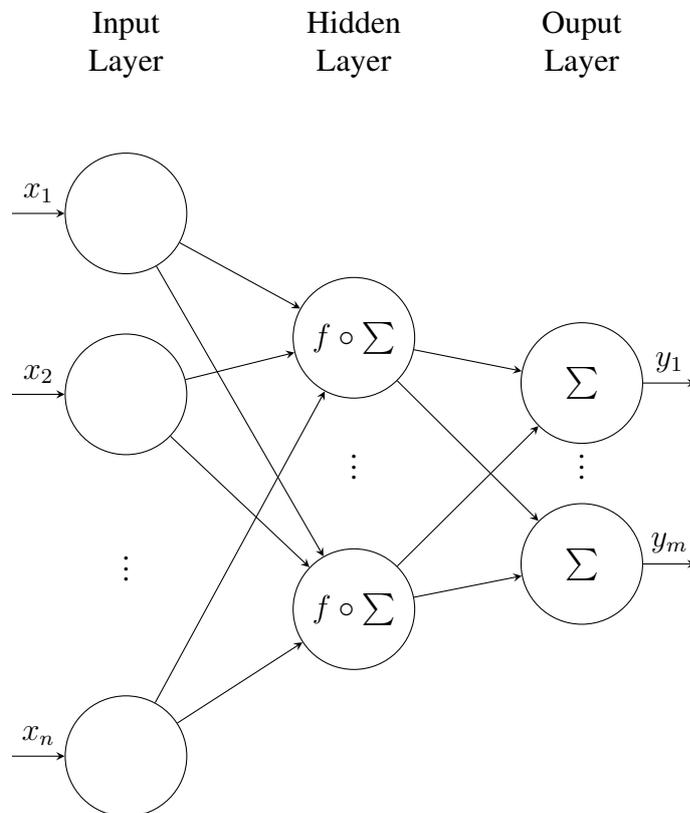


Figure 3.2: The structure of a basic artificial neural network, called multi-layer perceptron (MLP).

Despite the application purpose, the structure of a neural network always consist of three different types of layers:

- the input layer,
- the hidden layer and
- the output layer.

Each layer consists of a particular number of nodes, referred to as perceptrons or neurons. If every neuron of each layer is interconnected to every neuron in the following layer, the network is called *fully connected* (Haykin, 2009). Artificial neural networks (ANN) can be characterised by the way the signals are processed. If a neural network is

executed and the signal flows from the input layer through the hidden layer and exits at the output layer, the network structure is categorised as *feed-forward*. Such networks are often called *multi-layer perceptrons* (MLP). For most of the applications, a multi-layer feed-forward network is sufficient (Hornik et al., 1989). Another possible structure is the *recurrent neural network* (Nielsen, 2015). Hereby the edges of the network not only connect neurons of one layer to the next layer, but also back to the previous layers. Henceforth, only feed-forward networks are considered.

In Figure 3.2 a one layered feed-forward network is depicted. It consists of one input, one hidden and one output layer. The shape of the input and output layer is defined by the given inputs  $x \in \mathbb{R}^n$  and the corresponding outputs  $y \in \mathbb{R}^m$ . In contrast, there is no predefined structure for the hidden layers. The number of hidden layers as well as the amount of neurons in these layers are arbitrary.

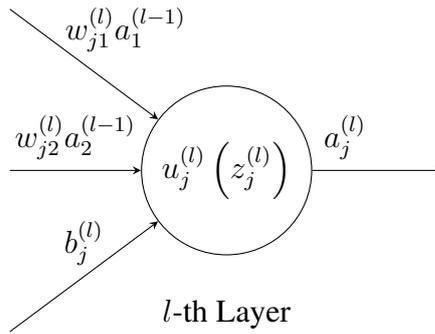


Figure 3.3: Input and output for one neuron in a multi-layer perceptron (MLP).

The connections between the neurons in the different layers are called edges. Each edge carries an individual *weight*  $w_{ij}^{(l)} \in \mathbb{R}$ , where  $l \in \mathbb{N}$  defines the target layer of the connection and  $i, j$  specify the end and starting neuron, respectively, as depicted in Figure 3.3. The value of the weight influences the incoming signal by either damping ( $w < 0$ ) or amplifying ( $w > 0$ ). The weighted sum of the inputs together with a *bias*  $b_j^{(l)} \in \mathbb{R}, j, l \in \mathbb{N}$  form the input of the neuron in most applications, henceforth denoted as  $z_j^{(l)}$  and given as

$$z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}.$$

The neurons of the hidden layer as well as the neurons of the output layer carry an *activation function*. Function  $u_j^{(l)}$  denotes the activation function for the  $j$ -th neuron in the  $l$ -th layer and can be defined as

$$u_j^{(l)}: \mathbb{R}^{n_{l-1}} \times \mathbb{R}^{n_l \times n_{l-1}} \rightarrow \mathbb{R}: (a^{(l-1)}, w^{(l)}) \mapsto a_j^{(l)}, j \in \{1, \dots, n_l\}, l \in \{2, \dots, L\},$$

where  $n_l \in \mathbb{N}$  defines the number of neurons in the  $l$ -th layer. The overall number of layers in the network is denoted as  $L \in \mathbb{N}$ . In most cases the dimension of the domain fulfils  $m = 1$ . The input vector  $x$  can be written as  $x =: a^{(1)}$ . The output of the network, which is simultaneously the output of the last layer, is labelled  $a^{(L)}$ .

Assuming a network structure as shown in Figure 3.2 with one hidden layer consisting of  $n_2$  neurons with the activation function  $f \circ \sum$ , the output of the  $j$ -th neuron in the hidden layer can be given as

$$a_j^{(2)} = u_j^{(2)}(z_j^{(2)}) = f\left(\sum_{k=0}^n w_{jk}^{(2)} x_k + b_j^{(2)}\right), j \in \{1, \dots, n_2\}. \quad (3.1)$$

In the basic perceptron, the activation function is a step function delayed by the threshold. In Table 3.1 a selection of commonly used continuous activation functions is given. For a long time, the sigmoid function was the most used activation function. Due to the fact that its range is not centred around zero it results in a slow convergence and has fallen out of popularity. In contrast, the range of the hyperbolic tangent is zero centred, hence optimisation is easier. Nevertheless, both functions suffer from vanishing gradient problem. Due to fast convergence as well as absence of the vanishing gradient problem, rectified linear units (ReLU) have become very popular.

Often the neurons are referred to as *unit* and are named after the applied activation function. Neurons applying a linear function are known as linear units and labelled  $\sum$ , whereas neurons with sigmoid functions can be labelled  $\sigma$ .

$$\begin{aligned} \sum &: u_j^{(l)}(a_j^{(l-1)}, w^{(l)}) = z_j^{(l)} \\ \sigma \circ \sum &: u_j^{(l)}(a_j^{(l-1)}, w^{(l)}) = \sigma(z_j^{(l)}) \end{aligned}$$

The structure of the hidden layers as well as the choice of the activation function depend on the application of the network and the given system specifications. For each layer in

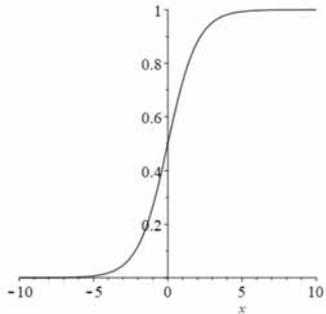
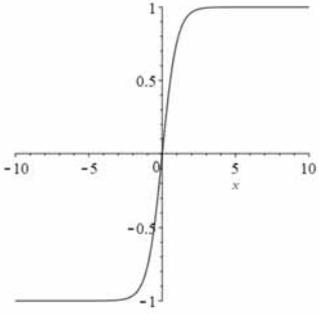
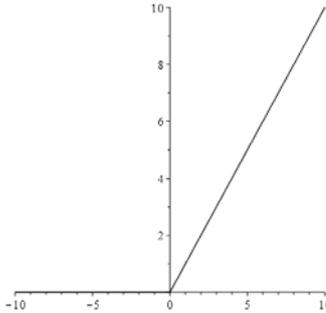
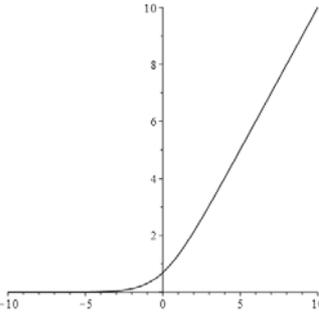
Sigmoid Function	Hyperbolic Tangent
$\sigma : \mathbb{R} \rightarrow (0, 1)$ $\sigma(x) = \frac{1}{1+e^{-x}}$	$\tanh : \mathbb{R} \rightarrow (-1, 1)$ $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
	
Rectified Linear Unit	Smooth Rectified Linear Unit
$\text{ReLU} : \mathbb{R} \rightarrow [0, \infty)$ $\text{ReLU}(x) = \max(0, x)$	$\text{ReLUS} : \mathbb{R} \rightarrow (0, \infty)$ $\text{ReLUS}(x) = \log(1 + e^x)$
	

Table 3.1: A list of commonly used activation functions in artificial neural networks.

the network, different activation functions can be applied. Due to the fact that the output values have to match the expected output data, the activation function of the neurons in the output layer are often different than in the hidden layers. The output layer usually consist of linear units, whereas the neurons in the hidden layer use the sigmoid function, hyperbolic tangent or rectified linear units (ReLU) as activation.

A MLP, as depicted above, is a fairly basic neural network. In literature different feed-forward structures can be found. In the following, two of them will be introduced in more detail.

### 3.2.1 High Order Neural Network

The general structure of a high order neural network (HONN) is similar to a MLP. It consists of one input, mostly only one hidden and one output layer. Unlike MLP networks the HONN applies so-called higher-order processing units (HPU), which realise higher-order input correlations. HPUs are usually labelled  $\Pi$ . The order of the network is defined by the highest order of the involved HPUs, whereby the order of a HPU is given as the maximum number of inputs multiplied. The output of a neural network with a HPU of third order can be given as

$$y = f \left( \sum_j w_j x_j + \sum_{j,k} w_{jk} x_j x_k + \sum_{j,k,l} w_{jkl} x_j x_k x_l \right), \quad (3.2)$$

where  $f$  is any nonlinear activation function,  $x_j$  are the inputs and  $w_{jkl}$  are the adjustable weights (Shin and Ghosh, 1991).

The Pi-Sigma network, as depicted in Figure 3.4, is an example for a HONN. It contains of linear units in the hidden layer and a HPU in the output layer of the following form

$$\begin{aligned} a^{(3)} &= u_j^{(3)} \left( a_j^{(2)} \right) := \sigma \left( \prod_{j=1}^{n_l} a_j^{(2)} \right), \\ a_j^{(2)} &= z_j^{(2)} = \sum_k w_{jk}^{(2)} a_k^{(1)} + b_j^{(2)}. \end{aligned} \quad (3.3)$$

There are of course other forms of high order neural networks, such as the HONEST and the Sigma-Pi network, see Abdelbar and Tagliarini (1996). The latter is an adaption of the Pi-Sigma network, exchanging linear units and HPUs. Both structures describe a higher-order neural network with one single hidden layer.

An advantage of the one-layered structure is a fast converging training. HONNs result in a functional structure similar to a polynomial function. The linear combination of the products of input values represents a powerful approximation tool for the functional relation between input and output (Epitropakis et al., 2010).

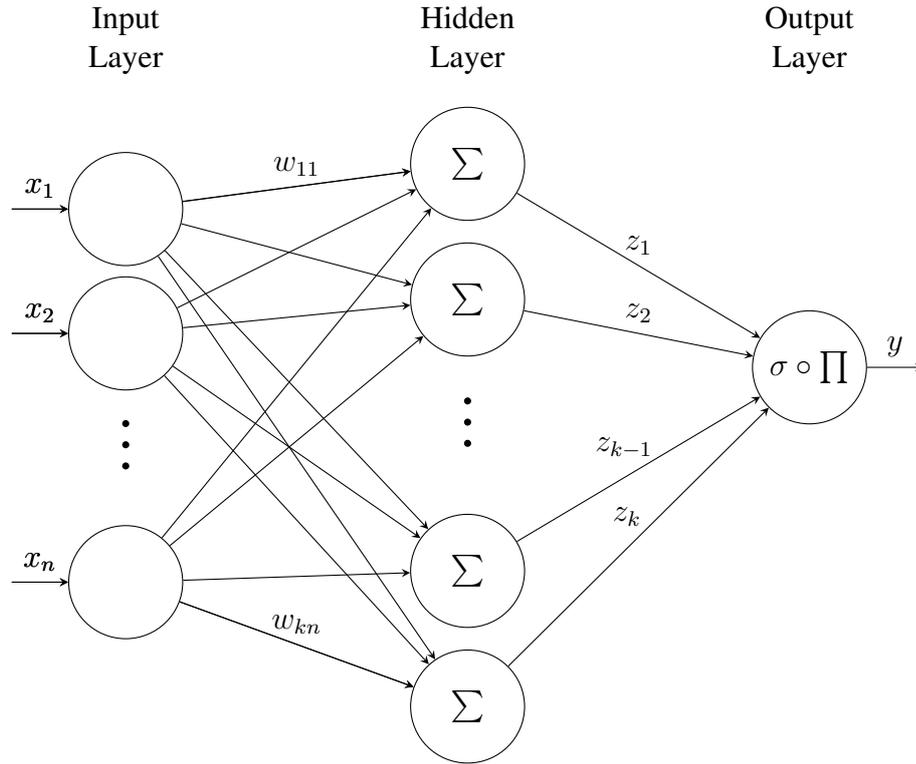


Figure 3.4: The structure of a Pi-Sigma network (Nayak, 2017).

### 3.2.2 Equation Learner

The equation learner network (EQL) emphasises the idea of high order neural networks. The intention of this network structure is to understand the functional relationship of input and output values. The goal is to enable not only an interpolation of the relation but also to gain insights to extrapolate the behaviour even for new input data. Similar to the units in the Pi-Sigma network, the EQL structure includes a multiplication unit of order two. Therefore, the order of the polynomial structure in the resulting output can be controlled by the number of hidden layers.

Similar to an MLP and Pi-Sigma network, this network defines the input signal of the neuron  $z_j^{(l)}$  as linear combination of the weighted outputs of the previous layer given as

$$z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}$$

Instead of applying the same activation function in every neuron of the hidden layers, each input signal is treated differently (Martius and Lampert, 2016). Each neuron applies one of the five distinct units given as identity, sine, cosine, sigmoid function and product function.

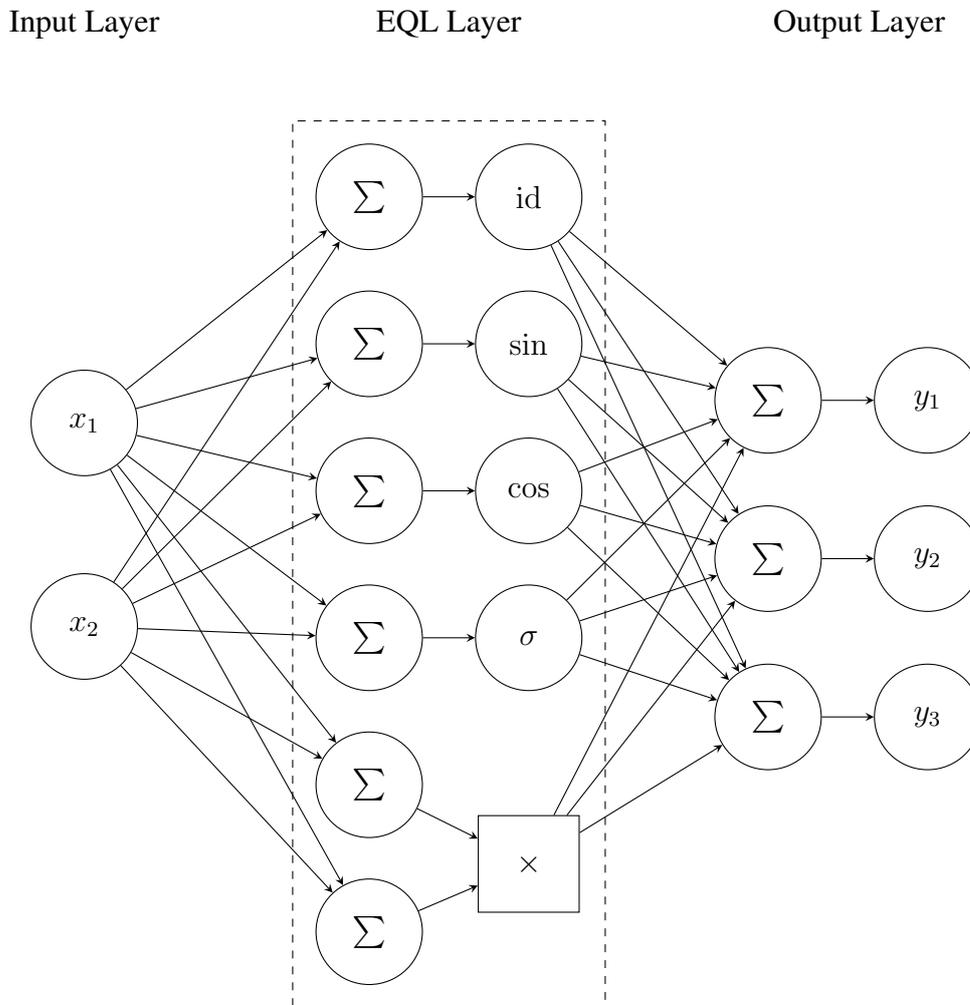


Figure 3.5: Equation learner network with one hidden layer based on Martius and Lampert (2016).

This layer structure, henceforth denoted as EQL layer, is depicted in Figure 3.5. Let  $a^{(l-1)} \in \mathbb{R}^{p+2v}$  be the output of the previous layer, a generalised EQL layer can be defined as follows

$$z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)},$$

$$u_j^{(l)}(z_j^{(l)}) := \begin{cases} z_j^{(l)}, & j \in I_1, \\ \sin(z_j^{(l)}), & j \in I_2, \\ \cos z_j^{(l)}, & j \in I_3, \\ \sigma(z_j^{(l)}), & j \in I_4, \end{cases} \quad (3.4)$$

with  $I_1 \cup I_2 \cup I_3 \cup I_4 = \{1, 2, \dots, p\}$ ,

$$g(z_j^{(l)}, z_{j+1}^{(l)}) = z_j^{(l)} \cdot z_{j+1}^{(l)}, \quad j \in \{p+1, p+3, \dots, p+2v-1\}.$$

As stated in Martius and Lampert (2016) this network is able to learn function relations and extrapolate them to unseen parts of the data space. If the underlying function of the problem can not be determined in the training process, the network results in an approximation of the exact function. In such cases, the extrapolation abilities of the network might be poor.

An improved version, the Equation Learner<sup>‡</sup> (EQL<sup>‡</sup>), has been developed, to increase the chances of the network to determine the exact function. Instead of the sigmoid unit in the hidden layer, a division unit has been added. This modified network improves accuracy in extrapolation experiments and even succeeds in performing the up-swing task for a pendulum (Sahoo et al., 2018).

### 3.2.3 Hybrid Neural Networks

Hybrid neural networks (HNN) represent no specific network structure, it is a descriptive term for the combination of different methods. On the one hand it can be used to describe the fact that a neural network is part of the model description (Psychogios and Ungar, 1992; Laursen et al., 2007; Yang et al., 2008; Sen et al., 2011; Cheng et al., 2012; Lu et al., 2016; Lian et al., 2017). On the other hand it is used if the approach combines different network structures (Gao et al., 2018).

In the following, the different interpretations of hybrid neural networks, found in literature, are summarised.

- The neural network is used to calibrate certain parameters in the model.
- The artificial neural network is embedded in a physical model to approximate a part of the model description.
- The model consists of a combination of different neural networks.

In Wernter and Sun (2000) the term *hybrid neural systems* is introduced. In addition to the categories introduced above, Wernter gives a characterisation of connections between symbolic and neural architectures of a model. It describes the interactions of different neural networks with other symbolic components, such as symbolic transformations and reasoning.

### 3.3 Training of Neural Networks

In order to obtain a good performing neural network, it is necessary to determine the biases and weights of the network structure. The iterative process determining these parameters is called training. The complexity of the model, the number of parameters as well as the accessibility, size and range of datasets affect the success of the training. In most cases, the input data is preprocessed to enable easier determination of the parameters and avoid overfitting. There are various methods for data conditioning, e.g. the minimum and maximum scaling as well as the mean and standard deviation scaling (Jayalakshmi and Santhakumaran, 2011; Shawash, 2012).

The structure of the neural network defines the number of parameters to be identified. There are two different categories of training methods: supervised and unsupervised. The latter includes a sub-characterisation, namely self-organised and reinforcement learning. Supervised training requires a predefined goal and a cost function. In case of reinforcement learning, no specific goal is set, instead certain rules and rewards are defined. In this study only supervised training methods are considered.

An iterative optimisation process uses the given dataset to improve the results of a pre-defined cost function, also referred to as loss function. At the beginning of the training the parameters are initialised randomly using a certain distribution. It is also possible to use optimisation algorithms to determine useful starting values, e.g. particle swarm optimisation (PSO), see Zaifei et al. (2009) and Ismail (2002). After the initialisation, a learning method has to be applied. The different elements of the training process are explained in the following.

### 3.3.1 Loss Function

The loss function of a neural network is similar to the cost function in optimisation problems. It evaluates the performance of the given neural network and by minimising the loss function the results are improving. Various factors have to be considered when choosing this function. First of all, the application purpose of the network has to be determined. For a regression task, one of the most common loss functions is the mean squared error (MSE) defined as

$$L_{\text{mse}} = \frac{1}{S} \sum_{i=1}^S \left( y_i - a^{(L)}(x_i) \right)^2, \quad (3.5)$$

where  $x_i \in \mathbb{R}^n$  and  $y_i \in \mathbb{R}^m$  denote one sample of the given input-output data. The size of the dataset is given as  $S$  and  $a^{(L)} \in \mathbb{R}^m$  stands for the output of the neural network. Other possible loss function are the mean absolute error (MAE), substituting the square in (3.5) with the absolute value and the mean bias error (MBE), calculating only the difference between the given output and the network result.

$$L_{\text{mae}} = \frac{1}{S} \sum_{i=1}^S \left| y_i - a^{(L)}(x_i) \right|, \quad (3.6)$$

$$L_{\text{mbe}} = \frac{1}{S} \sum_{i=1}^S \left( y_i - a^{(L)}(x_i) \right). \quad (3.7)$$

A special case of regression is the approximation of ordinary differential equations. For these problems particular structures of the loss function are suggested in literature (Lagaris et al., 1998; Chen et al., 2018; Bar-Sinai et al., 2019).

Considering a second order ODE

$$\frac{d^2y}{dx^2} = f(x, y) \quad (3.8)$$

with the according initial values  $y(0) = A$  and  $y'(0) = B$ , the approximation of the solution of equation (3.8) can be approximated using

$$\hat{y} = A + xB + x^2a^L. \quad (3.9)$$

The structure of the estimation  $\hat{y}$  is reminiscent of a Taylor approximation second order. Hereby, a specific loss function is suggested. It can be seen as a special MSE given as

$$L_{\text{ode}} = \sum_{i=1}^n \left( f(x_i, y_i) - \frac{d^2\hat{y}_i}{dx_i^2} \right)^2. \quad (3.10)$$

The network applied to calculate  $a^L$  can be of arbitrary structure.

For classification networks, a possible loss function for convex optimizers is the support vector machine (SVM) loss or Hinge loss. It is defined as

$$L_{\text{svm}} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1), \quad (3.11)$$

comparing the score of one category  $s_j$  against the sum of all other categories  $s_{y_i}$  including a safety margin (here set to 1). Another example is the cross entropy loss  $L_{\text{cross}}$ , which decreases when diverging from the actual category. It penalises classifiers that are confident about wrong predictions.

Generally, the loss function visualises a multidimensional plain. The goal of the optimisation of the parameters is to find the global minima by decreasing the loss function value.

### 3.3.2 Optimisation Algorithms

Most optimisation algorithms deal with the minimisation of convex problem definitions. Considering a two-dimensional loss function, the global minimum can be imagined as the lowest point of a paraboloid. Different optimisation algorithms, known from economics and control engineering, can be applied to adapt the parameters of the network

by minimising the loss function value. One method to determine the network parameters  $w$  and  $b$  is the Gradient Descent algorithm. Considering a loss function  $C$ , its change  $\Delta C$  can be defined as follows

$$\Delta C = \frac{\partial C}{\partial w} \Delta w + \frac{\partial C}{\partial b} \Delta b. \quad (3.12)$$

Assuming  $\Delta v = (\Delta w, \Delta b)$  and  $\nabla C = (\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b})$ , equation (3.12) can be reformulated resulting in

$$\Delta C = \nabla C \cdot \Delta v.$$

To guarantee convergence  $\Delta C$  has to be negative. Therefore  $\Delta v$  can be set to

$$\Delta v = -\eta d,$$

where  $\eta$  is called the *learning rate* and  $d$  the *direction* of the algorithm. In case of the Gradient Descent the direction  $d$  is defined as

$$d := -\nabla C.$$

In each iteration, the parameters of the network are updated as given in

$$w_k \rightarrow w_k - \eta \frac{\partial C}{\partial w_k}, \quad (3.13)$$

$$b_k \rightarrow b_k - \eta \frac{\partial C}{\partial b_k}. \quad (3.14)$$

Hereby, the learning rate  $\eta$  can be set individually. Apart from the Gradient Descent, common optimisation algorithms are the Newton's Method, Conjugate Gradient, Quasi Newton Method, Levenberg Marquardt and Adams Algorithm. For each algorithm only the definition of the direction  $d$  changes. In Table 3.2 the different optimisation directions are summarised (Haykin, 2009), where function  $f$  stands for the used loss function,  $H$  denotes the Hessian Matrix and  $J$  the Jacobian Matrix. Three different conjugate parameters  $\gamma$ , namely Fletcher-Reeves, Polak-Ribiere and Hestenes-Stiefel as well as two commonly used iterative processes for  $G$ , given as David-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS), can be found in literature.

Depending on the size of the input-output dataset, an adequate training algorithm has to be chosen. The Gradient Descent requires many iterations and therefore results in

Algorithm	Direction
Gradient Descent	$d = -\nabla f$
Newton's Method	$d = -H_f^{-1}\nabla f$
Conjugate Gradient	$d^{i+1} = g^{i+1} + d^i\gamma^i$ $d^0 = -\nabla f := g^0$
Quasi Newton's Method	$d = -G\nabla f$
Levenberg Marquardt	$d = -(J^T J + \lambda I)^{-1} 2J^T e$ $f = \sum_i e_i^2$

Table 3.2: Step directions of commonly used optimisation algorithms for neural networks.

slow convergences. Advantageously, it does not need the Hessian matrix to calculate  $d$ . Hence, it is best applied for networks with many parameters. For the Newton's Method, the Hessian matrix and also its inverse are required. However, less iterations are needed than for the Gradient Descent. In the Quasi Newton's method the expensive calculation of the inverse Hessian matrix is replaced by an approximation using the gradient. The Conjugate Gradient converges faster than the Gradient Descent and benefits from the fact, that no Hessian matrix is necessary. It is especially efficient if the neural network includes a great number of parameters. The Levenber Marquardt algorithm uses again an approximation of the Hessian matrix but requires large memory storage and is only applicable for the mean squared error as loss function.

At this point an appropriate loss function and optimisation algorithm can be chosen to determine the parameters of the network. Both elements have to be embedded into a learning method. The most common method is the back-propagation.

### 3.3.3 Back-Propagation Method

The back-propagation algorithm was originally introduced in the 1970s and its applicability was proven in Rumelhart et al. (1986). Today, the back-propagation algorithm is a well-established standard for the learning process of neural networks. The core of

back-propagation is the partial derivative of the loss function with respect to the weights and biases in the network. The derivative describes the rate by which the error changes relatively to the change in the parameters.

After initialising all parameters with random values, the derivative of the loss function is calculated according to the chosen algorithm. Starting at the end of the network and going backwards through the layers, the error of each neuron is determined. Considering a MLP with a sigmoid function as activation function, the Gradient Descent algorithm as optimiser, the steps of the back-propagation algorithm can be formulated.

**Algorithm 3.3.1** (Back-propagation of a MLP with Gradient Descent). *Considering the  $L_{\text{mse}}$  as loss function  $C$ , the learning rate  $\eta$ , the weights  $w^{(l)}$  and biases  $b^{(l)}$  as well as inputs  $x$  and outputs  $y$  the algorithm for the back-propagation can be given as*

- *Set input  $x$ :  $a^1 := x$*
- *Feed-forward:  $z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$ ,  $a^{(l)} = \sigma(z^{(l)})$ ,  $l \in \{2, \dots, L\}$*
- *Output error:  $\delta^{(L)} = (a^{(L)} - y) \circ \sigma'(z^{(L)})$*
- *Back-propagation:  $\delta^{(l)} = (w^{(l+1)})^T \delta^{(l+1)} \circ \sigma'(z^{(l)})$ ,  $l \in \{L-1, L-2, \dots, 2\}$*
- *Output:  $\frac{\partial C}{\partial b_j^{(l)}} := \delta_j^{(l)}$ ,  $\frac{\partial C}{\partial w_{ij}^{(l)}} := a_k^{(l-1)} \delta_j^{(l)}$*
- *Update parameter:  $w_{ij}^{(l)} \rightarrow w_{ij}^{(l)} - \eta a_k^{(l-1)} \delta_j^{(l)}$ ,  $b_j^{(l)} \rightarrow b_j^{(l)} - \eta \delta_j^{(l)}$*

This procedure needs to be repeated for each sample in dataset. After each round, often referred to as *epoch*, the algorithm starts again with the first sample until the network output fulfils the defined approximation requirements. It is also possible to set a maximum number of epochs. Instead of updating the parameter values for each sample separately, the mean value of a subset of samples can be considered. The size of these subsets is known as *batch size* and decreases the training time.

# 4

## Definition of the Modelling Framework

---

Based on the structure of the hybrid dynamical automaton, this chapter introduces a framework to include artificial neural networks in hybrid dynamical systems. It can be seen as an example of a hybrid neural network architecture. The framework offers a standard to apply neural networks in modelling of hybrid dynamical systems. Different cases, how neural network concepts can be included in the modelling process of hybrid dynamical systems, will be presented. First, these concepts are explored in an illustrative way, depicting the neural network approaches embedded in the hybrid automaton structure. Following that, each approach will be motivated and a detailed definition will be given.

### 4.1 Framework Overview

The illustration of the hybrid dynamical automaton in Figure 2.3 suggests three possible situations to apply neural network in hybrid dynamical models, as depicted in Figure 4.1. Case (a) shows a single neural network representing the entire hybrid system. In scenario (b) the nodes of the corresponding hybrid automaton, describing the local continuous dynamics, are replaced by neural networks. Proposition (c) applies neural networks to partly administrate the discrete changes. Henceforth, these cases are

referred to as (a) artificial hybrid model (HAM), (b) artificial hybrid dynamics (HAD) and (c) artificial hybrid events (HAE).

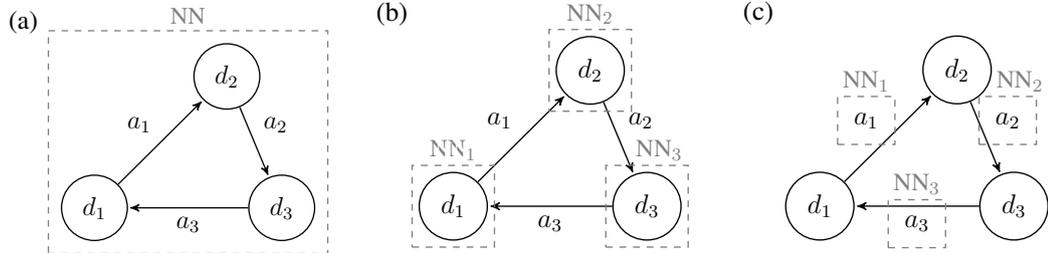


Figure 4.1: Three framework applications for including neural network concepts (red dashed rectangles) in modelling hybrid dynamical systems.

The framework also facilitates substituting individual elements. It is not necessary to replace every node and edge of the model, respectively. Regarding the modelling process of hybrid dynamical systems, the three scenarios represent every possible application of neural networks within the hybrid dynamical automaton structure. Merely combinations of case (b) and (c), replacing a mixture of nodes as well as edges, are possible.

Apart from the automaton description, there is one other possibility to utilise neural networks. In the simulation process of the local dynamics itself, hybrid neural network approaches can be applied to approximate system parameters, as introduced in 3.2.3. Since this approach affects only the simulation and not the structure of the hybrid automaton, no specific modifications of the framework are required to include such circumstances.

Each case depicted in Figure 4.1 represents a different application of the framework which will be introduced in this chapter. The framework is defined independently from any specific feed-forward network. Hence, these definitions facilitate the application of any network structure introduced in Chapter 3. By the same token, the only requirement for the chosen hybrid dynamical system is, that it can be described applying the automaton defined in Section 2.2.2. In the following, the definition of each framework application is given.

## 4.2 Artificial Hybrid Model

The first framework application discusses the replacement of an entire hybrid system by an artificial neural network. If data points are given and common dynamical model methods are not applicable, meaning that the error between model and data cannot be minimised, a hybrid model can be an appropriate approach. In case the implementation of the model will be embedded in a real time environment, a low run time of a trained ANN can be beneficial.

**Definition 4.2.1** (Artificial Hybrid Model). *An artificial hybrid model is given by the sextuple*

$$A_{\text{nn}} := (L; I; O; N; U; T),$$

where

- $L \in \mathbb{N}$  is the number of layers in the network,
- $I \subseteq \mathbb{R}^r$  is the input data,
- $O \subseteq \mathbb{R}^p$  is the output data,
- $N := (n_1, n_2, \dots, n_L) \in \mathbb{N}^L$  describes the structure of each network layer with  $n_i, i \in \{1, \dots, L\}$  defining the number of nodes in the  $i$ -th layer. Hence,  $n_1 = r$  and  $n_L = q$  refer to the input and output layer,
- $U := \{u_j^{(l)}\}_{j=1..n_l}^{l=2..L}$  is a finite set containing every unit of the network. The unit of the  $j$ -th neuron in the  $l$ -th layer is defined as

$$u_j^{(l)} : \mathbb{R}^{n_{l-1}} \times \mathbb{R}^{n_l \times n_{l-1}} \rightarrow \mathbb{R} : (a^{(l-1)}, w^{(l)}) \mapsto a_j^{(l)},$$

where  $a_j^{(l)}$  is the output of this neuron.

- $T := (M, A, C, S, V)$  characterises the training process. It contains two strings defining the used training method  $M$  and the training algorithm  $A$ . Function  $C$  denotes the applied loss function,  $S$  stands for the sample size of the dataset and vector  $V = (\alpha, \beta, \gamma) \in \mathbb{N}^3$  defines the percentages of the dataset used for training  $\alpha$ , testing  $\beta$  and verification  $\gamma$ .

**Remark.** *This formalism assumes the same method for training, testing and verification. The variable  $I$  describes the final input vector, previous data transformations are not included in the framework.*

The arguments  $M$  and  $A$  are assumed to be common methods and algorithms used for training. For  $A$ , common algorithms, such as the Gradient Descent and the Adams algorithm, are applicable optimisers for the parameter iteration. A selection of possible algorithms is given in Section 3.3.2. Back-propagation, as discussed in Section 3.3.3, is the most often used training method for feed-forward networks. Other feasible approaches for  $M$  can be found in Taylor et al. (2016).

The definition of the framework application does not answer the question if one neural network is feasible to approximate an entire hybrid system. The usability depends heavily on the requirements for the output. Two different perspectives can be distinguished. Having in mind the simulation of a physical model, the network requirements can include using the same input and output values as for a first principle model. Secondly the framework could be applied if certain behavioural aspects of the system are of interest. It can be sufficient to train the network for this specific task instead of trying to find a suitable model that describes the system's states over time. Hereby, the decision on the points of interest and omitting unnecessary variables can improve the network's accuracy which will be further discussed in Chapter 5 and 6.

Generally, any initial knowledge and system settings can be used as input data. Every initial condition as well as coefficients and parameters, knowingly influencing the system's behaviour, can be included in the training data for the neural network. The choice of the output data is more difficult. On the one hand, it can be of interest to approximate an entire time series of the model. On the other hand, considering a scenario, when only a part of the output data is significant to characterise the system's behaviour, these points can be used as training output for the neural network. Possible dataset structures that can be used to train the network are summarised in the following. *Input data* for this framework application are combinations of

- initial conditions  $x_0$ ,
- values of the state vector  $x$ ,
- inputs  $u$  and
- external variables  $w$  including system parameters  $p$ .

Hence the input can be written as  $I(x_0, x, u, w)$ . The choice of the *output data* depends on the applied network structure and its purpose. Two different cases can be distinguished:

- a single time series  $O(x_0, u, w, t) = x(t)$  for specific input values and
- output data  $O(x_0^i, u, w) = (x_*^{i1}, \dots, x_*^{im})$  representing special data points  $x_*^{ij}$  of the system for various initial values  $x_0^i$ ,  $i \in \mathbb{N}$ .

If the goal is to predict the state vector for the next time step or the system output for new input values, the training data can be restricted to a time series for specific initial values. In the second case, a variation of input values and according outputs are required for training. In this scenario the resulting network aims to approximate the system's behaviour with respect to the chosen data points. Considering hybrid systems, an example for such points are the events.

Definition 4.2.1 generalises not only the application for hybrid dynamical systems, but allows a standardised framework for feed-forward networks in general. An application example apart from hybrid dynamical systems is given in the appendix A.3. The broad application possibilities enhance interdisciplinary exchange of research results for feed-forward networks.

## 4.3 Artificial Hybrid Dynamics

In the second application of the framework, neural networks are used to approximate the dynamical behaviour of a hybrid dynamical system. Having the automaton in mind, this means that the description of each node is replaced by an artificial neural network, as depicted in Figure 4.1 (b). The framework is applicable regardless of whether one or more nodes are approximated by a neural network.

Considering a hybrid dynamical system with two nodes, it is possible, that the description of one node is either known or can be mathematically derived from basic physical laws, whereas the second process is unknown. In such cases it can be suitable to use observation data to substitute the unknown dynamical process with an ANN.

The definition of the hybrid automaton in 2.2.1 and the previous definition 4.2.1, can be combined to meet these requirements.

**Definition 4.3.1** (Artificial Hybrid Dynamics). *An automaton for Artificial Hybrid Dynamics is described by the tuple*

$$A_{\text{had}} := (D_{\text{an}}; X; B; W_{\text{an}}; E; \text{Inv}_{\text{an}}; \text{Act}_{\text{an}})$$

where

- $D_{\text{an}} := D_h \cup D_a$  is the finite set of discrete distinguishable states, where  $D_h$  belongs to the hybrid system and  $D_a$  to the network. Hence,  $D_h \cap D_a = \emptyset$  is fulfilled,
- $X$  is the continuous state space containing all continuous states  $x \in X \subseteq \mathbb{R}^n$ ,
- $W_{\text{an}} = W_h \cup W_a$  contains all external variables, where  $W_a$  refers to the external variables of the neural networks and  $W_h$  to the variables of the DAE,
- $B$  is a finite set of symbols,
- $E$  is a finite set of events. Every event  $e \in E$  is defined by  $e = (d; b; G_{d,d'}; J_{d,d'}; d')$ , labelled  $b \in B$  and depicted as an edge interconnecting different locations.  $G_{d,d'}$  symbolises a finite convex subset of  $X$ , the so called guard region.  $J_{d,d'}$  defines the jump relation for the transition from location  $d$  to  $d'$  on a subset of  $X \times X$  with  $d, d' \in D$ .
- $\text{Inv}_{\text{an}}: D_{\text{an}} \rightarrow \mathcal{P}(X)$  is called the location invariant and at location  $d \in D_h$  the corresponding state  $x$  must satisfy  $x \in \text{Inv}(d)$ ,
- $\text{Act}_{\text{an}}: D_{\text{an}} \rightarrow \mathcal{F}_{\text{an}}$  maps each location  $d \in D_h$  to the differential algebraic equation triplet  $(f_d, g_d, h_d)$ , as defined in (2.7), and  $d \in D_a$  to the neural network  $A_{\text{an}}^d$ . The neural network is given as sextuple  $A_{\text{an}}^d = (L^d; I^d; O^d; N^d; U^d; T^d)$ , as defined in 4.2.1.

**Remark.** Replacing one dynamical process with a neural network leads to  $|D_a| = 1$ . If the dynamics are entirely unknown  $D_h = \emptyset$  is valid. In some cases the communication spaces can overlap, hence  $W_h \cap W_a \neq \emptyset$  is possible.

If the dynamics of at least one node are unknown but data points are available, this approach enables an approximation of the local system behaviour. Compared to the first case, the event functions or at least the mathematical definition of the events are known.

Depending on the required output values different structures of neural networks are applicable. A time series with a conjecture about the order of the system can be approximated applying the network structure especially designed for ODE, see equation (3.9). If no pre-knowledge about the order of the system is considered, an ordinary MLP can be applied. An evaluation of the available datasets, regarding size and quality, might result in an inapplicability of multi-layer perceptron. The equation learner network structures presented in Section 3.2.2, can pose a suitable alternative. Their architecture supports time series approximation even if the structure of the local dynamical system is unknown. The *input data* for this framework application is a combination of

- the given initial conditions  $x_0$ ,
- state vector values  $x$ ,
- input  $u$  and
- external variables  $w$  including system parameters  $p$ .

The output of most first principle models is the state vector. Hence, the options for *output data* for the network training include the following:

- a specific time series  $O(x_0, u, w, t) = x(t)$  and
- the state vector  $O(x_0^i, u, w, t) = x_e^i$  initiating an event, given various initial values  $x_0^i$ ,  $i \in \mathbb{N}$ .

Instead of defining the mathematical and physical background of the system the modelling process focuses on narrowing down the network structure with the best approximation results. The accuracy of these results depends on the size and quality of the dataset for training and the requirements for the output data.

## 4.4 Artificial Hybrid Events

The last application of the framework proposed in this thesis, applies neural networks to administrate the event handling. The definition is given independently from any specific feed-forward network structure and facilitates every event type discussed in 2.2.3. The characterisation of the event, as introduced in the hybrid dynamical automaton, requires

at least three elements. The mapping  $\text{Act}$  is considered to be a-priori knowledge since the local processes and descriptions are accessible. The guard  $G$  defines the set of states initiating the events, whereas the jump map  $J$  executes the actual changes happening during the transition from one local description to the next.

Considering that the local dynamics are known, the jump relation is already implicitly defined. An investigation of the given dynamical descriptions determines a-priori if  $J$  describes a coordinate transform. A neural network approximation of that transform is unnecessary. Secondly, assuming a hybrid system with changing parameters and variables the dynamical behaviour of the system can be characterised mathematically and the jump relation is a consequence of the model description. Hence, the following framework focuses on the approximation of the guard region.

**Definition 4.4.1** (Artificial Hybrid Events). *An automaton including artificial hybrid events is defined as*

$$A_{\text{hae}} := (D; X; B_{\text{ae}}; W_{\text{ae}}; E_{\text{ae}}; \text{Inv}; \text{Act}),$$

where

- $D$  is the finite set of discrete states,
- $X$  is the continuous state space containing all continuous states  $x \in X \subseteq \mathbb{R}^n$ ,
- $W_{\text{ae}} = W_h \cup W_a$  contains all external variables, where  $W_a$  refers to the external variables of the neural networks and  $W_h$  to the variables of the DAE,
- $\text{Inv}: D \rightarrow \mathcal{P}(X)$  is called the location invariant, where  $\mathcal{P}(X)$  is the power set of  $X$ . At location  $d \in D$ , the corresponding state  $x$  must satisfy  $x \in \text{Inv}(d)$ ,
- $B_{\text{ae}} := B_h \cup B_a$  is the finite set of symbols, where  $B_h$  belongs to the known transitions and  $B_a$  to the transitions with network approximation. Hence, they fulfil  $B_h \cap B_a = \emptyset$ ,
- $E_{\text{ae}}$  is a finite set of events. Every event  $e \in E$  is defined by  $e = (d; b; G_{\text{ae}}; J_{d,d'}; d')$  with  $b \in B_{\text{ae}}$  labelling the edge. The function  $G_{\text{ae}}$  maps to the neural network  $A_{\text{ae}}^b$  for  $b \in B_a$ , and to the guard  $G_{d,d'}^b$  for  $b \in B_h$ . The neural network is given as tuple  $A_{\text{ae}}^b = (L^b; I^b; O^b; N^b; U^b; T^b)$ , as defined in 4.2.1.
- $\text{Act}: D \rightarrow \mathcal{F} : d \mapsto (f_d, g_d, h_d)$  maps each location  $d \in D$  to a set of differential algebraic equations, as defined in (2.7).

In definition 4.4.1, the local descriptions and their state variables are known, whereas the guard region has to be determined. In contrast to both previously explained applications, the output data for the training set of the ANN has to be defined differently. There is no feasible output of the system, which can be directly applied for training. Instead the data has to be analysed and classified. Two different categories can be defined:

$$O_j = \begin{cases} 0, & \text{no event occurred at } x_j, \\ 1, & \text{an event occurred at } x_j. \end{cases}$$

With this classification a training set for the network  $A_{ae}^b$  can be given. The *input data* for training the neural network include

- the state vector of the system  $x(t), t \in T \subset \mathbb{R}^+$ ,
- the given initial conditions  $x_0$ ,
- input  $u$  and
- external variables  $w$  including any system parameters  $p$ .

Hence, the input can be given as  $I(x_0, x, u, w, t)$ . The *output data* for the training process is  $O \in \{0, 1\}^q, q \in \mathbb{N}$ , where  $q$  depends on the number of classified state values included in the dataset. After embedding the trained network into the model structure, at each time step the state vector of the dynamical process is classified by the network and an event is initiated if the state vector is labelled 1.

Both previous definitions attempt to approximate the relation between input and output values. In contrast, the approximation of the event guard represents a classification task. Hence, neural network structures such as EQL and HONN are not applicable instead MLP suited for classification tasks are required for this framework application.



# 5

## Framework Application

---

For the application of the framework introduced in Chapter 4 two different hybrid dynamical systems are chosen, each a representative of a different example class. These examples are used to investigate the applicability of the framework scenarios. The experimental results are analysed and compared with solutions of common modelling approaches.

### 5.1 Definition of Application Examples

The case study consists of the bouncing ball and the pendulum with free fall phase. In the following, both example descriptions are motivated and a formulation of the hybrid dynamical automata are given. The examples differ in regard to the dimension of the state space as well as the number of nodes describing the local dynamics of the system.

#### 5.1.1 Bouncing Ball

The bouncing ball, as defined in Körner and Breiteneker (2016) and Altın et al. (2018), describes a ball, bouncing off the ground. The state variable of interest is the height over time  $t$ . The acting force in the observed system is the gravity, accelerating the ball to the ground. Thus, the ODE of the dynamical behaviour of the bouncing ball can be described as

$$\ddot{h}(t) = -g, \tag{5.1}$$

where  $g$  is the gravitational constant. Considering an initial height  $h_0$  and velocity  $v_0$ , two state variables can be defined, namely the height  $h(t) =: x_1(t)$  and the velocity  $x_2(t) := \dot{x}_1(t)$ . Hence, using the state vector  $x = (x_1, x_2)^T \in \mathbb{R}^2$  equation (5.1) can be transformed into a state space description resulting in

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ -g \end{pmatrix}, \quad x(0) = \begin{pmatrix} h_0 \\ v_0 \end{pmatrix}. \quad (5.2)$$

An advantage of the chosen academic example resides in the existence of an analytical solution given as

$$x(t) = \begin{pmatrix} -\frac{g}{2}t^2 + v_0t + h_0 \\ -gt + v_0 \end{pmatrix}. \quad (5.3)$$

In Figure 5.1 (a), the height of the ball is depicted over time. Two different processes can be distinguished, the flying and falling phase of the ball, respectively and the bounce. The latter affects the behaviour of the ball and therefore defines the state event of the system.

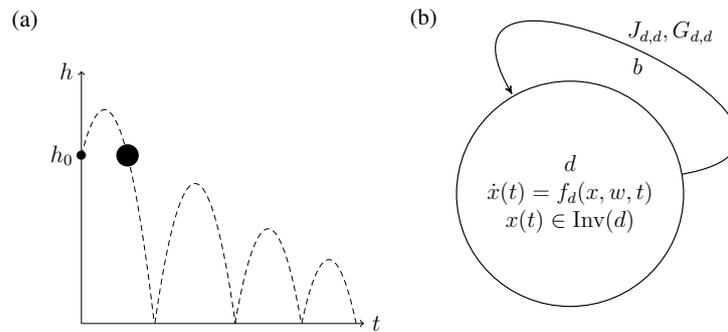


Figure 5.1: (a) Height of the ball over time. (b) Hybrid dynamical automaton of the bouncing ball.

After the event, the dynamical behaviour of the ball can again be described by equation (5.1). Based on definition 2.2.1 the elements of the hybrid dynamical automaton for the

bouncing ball can be formulated. The function  $\text{Act}$  is defined as

$$\text{Act}(d) := F_d = (f_d, h_d),$$

with

$$F_d = \begin{cases} \dot{x}(t) = f_d(x, w, t), \\ y(t) = h_d(x, w, t) = x(t). \end{cases} \quad (5.4)$$

The state space description in equation (5.2) defines function  $f_d$ . In this example an ODE system is sufficient and no additional algebraic equation  $g_d$  is required. Due to the fact that the free fall phase is interrupted when the ball reaches the ground, the *event guard*  $G_{d,d}$  can be defined. An event occurs if the state vector fulfils

$$x \in G_{d,d} := \{x(t) \in \mathbb{R}^2 : x_1(t) = 0 \wedge x_2(t) \leq 0\}. \quad (5.5)$$

Regarding the event formulations discussed in section 2.2.3, this state event can be given in all three forms. The *event function* can be defined as  $e_b(x) := x_1$  with  $e_b(x) = 0$  initiates the event, whereas the threshold for the event can be given as  $\Delta x_1 := 0$ .

Model descriptions are often implying a certain degree of abstraction. In this application, the bounce is described by a simplified assumption without modelling the physical process in detail. When the ball hits the ground the behaviour of the ball is affected. The friction is realised as simple damping factor  $\lambda$  with  $0 < \lambda < 1$ . The reflection of the ball results in inverting the velocity component. Hereby, the time delay due to any deformation work is neglected. Thus, the *jump map*  $J_{d,d}$  at the event can be characterised by the linear transform

$$x(t^+) = J_{d,d}(x(t^-)) := \begin{pmatrix} 1 & 0 \\ 0 & -\lambda \end{pmatrix} x(t^-). \quad (5.6)$$

The elements defined above combined with the set of external variables

$$W = \{\lambda, g, v_0, h_0\}$$

including the one-dimensional parameter vector  $p = g$  result in an hybrid dynamical automaton  $A_h$  in the form

$$A_h = (\{d\}, \mathbb{R}^2, \{b\}, W, E, \text{Inv} \subset \mathbb{R}^2, \text{Act}), \quad (5.7)$$

with  $E = (d, b, G_{d,d}, J_{d,d}, d)$ .

In Figure 5.1(b) an illustration of the automaton can be found.

### 5.1.2 Rotating Pendulum

The rotating pendulum considered in this work is a realisation of a thread pendulum. Here a mass, imagined as a point and suspended at one point by means of a massless pendulum rod, can swing back and forth in a vertical plane. The plane pendulum is a special case of the spherical pendulum. Assuming a non-elastic thread of length  $l$ , the mass can only reach points in the vertical plane within the circle defined by  $l$ . Therefore the overall state space  $X$  for this hybrid model can be defined as

$$X = \left\{ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2 : x_1^2 + x_2^2 \leq l^2 \right\}.$$

An observation of the system suggests three different model descriptions. First of all there is the typical swinging of the object, which can be called oscillation. Its mathematical description is given as

$$\ddot{\varphi} = \frac{g}{l} \sin \varphi - k_p \dot{\varphi}, \quad (5.8)$$

where  $k_p$  denotes the damping factor. The state vector consists of the angle  $\varphi$  and the angular velocity  $\dot{\varphi}$  and is denoted as  $x_p = (\varphi, \dot{\varphi})^T$ . The oscillation period is determined exclusively by the length of the pendulum and the prevailing gravitational acceleration  $g$ . For small oscillations, the period of oscillation is almost independent of the magnitude of the amplitude and shows an almost harmonic oscillation. If enough angular velocity is applied, the large excitations lead to "flashovers". Then the pendulum body orbits periodically on the circle in the vertical plane. To an observer the movement looks different, but the mathematical description is the same as for the oscillation described in (5.8). The force to keep the pendulum mass in orbit is given by the Lagrange function

$$F(x_p) = F(\varphi, \dot{\varphi}) := -mg \cos \varphi + mL\dot{\varphi}^2. \quad (5.9)$$

If  $F$  decreases and reaches zero, the pendulum body leaves the orbit and enters a free fall phase. This change in the dynamical behaviour marks the first state event of the model. The *guard region* for this event can be defined as

$$G_{p,r} = \{x_p \in \mathbb{R}^2 : F(x_p) = 0\}. \quad (5.10)$$

The changing model description results in a different state vector. The state variables for the free fall phase are denoted as

$$x_r = (x_1, \dot{x}_1, x_2, \dot{x}_2)^T \in \mathbb{R}^4,$$

with  $(x_1, x_2)$  begin the coordinates of the ball in the vertical plane and  $\dot{x}_1, \dot{x}_2$  the according velocities. The dynamical behaviour of the free fall can be described by an ODE system of the following form

$$\begin{cases} m\ddot{x}_1 = -k_r \dot{x}_1, \\ m\ddot{x}_2 = -mg - k_r \dot{x}_2. \end{cases} \quad (5.11)$$

Equation (5.11) can be rewritten as linear state space representation

$$f_r(x_r, w_r, t) := \dot{x}_r(t) = \frac{1}{m} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & -k_r & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -k_r \end{pmatrix} x_r(t) + \begin{pmatrix} 0 \\ 0 \\ 0 \\ -g \end{pmatrix}, \quad (5.12)$$

with the initial condition given as  $x_r(0) = (x_1^0, v_{x_1}, x_2^0, v_{x_2})^T$ . The vector  $w_r$  of the external variables is defined as

$$w_r = (g, k_r, x_1^0, v_{x_1}, x_2^0, v_{x_2})^T.$$

Due to the different state vector for the free fall phase, the *jump map*  $J_{p,r}$  is a coordinate transform which simultaneously sets the required initial conditions.

$$J_{p,r}(x_p(t)) := \begin{pmatrix} l \sin \varphi(t) \\ l \cos \varphi(t) \cdot \dot{\varphi}(t) \\ l \cos \varphi(t) \\ -l \sin \varphi(t) \cdot \dot{\varphi}(t) \end{pmatrix} = x_r(t). \quad (5.13)$$

The change in the model description is defined by the mapping

$$\text{Act}(r) = (f_r, h_r) \quad \text{with} \quad h_r = x_r(t).$$

In the free fall the pendulum body keeps on falling until the thread tightens at its maximum length  $l$  and the coordinates of the mass are again on the circular path. Thus, the second *event guard*  $G_{r,p}$  can be defined as

$$G_{r,p} = \{x_r \in \mathbb{R}^4 : x_1^2 + x_2^2 = l^2\}. \quad (5.14)$$

The mathematical description of the behaviour switches back to equation (5.8) and can be formulated as

$$F_p = \begin{cases} \dot{x}_p(t) = f_p(x_p, w_p, t) := (\dot{\varphi}, \frac{g}{l} \sin \varphi - k\dot{\varphi}), \\ 0 = g_p(x_p, w_p, t) := l^2 - x_1^2 - x_2^2, \\ y_p(t) = h_p(x_p, w_p, t) = x_p(t). \end{cases} \quad (5.15)$$

The output  $y_p$  is the state vector. For both models the finite set  $W$ , the parameter vector and the external variable vector are given as

$$\begin{aligned} W &= \{x_1^0, v_{x_1}, x_2^0, v_{x_2}, \varphi_0, \omega_0, g, k_p, k_r\}, \\ w_p &= (g, k_p, \varphi_0, \omega_0), \\ w_r &= (g, k_r, x_1^0, v_{x_1}, x_2^0, v_{x_2}), \end{aligned}$$

where  $\varphi_0$  denotes the initial angle of the ball and  $\omega_0$  the according angular velocity. The definition of the dynamics in the free fall can be formulated as

$$F_r = \begin{cases} \dot{x}_r(t) = f_r(x_r, w_r, t), \\ y_r(t) = h_r(x_r, w_r, t) = x_r(t). \end{cases} \quad (5.16)$$

Equation (5.12) defines the ODE system  $f_r$  where the output  $y_r$  is the state of the system. Putting the introduced elements together, the hybrid automaton is given as

$$\begin{aligned} A_h &= (\{p, r\}, X, \{b_1, b_2\}, W, E, \text{Inv} \subset \mathbb{R}^2, \text{Act}), \\ E &= \{(p, b_1, G_{p,r}, J_{p,r}, r), (r, b_2, G_{r,p}, J_{r,p}, p)\}, \\ \text{Inv}(p) &= \{(x_1, x_2)^T \in \mathbb{R}^2 : x_1^2 + x_2^2 = l^2\}, \\ \text{Inv}(r) &= \{(x_1, x_2)^T \in \mathbb{R}^2 : x_1^2 + x_2^2 < l^2\}, \\ \text{Act}(i) &= F_i, i \in \{p, r\}. \end{aligned} \quad (5.17)$$

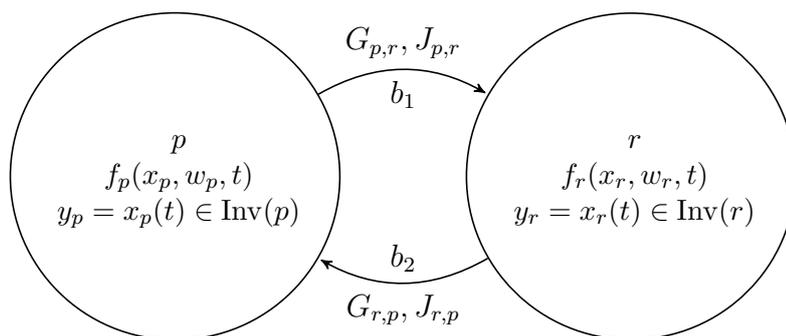


Figure 5.2: Hybrid automaton of the pendulum with free fall.

An illustration of the automaton can be found in Figure 5.2.

In conclusion, the rotating thread pendulum can be describe with two different state models, the oscillation and the free fall. If there is a high initial velocity, the oscillation can turn into a rotation in the orbit. Due to friction the driving forces are decreasing until the movement transitions into a free fall. Depending on the location where the Lagrange function reaches zero, the duration of the free fall varies and can be nonexistent. The final phase is always a damped oscillation around the equilibrium.

## 5.2 Artificial Hybrid Model Application

For the first framework application the bouncing ball example is considered. In this scenario the entire hybrid model, as given in equation (5.7), is replaced by a neural network, considering measurement data instead of a-priori knowledge. As mentioned before, the output data of the training set depends on the research question regarding the system's behaviour. Assuming, the main focus is the event localisation, then the approximation of the event time and its state vector are of interest. A possible approach is to identified the first three events in the measurement data and define them as output data for the network. The external variables and parameters of the system, consisting of the initial inputs  $h_0$  and  $v_0$  as well as the gravitational constant  $g$ , are possible input values, as discussed in Section 4.2.

	Dataset BB Train	Dataset BB Validate	Dataset BB Extra
$h_0$	$h_0 \in [0, 15]$	$h_0 \in [0, 20]$	$h_0 \in [20, 30]$
$\Delta h_0$	0.5	0.2	0.1
$v_0$	$v_0 \in [-10, 10]$	$v_0 \in [-10, 15]$	$v_0 \in [10, 20]$
$\Delta v_0$	0.5	0.2	0.1
$\Delta t$	0.001	0.005	0.01
$S$	1271	1989	10201

Table 5.1: Dataset definitions for the bouncing ball example.

For the following experiments the damping factor for the bounce is set as constant value  $\lambda = 0.8225$ . The existing analytical solution enables an arbitrary size of input-output datasets with varying initial condition. The variation of the initial conditions is given as  $h_0 \in [0, 15]$ ,  $v_0 \in [-10, 10]$ . The step size for the initial conditions as well as the time step for the analytical solution are set accordingly to Table 5.1. There, the dataset definitions for the following experiments are listed. In addition to the dataset for the training process an extended dataset *Dataset BB Extra* is created. It consists of data for new initial conditions and enables an evaluation of the extrapolation capabilities of the network structures. In addition to the extrapolation possibilities its application can also reveal if the training of the network parameters results in overfitting.

For each initial condition  $[h_0, v_0]$  in the Dataset the first three events ( $E_1, E_2, E_3$ ) of the according time series are stored as output. To reconstruct measurement data a white noise  $\mathcal{N}(0, 0.001)$  is added to the output. For each output the input data is defined as the initial values and the gravitational constant  $g$ . The resulting training data is permuted and divided, according to the percentages given in  $V = (\alpha, \beta, \gamma)$ , into training, testing and verification.

Applying the framework, the model for the bouncing ball can be given as

$$\begin{aligned}
 A_{\text{ham}} &= (4, I, O, N, U_{\text{mlp}}, T) \text{ with} & (5.18) \\
 I &:= (h_0, v_0, g), \\
 O &:= (E_1, E_2, E_3), \\
 U_{\text{mlp}} &:= \{u_j^{(l)}\}_{j=1, \dots, n_l}^{l=2, \dots, L} \text{ with}
 \end{aligned}$$

$$u_j^{(l)}(z_j^{(l)}) := \begin{cases} \sigma(z_j^{(l)}), & l \in \{2, \dots, L-1\}, j \in \{1, \dots, n_l\}, \\ z_j^{(l)}, & l = L, j \in \{1, \dots, n_L\}, \end{cases}$$

$$T := (M, A, C, S, V),$$

$$C := L_{\text{mse}}, V := (\alpha, \beta, 0),$$

where  $M$  is the back-propagation method. Two different optimisation algorithms  $A \in \{\text{Levenberg-Marquardt, Adam-Algorithms}\}$  and two different layer structures

$$N \in \{(3, 3, 3, 3), (3, 6, 6, 3)\}$$

are compared for the MLP. The training set can also be used to train and test an EQL set-up given as

$$A_{\text{ham}}^i = (L_i, I, O, N_i, U_{\text{eql}}, T), \quad i \in \{1, 2\} \quad \text{with} \quad (5.19)$$

$$I := (h_0, v_0, g),$$

$$O := (E_1, E_2, E_3),$$

$$U_{\text{eql}} := \{u_j^{(l)}\}_{j=1, \dots, n_l}^{l=2, \dots, L} \quad \text{with } l \in \{1, \dots, L-1\},$$

$$u^{(l)}(z_j^{(l)}) := \begin{cases} z_j^{(l)}, & j = 1, \\ \sin(z_j^{(l)}), & j = 2, \\ \cos(z_j^{(l)}), & j = 3, \\ \sigma(z_j^{(l)}), & j = 4, \\ z_j^{(l)} \cdot z_{j+1}^{(l)}, & j = 5, \end{cases}$$

$$u^L(z_j^{(L)}) := z_j^{(L)} \quad \forall j,$$

$$z_j^{(l)} := \sum_k w_{j,k} a_k^{(l-1)} + b_j^{(l)} \quad \forall l,$$

$$T := (M, A, C, S, V),$$

$$C := L_{\text{mse}}, V := (\alpha, \beta, 0),$$

with varying layers defined as

$$L \in \{3, 4\} \quad \text{with } N \in \{(3, 5, 3), (3, 5, 5, 3)\}.$$

MSE $\alpha$	MSE $\beta$	MSE $\gamma$ $E_{1-3}$	MSE $\gamma$ $E_{3-6}$	Method	Layer	$\alpha$	Epoch
3.6175 e-4	4.2658 e-4	6.6438 e-4	3.6754 e-2	EQL	2(5)	80	1000
3.2282 e-4	3.9579 e-4	5.3113 e-4	3.8308 e-2	EQL	2(5)	20	1000
7.5649 e-4	8.4607 e-4	8.4836 e-4	3.8488 e-2	MLP	2(6)	50	1000
4.5263 e-4	5.3456 e-4	4.8243 e-4	4.1186 e-2	MLP	2(6)	80	1000
4.0074 e-4	3.2488 e-4	4.7142 e-4	4.5297 e-2	EQL	2(5)	80	500
3.6031 e-3	3.6371 e-3	3.6309 e-3	5.0753 e-2	EQL	1(5)	80	1000
1.7402 e-2	1.8055 e-2	1.5773 e-2	7.0394 e-2	EQL	2(5)	50	100
2.7812	2.8264	2.7697	2.2196	MLP	2(3)	50	100

MSE $E_{1-3}$	MSE $E_{3-6}$	MSE Extra $E_{1-3}$	MSE Extra $E_{3-6}$	Method	Layer
3.0335 e-03	5.9518 e-02	4.5410 e-02	1.5169 e-01	MLP	1(12)
1.7923 e-01	3.9186 e-01	2.3156 e-02	1.7219 e-01	MLP	1(6)
2.5409 e-03	5.7638 e-02	2.1195 e-01	1.9962 e-01	MLP	1(9)
2.0243 e-02	1.5326 e-01	2.3040 e-01	2.2946 e-01	MLP	1(3)

Table 5.2: Comparison of different MLP and EQL settings for the approximation of the first three events and the extrapolation of the following three events for the bouncing ball.

In Table 5.2 the mean squared errors of the different neural networks are given. In the upper tabular format the MSE  $\alpha$  defines the error in training, where value  $\alpha$  denotes the percentage of the training set used. MSE  $\beta$  denotes the loss function value for the rest of the training set ( $\beta = 100\% - \alpha$ ). For MSE  $\gamma$  another training set, as stated in Table 5.1, is used. Hereby, not only the first three events are evaluated but also the following three events are predicted using the network. The column Layer defines the number of hidden layer(s) of the network and the value in the brackets defines the comprised neurons. Focusing on the last two rows, it is obvious that even with a small number of epochs the results of the EQL network are acceptable. The MLP needs at least 500 Epochs to show similar mean squared error values. The top 4 rows show that the two-layered EQL can be trained with only 20% of the training data and still reach an adequate approximation.

In the lower part of Table 5.2 the extrapolation capabilities of one-layered MLP networks are tested. The dataset used for MSE Extra consists of unseen samples. The network is executed to calculate the first three events as well as the following three events of the time series for each sample in the dataset. The results show, that this reduced output structure enables ordinary one-layered MLP networks to extrapolate event times for unseen initial conditions.

In some cases, the event location is not sufficient to answer questions about the system's behaviour, instead a time series is required. The MLP for this purpose can be defined similar to equation (5.18), except the layer structure and the dataset have to be modified. The input changes to  $I = (t_0, \dots, t_n)$  and the output consists of the according response of the system  $O = (x_0, \dots, x_n)$ . The second dataset of Table 5.1 contains  $S = 1989$  different initial values and for each of them a time series with 2001 time steps is available.

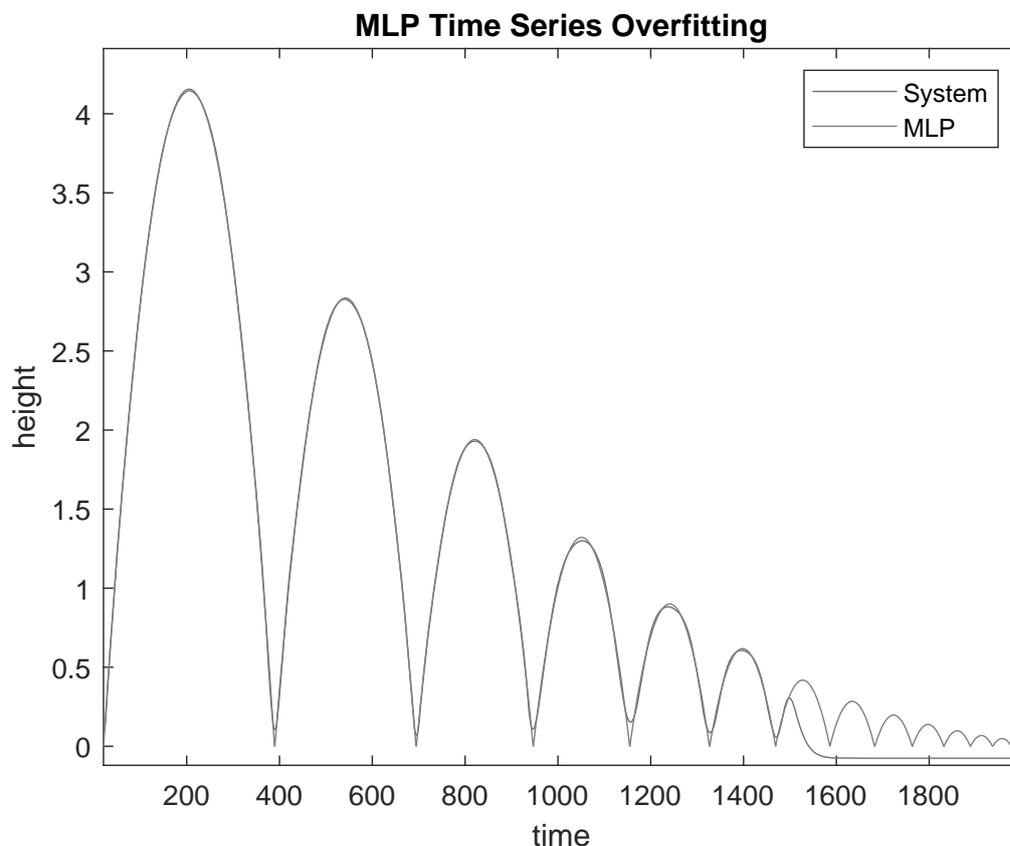


Figure 5.3: A MLP approximation of the time series for a specific initial value.

The first approach is a MLP with the layer structure  $N = [1989, 50, 1989]$ . Since MLP are not appropriate for learning the functional relation the training error results in  $L_{\text{mse}} \approx 173$ . Hence, a network using the entire time series as one input sample is not

useful. In the second approach, only one time series for one specific initial value is considered. In this case the input is defined as  $I = t_i$  and the according output is  $O = x_i$ . The according network structure is given as  $N = [1, 20, 1]$ . The defined MLP is able to learn the parameters and succeed the training process with  $L_{\text{mse}} < 8.85 \text{ e } -4$  for training, testing and validation. Though, as expected, this network is not able to extrapolate the behaviour of the system for future time instances which are not included in the training set, see Figure 5.3. An application of the EQL leads to a similar phenomenon, as shown in Figure 5.4.

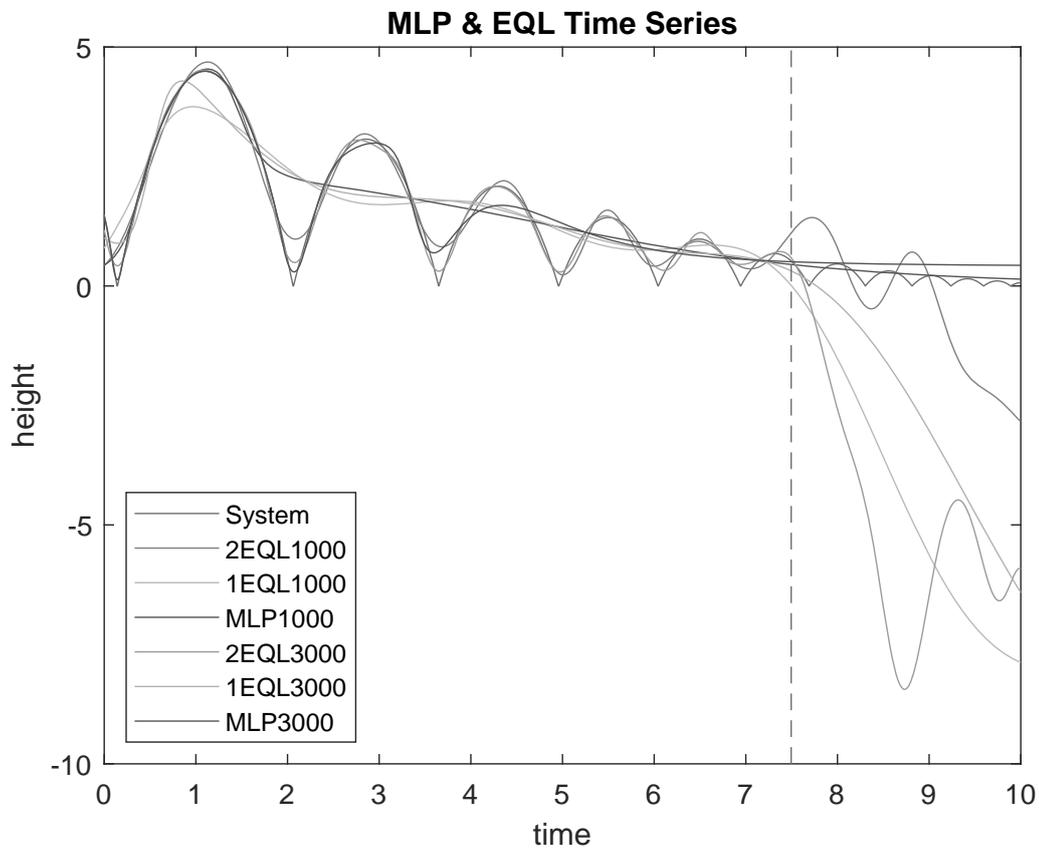


Figure 5.4: A EQL approximation of the time series for a specific initial value.

In Table 5.3 the results of EQL and MLP approaches are listed.  $\text{MSE } \alpha$  is the resulting error for the training using  $\alpha\%$  of the data.  $\text{MSE } \beta$  denotes again the MSE value for

the unused training data. Since only a part of the time series is taken as training data MSE  $TS_1$  defines the error resulting of the network evaluation for the entire available time series. Additionally a second time series for another set for initial conditions close to the original initial condition, is used for validation (MSE  $TS_2$ ). The last calculated error value MSE  $TS_{all}$  evaluates the mean squared error for all 1898 time series.

These results, especially column MSE  $TS_2$ , confirm the assumption drawn from Figure 5.4. A MLP and unfortunately also an EQL network are not applicable for time series approximation of the bouncing ball.

MSE $\alpha$	MSE $\beta$	Method	Layer	$\alpha$	Samples	Epoch
4.6301 e-2	5.9671 e-2	EQL	[1, 5, 5, 1]	50	1500	1000
5.2486 e-1	5.2861 e-1	EQL	[1, 5, 1]	50	1500	1000
3.1987 e-1	4.452 e-1	MLP	[1, 6, 6, 1]	50	1500	1000
1.2565 e-2	1.502 e-2	EQL	[1, 5, 5, 1]	50	1500	3000
4.1318 e-1	4.8689 e-1	EQL	[1, 5, 1]	50	1500	3000
7.3631 e-2	1.0438 e-1	MLP	[1, 10, 10, 1]	50	1500	3000

MSE $TS_1$	MSE $TS_2$	MSE $TS_{all}$	Samples TS	Samples all
5.1502 e-1	1.7552	42.1511	2001	1898 × 2001
7.1312	8.2132	53.0759	2001	1898 × 2001
2.9084 e-1	8.6156 e-1	40.3895	2001	1898 × 2001
7.6877	9.5234	55.3003	2001	1898 × 2001
3.1582	4.0192	46.7932	2001	1898 × 2001
8.8507 e-2	1.0754	40.8014	2001	1898 × 2001

Table 5.3: Comparison of different MLP and EQL for a time series approximation of the bouncing ball.

However, as discussed in Section 2.3, there are other established methods for approximating hybrid nonlinear models from measurement data. The algorithm used in the toolbox HIT is based on regression methods. Therefore the framework is not applicable. The PWA OAF NN approach, as presented in Shukla and Paul (1996) and Števek et al. (2012b), is based on a special network structure. Figure 5.5 shows an example structure of an OAF NN.

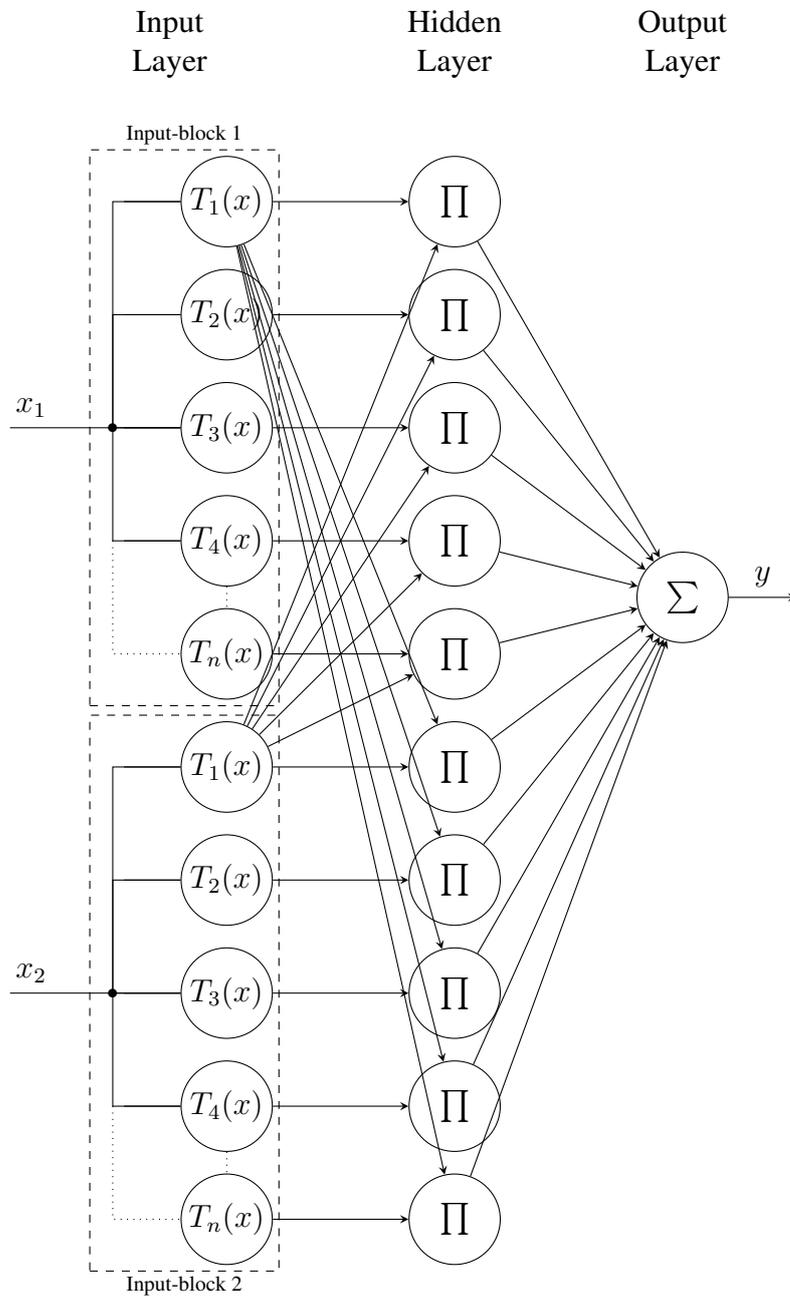


Figure 5.5: The structure of a neural network with orthogonal activation functions (OAF NN) based on (Kozák and Števek, 2011).

The application of this network results in a generalised Fourier series based on one-dimensional orthonormal functions (Števek and Kozák, 2011). The input layer in Figure 5.5 shows a preprocessing of the input data. Considering  $m$  inputs  $I = (x_1, \dots, x_m)$ , the  $n$  chosen orthonormal activation functions  $T_n(x)$  with order  $n$  are applied for each input resulting in

$$P(I) = (T_1(x_1), \dots, T_n(x_1), \dots, T_1(x_m), \dots, T_n(x_m))^T.$$

The structure of the hidden and output layer is similar to the Pi-sigma network and consists of product units in the hidden layer and a linear unit in the output layer. Only the edges between input and hidden layer are different. This network is not fully connected but requires specific connections between these layers to guarantee that the HPU multiplies only orthogonal functions of different inputs. This can be written as

$$\begin{aligned} z^{(2)} &= (Z^1, \dots, Z^m), \\ Z^i &= W^i \cdot I^i, \quad i \in \{1, \dots, m\}, \\ W^i &\in \{0, 1\}^{2m \times n}, \quad I^i = (I_i, \dots, I_{i+n})^T. \end{aligned}$$

Hence, the framework can be applied and the model description can be given as follows

$$\begin{aligned} A_{\text{oaf}} &= (3, I, O, N, U_{\text{oaf}}, T) \text{ with} \\ I &= P(x_1, \dots, x_m), \\ O &= y, \\ N &= (n \cdot m, n \cdot m, 1), \\ U_{\text{oaf}} &:= \{u_j^{(2)}\}_{j=1, \dots, n \cdot m}, \\ u_j^{(2)}(z_j^{(2)}) &= \prod_{i=1}^m z_{ji}^{(2)}, \\ u^{(3)}(z^{(3)}) &= z^{(3)} = \sum_{j=1}^{n \cdot m} w_j^{(3)} a_j^{(2)}, \\ T &= (M, A, C, S, V), \end{aligned} \tag{5.20}$$

where  $P$  denotes the preprocessing of the input data.

The model of the hybrid system actually consists of one neural network of the form  $A_{\text{oaf}}$  for each PWA system modelling the local dynamics in one region of the state space partition, see Section 2.3.4. Therefore, not one network is used, but multiple OAF NN are in place to model the hybrid system. Due to the fact that the starting point is a set of measurements, it is an approximation of the entire hybrid system using neural networks. Additionally, it is as well an application of the artificial hybrid dynamics approach since each local dynamical system is approximated by a network. The difference is that in the artificial hybrid dynamics framework the event conditions are a-priori known. In case of the OAF NN approach, the regions for the local models are determined using a regression based algorithm, see Section 2.3.3.

## 5.3 Artificial Hybrid Dynamics Application

This section investigates the application of the framework, apart from the OAF network approach, for systems where the dynamical processes are only given as input-output data. The event definition including guard and jump are considered a-priori knowledge.

### 5.3.1 Bouncing Ball

Figure 5.1 (b) shows one dynamical process and its definition is given in equation (5.18). The framework, as introduced in Section 4.3, can be applied for the dynamical behaviour as follows

$$\begin{aligned}
 A_{\text{had}}^d &= (d; \mathbb{R}^2; A; W; E; \text{Inv}_{\text{an}}; \text{Act}_{\text{an}}), \quad \text{with} & (5.21) \\
 &\text{Inv}_{\text{an}} \subset \mathbb{R} \text{ and } \text{Act}_{\text{an}} : d \mapsto A_{\text{nn}}^d, \text{ where} \\
 A_{\text{nn}}^d &= (3, I, O, N, U_{\text{mlp}}, T), \\
 I &= (h_0, v_0, g), \\
 O &= (t_E, v_E), \\
 T &= (M, A, C, S, V), \\
 C &= L_{\text{mse}}, \\
 V &= (\alpha, \beta, \gamma).
 \end{aligned}$$

The variables  $A$ ,  $W$  and  $E$  are inherited from equation (5.7), whereas  $U_{\text{mlp}}$ ,  $M$  and  $A$  are the same as in equation (5.18) with different layer structures  $N$ . The framework application for an EQL network is similar, except the set of units  $U_{\text{mlp}}$  has to be substituted with  $U_{\text{eql}}$  defined in equation (5.19).

MSE $\alpha$	MSE $\beta$	Method	Layer	$\alpha$	Epoch
1.1426 e-2	1.2294 e-2	EQL	[3, 5.5, 1]	50	100
6.5946 e-2	6.9320 e-2	MLP	[3, 6, 6, 1]	50	100
1.4988 e-3	1.8588 e-3	EQL	[3, 5.5, 1]	80	100
3.4077 e-2	4.1005 e-2	MLP	[3, 6, 6, 1]	80	100
3.5740 e-4	2.5978 e-4	EQL	[3, 5.5, 1]	50	1000
6.2314 e-4	7.2214 e-4	MLP	[3, 6, 6, 1]	50	1000
9.9819 e-5	9.3751 e-5	EQL	[3, 5.5, 1]	80	1000
2.5902 e-4	2.9231 e-4	MLP	[3, 6, 6, 1]	80	1000

MSE $E_1$	MSE $E_{1-3}$	MSE Extra $E_1$	MSE Extra $E_{1-3}$	Method
7.8549 e-2	0.5385	1.3667	17.1962	EQL
0.3845	2.3131	4.9861	26.8313	MLP
0.1879	0.4240	6.4578	61.4694	EQL
0.1404	0.3679	3.0436	13.3757	MLP
3.4185 e-2	4.9203 e-2	2.5090	6.3011	EQL
1.2948 e-2	5.3241 e-2	0.2744	2.5792	MLP
5.7118 e-3	0.1099	0.5731	0.41854	EQL
1.3797 e-2	8.1133 e-2	0.7764	4.7044	MLP

Table 5.4: Comparison of different MLP and EQL settings for the approximation of the first event and its extrapolation for the bouncing ball.

For the experiments to approximate the dynamics, the datasets in Table 5.1 are reused. Instead of focusing on the first three events this approach requires only storing the first events. The input data is defined as before, consisting of the initial condition  $[h_0, v_0]$  as well as the gravitational constant  $g$ . The output values consist of the event time  $t_e$  and the according velocity  $v_E$ . Hence, each of the 1271 samples is defined as  $I = (h_0, v_0, g)$  and  $O = (t_E, v_E)$ . For the training ( $\alpha$ ) 50 to 80% of this data is used, leaving 20 – 50% for testing ( $\beta$ ).

In Table 5.4 the simulation results for a MLP and an EQL are summarised. The changing epoch shows that an intense training is more important than the number of samples used. The different network structures show a high accuracy in predicting the second and third event for initial conditions in the domain of the training set. The extrapolation for new

Output	MSE $E_1$	MSE $E_{1-3}$	MSE Extra $E_1$	MSE Extra $E_{1-3}$	Layer
$(t_e, v_e)$	1.1097 e-02	8.3003 e+00	3.5516 e-02	2.7728 e+01	2(3)
$(t_e, v_e)$	1.3611 e-02	8.3250 e+00	7.7100 e-02	3.0234 e+01	1(9)
$(t_e, v_e)$	2.0303 e-02	8.4143 e+00	8.5657 e-02	2.8415 e+01	2(6, 2)
$(t_e, v_e)$	2.1441 e-02	8.3161 e+00	2.3364 e-01	2.9578 e+01	1(6)
$(t_e, v_e)$	1.6752 e-02	8.3683 e+00	2.3422 e-01	2.3866 e+01	1(9)
$(t_e, v_e)$	2.1446 e-01	6.2800 e+00	2.9381 e-01	2.1027 e+01	2(9, 1)
$(t_e, v_e)$	9.5669 e-03	8.3091 e+00	3.3316 e-01	3.0816 e+01	1(12)
$(t_e, v_e)$	5.1260 e-02	8.2436 e+00	4.7611 e-01	2.3711 e+01	1(3)
$(t_e, v_e)$	3.1343 e-03	8.3001 e+00	8.7262 e-01	3.0891 e+01	2(3, 6)
$(t_e)$	5.5952 e-04	8.8302 e-02	2.2811 e-03	2.8851 e-02	1(9)
$(t_e)$	1.3634 e-03	9.6518 e-02	7.3375 e-03	3.3124 e-02	2(6, 2)
$(t_e)$	5.6004 e-04	8.1020 e-02	1.5032 e-02	3.9409 e-02	2(3)
$(t_e)$	8.0539 e-04	8.3926 e-02	1.0225 e-02	5.0373 e-02	1(6)
$(t_e)$	4.1091 e-04	8.3086 e-02	1.2095 e-02	5.6290 e-02	1(12)
$(t_e)$	9.9303 e-05	8.1708 e-02	1.2096 e-02	8.6462 e-02	2(3, 6)
$(t_e)$	8.7971 e-03	1.5440 e-01	3.5814 e-02	9.2501 e-02	1(3)
$(t_e)$	3.8970 e-01	1.2481 e+00	5.2202 e-01	4.8766 e+00	2(9, 1)

Table 5.5: Comparison of different MLP networks for of the first event approximation and the extrapolation of the following two events.

initial conditions is rather bad. The best prediction outside of the training set can be achieved by the 2-layered EQL network.

In Tabel 5.5 different MLP networks are compared. The results suggest, that a one-layered network with nine neurons approximate better than a two-layered network with less neurons in each layer. Apart from the state output  $(t_e, v_e)$  the results are also tested with regards to the event time only. Here, the extrapolation results are much better.

Without simulating an entire time series, the model  $A_{\text{had}}$  is able to predict the first three events even if the initial condition is located outside of the training set. This framework application replaces the guard  $G_{d,d}$  of the hybrid model. The jump  $J_{d,d}$  can be applied to the output  $v_E$ , as defined in (5.6) and with the new initial conditions the network can approximate the next event. Due to the fact that there is only one dynamical node, function  $\text{Act}_{\text{an}}$  maps always to the same neural network  $A_{\text{nn}}^d$ . This process repeats itself until the predefined simulation time is reached.

Depending on the application purpose, the event time prediction can be inadequate and a time series might be required. Two different network structures are chosen for the time

series approach, an EQL and a MLP. First, an EQL network with an ARX approach is chosen to approximate the flying phase of the ball. Its training data consists of the current state vector  $(t_k, h_k, v_k)$  of a specific initial condition and aims to predict the future state vector  $O := (t_{k+1}, h_{k+1}, v_{k+1})$ . This data is extracted from the dataset introduced in Table 5.1. Only the first flying phase is including in the training set. Using the framework, this scenario can be defined as

$$\begin{aligned}
 A_{\text{nn}} &= (L, I, O, N, U_{\text{eql}}, T), \quad \text{with} & (5.22) \\
 I &:= (t_k, h_k, v_k), \\
 O &:= (t_{k+1}, h_{k+1}, v_{k+1}), \\
 T &:= (M, A, C, S, V), \\
 C &:= L_{\text{mse}}, \\
 V &:= (\alpha, \beta, \gamma).
 \end{aligned}$$

Only 20 – 50%( $\alpha$ ) of the dataset with  $S = 1675$  samples are used for training. Even for higher  $\alpha$  values the training error remains at  $L_{\text{mse}} \approx 3.4 \text{ e} - 1$ . The fact that it is not possible to reduce this error value even when using 80% of the data, suggests that the ARX approach is not feasible in this scenario. In Figure 5.6 the resulting networks are used to approximate a time series applying the jump relation  $J_{d,d}$  at the event times. It is obvious that neither MLP nor EQL are applicable in this case.

Another possible network structure for the framework defined in (5.21) is the ODE NN. The framework  $A_{\text{had}}$  has to be adapted to the following form

$$\begin{aligned}
 A_{\text{had}} &= (3, I, O, N, U_{\text{ode}}, T), \quad i \in \{1, 2\}, \quad \text{with} & (5.23) \\
 I &:= (h_0, v_0, t_i), \\
 O(a^L) &:= a^L t_i^2 + v_0 t_i + h_0 = x_i, \\
 N &:= [1, 10, 1], \\
 T &:= (M, A, C, S, V), C := L_{\text{ode}},
 \end{aligned}$$

where  $M$  is the back-propagation,  $A$  is the Gradient Descent algorithm and  $U_{\text{ode}}$  is equal to  $U_{\text{mlp}}$ . In contrast to the EQL approach, each sample in the training set is a point

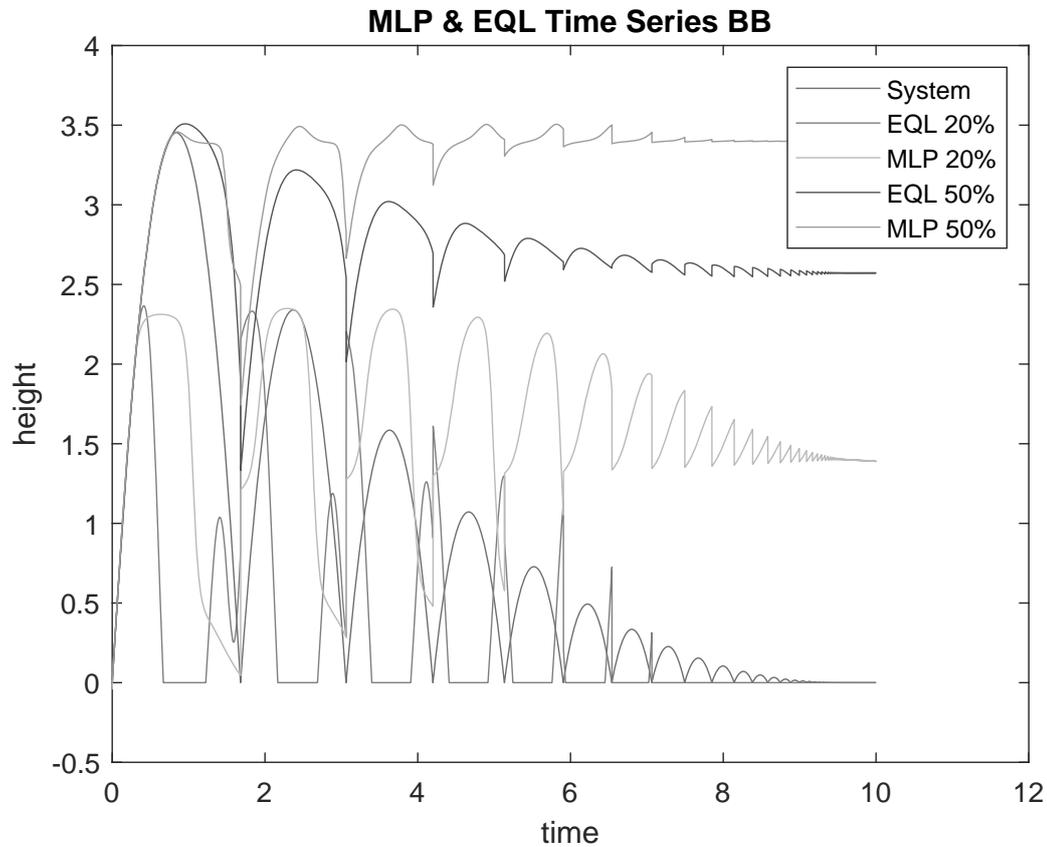


Figure 5.6: Time series approximation of MLP and EQL

in time  $t_i$  and the initial condition  $(h_0, v_0)$  as input resulting in  $I = (h_0, v_0, t_i)$ . The corresponding system answer  $O = x_i$  is defined as output and a white noise is added. The training samples are taken from a single time series with fixed initial values.

Applying the jump map  $J_{d,d}$  at every event, a complete time series can be approximated, see Figure 5.7. In Table 5.6 the results for different epochs and sample sizes are listed. Although the training set consists only of points for one specific initial condition, the network is able to approximate the system behaviour for other unseen initial value settings. The results also show that it is more important to invest in time for the training process than in a lot of training samples. It also confirms the superiority of this network approach compared to a MLP and EQL. This fact comes as no surprise, since the output

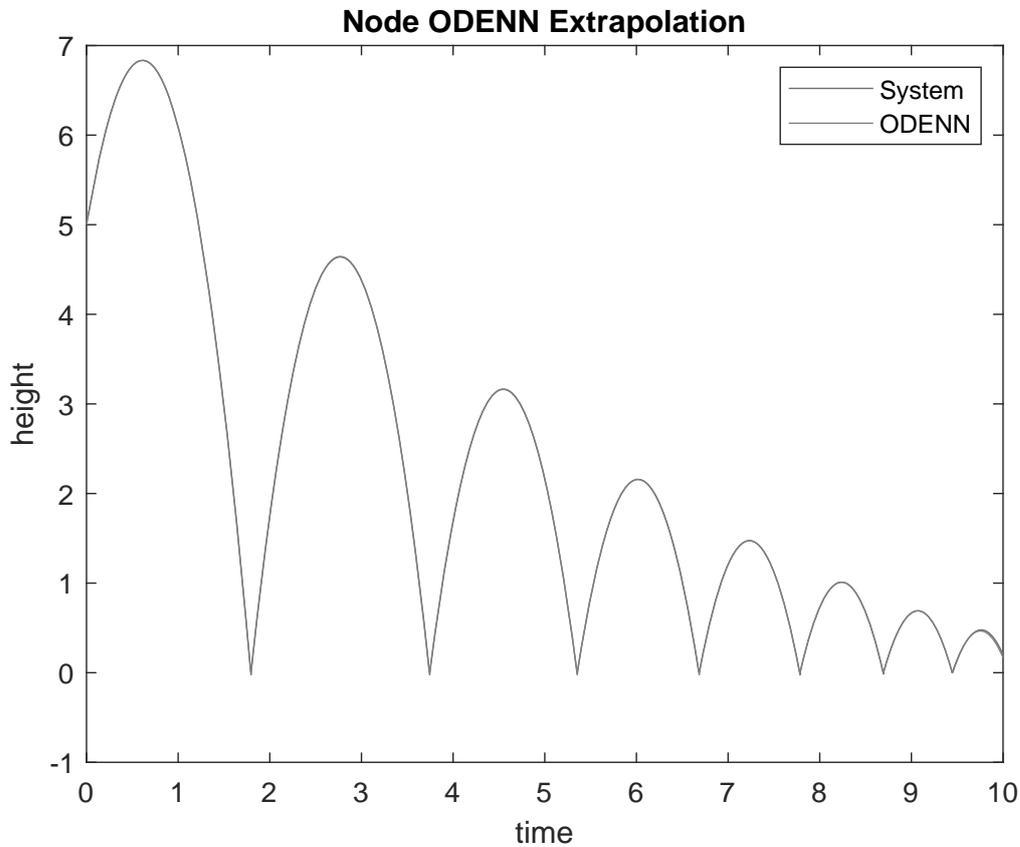


Figure 5.7: Approximation of the dynamics for the bouncing ball.

of the ODE NN approach, as defined in equation (5.23), is based on the same principles as a Taylor polynomial second order. Hence, the ODE NN network represents a solid alternative for approximation dynamical behaviour.

### 5.3.2 Pendulum with Free Fall Phase

In order to evaluate the performance of the framework applications a dataset is created. Since there is no analytical solution available, the differential algebraic equations defined in (5.15) and (5.12) are implemented and solved by an embedded ODE-solver in MATLAB<sup>®</sup>. The variation of the initial conditions is given as

$$\varphi \in [0.5, 6.2] \quad \text{and} \quad \dot{\varphi} \in [0, 5],$$

IC [3, 2]	IC [5, 5]	IC [5, 6]	Samples	Epoch
4.9141 e-4	1.0457 e-5	2.8280 e-5	30	1000
7.1115 e-4	1.9453 e-2	8.4800 e-3	100	100
4.4899 e-3	1.2394 e-2	3.0243 e-2	50	100
9.2812 e-4	7.1193 e-2	1.1078 e-1	30	100
IC [6, -2]	IC [6, 5]	IC [9, 7]	Samples	Epoch
5.4143 e-4	1.2945 e-5	1.2428 e-2	30	1000
1.9734 e-2	4.8782 e-3	2.4246 e-2	100	100
3.8697 e-3	3.2765 e-2	3.4682 e-1	50	100
3.8898 e-2	1.0601 e-1	2.2654 e-1	30	100

Table 5.6: Comparison of the ODE NN for different initial conditions for the bouncing ball.

with a step size of 0.1. The gravitational constant is approximated as  $g = 9.81$  and the damping parameters are given as  $k_p = 0.4$  and  $k_r = 0$ . The length of the pendulum rod is  $l = 1$  and the pendulum body has mass  $m = 1$ .

The application of the artificial hybrid dynamics framework for the pendulum with free fall is given as follows

$$\begin{aligned}
A_{\text{had}} &= (\{p, r\}; \mathbb{R}^2; \{b_1, b_2\}; W; E; \text{Inv}_{\text{an}}; \text{Act}_{\text{an}}), \quad \text{with} \\
\text{Inv}_{\text{an}} &\subset \mathbb{R}, \\
\text{Act}_{\text{an}} &: D \rightarrow \mathcal{F}_{\text{an}}, \\
\text{Act}_{\text{an}}(d) &= A_{\text{nn}}^d, \quad d \in \{p, r\}.
\end{aligned} \tag{5.24}$$

Both dynamical processes can be realised as neural networks denoted as  $A_{\text{nn}}^p$  for the oscillating behaviour and  $A_{\text{nn}}^r$  for the free fall. The framework definition for these networks differ solely in the used network approach is different.

$$\begin{aligned}
A_{\text{nn}}^p &= (3, I, O, N, U_{\text{mlp}}, T), \\
I &= (t_i, h_0, v_0, g), \\
O &= x_i, \\
T &= (M, A, C, S, V), \\
C &= L_{\text{mse}}, \\
V &= (\alpha, \beta, \gamma).
\end{aligned} \tag{5.25}$$

The MLP defined in (5.25) can be substituted by an EQL. In that case the layer structure  $N$  has to be adapted and the definition of the units has to be changed to  $U_{\text{eql}}$ . In Figure 5.8 the results of a MLP and an EQL for the oscillation of the pendulum are depicted. It is obvious, that the MLP can learn the training data to a certain extent. Unfortunately, the network is overfitted because the prediction of every point which was not included in the training data is incorrect. Whereas the EQL approximates the function also for new data with a small discrepancy.

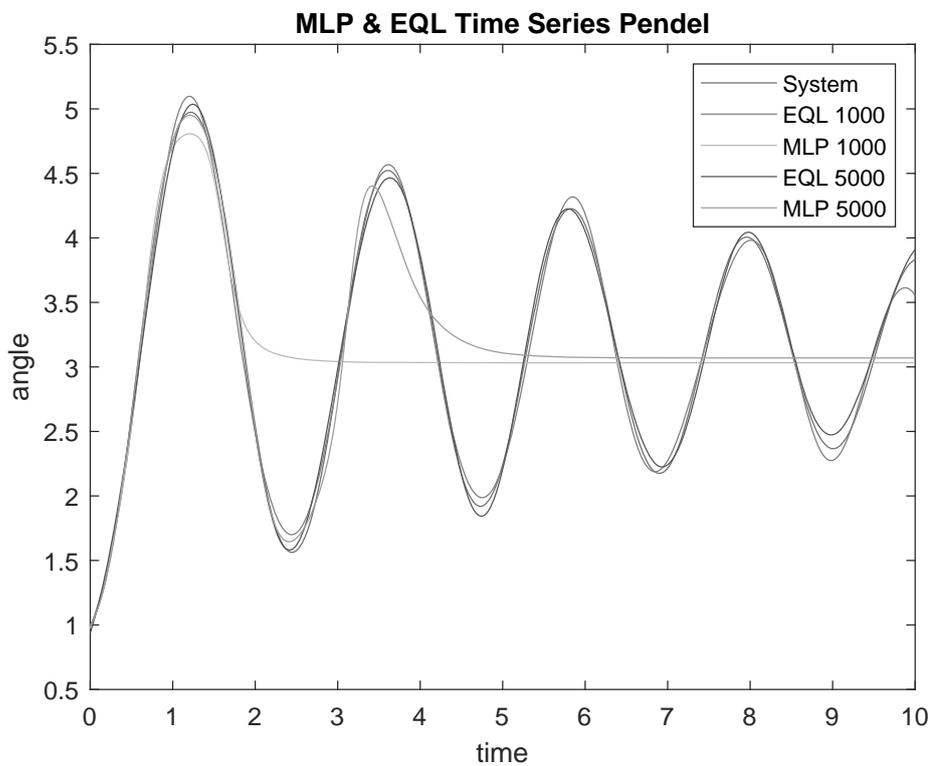


Figure 5.8: MLP and EQL approximation for the oscillation of the pendulum.

In case of the free fall phase, the framework for network  $A_{\text{nn}}^r$  applying a ODE NN can be given as

$$A_{\text{nn}}^r = (3, I, O, N^i, u_j^l, T), \quad i \in \{1, 2\}, \quad \text{with} \quad (5.26)$$

$$I := (t_0, \dots, t_n),$$

$$\begin{aligned}
O &:= (x_0, \dots, x_n), \\
u_j^{(l)}(z_j^{(l)}) &= \begin{cases} \sigma(z_j^{(l)}), & l = 2, j \in \{1, \dots, n_2\}, \\ z_j^{(l)}, & l = 3, j = 1, \end{cases} \\
N &:= (1, 10, 1), \\
T &:= (M, A, C, S, V), \\
C &:= L_{\text{ode}},
\end{aligned}$$

whereas  $M$  and  $A$  are the same as in (5.23).

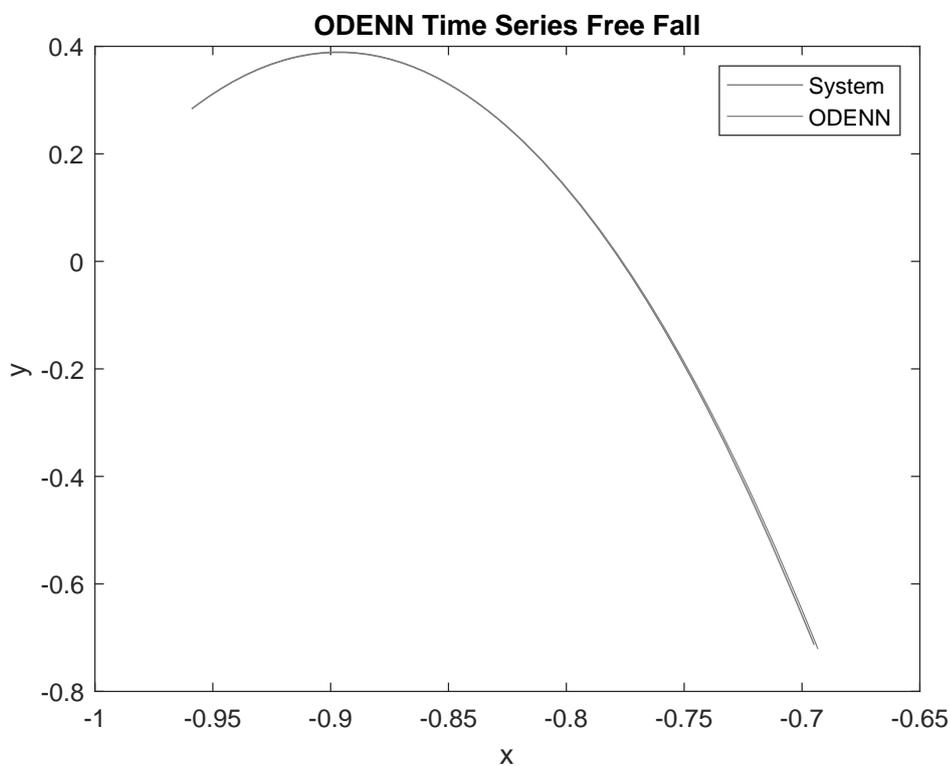


Figure 5.9: ODE NN approximation of the free fall phase of the pendulum.

In Figure 5.9 the results for a specific initial condition are depicted and in Table 5.7 the according results are shown. The network is trained with data of one specific initial condition. Similar to the experiments of the bouncing ball before, the network can

approximate also solutions for unseen initial conditions. There is almost no difference in the mean squared error values for the randomly chosen initial conditions in Table 5.7.

IC <sub>1</sub>	IC <sub>2</sub>	IC <sub>3</sub>	IC <sub>4</sub>	IC <sub>5</sub>	Samples	Epoch
1.4536 e−05	1.0596 e−05	1.5075 e−05	2.6757 e−05	1.1531 e−04	30	100
3.7666 e−06	1.9892 e−06	3.7182 e−06	4.3021 e−06	3.7139 e−06	100	100
3.2992 e−06	1.4747 e−06	3.2822 e−06	4.2290 e−06	4.6056 e−06	30	1000
1.8490 e−06	1.5378 e−06	1.8037 e−06	1.1336 e−06	1.3347 e−06	100	1000

Table 5.7: ODE NN for different initial conditions for the free fall phase of the pendulum.

## 5.4 Artificial Hybrid Event Application

The last application scenario focuses on the problem of event detection with neural networks. As mentioned in the framework definition 4.4, the data consists of different state values of the system and its label states if an event occurred. Neither a EQL nor a ODE NN is applicable for such a classification task. Hence, a MLP is chosen as network structure. The framework application to replace the guard region in the hybrid model of the bouncing ball is given as

$$\begin{aligned}
 A_{\text{hae}} &= (\{d\}; \mathbb{R}^2; \{b\}; W_{\text{ae}}; E_{\text{ae}} = (d, b, G_{\text{ae}}, J_{d,d}, d); \text{Inv}; \text{Act}), \\
 \text{Act}(d) &:= F_d, \\
 G_{\text{ae}}(b) &:= A_{\text{ae}}^b.
 \end{aligned} \tag{5.27}$$

where  $W$ ,  $\text{Inv}$  and  $J_{d,d}$  remain as defined in the hybrid automaton in equation (5.7). In case of a pendulum with free fall the according framework can be defined as

$$\begin{aligned}
 A_{\text{hae}} &= (\{r, p\}; X; \{b_1, b_2\}; W_{\text{ae}}; E_{\text{ae}}; \text{Inv}; \text{Act}), \\
 E_{\text{ae}} &= \{(p, b_1, A_{\text{ae}}^p, J_{p,r}, r), (r, b_2, A_{\text{ae}}^r, J_{r,p}, p)\},
 \end{aligned} \tag{5.28}$$

where  $W$ ,  $\text{Inv}$ ,  $\text{Act}$  as well as the jump maps  $J_{p,r}$  and  $J_{r,p}$  are the same as in definition (5.17). The network definition does not depend on the application example. Therefore one network definition can be used for events of both systems, occurring during the bounce, the free fall and the pendulum phase.

The framework application for the network is given as

$$\begin{aligned}
A_{\text{ae}}^i &= (4, I, O, N_i, \{u_j^{(l)}\}_{j=1, \dots, n_l}^{l=2,3,4}, T), \quad i \in \{d, r, p\}, \\
u_j^{(l)}(z_j^{(l)}) &= \sigma(z_j^{(l)}), \quad l = 2, 3, \quad j = \{1, \dots, n_l\}, \\
u^{(4)}(z^{(4)}) &= z^{(4)}, \\
T &= (M, A, C, S, V), \\
C &= L_{\text{cross}},
\end{aligned} \tag{5.29}$$

where  $A$  is the Scaled Conjugate Gradient Algorithm,  $M$  is the back-propagation method and  $N_i$  depends on the application example. The input data consists of the state vector for various initial conditions and various points in time. For each data point the occurrence of an event is evaluated

$$O_i = \begin{cases} 0, & \text{no event at } I_i, \\ 1, & \text{event occurs at } I_i. \end{cases}$$

The resulting dataset  $O_i \in \{0, 1\}, i \in \mathbb{N}$  forms the output data for training. The size and structure of the state vector depends on the application example. Hence, the input for the training data is given as

$$I_i = \begin{cases} (t_i, x_1(t_i), x_2(t_i), g), & D = \{d\}, \\ (t_i, \varphi(t_i), \dot{\varphi}(t_i), x_1(t_i), \dot{x}_1(t_i), x_2(t_i), \dot{x}_2(t_i), g, k_r, k_p), & D = \{r, p\}. \end{cases}$$

The event classification is especially challenging. In most hybrid systems the occurrence of an event is rather rare. A dataset for the pendulum with various initial conditions results in  $\approx 0.6\%$  events. This problem is often referred to as binary classification of imbalanced datasets (Wang et al., 2012; Chawla, 2010). In Burnaev et al. (2015) different resampling methods are discussed, to balance the dataset and enable successful classification. One method suggests to create synthetic observations drawn from a uniform distribution within the data of the small category. Due to the fact that the events are defined by physics this approach is not applicable. Another possibility is to remove a certain amount of random data from the majority class to balance the dataset. A parameter  $k$  is introduced to coordinate the balance in the training set. For each data point

Ex	$S = (E/\text{NoE})$	Corr. pos.	Corr. neg.	False. pos.	False. neg.	$k$	$V$
FF <sub>1</sub>	310/615	33.9%	66.1%	0%	0%	2	[0.6, 0.2, 0.2]
FF <sub>2</sub>	310/615	33.9%	66.1%	0%	0%	$\frac{3}{2}$	[0.8, 0.1, 0.1]
FF <sub>3</sub>	615/598200	0%	99.9%	0%	0.1%	—	[0.6, 0.2, 0.2]
PE <sub>1</sub>	502/502	50%	50%	0%	0%	1	[0.6, 0.2, 0.2]
PE <sub>2</sub>	502/96827	0.3%	99.4%	0.0001%	0.3%	—	[0.6, 0.2, 0.2]
BB <sub>1</sub>	98485/6010853	1.2%	98.4%	0%	0.4%	—	[0.75, 0.15, 0.15]
BB <sub>2</sub>	49243/49243	49.9%	50%	0%	0.1%	$\frac{1}{2}$	[0.75, 0.15, 0.15]
BB <sub>3</sub>	49243/49243	50%	50%	0%	0.001%	$\frac{1}{2}$	[0.75, 0.15, 0.15]

Ex	$S = (E/\text{NoE})$	Corr. pos.	Corr. neg.	False. pos.	False. neg.	$k$	$N$
FF <sub>1</sub>	615/1230	33.3%	56.9%	9.7%	0%	2	[10, 20, 10, 1]
FF <sub>2</sub>	615/1845	25.0%	64.0%	1.0%	0%	$\frac{3}{2}$	[10, 40, 10, 1]
PE <sub>1</sub>	502/131264	0.6%	41.1%	58.5%	0%	—	[10, 20, 10, 1]
PE <sub>2</sub>	502/131264	0.1%	99.5%	0.1%	0.2%	—	[10, 20, 10, 1]
BB <sub>1</sub>	98485/12021707	0.6%	49.6%	49.6%	0.2%	—	[4, 30, 20, 1]
BB <sub>2</sub>	98485/98485	49.9%	50%	0%	0.001%	1	[4, 30, 20, 1]
BB <sub>3</sub>	98485/147728	40%	0.6%	59.4%	0.001%	$\frac{3}{2}$	[4, 30, 20, 1]

Table 5.8: Comparison of the classification MLP for both examples.

characterising an event,  $k$  data points classified as 'no event occurred' are added to the training set.

In Table 5.8 the results for all three events of the two examples are given. The classification of imbalanced data with regards to hybrid events is a challenge. Even though some of the values for wrong categorisation seem sufficiently small, the reason is not a good approximation but the ratio between the two categories.

For all three event types this approach is inapplicable. Even resampling the dataset only improves the network performance in training but the testing is still not feasible. In case of the bouncing ball example, not even the events in the training set can be classified correctly by the network. In case BB<sub>3</sub> at least the new data is classified correctly but the 50 events which have been classified false in training, stay incorrect.



# 6

## Discussion

---

The results in Chapter 5 show that artificial neural network approaches can be part of the modelling of hybrid dynamical systems. In cases, where certain elements of the hybrid model are missing, neural networks can be used to substitute these elements.

The framework can also be applied to describe neural network applications in other research fields, as shown in A.3. It enables a characterisation of commonly used neurons and a clear structure for networks such as higher order neural networks and equation learner. The modular construction of the proposed network description offers the possibility to reuse and exchange individual elements, as presented in Section 5.3.2. Here, only the definitions of the units have to be replaced to switch from an ODE NN to an EQL description. The framework can be applied to characterise as well advanced approaches, such as OAF NN in a structured way, see equation (5.20). Therefore, an interdisciplinary exchange of research results in regards to neural network approaches can be facilitated by the framework application.

If a local dynamical process of a hybrid dynamical system is only described by a set of input-output data, the application of the framework for a neural network inclusion shows promising results. Dependent on the research question and the structure of the required output data an applicable neural network can be chosen. Focusing on specific data describing the behaviour of the system, an ordinary one-layered MLP provides good approximation results, as shown in Table 5.5. It even enables predicting the defined data points for unknown system settings, without including new data into the learning

process of the network. The approximation of the state vector of the next event can be feasible even if the model description of the dynamical process is available. It can be used to narrow the region for the next event. If the research question requires an approximation of the time series, an ODE network approach can be applied. Based on the given data, the first or second derivative of the system variables can be estimated. The derivation is used to optimize the network performance, as explained in Section 3.3.1, and results in a trained network, that predicts not only time series for the training data but also extrapolates the behaviour for new initial conditions of the system. The outcomes listed in Table 5.6 and 5.7 show the accuracy of this network approach. An hypotheses about the differential order, the mathematical description of the process is based on, can improve the network's results.

For the approximation of a time series of the entire hybrid system, the application of one single neural network is not appropriate. Applying a network to the data where different dynamical systems are included, the training process leads to bad approximations or overfitting, and a generalised application is impossible, see Figure 5.3 and 5.4. Network structures such as the equation learner and the multi-layered perceptron can be used to approximate a selection of specific data points, as shown in the bouncing ball example in Table 5.2. For the approximation of a time series, these networks lead to overfitting behaviour. A possible alternative is presented in Ferrari-Trecate and al (2003): The given input-output data can be clustered beforehand and the results are used to partition the state space into small regions. For each region, an approximation of the local dynamics is required. As discussed in Kozák and Števek (2011) and Števek et al. (2012b) specially designed neural networks called OAF NN can be applied for this task. Hence, after preprocessing the given data, the neural networks used to approximate the entire hybrid system can be characterised using the proposed framework definition as in equation (5.20). Therefore, the dynamical process can either be replaced by defining specific points of interest and applying standard network structures or the OAF NN approach can be used to receive a time series approximation of the system.

In the case of the event detection approach, the available descriptions of the local dynamics can be used to classify the available data. If the discrepancy between model output and system data increases an event can be identified. The gathered state vectors

where the discrepancies are within a certain range, can be labelled 'no event'. This procedure results in an imbalanced dataset with two classes, the majority class 'no event' and the minority 'event'. The results listed in Table 5.8 show the results of a trained MLP for the classification of imbalanced binary datasets. If the data is resampled, as suggested in Nguyen et al. (2008) and Burnaev et al. (2015), the training can be successful. Unfortunately, even then, the generalised classification of events often does not work sufficiently. The case study confirms the findings in Murphey et al. (2004), that the traditional feed-forward neural network has difficulties to learn from imbalanced datasets (Wang et al., 2012). To cope with imbalanced binary datasets in classification scenarios possible alternatives to neural networks are presented in Lin et al. (2009). Therefore, facilitating methods such as regression and projection into the third framework application is a future objective.

In the case study, three different network structures are applied. The advantages of the ODE NN structure are, that a small sample size and a moderate number of epochs is sufficient, to approximate the underlying ODE. Additionally, the samples only have to cover a small part of the actual time series to succeed in training of a one-layered network with 10 neurons. The lack of hypotheses about the order of the ODE can decrease the accuracy of the approach. For the multi-layered perceptron it is crucial to provide enough data samples for training. Two different layer structures are investigated, a one- and two-layered network. The results showed, that a one-layered network with 9 – 12 sigmoid neurons is sufficient and provides a better approximation than some two-layered networks, see Table 5.5. An increase in neurons does not increase the accuracy of the network, only the risk of overfitting increases. For the equation learner, two things have to be mentioned: If the EQL layer structure is able to find a plausible functional relation of the input-output data, a small percentage of training data is enough to achieve a good approximation. If the function can not be approximated properly, the results can be worse than for a ordinary one-layered MLP.

Apart from neural network approaches, alternatives for hybrid systems with unknown model equations can be found in Ferrari-Trecate and al (2003) and Bemporad et al. (2009). Dependent of the a-priori knowledge of the hybrid system a variety of toolboxes like HIT, YAMLIP and MPT 3.0 can be used (Herceg et al., 2013).



# 7

## Conclusion

---

This research aimed to identify useful applications of neural networks in the modelling of hybrid dynamical systems. A framework was introduced, to standardise the replacement of elements in hybrid dynamical models and to investigate different application scenarios. In addition, a formalism for multi-layer perceptrons was defined, facilitating as well the original concept of neural networks, the perceptron, as other network structures. The application of the introduced framework itself is straight forward. Based on the analysis of the bouncing ball and pendulum example, it can be concluded that the usability of the introduced framework depends on the goal of the application. The required structure of the network's output is an important factor for the applicability.

The results of the case study show that neural networks are able to approximate the behaviour of hybrid dynamical systems if an appropriate network structure is applied. The ODE neural network has shown accurate approximations for different applications undergoing only a short training process. The EQL network presented a good approximation when confronted with oscillating behaviour. If the EQL is not able to find the right functional relationship between the data within the training process, the resulting approximation is as good as a multi layered perceptron. If the function can be found by the network, the resulting expression can be used as analytical approximation. However, the research group who introduced this network approach is constantly improving the layer structure to enable better approximations. In cases which aim to approximate specific points of interest, also a well-trained MLP can provide feasible results.

An interesting aspect of this thesis is the extrapolation capabilities of networks. In this context, the resulting network applications were not only tested for data in the learning domain but also for data of unknown domain. Dependent on the required information the accuracy of the results differed. The scenarios in which an ODE NN is applicable have shown promising results. The network structure has to be chosen based on the given quality and quantity of the data. An ODE NN leads to accurate prediction even if little data is available, whereas multi-layer networks are powerful if enough training data is given. The dimension of the state space of the hybrid system does not influence the applicability of the framework. For a more generalised application analysis an extended case study will be used for future studies.

The focus of the framework was, to investigate different application scenarios for neural networks. In the case of classifying state vectors in 'event' and 'no event', the neural network performance was not sufficient. The classification experiments for both application examples have shown, that neural networks are not applicable for imbalanced datasets like this. Other methods might be more effective. Therefore, adding elements to the framework to enable an inclusion of other methods in the hybrid dynamical model is a future objective. In this context, also the preprocessing of the input data in neural network applications can be included in the framework definition. A future review of networks other than feed-forward, might require adding new elements. For example a recurrent network might require more descriptive elements than defined in the framework at the moment. If the dynamical description of each node involves several implicit formulations, it might be laborious or even impossible to formulate an explicit jump relation. In such cases, the measured output and input data of the different nodes could be used to train another network  $J_{ae}^b$  to replace the jump relation  $J$  in the model description. In addition, a combined framework application, to facilitate the replacement of events and local dynamics at the same time, can be one of the next steps.

In conclusion, the framework is easy to apply. In the approximation of an entire hybrid system, the framework helps to administrate the different applied network approaches. It is especially beneficial in cases where local dynamical descriptions are absent.

# A

## Appendix

---

### A.1 Switched Affine Systems

A switched affine system represents a sampled continuous hybrid system and can be described by the following set of linear affine equations.

**Definition A.1.1** (Switched Affine System). *A switched affine system (SAS) is described as set of linear affine equations as follows*

$$\begin{aligned}x_r(k+1) &= A_{i(k)}x_r(k) + B_{i(k)}u_r(k) + f_i(k), \\y_r(k) &= C_{i(k)}x_r(k) + B_{i(k)}u_r(k) + g_i(k),\end{aligned}\tag{A.1}$$

with  $k \in \mathbb{N}_0$ ,  $x_r \in X_r \subset \mathbb{R}^{n_r}$  as state vector,  $u_r \in U_r \subset \mathbb{R}^{q_r}$  the input vector and  $y_r \in Y_r \subset \mathbb{R}^{p_r}$  as continuous output vector. The variables  $\{A_i, B_i, f_i, C_i, D_i, g_i\}_{i \in I}$  are matrices of suitable size and  $i(k) \in I$  selects the system dynamics.

Every time a switch occurs,  $i(k)$  changes and another update rule is activated. This principle can be realized by combining all dynamic descriptions using an if-else statement to activate the according system. Another possibility to execute a switch is to introduce a logical signal, called the event generator (EG), defined by

$$\delta_e(k) = f_H(x_r(k), u_r(k), k).$$

The event generator is the logical analogy to the continuous event function  $e(x)$  in section 2.2.2. The two possible types of events are realised using logical expressions verifying the predefined conditions.

$$\text{time event: } [\delta_e^i(k) = 1] \Leftrightarrow [kTs \geq t_i], \quad (\text{A.2})$$

$$\text{state event: } [\delta_e^i(k) = 1] \Leftrightarrow [a_i^T x_r(k) + b_i^T u_r(k) \leq c_i]. \quad (\text{A.3})$$

The variables  $a_i$ ,  $b_i$  and  $c_i$  are parameters of a linear hyperplane characterising the partition of the affine system (Potočnik et al., 2004, p. 5).

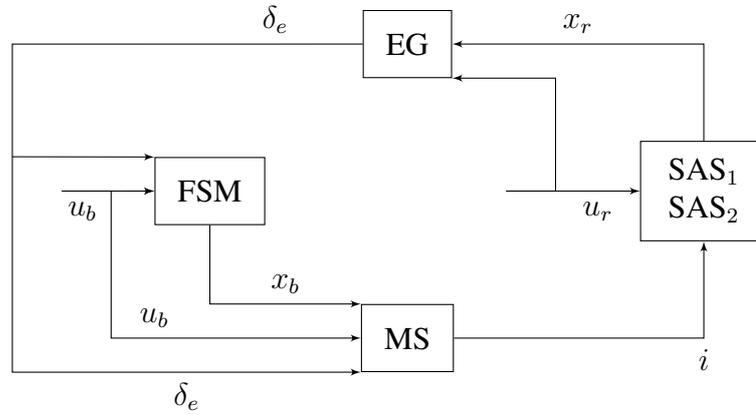


Figure A.1: Schematic layout of a switched affine system (SAS) including the finite state machine (FMS), the event generator (EG) and the mode selector (MS).

In Figure A.1 a discrete hybrid automata is shown. It contains the switched affine system descriptions to implement the continuous dynamics and a FSM (Finite State Machine) for the discrete behaviour. The system changes are realised with the event generator, similar to the guard, and the mode selector realising the jump relation. Compared to Definition 2.2.1 this automata introduces additional variables including a binary input vector  $x_b$  and a binary output vector  $u_b$ .

In the time-continuous case the dependence on  $k$  is replaced by the continuous time variable  $t$ . A special case of continuous-time hybrid systems is given in (A.4).

$$\begin{aligned} \dot{x}(t) &= A_{i(t)}x(t) + f_i(t), & i(t) &\in S, \\ x(t^+) &= J_{i,j}x(t^-), & i(t^-) &= i, i(t^+) = j. \end{aligned} \quad (\text{A.4})$$

An advantage of this subclass is, that it can be rewritten as linear dynamics by augmenting the state-space and to ease the construction of switching tables. (Seatzu et al., 2006),(Zhu and Antsaklis, 2013).

## A.2 Mixed Logical Dynamical System

In the case of the mixed logical dynamical (MLD) system description discrete and dynamical behaviour is not totally separable. Therefore MLD contains linear dynamical equations and linear mixed-integer inequalities including both continuous and binary variables as explained in detail in Bemporad and Morari (1999). Using MLD systems various models as hybrid systems, finite state machines, special classes of discrete event systems and nonlinear systems formulated as PWA are formulated.

**Definition A.2.1** (Mixed Logical Dynamical System). *Let*

$$x = \begin{pmatrix} x_c \\ x_l \end{pmatrix}, \quad x_c \in \mathbb{R}^{n_c}, x_l \in \{0,1\}^{n_l},$$

*be the states of the system,*

$$y = \begin{pmatrix} y_c \\ y_l \end{pmatrix}, \quad y_c \in \mathbb{R}^{p_c}, y_l \in \{0,1\}^{p_l},$$

*the output and*

$$u = \begin{pmatrix} u_c \\ u_l \end{pmatrix}, \quad u_c \in \mathbb{R}^{u_c}, u_l \in \{0,1\}^{u_l},$$

*the command input. Let  $\delta \in \{0,1\}^{r_l}$  and  $z \in \mathbb{R}^{r_c}$  be the auxiliary logical and continuous variables then*

$$x(t+1) = A_t x(t) + B_{1t} u(t) + B_{2t} \delta(t) + B_{3t} z(t), \quad (\text{A.5a})$$

$$y(t) = C_t x(t) + D_{1t} u(t) + D_{2t} \delta(t) + D_{3t} z(t), \quad (\text{A.5b})$$

$$E_{2t} \delta(t) + E_{3t} z(t) \leq E_{1t} u(t) + E_{4t} x(t) + E_{5t}, \quad (\text{A.5c})$$

*describe a mixed logical dynamical (MLD) system.*

The variables  $A, B_1, B_2, B_3, C, D_1, D_2, D_3, E_1, E_2, E_3, E_4, E_5$  are matrices of suitable dimension. For a given state  $x(k)$  and input  $u(k)$  the evolution of the MLD system (A.5)

is determined by solving  $\delta(k)$  and  $z(k)$  in equation (A.5c) to update  $x(k+1)$  and  $y(k)$ . Using computational inference technique to reformulate soft and hard constraints of the hybrid system, as discussed in Williams (2013), transforms any logical conditions and constraints of the system into equation (A.5c). The MLD system (A.5) is *completely well-posed* if for a given state  $x(k)$  and input  $u(k)$  the inequality (A.5c) has a unique solution with respect to  $\delta(k)$  and  $z(k)$ .

The mentioned reformulation of constraints can then be used to transform SAS and PWA into an MLD to enable a common simulation interface. Discrete hybrid automata can be transformed into MLD systems for control designs. Due to issues and shortcomings of this MLD formulation the definition was updated by Michael Kvasnica in 2010 adding new variables to enable a clear differentiation between continuous and logical variables (Kvasnica et al., 2009).

### A.3 Framework Application - Neural Network

The recognition of hand-written digits is one of the classical introduction examples for MLPs and will be used to demonstrate the introduced framework for an application besides HDS. The parameter estimation is done using the data sets for training 60000 and validation 10000, offered by MNIST (Modified National Institute of Standards and Technology database). After transforming the input data  $B \in \mathbb{R}^{28 \times 28}$  containing the colour values of each pixel of all 60000 pictures, the input set is given as vector  $I \in [0, 1]^{784}$ . The output is defined as  $O \in \{0, 1\}^{10}$  where the entry 1 marks the prediction of the digit on the picture.

Choosing one hidden layer with 15 nodes and applying the sigmoid function  $\sigma$  as activation function, results in a structure as depicted in Figure A.2. The framework application for this network is given as

$$\begin{aligned}
 A_{\text{nn}} &:= (4, I, O, N, \{u_j^2\}_j, T) \quad \text{with} \\
 N &:= (784, 15, 10), \\
 u_j^2(z_j^2) &= \sigma(z_j^2), \forall j \in \{1, \dots, 15\}, \\
 T &= (M, A, V), \\
 M &= \text{'Back-propagation'}, \\
 A &= \text{'Levenberg-Marquardt'}, \\
 V &= (50000, 10000, 20000).
 \end{aligned} \tag{A.6}$$

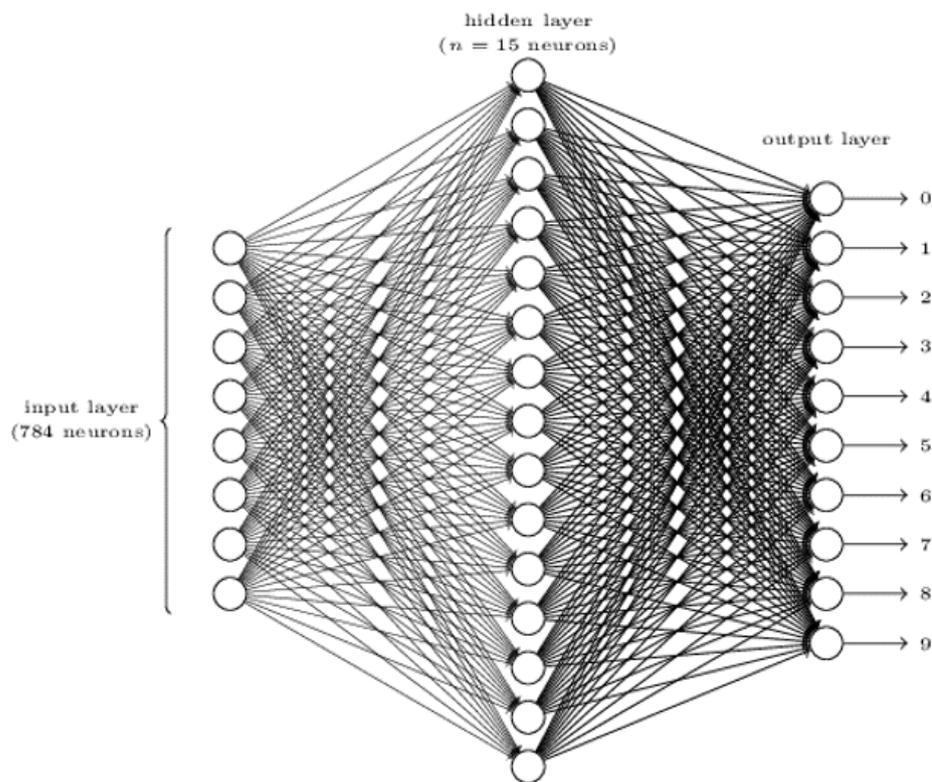


Figure A.2: The Illustration shows a neural network structure with 15 neurons in the hidden layer. It is used to recognise hand-written digits as presented in Nielsen (2015).



# List of Figures

- 1.1 The modelling and simulation process, motivated by Sargent et al. (2016). . . . . 4
- 1.2 An overview of the different modelling approaches within white and black box categories. . . . . 8
- 2.1 A block diagram of the linear time-invariant state space representation. . . . . 16
- 2.2 A block diagram of the extended LTI including reference signal  $r$  and feedback loop  $K$ . . . . . 17
- 2.3 Conceptional structure of a hybrid automaton based on the work of Körner (2015). . . . . 24
- 2.4 The different event constraints are illustrated, depicting the threshold  $\Delta x$  in (a), an event function  $h(x)$  in (b) and the guard region  $G$  in (c). . . . . 25
- 2.5 The flow chart describes the simulation steps of the HYSDEL compiler, including the improvements (dashed lines) made in HYSDEL 3.0. . . . . 28
- 2.6 PWA approximation structure with OAF neural networks. . . . . 31
- 3.1 An overview of the different areas of machine learning based on Rodríguez (2019). . . . . 34
- 3.2 The structure of a basic artificial neural network, called multi-layer perceptron (MLP). . . . . 35
- 3.3 Input and output for one neuron in a multi-layer perceptron (MLP). . . . . 36
- 3.4 The structure of a Pi-Sigma network (Nayak, 2017). . . . . 40
- 3.5 Equation learner network with one hidden layer based on Martius and Lampert (2016). . . . . 41

4.1	Three framework applications for including neural network concepts (red dashed rectangles) in modelling hybrid dynamical systems. . . . .	50
5.1	(a) Height of the ball over time. (b) Hybrid dynamical automaton of the bouncing ball. . . . .	60
5.2	The conceptional structure of a hybrid automaton based on Körner (2015). . . . .	65
5.3	A MLP approximation of the time series for a specific initial value. . . . .	69
5.4	A EQL approximation of the time series for a specific initial value. . . . .	70
5.5	The structure of a neural network with orthogonal activation functions, called multi-layer perceptron (MLP), see (Kozák and Števek, 2011). . . . .	72
5.6	Time series approximation of MLP and EQL . . . . .	78
5.7	Approximation of the dynamics for the bouncing ball. . . . .	79
5.8	MLP and EQL approximation for the oscillation of the pendulum. . . . .	81
5.9	ODE NN approximation of the free fall phase of the pendulum. . . . .	82
A.1	Schematic layout of a switched affine system (SAS) including the finite state machine (FMS), the event generator (EG) and the mode selector (MS). . . . .	94
A.2	The Illustration shows a neural network structure with 15 neurons in the hidden layer. It is used to recognise hand-written digits as presented in Nielsen (2015). . . . .	97

# List of Tables

- 2.1 Basic Elements for simulation models of hybrid systems. . . . . 22
- 2.2 Modelling components for HDS using HYSDEL, see Torrisi and Bemporad (2004). . . . . 27
- 3.1 A list of commonly used activation functions in artificial neural networks. . . 38
- 3.2 Step directions of commonly used optimisation algorithms for neural networks. . . . . 47
- 5.1 Dataset definitions for the bouncing ball example. . . . . 66
- 5.2 Comparison of different MLP and EQL settings for the approximation of the first three events and the extrapolation of the following three events for the bouncing ball. . . . . 68
- 5.3 Comparison of different MLP and EQL for a time series approximation of the bouncing ball. . . . . 71
- 5.4 Comparison of different MLP and EQL settings for the approximation of the first event and its extrapolation for the bouncing ball. . . . . 75
- 5.5 Comparison of different MLP networks for of the first event approximation and the extrapolation of the following two events. . . . . 76
- 5.6 Comparison of the ODE NN for different initial conditions for the bouncing ball. . . . . 80
- 5.7 ODE NN for different initial conditions for the free fall phase of the pendulum. 83
- 5.8 Comparison of the classification MLP for both examples. . . . . 85



# Bibliography

---

- Abdelbar, A. M. and Tagliarini, G. A. (1996). HONEST: a new high order feedforward neural network. In *Proceedings of International Conference on Neural Networks*, volume 2. IEEE.
- Altun, B., Ojaghi, P., and Sanfelice, R. G. (2018). A model predictive control framework for hybrid dynamical systems. *IFAC-PapersOnLine*, 51(20):128–133.
- Antsaklis, P. (2000). Special issue on hybrid systems: theory and applications a brief introduction to the theory and applications of hybrid systems. *Proceedings of the IEEE*, 88(7):879–887.
- Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. (2019). Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, page 201814058.
- Bemporad, A., Di Cairano, S., Ferrari-Trecate, G., Kvasnica, M., Morari, M., and Paoletti, S. (2009). Tools for modeling, simulation, control, and verification of piecewise affine systems. In Lunze, J. and Lamnabhi-Lagarrigue, F., editors, *Handbook of Hybrid Systems Control*, pages 297–324. Cambridge University Press.
- Bemporad, A. and Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427.
- Birta, L. G. and Arbez, G. (2013). *Modelling and simulation: exploring dynamic system behaviour*. Simulation foundations, methods and application. Springer, 2. ed edition. OCLC: 935142444.
- Borshchev, A. (2014). Multi-method modelling: AnyLogic. In *Proceedings of 2013 Winter Simulation Conference*, pages 248–279. IEEE.

- Breen, P. G., Foley, C. N., Boekholt, T., and Zwart, S. P. (2019). Newton vs the machine: solving the chaotic three-body problem using deep neural networks. *arXiv:1910.07291 [astro-ph, physics:physics]*.
- Burnaev, E., Erofeev, P., and Papanov, A. (2015). Influence of resampling on accuracy of imbalanced classification. *arXiv:1707.03905 [cs, stat]*, page 987521.
- Canty, N., O'Mahony, T., and Cychowski, M. T. (2012). An output error algorithm for piecewise affine system identification. *Control Engineering Practice*, 20(4):444–452.
- Carloni, L. P., Passerone, R., Pinto, A., and Angiovanni-Vincentelli, A. L. (2006). Languages and tools for hybrid systems design. *Foundations and Trends® in Electronic Design Automation*, 1(1):1–193.
- Chawla, N. V. (2010). Data mining for imbalanced datasets: An overview. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, pages 875–886. Springer US.
- Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. *arXiv:1806.07366 [cs, stat]*.
- Cheng, M.-Y., Tsai, H.-C., and Sudjono, E. (2012). Evolutionary fuzzy hybrid neural network for dynamic project success assessment in construction industry. *Automation in Construction*, 21:46–51.
- Choy, K., Chow, H. K., Tan, K., Chan, C.-K., Mok, E. C., and Wang, Q. (2008). Leveraging the supply chain flexibility of third party logistics – hybrid knowledge-based system approach. *Expert Systems with Applications*, 35(4):1998–2016.
- Christin, A., Rosenblat, A., and Boyd, D. (2015). Courts and predictive algorithms. page 11.
- Deatcu, C. and Pawletta, T. (2012). A qualitative comparison of two hybrid DEVS approaches. *SNE Simulation Notes Europe*, 22(1):15–24.

- Epitropakis, M., Plagianakos, V., and Vrahatis, M. (2010). Hardware-friendly higher-order neural network training using distributed evolutionary algorithms. *Applied Soft Computing*, 10(2):398–408.
- Ferrari-Trecate, G. and al, e. (2003). A clustering technique for the identification of piecewise affine systems. *Automatica*, 39:205–217.
- Ferrari-Trecate, G., Muselli, M., Liberati, D., and Morari, M. (2001). Identification of piecewise affine and hybrid systems. In *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, pages 3521–3526 vol.5. IEEE.
- Ferrari-Trecate, G., Muselli, M., Liberati, D., and Morari, M. (2005). Hybrid identification toolbox.
- Fritzson, P. A. (2004). *Principles of object-oriented modeling and simulation with Modelica 2.1*. IEEE Press ; Wiley-Interscience. OCLC: ocm52920516.
- Gao, G., Liu, F., San, H., Wu, X., and Wang, W. (2018). Hybrid optimal kinematic parameter identification for an industrial robot based on BPNN-PSO. *Complexity*, 2018:1–11.
- Glock, B., Popper, N., and Breitenacker, F. (2015). Various aspects of multi-method modelling and its applications in modelling large infrastructure systems like airports.
- Hafner, I. and Popper, N. (2017). On the terminology and structuring of co-simulation methods. In *EOOLT*.
- Haykin, S. (2009). *Neural networks and learning machines*. Prentice Hall, 3rd ed edition. OCLC: ocn237325326.
- Henzinger, T. A. (1996). The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292.
- Herceg, M., Kvasnica, M., Jones, C. N., and Morari, M. (2013). Multi-parametric toolbox 3.0. In *2013 European Control Conference (ECC)*, pages 502–510. IEEE.

- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.
- Huynh, Q.-N. (2016). *CoModels, engineering dynamic compositions of coupled models to support the simulation of complex systems*. phdthesis, Pierre and Maria Curie University.
- Ismail, A. (2002). *Training and optimization of product unit neural networks*. phdthesis, University of Pretoria.
- Jayalakshmi, T. and Santhakumaran, A. (2011). Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3(1):pp. 89 – 93.
- Juditsky, A., Hjalmarsson, H., Benveniste, A., Delyon, B., Ljung, L., Sjöberg, J., and Zhang, Q. (1995). Nonlinear black-box models in system identification: Mathematical foundations. *Automatica*, 31(12):1725–1750.
- Körner, A., Winkler, S., and Breiteneker, F. (2018). Possibilities in state event modelling of hybrid systems. *SNE Simulation Notes Europe*, 28(3):109–111.
- Kozák, S. and Števek, J. (2011). Improved piecewise linear approximation of nonlinear functions in hybrid control. *IFAC Proceedings Volumes*, 44(1):14982–14987.
- Krüger, I., Mehlhase, A., and Schmitz, G. (2012). Variable structure modeling for vehicle refrigeration applications. pages 927–934.
- Körner, A. (2015). *Mathematical Characterisation of State events in Hybrid Modelling*. phdthesis, Technische Universität Wien.
- Körner, A. and Breiteneker, F. (2016). State events and structural-dynamic systems: Definition of ARGESIM benchmark c21. *SNE Simulation Notes Europe*, 26(2):117–128.
- Kvasnica, M. (2008). *Efficient Software Tools for Control and Analysis of Hybrid Systems*. PhD thesis, ETH Zurich, ETH Zurich, Physikstrasse 3, 8092 Zurich, Switzerland.

- Kvasnica, M., Grieder, P., Baotić, M., and Morari, M. (2004). Multi-parametric toolbox (MPT). In Alur, R. and Pappas, G. J., editors, *Hybrid Systems: Computation and Control*, volume 2993, pages 448–462. Springer Berlin Heidelberg.
- Kvasnica, M., Herceg, M., Morari, M., Gaulocher, S., and Poland, J. (2009). HYS-DEL3\_manual.pdf.
- Kvasnica, M., Szücs, A., and Fikar, M. (2011). Automatic derivation of optimal piecewise affine approximations of nonlinear systems. *IFAC Proceedings Volumes*, 44(1):8675–8680.
- Kwiatkowski, A., Lichtenberg, G., and Schild, A. (2003). Event prediction for switching linear systems with time varying thresholds using orthogonal functions. In Maler, O. and Pnueli, A., editors, *Hybrid Systems: Computation and Control*, volume 2623, pages 314–327. Springer Berlin Heidelberg.
- Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000.
- Laursen, S. ., Webb, D., and Ramirez, W. F. (2007). Dynamic hybrid neural network model of an industrial fed-batch fermentation process to produce foreign protein. *Computers & Chemical Engineering*, 31(3):163–170.
- Lian, J., Liu, S., Li, L., Liu, X., Zhou, Y., Yang, F., and Yuan, L. (2017). A mixed logical dynamical-model predictive control (MLD-MPC) energy management control strategy for plug-in hybrid electric vehicles (PHEVs). *Energies*, 10,74.
- Lin, J. . and Unbehauen, R. (1992). Canonical piecewise-linear approximations. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(8):697–699.
- Lin, S.-C., Chang, Y.-c. I., and Yang, W.-N. (2009). Meta-learning for imbalanced data and classification ensemble in binary classification. *Neurocomputing*, 73(1):484–494.

- Lofberg, J. (2004). YALMIP : a toolbox for modeling and optimization in MATLAB. In *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pages 284–289.
- Lu, W., Zhu, P., and Ferrari, S. (2016). A hybrid-adaptive dynamic programming approach for the model-free control of nonlinear switched systems. *IEEE Transactions on Automatic Control*, 61(10):3203–3208.
- Martius, G. and Lampert, C. H. (2016). Extrapolation and learning equations. *arXiv:1610.02995 [cs]*. arXiv: 1610.02995.
- Maschewski, F. and Nosthoff, A.-V. (2018). Res publica ex machina: On neocybernetic governance and the end of politics. page 25.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.
- Mehlhase, A. (2015). *Konzepte für die Modellierung und Simulation strukturvariabler Modelle*. PhD thesis, Technische Universität Berlin.
- Miková, u., Gmíterko, A., and Hroncová, D. (2016). State space representation of dynamical systems. *American Journal of Mechanical Engineering*, page 5.
- Minsky, M. (1965). Matter, minds and models. In *Proceedings of the International Federation for Information Processing (IFIP) Congress*, pages pp. 45–49. Spartan Books.
- Murphey, Y. L., Guo, H., and Feldkamp, L. A. (2004). Neural learning from unbalanced data. *Applied Intelligence*, 21(2):117–128.
- Nagendra, S., Podila, N., Ugarakhod, R., and George, K. (2017). Comparison of reinforcement learning algorithms applied to the cart-pole problem. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 26–32. IEEE.

- Narayanan, V. and Jagannathan, S. (2018). Event-triggered distributed control of non-linear interconnected systems using online reinforcement learning with exploration. *IEEE Transactions on Cybernetics*, 48(9):2510–2519.
- Nayak, S. C. (2017). Development and performance evaluation of adaptive hybrid higher order neural networks for exchange rate prediction. *International Journal of Intelligent Systems and Applications*, 9(8):71–85.
- Nguyen, G. H., Bouzerdoum, A., and Phung, S. L. (2008). A supervised learning approach for imbalanced data sets. In *2008 19th International Conference on Pattern Recognition*, pages 1–4. IEEE.
- Nielsen, M. A. (2015). Neural networks and deep learning.
- Paoletti, S., Roll, J., Garulli, A., and Vicino, A. (2010). On the input-output representation of piecewise affine state space models. *IEEE Transactions on Automatic Control*, 55(1):60–73.
- Potočnik, B., Mušič, G., and Zupančič, B. (2004). A new technique for translating discrete hybrid automata into piecewise affine systems. *Mathematical and Computer Modelling of Dynamical Systems*, 10(1):41–57.
- Preyser, F., Heinzl, B., Raich, P., and Kastner, W. (2016). Towards extending the parallel-DEVS formalism to improve component modularity. In *ASIM-Treffen STS/G-MMS 2016 Workshop der ASIM/GI Fachgruppen STS und GMMS Tagungsband*, volume AR 51, page 8. ARGESIM Publisher Vienna.
- Preyser, F. J., Heinzl, B., and Kastner, W. (2019). Rpdevs abstract simulator. *SNE Simulation Notes Europe*, 29(2):79–84.
- Psichogios, D. C. and Ungar, L. H. (1992). A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511.
- Ragazzini, J. R., Randall, R. H., and Russell, F. A. (1947). Analysis of problems in dynamics by electronic circuits. *Simulation*, 3.

- Rodríguez, N. G. (2019). Machine learning for everyone in simple words. with real-world examples. [Online; posted 21-May-2019].
- Roll, J., Bemporad, A., and Ljung, L. (2004). Identification of piecewise affine systems via mixed-integer programming. *Automatica*, 40(1):37–50.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:4.
- Sahoo, S. S., Lampert, C. H., and Martius, G. (2018). Learning equations for extrapolation and control. *arXiv:1806.07259 [cs, stat]*.
- Sargent, R. G., Goldsman, D. M., and Yaacoub, T. (2016). A tutorial on the operational validation of simulation models. In *2016 Winter Simulation Conference (WSC)*, pages 163–177. ISSN: 1558-4305.
- Seatzu, C., Corona, D., Giua, A., and Bemporad, A. (2006). Optimal control of continuous-time switched affine systems. *IEEE Transactions on Automatic Control*, 51(5):726–741.
- Sen, D., Roy, A., Bhattacharya, A., Banerjee, D., and Bhattacharjee, C. (2011). Development of a knowledge based hybrid neural network (KBHNN) for studying the effect of diafiltration during ultrafiltration of whey. *Desalination*, 273(1):168–178.
- Shawash, J. (2012). *Generalised Correlation Higher Order Neural Networks, Neural Network operation and Levenberg-Marquardt training on Field Programmable Gate Arrays*claration Of Authorship. phdthesis, University College London.
- Shin, Y. and Ghosh, J. (1991). The pi-sigma network: an efficient higher-order neural network for pattern classification and function approximation. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume i, pages 13–18 vol.1.
- Shukla, D. and Paul, F. W. (1996). Computationally efficient control of nonlinear systems using orthonormal activation function based neural networks. *IFAC Proceedings Volumes*, 29(1):4995–5000.

- Siegfried, R. (2014). *Modeling and simulation of complex systems: a framework for efficient agent-based modeling and simulation*. Research. Springer Vieweg. OCLC: 931670812.
- Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and Juditsky, A. (1995). Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724.
- Solcany, V. (2008). Simulation algorithms for DEVS models. page 10.
- Szűcs, A., Kvasnica, M., and Fikar, M. (2012). Optimal piecewise affine approximations of nonlinear functions obtained from measurements. *IFAC Proceedings Volumes*, 45(9):160–165.
- Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. (2016). Training neural networks without gradients: A scalable ADMM approach. *Proceedings of the 33rd International Conference on Machine Learning*, page 10.
- Števek, J. and Kozák, . (2011). Matlab toolbox for PWA identification of nonlinear systems. In *Proceedings of the 18th International Conference on Process Control*, pages pp. 111–118. Fikar, M., Kvasnica, M.
- Števek, J., Szűcs, A., Kvasnica, M., Fikar, M., and Kozák, t. (2012a). Two steps piecewise affine identification of nonlinear systems. *Archives of Control Sciences*, 22(4):371–388.
- Števek, J., Szucs, A., Kvasnica, M., Kozák, S., and Fikar, M. (2012b). Smart technique for identifying hybrid systems. In *2012 IEEE 10th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 383–388. IEEE.
- Torrise, F. and Bemporad, A. (2004). HYSDEL—a tool for generating computational hybrid models for analysis and synthesis problems. *IEEE Transactions on Control Systems Technology*, 12(2):235–249.
- Vangheluwe, H. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *CACSD. Conference Proceedings. IEEE International*

- Symposium on Computer-Aided Control System Design (Cat. No.00TH8537)*, pages 129–134. IEEE.
- Velten, K. (2009). *Mathematical Modeling and Simulation*. WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim.
- Wang, J., You, J., Li, Q., and Xu, Y. (2012). Extract minimum positive and maximum negative features for imbalanced binary classification. *Pattern Recognition*, 45(3):1136–1145.
- Wermter, S. and Sun, R. (2000). *Hybrid Neural Systems*, volume 1778. Springer Berlin Heidelberg.
- Williams, H. P. (2013). *Model Building in Mathematical Programming*. John Wiley & Sons Ltd, 5th edition edition.
- Winkler, S., Körner, A., Bicher, M., and Breitenecker, F. (2017). A comparison of different modelling and simulation approaches for hybrid dynamical systems. In *UKSim-AMSS 19th International Conference on Computer Modelling and Simulation (UKSim)*, pages 97–102. IEEE.
- Yang, S., Lu, M., and Xue, H. (2008). Hybrid partial least squares and neural network approach for short-term electrical load forecasting. *Journal of Control Theory and Applications*, 6(1):93–96.
- Zaifei, L., Binglei, G., and Shiguan, Z. (2009). A neural network learning algorithm based on hybrid particle swarm optimization. In *2009 Chinese Control and Decision Conference*, pages 3255–3259.
- Zeigler, B. P. (1976). *Theory of Modeling and Simulation*. John Wiley.
- Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Second Edition. Academic Press.
- Zhu, F. and Antsaklis, P. J. (2013). Optimal control of switched hybrid systems: A brief survey.

# Stefanie Nadine Winkler

---

## *Curriculum Vitae*

### Education

- 2014–2020 **Doctoral Program in Technical Mathematics**, *TU Wien*, Vienna.  
Modelling and Simulation of Hybrid Systems
- 2013–2014 **Master of Science in Technical Mathematics**, *TU Wien*, Vienna.  
Specialized in Modelling and Simulation
- 2006–2012 **Bachelor of Science in Technical Mathematics**, *TU Wien*, Vienna.  
Specialized in Applied Mathematics
- 1998–2006 **Musikgymnasium Wien**, *Music High School*, Vienna.

### Doctoral Thesis

- Title *A Framework Including Artificial Neural Networks in Modelling Hybrid Dynamical Systems*
- Supervisors Felix Breitenecker & Andreas Körner
- Description This thesis defines a framework for applying artificial neural networks within hybrid modelling. The results of a case study involving two hybrid systems is analysed and conclusions regarding feasibility of the framework as well as extrapolation capabilities of the neural network approaches are drawn.

### Masters Thesis

- Title *Comparative Mathematical Modelling of Groundwater Pollution*
- Supervisors Felix Breitenecker & Martin Bicher
- Description In this work the focus is on the convection-diffusion equation. Using this equation the distributive behavior of the pollution influenced by a velocity field is described. Several approaches, ranging from analytical solutions to some chaotic particle movement, are used for realization. An important part of this work is the comparison of these different approaches regarding efficiency, accuracy and implementation.

### Experience

#### Vocational

- 2018–Present **Education Consultant**, *in Cooperation with DIGITALEd*, Waterloo, Canada.  
Educational Consultant for University Teacher, Sales Engineer, Trainer.
- 2016–2018 **Sales Engineer**, *in Cooperation with MAPLESOFT EUROPE*, Cambridge, UK.  
Team member of the regional sales team

*Gernotgasse 5/10 – 1150 Vienna, Austria*  
✉ [stefanie.winkler@tuwien.ac.at](mailto:stefanie.winkler@tuwien.ac.at)

- 2014–Present **Project Assistant**, TU WIEN, Vienna.
- 2013–2014 **Student Assistant**, TU WIEN, Vienna.  
Creating Online Examples and Teaching Mathematics for Freshmen
- 2013 **Project Collaborator**, UNIVERSITY OF VIENNA, Department of Business Administration, Vienna.  
Algorithm Development for Simulations of Competitive Buyer's Behaviour.
- Miscellaneous
- 2009–2013 **Backoffice**, DIEHL METERING GESMBH, Vienna.  
Tender Preparation, Accounting  
Customer and Order Administration via SAP.
- 2012 **Mathematics Teacher**, VHS Favoriten, Vienna.  
Adult Education Centre
- 2009–2012 **Violine Teacher**, VHS Penzing, Vienna.  
Adult Education Centre

---

## Research Interests

- Modelling Mathematical Modelling and Simulation  
Data Modelling and Deep Learning  
Comparison and Benchmarks in Simulation  
Applied Mathematics in Engineering
- Education Online Assessment in STEM Education  
Analysis of Students' Learning Paths  
Gamification in Higher Education

---

## Projects

- 2015–2016 **WTZ Albania**, *Mathematical Modelling and Simulation in Business Informatics.*
- 2014–2015 **ÖAD Hungary**, *Development of mathematical competence training modules with e-learning support.*
- 2013–2015 **BLAdEdu**, *Blended Learning Adult Education.*
- 2013–2014 **Kolleg LehreN**, *Mathematical Education for Engineers.*

---

## Selected Publications

- 2018 **S. Winkler, A. Körner, F. Breitenecker**, *Data modelling approach for physical systems*, in: 'Proceedings of the 30th European Modeling and Simulation Symposium, 2018', M. Affenzeller, A. Bruzzone, E. Jimenez, F. Longo, Y. Merkurjev, M. Piera (Hrg.); DIME Università di Genova, DIMEG University of Calabria, 30/Rende (2018), p. 310.  
ISBN: 978-88-85741-03-4

- 2018 **S. Winkler, M. Bicher, A. Körner, F. Breitenecker:**, *Modelling and Simulation of Hybrid Systems with Neural Networks*, in: 'MATHMOD 2018 Extended Abstract Volume', F. Breitenecker, W. Kemmetmüller, A. Körner, A. Kugi, I. Troch (Hrg.); ARGESIM Publisher, ARGESIM Report No. 55 (2018), p. 111 - 112..  
ISBN: 978-3-901608-91-9
- 2018 **S. Winkler, R. Leskovar, F. Gorgas, A. Körner:**, *Blended Marking in Online Prüfungen*, in: e-Prüfung Symposium 2018, Aachen, Deutschland, 'ePS 2018 Kompetenzorientiertes Lehren, Lernen und Prüfen', (2018); p. 17 - 18..  
ISBN: 978-2-87352-017-5
- 2018 **S. Winkler, R. Leskovar, A. Körner, F. Gorgas:**, *Influence on learning outcomes by human factors*, in: 'The 19th SEFI Mathematics Working Group Seminar on Mathematics in Engineering Education - Proceedings', European Society for Engineering Education (SEFI), 19 Brussels (2018); p. 121 - 126..  
ISBN: 978-2-87352-017-5
- 2017 **S. Winkler, A. Körner, M. Bicher, F. Breitenecker**, *A Comparison of Different Modelling and Simulation Approaches for Hybrid Dynamical Systems*, in: 'UKSim2017-AMSS 19th Intl. Conference on Mathematical Modelling & Computer Simulation', D. Al-Dabass, A. Orsoni, R. Cant, G. Jenkins (Hrg.); IEEE Computer Society Order Number E6161, 19/New York, USA (2017), p. 97 - 102..  
ISBN: 978-1-5386-2735-8
- 2016 **F. Breitenecker, A. Körner, S. Winkler:**, *The Assessment*, in: 'Blended Learning Quality - Concepts Optimized for Adult Education', P. Mazohl, H. Makl (Hrg.);Mazohl Publish, Wiener Neustadt, 2016, S. 95 - 102..  
ISBN: 978-3-901679-11-7
- 2016 **A. Körner, S. Winkler, F. Breitenecker:**, *Possibilities in State Event Modelling of Hybrid Systems*, in: 'Proceedings of the 9th EUROSIM Congress on Modelling and Simulation', Linköping University Electronic Press, (2016), 2 p..  
ISBN: 978-1-5090-4119-0
- 2016 **S. Winkler, A. Körner, M. Bicher, F. Breitenecker:**, *A New Approach Teaching Mathematics, Modelling and Simulation*, in: 'Proceedings of the 9th EUROSIM Congress on Modelling and Simulation', Linköping University Electronic Press, (2016), p. 416 - 421..  
DOI: 10.3384/ecp17142
- 2015 **S. Winkler, M. Bicher, F. Breitenecker:**, *Alternative approaches for groundwater pollution*, in: 'IFAC PapersOnLine', 48 (2015),1, p. 159 - 164..
- 2014 **A. Körner, S. Winkler, C. Rößler, F. Breitenecker:**, *Erlangen von Verständnis und Erlernen von Fertigkeiten in der Mathematik einmal anders*, in: 'Zeitschrift für Hochschulentwicklung', 9 (2014), 14, p. 101 - 115..
- 2013 **A. Körner, S. Winkler:**, *Making higher mathematics teachable in MTA*, in: '9th International PhD & DLA Symposium', B. Bachmann, A. Fülöp, E. Györi (Hrg.); Pollak Mihaly Faculty of Engineering and Information Technology, University of Pecs, Hungary, 2013, p. 90 - 91..  
ISBN: 978-963-7298-54-7

- 2012 **I. Hafner, M. Bicher, S. Winkler, U. Fitsch:**, *MMT - An E-Learning System based on Computer Numeric System for Teaching Mathematics*, in: 'Preprints MATHMOD 2012 Vienna - Full Paper Volume', F. Breitenecker, I. Troch (Hrg.); ARGESIM Publisher, ARGESIM Report No. 38 (2012), p. 251 - 252..
- 2010 **A. Körner, B. Heinzl, M. Rößler, S. Winkler, I. Hafner, G. Schneckenreither, G. Zauner, N. Popper:**, *BCP-A Benchmark for Hybrid Modelling and State Event Modelling*, in: 'Proceedings of the 7th Congress on Modelling and Simulation', M. Snorek, M. Cepek, Z. Buk, J. Drchal (Hrg.); Vol.2 Full Papers (2010), p. 1032 - 1042..  
ISBN: 978-80-01-04589-3
- 2010 **A. Zimmermann, V. Urbonaite, A. Körner, S. Winkler, S. Krause, M Kleinert:**, *Advanced Randomization and Grading in the E-Learning System Maple T.A.*, in: 'Proceedings of the 7th Congress on Modelling and Simulation', M. Snorek, M. Cepek, Z. Buk, J. Drchal (Hrg.); Vol.2 Full Papers (2010), p. 1209 - 1214..  
ISBN: 978-80-01-04589-3

## Computer skills

Advanced	MATLAB, Simulink, Maple, L <sup>A</sup> T <sub>E</sub> X, Moodle, Microsoft Office
Intermediate	html, CSS, phyton, OpenOffice, iOS Applications
Basic	Photoshop, Javascript, Inkscape, R, C, Swift

## Skills Summary

- Conversation** Great customer service skills. Deals with internal and external customers at all levels via online meeting, email and face-to-face to ensure successful communication through active listening and thoughtful questions.
- Problem Solving** Finds weak spots and bugs in processes to improve efficiency and benefits for customer and company. Crafting creative and suitable resolutions within the boundaries of customer and environment, all with a high level of quality.
- Team Player** Enjoys sharing knowledge and encouraging the development of others to achieve specific team goals. Merely talking about an issue can resolve the problem itself or create new ideas.
- Planning and Organizing** Refined planning and organizational skills that balance work, team support, and ad-hoc responsibilities in a timely and professional manner. Organises courses reaching from 20 to 2000 students including training courses for tutors. Plans and administrates conferences with 400 to 500 participants.

## Languages

German	<b>Native</b>	● ● ● ● ●
English	<b>Advanced</b>	● ● ● ● ○
French	<b>Intermediate</b>	● ● ○ ○ ○
Swedish	<b>Basic</b>	● ○ ○ ○ ○



## About the Author

*Stefanie Nadine Winkler* studied Technical Mathematics at the Vienna University of Technology (TU Wien). Already in her bachelor curriculum she got involved in modelling and simulation. At master level, she put emphasis on two subjects – modelling methods and system simulation, as well as Blended Learning for modelling and simulation and basic mathematics. The first topic led her to a benchmarking master thesis on *Comparative Mathematical Modelling of Groundwater Pollution*, where she showed her interest and her abilities in analysing different modelling approaches. The work in E-Learning opened her a door into the TU Wien AKMATH Group, where she organised and executed the introductory mathematic courses for all beginners at TU Wien including managing the E-Learning content team developing interactive online examples for mathematics. After the master thesis, she continued her work as part of the E-Learning group as well as her scientific work in the PhD curriculum. She investigated various approaches for modelling hybrid systems and analysed neural network approaches, which she combined in her PhD on neural net modelling for hybrid systems. Her deep knowledge in the Maple-based Math E-Learning system (Maple-TA/Möbius) resulted in consultancy work at European universities representing *Maplesoft Europe* and *DigitalEd*. After her PhD, she left university and started working for the management consulting firm *d-fine Austria*.



## About the Book

Modelling dynamical systems by equations and modelling dynamical behaviour by neural nets are up to now different worlds. This PhD thesis tries to combine these two worlds in the area of hybrid dynamical systems. The author first introduces modelling standards for hybrid dynamical systems using hybrid state automata followed by neural network practices for dynamic behaviour.

Based on these two areas the author develops a framework, which allows to replace certain elements of hybrid models by neural networks, as sketched by the cover pictures. On a mathematical basis of extending hybrid state automata by neural nets and training methods, the thesis discusses three different possibilities for application: the approximation of local dynamic behaviour, the prediction of the discrete processes and the replacement of the entire hybrid system applying neural networks. The defined formalism standardises the use of feed-forward networks in hybrid modelling and in general to enable an analysis of different network structures.

## About the Series

The **ASIM** series *Advances in Simulation / Fortschrittsberichte Simulation* presents new and recent approaches, methods, and applications in modelling and simulation. The topics may range from theory and foundations via simulation techniques and simulation concepts to applications. As the spectrum of simulation techniques and applications is increasing, books in these series present classical techniques and applications in engineering, natural sciences, biology, physiology, production and logistics, and business administration, upcoming simulation applications in social sciences, media, data management, networking, and complex systems, and upcoming new simulation techniques as agent-based simulation, co-simulation, and deep learning, etc.

The series puts emphasis on monographs with special character, as PhD theses, habilitation treatises, project reports and overviews on scientific projects. **ASIM** -- Arbeitsgemeinschaft Simulation, the German Simulation Society (part of **GI** – Gesellschaft für Informatik) has founded the series *Advances in Simulation / Fortschrittsberichte Simulation* together with **ARGESIM** Publisher Vienna in order to provide to the international simulation community a quick and cost-efficient print and e-book series with open access.

ISBN print  
ISBN 978-3-903311-12-1  
TU Verlag, Vienna, 2020  
[www.tuverlag.at](http://www.tuverlag.at)

ISBN ebook  
ISBN 978-3-903347-34-2  
ARGESIM Publisher, Vienna, 2020  
[www.argesim.org](http://www.argesim.org)

DOI ID  
DOI 10.11128/fbs.34