Doktoravhandlinger ved NTNU, 2023:168

Haakon Rennesvik Robinson

Trustworthy Machine Learning for Controlled Dynamical Systems

Doktoravhandling

NTNU
Norges teknisk-naturvitenskapelige universitet
Avhandling for graden
philosophiae doctor
Fakultet for informasjonsteknologi og
elektroteknikk
Institutt for teknisk kybernetikk

NTNU
Kunnskap for en bedre verden

Haakon Rennesvik Robinson

Trustworthy Machine Learning for Controlled Dynamical Systems

Thesis for the Degree of Philosophiae Doctor

Trondheim, June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

ISBN 978-82-326-7040-6 (printed ver.)
ISBN 978-82-326-7039-0 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

IMT-report 2023-11-W

Doctoral theses at NTNU, 2023:168

Printed by NTNU Grafisk senter

Summary

Recent advances in machine learning (ML) have helped solve many problems once thought impossible for computers to solve. These methods are increasingly used to develop flexible agents that learn to model and control through direct interaction with real-world systems. Despite encouraging results, it is difficult to predict the behaviour of such agents, which prevents their use in safety-critical applications. This thesis is a broad investigation into the safe application of machine learning methods to three different problems in the context of *controlled dynamical systems*, namely (i) Control design, (ii) System identification, and (iii) Verification.

One of the main ways to realise a learning controller is through Reinforcement learning (RL), where a learning agent is rewarded when it reaches a goal or performs a desired behaviour. A variety of RL methods are applied to the problems of path-following and collision avoidance (COLAV) for an autonomous ship. These two goals are occasionally in conflict. A parameter controlling the tradeoff between path-following and COLAV is given to the agent as an additional "insight", allowing the agent's priorities to be controlled during operation. However, even though the "conservativeness" of the agent is controllable using this scheme, experiments show that collisions can still occur on rare occasions, thus confirming the need for improved safety measures when using machine learning (ML) methods. A guarantee of safety can be achieved by introducing a failsafe controller known to be safe, a so-called *safety filter*. The second contribution of this thesis is the implementation of a predictive safety filter for the milliAmpere ferry. This auxiliary system is formulated as an optimal control problem (OCP), where the optimal solution is a minimal perturbation to a nominal input to the system such that the safety constraints are satisfied. The safety filter can be used with arbitrary controllers (e.g. RL agents) while guaranteeing safe operation.

System identification is closely related to ML. In both cases, a set of model structures is chosen, the parameters are selected such that the predictions match the available data without overfitting, and the model candidates are evaluated according to task-specific metrics. Model structures are often designed from first principles following physical laws, an approach referred to as physics-based modelling (PBM) in this thesis. PBM becomes more challenging as the complexity or scale of the system increases, and assumptions typically have to be made to make the resulting model tractable. Modern ML practitioners are increasingly turning to more flexible model structures such as neural networks (NNs) that can scale well without the need for bespoke model structures; this is referred to as Data-driven modelling (DDM) in this thesis. The combination of PBM and DDM techniques is also investigated in this thesis. We propose the physics-guided neural network (PGNN), a novel NN architecture where physics-based priors are injected into the intermediate layers of a NN. This method is found to improve accuracy and generalisation on a variety of dynamical system modelling tasks. However, the choice of injection layer plays a significant role, and there is no principled way to choose the correct layer a priori. Another way to augment PBM with DDM is through "boosting", i.e. training a second model to correct the errors of the first. This method is also known as the Corrective source term approach (CoSTA), and we apply it to an ablated model of an aluminium electrolysis cell and show that the corrected model is more accurate and stable than a purely data-driven model. Stability is a common issue when using NNs to model dynamical systems. This thesis investigates how regularisation and network architecture can affect the stability of the resulting system. The results show that introducing ℓ_1 regularisation and skip connections can significantly improve the predictive stability of NNs and that these measures are most effective when used together. These effects persist even when the amount of training data is reduced.

The third and last part of this thesis studies NNs as piecewise affine (PWA) systems, which is an exact correspondence when the only nonlinearities present are PWA functions. The work represents a step towards practical algorithms that can be used to verify the safety of black-box models. The first contribution is a memory-efficient algorithm for computing the linear pieces of a NN. However, the number of pieces grows exponentially with the network's depth and the input space's dimension, limiting the method to relatively small networks. Despite this, studying smaller systems can still yield insights. A series of experiments were performed on NNs trained to mimic a damped pendulum with different forms of regularisation. The linear regions of the network are recorded regularly during the training process. It is found that ℓ_1 regularisation significantly reduces the apparent number of regions, and a simple mechanism is proposed to explain this. Regularising using the ℓ_2 norm has a similar but lesser effect to ℓ_1 regularisation. Dropout regularisation

is not found to change the number of regions significantly but instead affects the structure of the regions. Weight normalisation is found to negate the observed effects. One motivation for these methods is that reducing the number of regions of a NN may make verification methods tractable. However, regularisation by itself appears insufficient. Instead, an additional algorithm to discard insignificant regions is proposed. A nonlinear benchmark based on modelling the vibrations of a wing/payload system is chosen as a case study. It is shown that regularisation can significantly reduce the number of regions at the cost of accuracy. Weight pruning is shown to have little effect. In comparison, the PWA approximation algorithm runs efficiently on a network with ten inputs and sacrifices little accuracy.

Contents

List of Tables

List of Figures

Acronyms

ACKTR Actor Critic Using Kronecker-Factored Trust Region
AI Artificial Intelligence
AIS Automatic Identification System
AN-RFMSE Average Normalised Rolling Forecast Mean Squared Error
APRBS Amplitude-modulated Pseudo-Random Binary Signal
ASV Autonomous Surface Vehicle
ATE Along-Track Error
CNN Convolutional Neural Networks
COLAV Collision Avoidance
COLREGS Convention On The International Regulations For Preventing Collisions At Sea
CoSTA Corrective Source Term Approach
CSS Convex Safe Set
CTE Cross-Track Error
DDM Data-driven Modelling
DDPG Deep Deterministic Policy Gradient
DL Deep Learning
DNN Deep Neural Network
DOF Degrees Of Freedom
GAE Generalised Advantage Estimation
HAM Hybrid Analysis And Modelling
IC Initial Conditions
LOS Line-of-Sight
LP Linear Programming
LQE Linear-quadratic Estimation
LQG Linear-quadratic Gaussian
LQR Linear-quadratic Regulator

LTI Linear Time-invariant

MDP Markov Decision Process

ML Machine Learning

MPC Model Predictive Control

MSE Mean Squared Error

NE North-East

NED North-East-Down

NLP Nonlinear Program

NN Neural Network

OCP Optimal Control Problem

ODE Ordinary Differential Equation

PBM Physics-based Modelling

PCA Principal Component Analysis

PCHIP Piecewise Cubic Hermite Interpolator

PCR Principal Component Regression

PDE Partial Differential Equation

PE Persistency Of Excitation

PF Potential Field

PGML Physics-guided Machine Learning

PGNN Physics-guided Neural Network

PINN Physics-informed Neural Network

PPO Proximal Policy Optimisation

PWA Piecewise Affine

PWANN Piecewise Affine Neural Network

PWL Piecewise Linear

ReLU Rectified Linear Unit

RFMSE Rolling Forecast Mean Squared Error

RK4 Runge-Kutta 4

RKF45 Runge-Kutta-Fehlberg Method

RL Reinforcement Learning

RMSE Root Mean Square Error

RNN Recurrent Neural Network

ROM Reduced Order Model

SGD Stochastic Gradient Descent

SNAME Society Of Naval Architects And Marine Engineers

TD Temporal Difference

TRPO Trust Region Policy Optimisation

VC Vapnik-Chervonenkis

XAI Explainable Artificial Intelligence

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work has been carried out at the Department of Engineering Cybernetics (ITK), with Professor Adil Rasheed as the supervisor and Professor Damiano Varagnolo as co-supervisor. This work has been supported through the EXAIGON project: Explainable AI systems for gradual industry adoption (grant no. 304843).

Acknowledgements

I would like to express my gratitude to my supervisors, Adil Rasheed and Damiano Varagnolo. Thank you for all our discussions and your endless enthusiasm and patience. The time and energy that you dedicate to your students is incredible, and I still don't know how you do it.

Despite a global pandemic putting a stop to many of my plans, I've been lucky enough to collaborate with many talented people. Thanks to Erlend Torje Berg Lundby for our many late nights, chats in the sauna, and for putting up with my perfectionism. Thanks to Aksel and Eivind, I really enjoyed working with you both.

I would also like to express my appreciation towards the people of the department. My time as a PhD would not have been the same without the many strange and wonderful characters roaming the halls of ITK. Thanks to my office mates, Trym Tengesdal, Asbjørn Espe, and Andreas Våge, who made my time in the office truly memorable. Sorry to the neighbouring offices for the weird noises, especially just before lunch. I'm grateful to everyone who took the time to listen to my problems and encourage me (shoutouts to Amer, Sverre and Jostein); this thesis would not have been possible without you. It has been a pleasure to get to know you all.

Thanks to my parents, Clive and Sissel, and my sister Helene for their unwavering love, encouragement, and support. And thanks to Alicia, for your love and positivity.

February 2023, Trondheim
Haakon

Chapter 1

Introduction

The exponential growth of cheap sensors, connected devices, and computing power has fuelled the development of machine learning (ML) algorithms that promise to solve problems that were widely thought impossible to tackle using computers. These advances have led to numerous breakthroughs in computer vision, natural language processing, drug design, and robotics. ML is already being used to automate many aspects of our lives to improve the productivity, efficiency, and safety of the many systems we depend on.

Data-driven modelling (DDM) has proved to be an attractive approach from an economic perspective because it reduces development costs and (in theory) yields a product that can be continuously improved simply by acquiring more data and computational resources. The success of this approach has inspired terms such as "Software 2.0" [1], where DDM is the preferred way to solve engineering problems. Furthermore, large-scale ML methods such as Deep learning (DL) are often based on the composition of many simple operations, which allows them to run efficiently in parallel and be applied to very large and high-dimensional datasets. However, relying on a pure DDM approach is problematic when dealing with constrained systems, e.g. self-driving cars, because many methods are black boxes, and it is challenging to place guarantees on their behaviour.

For safety-critical problems, it is preferable to use a physics-based modelling (PBM) approach where solutions are based on solid theory that can be used to reason about the current and future behaviour of the system. Analyses of this kind allow us to make preemptive design choices or protocols that ensure safe operation, which is the domain of control theory, feedback systems, and formal verification. However, modelling every aspect of a system to a high level of fidelity is often computationally intractable, forcing us to make assumptions, simplifications, and discretisations. These measures are almost always necessary when developing low-level control systems, which must be able to respond to changes in state hundreds of times a second while running on low-power microcontrollers. Furthermore, we may simply fail to accurately describe aspects of our data that we do not fully understand, resulting in an incomplete, unfaithful, or overly simplified representation of the original system. In addition, when the equations are solved using numerical algorithms, the stability of these must be verified. These issues are becoming increasingly important as our systems become more complex, interconnected, and sophisticated.

This thesis studies how DDM and PBM can be combined effectively in the context of *controlled dynamical systems*, a strategy which we refer to as hybrid analysis and modelling (HAM). Fig. 1.1 shows a generic block diagram of such a system, which also serves to structure the thesis. In the following sections, we will briefly discuss each of the blocks and symbols shown in the figure and how modern control methods approach them.

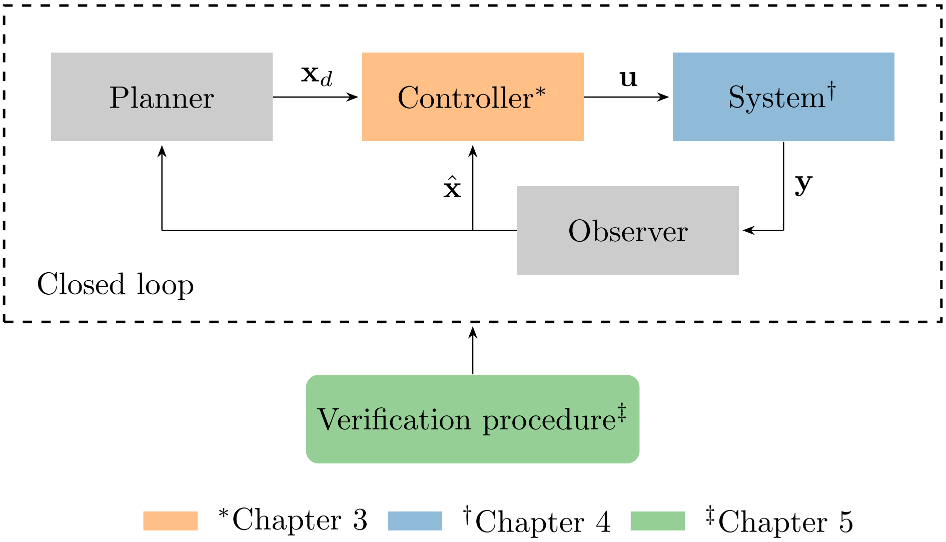


Figure 1.1: Structure of the thesis as a generic feedback control system.

1.1 Modelling and control of dynamical systems

Modern control methods are largely based on the state-space representation of controlled dynamical systems [2]:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{h}(\mathbf{x}, \mathbf{u})\end{aligned} \tag{1.1}$$

where $\mathbf{x}(t)$ is the state of the system at time t, $\mathbf{u}(t)$ is the input signal, $\mathbf{y}(t)$ is what we can measure from the system, and $\mathbf{f}, \mathbf{h}$ are the dynamics and measurement functions respectively. Preliminary details about dynamical systems and relevant examples are given in Chapter 2.

Successfully modelling $\mathbf{f}$ can allow us to predict how our system will change in response to some input $\mathbf{u}$, which can help us control the system as we wish. An algorithm that computes $\mathbf{u}$ is known as a *control law*, *control policy*, or simply a *controller*. When a controller provides feedback to some process, we refer to the whole system as a *closed-loop*, as illustrated in Fig. 1.1. However, designing a controller that automatically chooses $\mathbf{u}$ is nontrivial.

In the field of model checking and formal verification, there are two fundamental properties that we wish to build into our systems: *liveness* and *safety* [3]. While the former is not commonly used within the context of control theory, it nonetheless transfers well. The liveness property is often summarised as "good things will happen", while safety implies that "bad things will not happen", e.g. a self-driving car should reach some destination without driving recklessly. Liveness and safety are often two sides of the same coin; unsafe behaviour typically prevents the system from achieving the desired goal. It is often trivial to stay safe if no action needs to be taken. Most real-world systems are stochastic to some degree; in these cases, we can only try to maximise the probability of liveness and safety. The *performance* of the system is also often of interest (e.g. fuel consumption), which implies that some control policies are "better" than others. Methods that select a controller that provably maximises some performance metric are known as *optimal control methods*.

There are multiple subproblems to consider when designing "live" and "safe" systems.

(System identification)	Determine $\mathbf{f}$ and $\mathbf{h}$
(Control design)	Develop an algorithm to choose $\mathbf{u}$
(State estimation)	Estimate $\mathbf{x}$ from noisy data $\mathbf{y}$
(Verification)	Check that Eq. (1.1) behaves as intended

Each problem corresponds to one of the blocks in Fig. 1.1, where the planner block also falls under the field of control design. This is because a planning algorithm can be viewed as just another controller. In practice, it is common to nest multiple control loops within one another where each outer loop passes on the desired state to the inner loops. Connecting multiple modules in a feedback system can have unexpected, sometimes destabilising consequences [2]. This fact highlights the importance of the verification problem. Care must always be taken to ensure that the modules play nicely together and continue to do so for all situations the system might encounter.

1.1.1 System identification

System identification can be summarised in three steps: *data collection*, *model structure selection*, and *model fitting/validation* [4]. In this thesis, we will mostly assume that the data has already been collected and focus on the last two steps. In broad terms, the structure of a model refers to some equation or process through which the prediction $\hat{\mathbf{y}}(t)$ is computed from past data and parameters. Following the notation of [5], this can be seen as the mapping:

$$\begin{aligned} \hat{\mathbf{y}}(t \mid \boldsymbol{\theta}) &= \hat{\mathbf{f}}(\boldsymbol{\psi}(t), \boldsymbol{\theta}) \\ \boldsymbol{\psi}(t) &= \boldsymbol{\psi}(Z_t) \end{aligned} \tag{1.2}$$

where $\boldsymbol{\theta}$ represents the parameters of the model, $Z_t = \{(\mathbf{y}_\tau, \mathbf{u}_\tau) \mid \forall \tau < t\}$ is the set of all past observations, and $\boldsymbol{\psi}(t)$ represents the vector of *regressor variables* that summarise the past observations at time t. For example, $\boldsymbol{\psi}(t)$ might include the last k measurements made, some running statistics, or a state estimate $\hat{\mathbf{x}}(t)$. Multiple model structures might be considered and ranked using task-specific metrics in the validation/test stage.

One of the most fundamental structures that can be selected is the linear time-invariant (LTI) system:

$$\hat{\mathbf{f}}(\boldsymbol{\psi}(t), \boldsymbol{\theta}) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1.3}$$

where the parameters $\boldsymbol{\theta}$ take the form of the matrices $\mathbf{A}$ and $\mathbf{B}$. Classical control theory has given us many strong results for LTI systems, allowing us to determine many properties, e.g. stability, controllability, and observability. Many of these results also play an essential role in the local analysis of nonlinear systems. The *superposition principle* for linear systems implies that they can be characterised by their frequency response to sinusoidal input signals, enabling a host of powerful frequency domain methods based on manipulating the poles and zeros of the resulting transfer function. Other linear methods include ARMAX, output-error, and finite-impulse response models.

If a linear system is insufficient, many nonlinear model structures have been proposed over the years. Despite the near-infinite ways this can be done, many existing methods are readily described as a basis expansion of the form [5]:

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{A}\boldsymbol{\kappa}\left(\mathbf{W}(\boldsymbol{\psi} - \mathbf{b})\right) \\ \boldsymbol{\kappa}(\mathbf{z}) &= \left[\kappa_1(z_1), \ldots, \kappa_n(z_n)\right]^\top \end{aligned} \tag{1.4}$$

where the rows of $\mathbf{W}$ and elements of $\mathbf{b}$ are parameters associated with the kth scalar basis function κ_k, and $\mathbf{A}$ consists of some weighting parameters. These basis functions are often parameterised using a single "mother-basis" $\kappa_1 = \cdots = \kappa_n = \kappa$ that is scaled and translated using the parameters $\mathbf{W}$ and $\mathbf{b}$. Fourier series can be expressed in this way by setting $\kappa = \cos$ and setting frequency and phase using $\mathbf{W}$ and $\mathbf{b}$, respectively. The wavelet transform can be expressed similarly, as can B-splines. Changing the input to the basis functions to a quadratic of $\boldsymbol{\psi}$ yields radial basis functions, which can represent sums of Gaussians and even nearest neighbour algorithms. Eq. (1.4) is also the structure of a 1-layer neural network, with activation function $\boldsymbol{\kappa}(\mathbf{z})$. Adding additional layers is simple: take the output of Eq. (1.4) as a new set of regressor variables $\boldsymbol{\psi}^{[2]}$, and repeat with a new set of parameters $\mathbf{W}^{[2]}$ and $\mathbf{b}^{[2]}$.

A suitable model structure can often be derived from the mechanics and physics of a system. This strategy is sometimes known as *white-box modelling* and falls under the PBM approach in this thesis. Conversely, if a model is selected without any physical insight, it is known as a *black-box model*, which corresponds to DDM. Once a suitable structure has been determined, the optimal (sometimes sub-optimal) parameters for the model are estimated from the data. When the parameters appear linearly in the equation (e.g. Eq. (1.4)), linear regression can be used to identify them.

1.1.2 Control design

Similarly to system identification, control design starts with selecting an appropriate controller structure (often highly dependent on the model structure). Then the parameters of this structure are chosen according to some procedure. The parameters are then commonly tuned experimentally by observing the response of the closed-loop system. These procedures can be broadly grouped into two categories: model-based control and model-free control.

Model-based methods typically obtain the closed-loop system by plugging the controller equations into the model and directly computing properties of interest. The parameters can then be specified to get the desired response. For example, the *pole placement* method applies linear state feedback to an LTI system and the poles of the closed-loop system can be specified as desired. There are many methods for nonlinear systems, and we cannot hope to cover all of them here. Notable examples include (i) Feedback linearisation: the controller additively cancels out the nonlinear dynamics and replaces them with stable LTI dynamics (ii) Gain scheduling: linearise at multiple states and apply different linear state feedback at each (iii) Integrator backstepping: Recursively stabilise nested feedback loops (iv) Sliding-mode control: forces the state onto a lower-dimensional manifold with desirable dynamics.

Optimal control methods represent the controller as an optimisation problem over a time horizon with the model equations as a constraint. The infinite horizon optimal control problem (OCP) has the form:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \int_0^\infty & J\left(\mathbf{x}(t), \mathbf{u}(t)\right) \\ s.t. \quad \dot{\mathbf{x}}(t) &= \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t)\right) \\ \mathbf{x}(0) &= \mathbf{x}_0 \\ \mathbf{x}(t) &\in \mathcal{X} \quad \forall t \\ \mathbf{u}(t) &\in \mathcal{U} \quad \forall t \end{aligned} \tag{1.5}$$

The linear-quadratic regulator (LQR) problem is a quadratic OCP for linear systems with an infinite time horizon that can be solved explicitly using the algebraic Ricatti equation, resulting in an optimal linear state-feedback law. Suppose additional constraints are added to the LQR problem. In that case, it can be discretised and solved over a finite horizon for a given initial state to obtain an optimal trajectory and input sequence. Repeatedly solving the problem and applying the first input in the sequence is known as model predictive control (MPC). Although suboptimal, this works well in practice and is widely used due to the ease of incorporating constraints into the controller. Interestingly, the linear MPC problem can be solved explicitly for all initial states, yielding a constant piecewise affine (PWA) controller. This method also works if the system dynamics are PWA. However, despite the savings in execution time, offline computations are demanding, and the memory cost is often too prohibitive for low-level controllers. With developments in computing power and optimisation algorithms, MPC is also becoming more and more applicable to nonlinear systems.

Model-free methods estimate control parameters directly from data. These methods are commonly associated with Reinforcement learning (RL), but many adaptive control methods also fall under this category. Most RL methods do not learn a model and instead try to learn the expected future reward associated with states and actions (as a *value function*). The control strategy is to pick the action that maximises this value function at the current state. One of the issues with RL is that it is relatively data-inefficient. Model-based RL methods that attempt to solve the dual control problem show some promise here. However, current model-free RL methods yield better performance because they explore the state space more thoroughly and find better solutions [6].

1.1.3 State estimation

State estimation, also known as *sensor fusion*, is a family of methods that attempt to reconstruct accurate estimates of $\mathbf{x}$ from noisy data. The Luenberger observer is one of the most straightforward approaches; the state estimate is forced to track the measurements $\mathbf{y}$ by using linear feedback [7]. The Kalman filter is one of the most celebrated state estimation methods for linear systems and is also known as linear-quadratic estimation (LQE) because it is the dual of the LQR problem [8]. When LQR and LQE are used together, this is referred to as linear-quadratic Gaussian (LQG) control. This duality can also be extended to nonlinear systems, implying that control and state estimation methods are generally interchangeable. However, this only holds for a subset of control problems [9]. The Extended Kalman filter can be applied to nonlinear systems that again utilise local linearisation to update the estimate [10]. However, the linearisation can lead to poor estimates of the state covariance. The unscented Kalman filter addresses this by sampling a small number of points around the current mean, propagating these through the system equations, and then using the result to estimate the covariance instead [11]. Moving horizon estimators frame the estimation problem as an optimisation problem and are analogous to MPC.

In ML applications, state estimation methods have not been widely applied. For classification and regression problems, injecting noise into the training data is seen as a way to counter overfitting and improve the generalisation of a model. On the other hand, architectures such as autoencoders are often used to learn a *latent space* from which it is possible to reconstruct the input data [12]. This concept has also been applied to measurements over time [13], and thus can be linked to state estimation. A key difference is that state estimation techniques attempt to reconstruct a predefined set of variables from measurements. In contrast, autoencoding ML methods attempt to learn a mapping onto an undefined latent space that allows for maximal retention of information. The latent space trick has also been applied to model-based RL methods, such as the Dreamer methods from Google [14, 15].

1.1.4 Verification

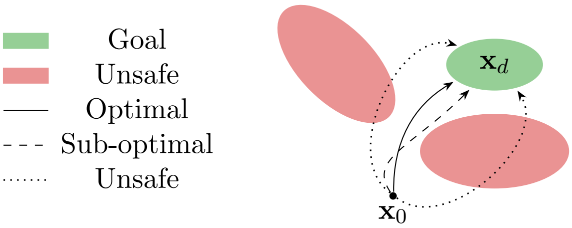


Figure 1.2: Illustration of safe vs. unsafe and sub-optimal trajectories.

Formally, a verification procedure checks that a system has some property [16]. As previously mentioned, the two fundamental properties of interest for controlled dynamical systems are liveness and safety. A closed-loop system follows specific trajectories determined by the system dynamics and the current control policy. Fig. 1.2 shows the difference between safe, unsafe, and sub-optimal trajectories.

The most obvious way to verify a system is to test it in vivo or in simulation under different conditions, which should always be done to benchmark its performance accurately. However, there is always the possibility that an unsafe trajectory is missed.

Typically, the goal of a controller is to force the state $\mathbf{x}$ towards some desired value, denoted $\mathbf{x}_d$, s.t. the error $\mathbf{x}_d - \mathbf{x} \to \mathbf{0}$ over time. In other words, the origin $\mathbf{0}$ should be *asympotically stable* for the closed-loop system $\mathbf{f}(t, \mathbf{x}, \mathbf{u})$ [17]. A “good” controller should ensure this for a large set of initial states (known as the *region of attraction*) while also resulting in a well-behaved trajectory (fast, non-oscillatory).

Many criteria for checking stability have been developed. Linear stability involves linearising the closed-loop system at $\mathbf{0}$ and checking that all eigenvalues of the system are negative [18]. Lyapunov stability is a more general property; if a non-negative, energy-like function $V(\mathbf{x})$ of the variables can be found, and V decreases over time, then the system should converge to the lowest energy state ($\mathbf{x} = \mathbf{0}$) [17].

Safety w.r.t. a constraint set can be verified using barrier functions, which involves showing that the closed-loop system never crosses some level set $b(\mathbf{x}) = 0$ of the barrier function $b(\mathbf{x})$ for a given set of initial conditions. If this level set completely separates the initial conditions and the unsafe states, then safety is guaranteed.

Optimal control methods with the form Eq. (1.5) formulate both liveness and safety requirements via constrained optimisation. The desired properties are therefore guaranteed by construction and hold for MPC under the additional conditions of recursive feasibility and a terminal constraint/cost [19].

The issue with verification via testing/simulation is that it is impossible to verify all feasible trajectories. This problem becomes more and more difficult in higher dimensions. In some cases, it is possible to perform set-based computations that predict all reachable states of a system, given a set of initial conditions and admissible inputs. This approach is known as *reachability analysis* and is becoming more and more practical with advances in algorithms and computational power. Reachability analysis methods have been applied to linear systems up to a billion dimensions [20]. A challenge is that reachable sets usually require some overapproximation. The *wrapping effect* occurs when reachable sets are overapproximated over multiple timesteps, resulting in a very conservative estimate [21]. Reachability methods for general nonlinear systems is an ongoing research topic; see, e.g. GoTube [22].

1.1.5 Connections to Explainable AI and Model Interpretability

This thesis began with the idea that improving the *interpretability* of DDM could help build more trustworthy systems. One approach to this issue is explainable artificial intelligence (XAI), where the aim is to develop algorithms for identifying the causes and reasons behind the output of a black-box model. However, it quickly became apparent that many methods for *explainability* could only provide insight into the decision-making of a model after the fact. While this is helpful for learning from accidents and errors, it would be preferable to make preemptive design choices or protocols that ensure safe operation. This is the domain of control theory, feedback systems, and formal verification. A brief survey of model interpretability and explainability is nonetheless given for completeness.

The right to explanation is a legal right [23], which is best understood in the context of decision-making where it is vital to justify the reasoning behind a decision. For example, we might want to know the factors behind a particular classification of an image.

In traditional statistical modelling, a model's *interpretability* is seen as important because it enables researchers to make inferences based on the model itself and derive more knowledge from the system [24]. Because of this, although there is often a tradeoff between interpretability and predictive accuracy (because simpler models are easier to understand), the interpretable model is usually chosen.

The increasing amount of available data has enabled the development of learning models with startling predictive power at the cost of interpretability. In particular, neural networks (NNs) and deep learning models have seen extraordinary success and interest. One of the most surprising discoveries is that these models can avoid overfitting and generalise reasonably well, despite being highly over-parameterised. For example, the widely publicised natural language model GPT-3 by OpenAI contains roughly 175 billion parameters [25]. Interpreting the outputs of such models by examining the internals is daunting. Naturally, there is a growing debate within the field on the role of interpretability and XAI, mainly around the various conflicting and ill-posed definitions, if model interpretability is necessary, and if it is even ethical to compromise predictive ability in the pursuit of it.

The notion of an interpretable model can be divided into three concepts [26]:

- Simulatability: Can a human follow the internal logic of the model?
- Decomposability: Is the model separable into multiple modules?
- Learning/algorithmic transparency: Is the learning process well understood?

According to these definitions, even linear models are not necessarily more interpretable than NNs, as the only trade-off is between decomposability and learning transparency. Indeed, NNs can identify useful and complex features, while a linear model must rely on the developer to supply existing features. They also argue that calls for interpretability usually arise when there is a mismatch between the real objective and the objective that a model is trained on (e.g. mean squared error), which can happen when the interpretability requirements are subjective. From this perspective, the danger lies in sacrificing model performance or generating misleading post hoc explanations to satisfy ill-conceived notions of interpretability.

There is also considerable debate on what constitutes an effective explanation. There have been many proposals, e.g. textual and visual explanations, linearisations near a specific input, and finding similar examples (i.e. an analogy). Gilpin et al. [27] define two essential qualities of explanations: *interpretability* and *completeness*, where an interpretable explanation is understandable by a human, and a complete explanation describes the operation of the model to a high level of accuracy. For example, a list of all the mathematical operations that a model performs is complete but not always very interpretable. Likewise, a simple cause-and-effect explanation is interpretable but does not necessarily reflect the model's inner workings. Herman et al. [28] argue that defining interpretability this way biases explanations towards human requirements of simplicity, encouraging *persuasive* explanations that might be misleading. However, there will always be a trade-off between interpretability and completeness, and the trick might be to allow the user to control this trade-off [27]. Other recent attempts to demystify the concept of interpretability and make the discussion more rigorous have focused on the human aspect of explanations and model interpretation [29].

The definition of interpretability that a researcher subscribes to is naturally determined by their aims and that of their field. This thesis uses the following definition:

the ability to verify a controlled dynamical system's liveness and safety properties.

1.2 Objectives and Contributions

The thesis is based on the following original works.

Journal publications

Paper A Eivind Meyer, Haakon Robinson, Adil Rasheed and Omer San. 'Taming an Autonomous Surface Vehicle for Path Following and Collision Avoidance Using Deep Reinforcement Learning'. In: *IEEE Access* 8 (2020), pp. 41466–41481

Paper B Haakon Robinson, Suraj Pawar, Adil Rasheed and Omer San. 'Physics Guided Neural Networks for Modelling of Non-Linear Dynamics'. In: *Neural Networks* 154 (2022), pp. 333–345

Paper C Haakon Robinson, Erlend Lundby, Adil Rasheed and Jan Tommy Gravdahl. 'Deep Learning Assisted Physics-Based Modeling of Aluminum Extraction Process'. Conditional Acceptance: Engineering Applications of Artificial Intelligence. 2023

Conference publications

Paper D Haakon Robinson. 'Approximate Piecewise Affine Decomposition of Neural Networks'. In: *Proceedings of the 19th Symposium on System Identification (SYSID)*. vol. 54. 2021, pp. 541–546

Paper E Aksel Vaaler, Haakon Robinson, Trym Tengesdal and Adil Rasheed. 'Safety Filter for Small Passenger Ferry'. Accepted: 42nd International Conference on Ocean, Offshore & Arctic Engineering. 2023

Preprints

Paper F Haakon Robinson, Adil Rasheed and Omer San. 'Dissecting Deep Neural Networks'. Jan. 2020. arXiv: arXiv:1910.03879

Paper G Erlend Torje Berg Lundby, Haakon Robinsson, Adil Rasheed, Ivar Johan Halvorsen and Jan Tommy Gravdahl. 'Sparse Neural Networks with Skip-Connections for Nonlinear System Identification'. Pending Review: CDC 2023. Jan. 2023. arXiv: arXiv:2301.00582

Objectives and Contributions of the candidate

Papers A to G address the following research questions (RQs).

RQ 1:	What safety issues can arise when using ML controllers?	Paper A
RQ 2:	How can safety be guaranteed when using ML controllers?	Paper E
RQ 3:	What benefits and issues arise when using NNs for system identification, and what role does the architecture of the NNs play?	Papers B, C and G
RQ 4:	How can domain knowledge be used together with NNs for system identification?	Paper B
RQ 5:	What are the challenges related to the verification of NN-based systems?	Paper F
RQ 6:	How can the NN verification problem be simplified?	Paper D

Papers B, D and F were mainly written by the candidate, with coauthors assisting with discussions and proofreading. Papers C and G are the result of collaborations with colleagues who contributed equally to all aspects of the work. Papers A and E are the products of master student projects the candidate co-supervised, where the students contributed to the software implementation, computational experiments, writing, and proofreading. In addition to the supervision, the candidate made significant contributions to the conceptualisation, methodology, software implementation, writing, and proofreading steps.

1.3 Outline of the thesis

Chapter 2 presents the relevant background for this thesis and is best used as a reference. Chapter 3 investigates how machine learning methods can be used to learn controllers from data and is based on Papers A and E. Chapter 4 describes some ways to develop dynamic models based on machine learning techniques and how to utilise prior knowledge to improve their generalisation and is based on Papers B, C and G. Chapter 5 describes a family of methods that treat many ML methods as PWA functions, opening up possibilities for safety verification and control synthesis, and is based on Papers D and F. Chapter 6 concludes the thesis by discussing future work.

Chapter 2

Preliminaries

This chapter develops the fundamental concepts described in the introduction to provide the theoretical background necessary for the remainder of the thesis.

2.1 Dynamical systems

This thesis studies the application of machine learning techniques to *dynamical systems*. A dynamical system can be described as a set of differential equations that describe the evolution of a state $\mathbf{x}$ on some domain $\Omega \subset \mathbb{R}^n$. Placing additional conditions on the initial and boundary values of the state gives the following boundary-value problem:

$$\begin{aligned} \mathcal{L}\mathbf{x}(t, \mathbf{z}) &= \mathbf{f}(t, \mathbf{x}, \mathbf{z}, \mathbf{u}) && \forall (t, \mathbf{z}) \in \Omega \\ \mathcal{B}\mathbf{x}(t, \mathbf{z}) &= \mathbf{g}(t, \mathbf{z}) && \forall (t, \mathbf{z}) \in \partial\Omega \end{aligned} \tag{2.1}$$

In this formulation, the state $\mathbf{x}$ can vary in time t and along some spatial dimensions $\mathbf{z}$. The differential operator $\mathcal{L}$ maps $\mathbf{x}$ to some function of its partial derivatives (e.g. $\partial/\partial t$), and $\mathbf{f}$ is a source term that describes the dynamics of the system as a function of $(t, \mathbf{x}, \mathbf{u})$ on the domain Ω. The differential operator $\mathcal{B}$ and function $\mathbf{g}$ determine the initial and boundary conditions of the problem on $\partial\Omega$.

Eq. (2.1) generalises many typical ordinary differential equations (ODEs) and partial differential equations (PDEs), and rarely has a closed form solution. Instead, the equation can be solved numerically using, e.g. finite differences, finite element analysis, or by projecting the equations onto an orthogonal basis, among many other methods. The following sections give examples of dynamical systems and how to rewrite them in the form of Eq. (2.1). The systems in this section exhibit a wide range of nonlinear phenomena, including periodic and aperiodic solutions, limit cycles, and chaos. The motivation for investigating multiple systems in this thesis is to show that ML methods are broadly applicable to various use cases, despite their generality.

Example 2.1.1 (Damped pendulum)**.** The equations of motion for a damped pendulum are written as follows:

$$\ddot{\theta} + \frac{d}{m}\dot{\theta} = -\frac{g}{L}\sin\theta, \quad \theta(0) = \theta_0 \tag{2.2}$$

where $g = 9.81\,\mathrm{m\,s^{-1}}$ is the gravitational acceleration, L is the length of the (massless) pendulum arm, m is the mass of the ball, d is the damping coefficient, and θ_0 is the initial angle of the pendulum. In this case, we have

$$\mathcal{L} = \left(\frac{\partial^2}{\partial t^2} + \frac{d}{m}\frac{\partial}{\partial t}\right), \quad \mathbf{f}(\theta) = -\frac{g}{L}\sin\theta \tag{2.3}$$

Naturally, Eq. (2.2) can also be converted to two first order ODEs.

$$\frac{\partial}{\partial t}\begin{bmatrix}\theta \\ \omega\end{bmatrix} = \begin{bmatrix}\omega \\ -\frac{g}{L}\sin\theta - \frac{d}{m}\omega\end{bmatrix} \tag{2.4}$$

Now we have:

$$\mathcal{L} = \frac{\partial}{\partial t}, \quad \mathbf{f}(\theta, \omega) = \begin{bmatrix}\omega - \frac{g}{L}\sin\theta - \frac{d}{m}\omega\end{bmatrix} \tag{2.5}$$

This last formulation shows that the choice of $\mathcal{L}$ and $\mathbf{f}$ are not always unique.

Example 2.1.2 (Generalised ship model)**.** The model formulation is based on the Robot-Inspired Model for marine craft [37], where the motion of the ship is restricted to the xy plane:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{R}(\psi)\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} &= \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{wind}} - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} \end{aligned} \tag{2.6}$$

where the pose of the ship (position and heading) is denoted $\boldsymbol{\eta} = [x \quad y \quad \psi]^\top$, and the velocities are written as $\boldsymbol{\nu} = [u \quad v \quad r]^\top$. The $\mathbf{M}$, $\mathbf{C}(\boldsymbol{\nu})$, and $\mathbf{D}(\boldsymbol{\nu})$ are the inertia, Coriolis, and Damping matrices, respectively. The rotation matrix $\mathbf{R}(\psi)$ is written as:

$$\mathbf{R}(\psi) = \begin{bmatrix}\cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1\end{bmatrix} \tag{2.7}$$

[38] expresses the inertia and Coriolis matrices as:

$$\mathbf{M} = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & m_{23} \\ 0 & m_{32} & m_{33} \end{bmatrix}, \quad \mathbf{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & c_{13}(\boldsymbol{\nu}) \\ 0 & 0 & c_{23}(\boldsymbol{\nu}) \\ c_{31}(\boldsymbol{\nu}) & c_{32}(\boldsymbol{\nu}) & 0 \end{bmatrix}$$

$$\begin{aligned} c_{13}(\boldsymbol{\nu}) &= -m_{22}v - m_{23}r \\ c_{23}(\boldsymbol{\nu}) &= m_{11}u \\ c_{31}(\boldsymbol{\nu}) &= -c_{13}(\boldsymbol{\nu}) \\ c_{32}(\boldsymbol{\nu}) &= -c_{23}(\boldsymbol{\nu}) \end{aligned} \tag{2.8}$$

Note that these matrices combine rigid body and *added mass* terms, the latter being a virtual mass added to the system due to the volume of fluid accelerated along with the vessel. This trick simplifies the identification problem. The decoupled damping matrix $\mathbf{D}(\boldsymbol{\nu})$ can be written as:

$$\begin{aligned} \mathbf{D}(\boldsymbol{\nu}) &= \begin{bmatrix} d_{11}(\boldsymbol{\nu}) & 0 & 0 \\ 0 & d_{22}(\boldsymbol{\nu}) & d_{23}(\boldsymbol{\nu}) \\ 0 & d_{32}(\boldsymbol{\nu}) & d_{33}(\boldsymbol{\nu}) \end{bmatrix} \\ d_{11}(\boldsymbol{\nu}) &= -X_u - X_{|u|u}|u| - X_{uuu}u^2 \\ d_{22}(\boldsymbol{\nu}) &= -Y_v - Y_{|v|v}|v| - Y_{|r|v}|r| - Y_{vvv}v^2 \\ d_{23}(\boldsymbol{\nu}) &= -Y_r - Y_{|v|r}|v| - Y_{|r|r}|r| \\ d_{32}(\boldsymbol{\nu}) &= -N_v - N_{|v|v}|v| - N_{|r|v}|r| \\ d_{33}(\boldsymbol{\nu}) &= -N_r - N_{|v|r}|v| - N_{|r|r}|r| - N_{rrr}r^2 \end{aligned} \tag{2.9}$$

where the coefficients appearing in $d_{ij}(\boldsymbol{\nu})$ are ship-specific parameters that can be identified experimentally. The subscripts refer to the corresponding product of terms in $\mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu}$, X, Y correspond to forces along the $\{x\}$ and $\{y\}$ axes of the BODY frame respectively, and N refers to the torque about the $\{z\}$ axis of the BODY frame.

Example 2.1.3 (Hall-Héroult process)**.** This system describes an industrial process for extracting aluminium using electrolysis. The system is described by a set of ODEs:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{2.10}$$

where $\mathbf{x} \in \mathbb{R}^8$ and $\mathbf{u} \in \mathbb{R}^5$ represent the time-varying states and inputs of the system, respectively. The complete set of equations are:

$$\dot{x}_1 = \frac{k_1(g_1 - x_7)}{x_1 k_0} - k_2(x_6 - g_1) \tag{2.11a}$$

$$\dot{x}_2 = u_1 - k_3 u_2 \tag{2.11b}$$

$$\dot{x}_3 = u_3 - k_4 u_1 \tag{2.11c}$$

$$\dot{x}_4 = -\frac{k_1(g_1 - x_7)}{x_1 k_0} + k_2(x_6 - g_1) + k_5 u_1 \tag{2.11d}$$

$$\dot{x}_5 = k_6 u_2 - u_4 \tag{2.11e}$$

$$\begin{aligned}\dot{x}_6 = \frac{\alpha}{x_2 + x_3 + x_4}\Bigg[& u_2 g_5 + \frac{u_2^2 u_5}{2620 g_2} - k_7(x_6 - g_1)^2 \\ & + k_8 \frac{(x_6 - g_1)(g_1 - x_7)}{k_0 x_1} - k_9 \frac{x_6 - x_7}{k_{10} + k_{11} k_0 x_1}\Bigg]\end{aligned} \tag{2.11f}$$

$$\begin{aligned}\dot{x}_7 = \frac{\beta}{x_1}\Bigg[& \frac{k_9(g_1 - x_7)}{k_{15} k_0 x_1} - k_{12}(x_6 - g_1)(g_1 - x_7) \\ & + \frac{k_{13}(g_1 - x_7)^2}{k_0 x_1} - \frac{x_7 - x_8}{k_{14} + k_{15} k_0 x_1}\Bigg]\end{aligned} \tag{2.11g}$$

$$\dot{x}_8 = k_{17} k_9 \left(\frac{x_7 - x_8}{k_{14} + k_{15} k_0 \cdot x_1} - \frac{x_8 - k_{16}}{k_{14} + k_{18}} \right), \tag{2.11h}$$

where the intrinsic properties g_i of the bath mixture are given as:

$$\begin{aligned} g_1 = {} & 991.2 + 112 c_{x_3} + 61 c_{x_3}^{1.5} - 3265.5 c_{x_3}^{2.2} \\ & - \frac{793 c_{x_2}}{-23 c_{x_2} c_{x_3} - 17 c_{x_3}^2 + 9.36 c_{x_3} + 1} \end{aligned} \tag{2.12a}$$

$$g_2 = \exp\left(2.496 - \frac{2068.4}{273 + x_6} - 2.07 c_{x_2}\right) \tag{2.12b}$$

$$\begin{aligned} g_3 = {} & 0.531 + 3.06 \cdot 10^{-18} u_1^3 - 2.51 \cdot 10^{-12} u_1^2 \\ & + 6.96 \cdot 10^{-7} u_1 - \frac{14.37(c_{x_2} - c_{x2,crit}) - 0.431}{735.3(c_{x_2} - c_{x2,crit}) + 1} \end{aligned} \tag{2.12c}$$

$$g_4 = \frac{0.5517 + 3.8168 \cdot 10^{-6} u_2}{1 + 8.271 \cdot 10^{-6} u_2} \tag{2.12d}$$

$$g_5 = \frac{3.8168 \cdot 10^{-6} g_3 g_4 u_2}{g_2(1 - g_3)}. \tag{2.12e}$$

See Table 2.1 for a description of these quantities.

Lundby et al. present a more in-depth derivation of the model and the parameter values [39]. The dynamics of the system are relatively slow. The control inputs

$u_1,\ u_3$ and u_4 are therefore well modelled as impulses representing discrete events involving the addition or removal of substances. This results in step changes in the linear states x_2, x_3, x_5, which act as accumulator states for the mass of the corresponding substance (see Table 2.1). The control inputs u_2, u_5 are piecewise constant and always nonzero. The inputs $\mathbf{u}$ are determined by a simple proportional controller $\boldsymbol{\pi}(\mathbf{x})$. Lundby et al. derived the simulation model from the mass/energy balance of the cell [39]. Fig. 2.1 shows a schematic of the setup.

Table 2.1: Table of states, inputs, and other quantities used to model the electrolysis cell

Variable	Physical meaning	Units
x_1	Mass side ledge	kg
x_2	Mass Al_2O_3	kg
x_3	Mass AlF_3	kg
x_4	Mass Na_3 AlF_6	kg
x_5	Mass metal	kg
x_6	Temperature bath	°C
x_7	Temperature side ledge	°C
x_8	Temperature wall	°C
u_1	Al_2O_3 feed	kg/s
u_2	Line current	kA
u_3	AlF_3 feed	kg/s
u_4	Aluminium tapping	kg/s
u_5	Anode-cathode distance	cm
c_{x_2}	Al_2O_3 mass ratio $x_2/(x_2 + x_3 + x_4)$	-
c_{x_3}	AlF_3 mass ratio $x_3/(x_2 + x_3 + x_4)$	-
g_1	Liquidus temperature	°C
g_2	Electrical conductivity	S m
g_3	Bubble coverage	-
g_4	Bubble thickness	cm
g_5	Bubble voltage	V

Example 2.1.4 (Lotka-Volterra system)**.** Also known as the predator-prey model. The Lotka-Volterra equations are often used to describe the interactions of a population of predators x and a population of prey y:

$$\begin{aligned} \dot{x} &= \alpha x - \beta xy, \\ \dot{y} &= \delta xy - \gamma y, \end{aligned} \tag{2.13}$$

where $\dot{y}$ and $\dot{x}$ represent the instantaneous growth rates of the two populations due to predation, overpopulation, and starvation. The solutions are periodic; as the prey population x grows, the population of predators y can eat more and reproduce. The

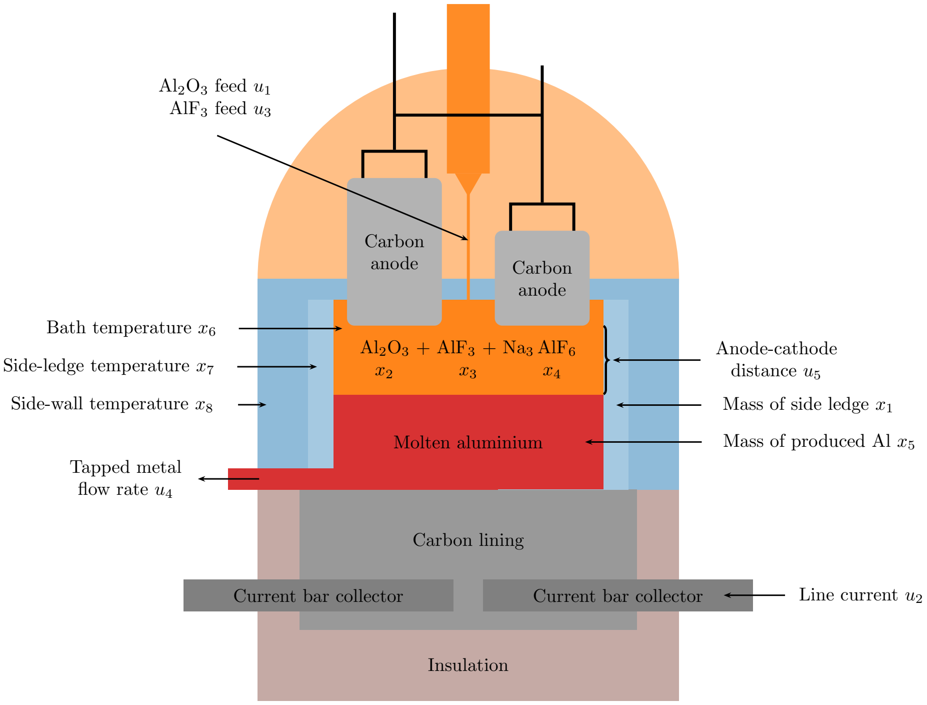


Figure 2.1: Schematic of the aluminium electrolysis cell for the Hall-Héroult process

prey population x then declines, causing y to drop as the predators starve. The two variables thus appear as similar waves, with y lagging behind x. The values $\alpha = 0.1, \beta = 0.05, \delta = 0.1, \gamma = 1.1$ were used.

The Lotka-Volterra equations, and predator-prey models in general, remain of theoretical and practical interest today. Such systems have been successfully used to model ecological communities [40], the spread of disease [41], and economic growth cycles [42, 43, 44].

Example 2.1.5 (Duffing oscillator)**.** The Duffing equation is a non-linear second-order differential equation that describes an oscillator with complex, sometimes chaotic behaviour. The Duffing equation was originally the result of Georg Duffing's systematic study of nonlinear oscillations [45]. Interest in the equation was later revived with the advent of chaos theory. Since then, the system has come to be regarded as one of the prototype systems in chaos theory [46], and related equations continue to find applications today, e.g. to describe the rolling of ships [47]. The equation is

$$\ddot{x} = \gamma\cos(\omega t) - \delta\dot{x} - \alpha x - \beta x^3 \tag{2.14}$$

where $x(t)$ is the displacement at time t and the term $\gamma\cos(\omega t)$ represents a sinusoidal driving force. The cubic term describes an asymmetry in a spring's restoring force that softens or stiffens as it stretches. The parameters used in this thesis are $\delta = 1, \alpha = 0.5, \beta = 1, \gamma = 3$ and $\omega = 0.4$. This time-varying system depends on t, making it challenging to model using DDM as-is, as the input t is unbounded. Instead, the system can be parameterised as follows:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= \gamma\psi - \delta y - \alpha x - \beta x^3 \\ \dot{\psi} &= -\omega\theta \\ \dot{\theta} &= \omega\psi \\ \psi(0) = 1,& \quad \theta(0) = 0 \end{aligned} \tag{2.15}$$

This formulation enables us to treat the system as if it were time-invariant. From a ML perspective, this is equivalent to feature engineering, as the features $\cos(\omega t)$ and $\sin(\omega t)$ are provided as additional inputs to the model.

Example 2.1.6 (Van der Pol oscillator)**.** The Van der Pol equations were discovered through the study of triode vibrations [48]. It describes a nonlinear oscillator that approaches a limit cycle over time. Systems like this are immensely useful in a variety of fields. For example, coupled Van der Pol systems have been used to model biological circadian rhythms [49] and to model the asymmetries in vocal folds[50]. Kuiate et al. [51] have even applied a variant of the system to encrypt images on-the-fly. The Van der Pol oscillator can be written as:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= \mu(1 - x^2)y + x. \end{aligned} \tag{2.16}$$

where $x(t)$ is the displacement, and $\mu = 3$ is a scalar that controls the effects of the nonlinear damping term. In this thesis, the value $\mu = 3$ is used. The system approaches a stable limit cycle for all initial conditions. As x approaches the maximum amplitude of the oscillation, $\dot{x}$ increases. When reaching the maximum, $\dot{x}$ rapidly switches sign, and x begins to decrease slowly, building up speed in the same way as it approaches the minimum. The system can exhibit chaos when forced with an additional sinusoidal term.

Example 2.1.7 (Lorenz system)**.** Lorenz initially developed this system to describe atmospheric convection [52]. However, it later became one of the most well-studied systems in chaos theory and is often credited with the explosion of interest in the subject [46]. The Lorenz equations have since been studied in connection with real physical phenomena, such as unstable spiking in lasers [53] and turbulence [54]. The system has the following form:

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{2.17}$$

For $\rho < 1$, the origin is globally stable. When $\rho > 1$, the system has three fixed points: $(0,0,0)$, and $(\pm\sqrt{(\beta(\rho-1)}, \pm\sqrt{(\beta(\rho-1)}, \rho-1)$, the latter of which are referred to as C_+ and C_-. For $\rho > \frac{\sigma(\sigma+\beta+3)}{\sigma-\beta-1}$ the solutions of the system become non-periodic and chaotic where almost all initial states will converge to an invariant fractal set called the Lorenz attractor [55]. This thesis uses the values originally used by Lorenz, namely $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$.

Example 2.1.8 (Henon–Heiles system)**.** Henon and Heiles originally developed these equations to study the movement of a star around a galactic centre while restricted to a plane [56]. The system is still used to study the escape dynamics of orbits [57]. The following Hamiltonian describes the dynamics:

$$H = \tfrac{1}{2}(\dot{x}^2 + \dot{y}^2) + \tfrac{1}{2}(x^2 + y^2) + x^2 y - \tfrac{1}{3}y^3 \tag{2.18}$$

This Hamiltonian can be reformulated as a set of ODEs:

$$\begin{aligned} \ddot{x} &= -x - 2\lambda xy \\ \ddot{y} &= -y - 2\lambda(x^2 - y^2) \end{aligned} \tag{2.19}$$

This thesis uses the value $\lambda = 1$. The solution set features many periodic orbits, chaotic orbits, and escape trajectories when the system's energy is sufficiently high [58]. The escape sets exhibit a rich fractal structure, adding complexity to the system's behaviour [59].

2.2 Neural networks

This section presents the general structure of a neural network and explains the notation used in this thesis. A more in-depth discussion can be found in the textbook by Goodfellow [60].

2.2.1 Layered neural networks

Fig. 2.2 shows the general structure of a simple NN $\mathcal{N} : \mathcal{X} \subset \mathbb{R}^{n_x} \mapsto \mathcal{Y} \subset \mathbb{R}^{n_y}$ with L layers, where information is processed layer-by-layer in a pipeline. A

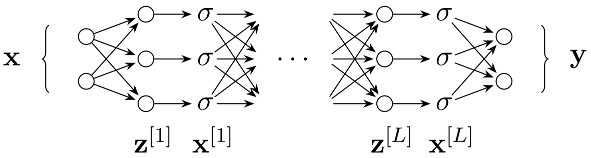


Figure 2.2: Neural network with L fully connected layers with $\boldsymbol{\sigma}$ as an activation function. The kth fully connected layer takes in the output of the previous layer $\mathbf{x}^{[k-1]}$ and produces the *pre-activation* $\mathbf{z}^{[k]} = \mathbf{W}^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}$, where $\mathbf{x}^{[0]} = \mathbf{x}$. This value is then passed through the nonlinear activation function $\boldsymbol{\sigma}$ which operates element-wise on $z^{[k]}$, yielding $\mathbf{x}^{[k]} = \boldsymbol{\sigma}(\mathbf{z}^{[k]})$.

network where information only flows in one direction is known as a *feedforward* network. The output of the network can be written as:

$$\begin{aligned} \mathcal{N}(\mathbf{x}) &= \left(\mathbf{f}^{[L]} \circ \cdots \circ \mathbf{f}^{[1]}\right)(\mathbf{x}) \\ \mathbf{f}^{[k]}(\mathbf{x}) &= \boldsymbol{\sigma}^{[k]}(\mathbf{z}^{[k]}) \end{aligned} \tag{2.20}$$

where $\boldsymbol{\sigma}^{[k]}$, $\mathbf{W}^{[k]}$ and $\mathbf{b}^{[k]}$ are the nonlinear activation function, connection weight matrix, and bias vector for the kth layer of $\mathcal{N}$ respectively. As is common in the literature [60], additional (purely convenient) notation is introduced to denote the values at each "stage" in the pipeline:

$$\begin{aligned} \mathbf{z}^{[k]} &= \mathbf{W}^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]} \\ \mathbf{x}^{[k]} &= \mathbf{f}^{[k]}(\mathbf{x}) = \boldsymbol{\sigma}^{[k]}(\mathbf{z}^{[k]}) \end{aligned} \tag{2.21}$$

The superscript $\cdot^{[k]}$ with square brackets is used to associate some vector or matrix value with the kth layer of a network. In Eq. (2.21) layer "0" represents the input, such that $\mathbf{x}^{[0]} = \mathbf{x}$. Note that the first k layers of the network also form a valid neural network. The *subnetwork* $\mathcal{N}_k$ is defined as:

$$\mathcal{N}_k(\mathbf{x}) = \left(\mathbf{f}^{[k]} \circ \cdots \circ \mathbf{f}^{[1]}\right)(\mathbf{x}) \tag{2.22}$$

From this, it can be seen that a simple feedforward NN can be understood as the application of alternating affine and nonlinear transformations of an initial input $\mathbf{x}$ and choosing a linear activation function $\boldsymbol{\sigma}$ would cause the network to simplify to a linear transformation.

The size of the matrix $\mathbf{W}^{[k]}$ can become intractable for large $x^{[k-1]}$, which is often the case when processing images. This issue can be mitigated by imposing some structure on the weights. For example, a *convolutional layer* assumes that adjacent input regions will be processed similarly. A memory-efficient convolution operation can then be used instead of multiplication with the large matrix $\mathbf{W}^{[k]}$. These networks are known as convolutional neural networkss (CNNs) and have proven to be instrumental image processing tasks [61].

2.2.2 Neural networks as computational graphs

Sometimes it is also necessary to talk about individual neurons in the network. Using the layer notation for feedforward networks, the connection weights of the ith neuron in the kth layer of $\mathcal{N}$ correspond to the ith row of $\mathbf{W}^{[k]}$, which is denoted as $\mathbf{w}_i^{[k]}$. Likewise, the bias of the same neuron is denoted $b_i^{[k]}$, and its output is $x_i^{[k]}$.

An issue with this notation is that organising a NN into layers can be restrictive. Modern network architectures introduce tricks that do not fit the notation, e.g. *skip connections* that "jump" over some layers [62], or models that combine the outputs of multiple separate subnetworks.

In these cases, it is simpler to represent the NN as a graph with weighted edges and designate a subset of the nodes as inputs or outputs. Every neuron is then given a unique index n, and we can instead refer to its parameters using the parenthesised superscript $\cdot^{(n)}$. Using this notation we refer to the weights $\mathbf{w}^{(n)}$, bias value $b^{(n)}$, pre-activation $z^{(n)}$, and output $x^{(n)}$. The initial values of the input nodes are then set to $\mathbf{x}$, and the following update equation is run until the value of all output nodes is determined:

$$x^{(n)} = \sigma\left(z^{(n)}\right) = \sigma\left(\sum_{j\in\mathcal{P}(n)}\left(w_j^{(n)}x^{(j)} + b^{(n)}\right)\right) \tag{2.23}$$

where $\mathcal{P}(n)$ is the set of neurons that are connected to neuron n, and $w_j^{(n)}$ is the weight of the edge from the jth neuron to the nth. In an attempt to reduce possible confusion between, e.g. $\mathbf{w}_i^{[k]}$ and $\mathbf{w}^{(n)}$, the indices k and n are used to refer to layers and neurons, respectively.

Most DL software frameworks, e.g. PyTorch [63] or Tensorflow [64], also take this approach and represent NNs as *computational graphs* where each node represents a differentiable function that operates on tensors of arbitrary order. For example, a video can be represented as a 4th-order tensor, where the dimensions correspond to the frame number, two indices for the pixel, and the colour channel. This representation yields compact and concise code to process large amounts of data and efficiently compile the model.

2.2.3 Activation functions

There are many possible choices for the activation function σ. The activation function σ is chosen to be any nonlinear function that maps $\mathbb{R}$ to some interval and is applied element-wise to the layer input $\mathbf{x}^{[k]}$. Historically σ has been selected as the sigmoid function $(1 + \exp(-x))^{-1}$, as this resembles the action potential exhibited in biological neurons. However, the sigmoid function is associated with the *vanishing gradient problem* in deep networks, making them challenging to train [65, 66]. A popular activation function that mitigates this issue is the Rectified Linear Unit rectified linear unit (ReLU), a PWA function:

$$\sigma(z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{2.24}$$

The surprising effectiveness and simplicity of ReLU have made it one of the most popular activation functions today. However, it can also result in *dead neurons*, which output zero for all inputs in the dataset. Therefore, the gradient of a dead neuron is always zero, and gradient descent algorithms will never adjust the corresponding parameters [60]. Similar activation functions such as Leaky ReLU [67] or Swish [68] address this by ensuring there is a slight slope for $z < 0$.

2.3 Reinforcement Learning

This section briefly presents some fundamental concepts in RL. The standard reference for the topic is the book by Sutton and Barto [69]. The aim of RL is to train an autonomous agent by feeding it rewards (represented as a scalar value) in response to reaching certain states $\mathbf{x}$ by taking some action $\mathbf{u}$. There are deep connections between RL and optimal control, although the aim of an RL agent differs in that it seeks to maximise an unbounded reward. In contrast, optimal control methods typically minimise a non-negative cost function.

The problem is traditionally separated into the *agent*, which applies actions $\mathbf{u}$ to the *environment*, which in turn returns a measurement $\mathbf{y}$ (correlated with the state $\mathbf{x}$) and an associated reward r. Fig. 2.3 illustrates this process.

2.3.1 Fundamentals of RL

At each discrete time step of the learning process, an agent chooses an action u based on its current state s within its environment. The agent does not usually know the true state of the environment. In this case, we would call s an observation. How the agent chose the specific action (i.e. the agent's strategy) is commonly referred

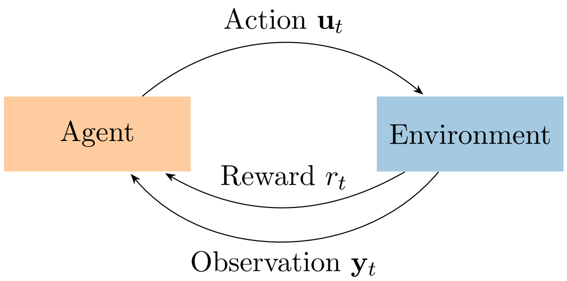


Figure 2.3: Structure of reinforcement learning.

to as the policy and denoted by π. Thus, the policy π can be considered a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from the state space to the action space. In order to learn, i.e. improve the policy π, the agent then receives a numerical reward r from the environment. The fundamental goal of the agent is to maximise its long-term reward (also known as the return), and updates to the agent's policy are intended to improve the agent's ability to do this. These concepts (i.e. agents, environments, observations/states, policies, actions and rewards) are fundamental to the study of RL.

Remark 2.3.1. The reward may not solely depend on the latest action made. An immediately attractive action may have downsides in the long term. Similarly, an unexciting action in the short term may be optimal in the long term. Delayed rewards are standard in RL environments.

Remark 2.3.2. The policy need not be deterministic. In rock-paper-scissors, the optimal policy is stochastic.

Remark 2.3.3. Classic RL algorithms deal with discrete action spaces. However, recent advances in the field have led to state-of-the-art algorithms that are naturally compatible with continuous action spaces without any undesirable discretisation [70].

As the environment may be stochastic, it is common to think of the process as a Markov decision process (MDP) with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $r(s_t, a_t)$, transition dynamics $p(s_{t+1}|s_t, a_t)$ and an initial state distribution $p(s_0)$ [71]. The combined MDP and agent formulation allows us to sample trajectories from the process by first sampling an initial state from $p(s_0)$ and then repeatedly sampling the agent's action a_t from its policy $\pi(s_t)$ and the next state s_{t+1} from $p(s_{t+1}|s_t, a_t)$. The agent receives a reward after each iteration, such that its total reward is:

$$R_t \triangleq \sum_{i=t}^{\infty} r(s_i, a_i) \tag{2.25}$$

Remark 2.3.4. It is common to introduce a discount factor $\gamma \in (0, 1]$ to encode a preference for short-term rewards and to ensure that the infinite sum of rewards will not diverge This factor is analogous to discount functions used in economics. The discounted sum of rewards is then given by $\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$. In the following derivations, the discount factor is assumed to be incorporated into a time-dependent reward function and is therefore neglected.

In stochastic environments, this is replaced with an expectation over many trials. When the MDP is too large to admit a closed-form solution or to be tabulated, the expected return is often approximated using the state-value function $V^\pi(s)$ and the action-value function $Q^\pi(s, a)$. $V^\pi(s)$ represents the expected return from time t onwards given an initial state s, while $Q^\pi(s, a)$ represents the expected return from time t onwards conditioned on the initial action a_t.

$$V^\pi(s_t) \triangleq \mathbb{E}_{s_{i>=t}, a_{i>=t} \sim \pi} \left[R_t | s_t \right] \tag{2.26}$$

$$Q^\pi(s_t, a_t) \triangleq \mathbb{E}_{s_{i>=t}, a_{i>=t} \sim \pi} \left[R_t | s_t, a_t \right] \tag{2.27}$$

where the expectations are taken over all states s_i after time t, and the corresponding actions a_i sampled according to the policy π, such that $a_i \sim \pi$.

2.3.2 Policy gradients

Value-based methods attempt to estimate the state-value function and then infer the optimal policy. Policy-based methods instead try to optimise the policy directly. For high-dimensional or continuous action spaces, policy-based methods are considered the more efficient approach [72].

From now on, we consider the policy $\pi(\theta)$ to be stochastic (i.e. $\pi(\theta) : \mathcal{S} \times \mathcal{A} \to [0, 1]$) and assume that it is defined by some differentiable function parametrised by θ, enabling us to optimise it through policy-gradient methods. In general, these methods are concerned with using gradient ascent approximations to gradually adjust the policy function parameterisation vector in order to optimise the performance objective:

$$J(\theta) \triangleq \mathbb{E}_{s_i, a_i \sim \pi(\theta)} \left[R_0 \right] \tag{2.28}$$

More formally, policy-gradient methods approach gradient ascent by updating the parameter vector θ according to the approximation $\theta_{t+1} \leftarrow \alpha\theta_t + \widehat{\nabla_\theta J(\theta)}$, where $\widehat{\nabla_{\theta} J(\theta)}$ is a stochastic estimate of $\nabla_{\theta} J(\theta)$ satisfying $\mathbb{E}\left[\widehat{\nabla_{\theta} J(\theta)}\right] = \nabla_{\theta} J(\theta)$. Intuitively, estimating the policy gradient might be considered intractable, as the state transition dynamics, which affect the expected reward and hence our performance objective, are influenced by the agent's policy in an unknown fashion. However, the policy gradient theorem [73] establishes that the policy gradient $\nabla_{\theta} J(\theta)$ satisfies:

$$\begin{gathered}\nabla_{\theta} J(\theta) \propto \sum_{s} \mu(s) \sum_{a} \nabla_{\theta} \pi(a|s) Q^{\pi}(s,a) \\ \mu(s) = \lim_{t \to \infty} Pr\{S_t = s | A_{0:t-1} \sim \pi\}\end{gathered} \tag{2.29}$$

where μ is the steady state distribution under π and S_t and $A_{0:t-1}$ are random variables representing the state at time-step t, and the actions up to that point, respectively. Interestingly, the expression for the policy gradient does not contain the derivative $\nabla_{\theta} \mu(s)$, implying that approximating the gradient by sampling is feasible because calculating the effect of updating the policy on the steady-state distribution is not needed. By replacing the probability-weighted sum over all possible states in Eq. (2.29) by an expectation of the random variable S_t under the current policy, we have that

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi}\left[\sum_{a} \nabla_{\theta} \pi(a|S_t) Q^{\pi}(S_t, a)\right] \tag{2.30}$$

Similarly, we can replace the sum over all possible actions with an expectation of the random variable A_t after multiplying and dividing by the policy $\pi(a|S_t)$:

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi}\left[\sum_{a} \frac{\pi(a|S_t)}{\pi(a|S_t)} \nabla_{\theta} \pi(a|S_t) Q^{\pi}(S_t, a)\right]$$

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi}\left[\frac{\nabla_{\theta} \pi(A_t|S_t)}{\pi(A_t|S_t)} Q^{\pi}(S_t, A_t)\right] \tag{2.31}$$

Furthermore, it follows from the identity $\nabla \ln x = \frac{\nabla x}{x}$ that:

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi}[\nabla_{\theta} \ln \pi(A_t|S_t) Q^{\pi}(S_t, A_t)] \tag{2.32}$$

Also, by considering that:

$$\begin{aligned}\sum_a b(s)\nabla\pi(a|s) &= b(s)\nabla\sum_a \pi(a|s) \\ &= b(s)\nabla\mathbf{1} = 0\end{aligned} \tag{2.33}$$

It is straightforward to see that one can replace the state-action value function $Q^\pi(s,a)$ in Eq. (2.29) by $Q^\pi(s,a) - b(s)$, where the baseline function $b(s)$ can be an arbitrary function not depending on the action a, without introducing a bias in the estimate. However, it can be shown that the estimator's variance can be significantly reduced by introducing such a baseline. It is possible to calculate the optimal (i.e. variance-minimizing) baseline [74], but commonly the state value function V^π is used, yielding an almost optimal variance [75]. The result is known as the advantage function:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.34}$$

Intuitively, Eq. (2.34) represents the expected improvement obtained by an action compared to the default behaviour. Furthermore, by following the same steps as outlined above, we end up with the expression:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi[\nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t) A^\pi(s,a)] \tag{2.35}$$

Thus, an unbiased empirical estimate based on N episodic trajectories (i.e. independent rollouts of the policy in the environment) of the policy gradient is:

$$\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty} \hat{A}_t^n \nabla_{\boldsymbol{\theta}} \log \pi(a_t^n|s_t^n) \tag{2.36}$$

2.3.3 Advantage function estimation

As both $Q^\pi(s,a)$ and $V^\pi(s)$ are unknown in general, it follows that $A^\pi(s,a)$ is also unknown. Thus, it is commonly replaced by an advantage estimator $\hat{A}^\pi(s,a)$. Various estimation methods have been developed for this purpose. Generalised Advantage Estimation (GAE), as originally outlined by Schulman et al. [75], is one of the most widely used methods. GAE uses discounted temporal difference (TD) residuals of the state value function as the fundamental building blocks. For this, we reintroduce the discount parameter γ. However, even if γ corresponds to the discount factor discussed in the context of MDPs, we now treat it as a variance-reducing parameter in an undiscounted MDP. TD residuals [69], which are in widespread use within RL, and give a basic estimate of the advantage function, are defined by:

$$\delta_t^V = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t) \tag{2.37}$$

where $\hat{V}$ is an approximate value function. The estimate is unbiased whenever $\hat{V} = V^\pi$, i.e. when the approximation equals the real value function. However, this is unlikely to be the case in practice, so a common approach is to look further ahead than just one step to reduce bias. More formally, by defining $\hat{A}_t^{(k)}$ as the discounted sum of the k next TD residuals, we have that

$$\begin{aligned}
\hat{A}_t^{(1)} &= \delta_t^{\hat{V}} = -\hat{V}(s_t) + r_t + \gamma\hat{V}(s_{t+1}) \\
\hat{A}_t^{(2)} &= \delta_t^{\hat{V}} + \gamma\delta_{t+1}^{\hat{V}} = -\hat{V}(s_t) + r_t + \gamma r_{t+1} + \gamma^2\hat{V}(s_{t+2}) \\
&\vdots \\
\hat{A}_t^{(k)} &= \sum_{l=0}^{k-1}\gamma^l\delta_{t+l}^{\hat{V}}
\end{aligned} \tag{2.38}$$

The defining feature of GAE is that, instead of choosing some k-step estimator $\hat{A}_t^{(k)}$, we use an exponentially weighted average of the k first estimators, letting $k \to \infty$. Thus, we have that:

$$\hat{A}_t^{GAE(\gamma,\lambda)} \triangleq (1-\lambda)\left(\hat{A}_t^1 + \lambda\hat{A}_t^2 + \lambda^2\hat{A}_t^3 + \dots\right) \tag{2.39}$$

which can be shown by insertion of the definition of $\hat{A}_t^{(k)}$ to equal:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^{\hat{V}} \tag{2.40}$$

Here, $\lambda \in [0,1]$ serves as a trade-off parameter controlling the compromise between bias and variance in the advantage estimate; using a small value lowers the variance as the immediate TD residuals make up most of the estimate, whereas using a large value lowers the bias induced from inaccuracies in the value function approximation.

Due to the recent advances made within DL, a common approach is to use a Deep neural network (DNN) for estimating the value function, which is trained on discounted empirical returns. More specifically, the DNN state value estimator $\hat{V}_\theta(s_t)$, which is parametrised by θ_{VF}, is trained by minimizing the loss function:

$$L_t^{VF}(\theta) = \hat{\mathbb{E}}_t\left[\hat{V}_\theta(s_t) - \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)\right] \tag{2.41}$$

where the expectation $\hat{\mathbb{E}}_t[\dots]$ represents the empirical average obtained from a finite batch of samples. The reader is referred to the textbooks by Goodfellow et al. for a comprehensive introduction to DL [60], and Bishop et al. [76], which covers supervised machine learning, of which DL is a subfield.

2.3.4 A surrogate objective

Optimising the performance objective using the empirical policy gradient approximation from Eq. (2.36) is feasible. This approach yields the vanilla policy gradient algorithm proposed by Williams et al. [77]. However, this inefficient approach requires many samples to accurately estimate the policy gradient direction [78]. Accordingly, unless the step size is trivially small (yielding unacceptably slow convergence), it is not guaranteed that the policy update will improve the performance objective, which leads to the algorithm having poor stability and robustness characteristics [79].

Instead, state-of-the-art policy gradient methods such as Trust Region Policy Optimisation (TRPO) [80] and Proximal Policy Optimisation [81] optimise a surrogate objective function which provides theoretical guarantees for policy improvement even under nontrivial step sizes. Fundamentally, these methods rely on the relative policy performance identity proven by Kakade et al. [78], which states that the improvement in the performance objective $J(\theta)$ achieved by a policy update $\theta \rightarrow \theta'$ is equal to the expected advantage (see Eq. (2.34)) of the actions sampled from the new policy $\pi'_{\theta'}$ calculated w.r.t. the old policy π_θ. More formally, this translates to:

$$J(\theta') - J(\theta) = \mathbb{E}_{\pi'_\theta}\left[\sum_t \gamma^t A^{\pi_\theta}(s_t, a_t)\right] \tag{2.42}$$

This expectation is defined under the next (i.e. unknown) policy $\pi_{\theta'}$, which we cannot use to sample trajectories. However, Eq. (2.42) can be rewritten and finally approximated by:

$$\begin{aligned}
&J(\theta') - J(\theta) \\
&= \sum_t \mathbb{E}_{s_t \sim \pi_{\theta'}} \left[\mathbb{E}_{a_t \sim \pi_{\theta'}} \left[\gamma^t A^{\pi_\theta}(s_t, a_t) \right] \right] \\
&= \sum_t \mathbb{E}_{s_t \sim \pi_{\theta'}} \left[\mathbb{E}_{a_t \sim \pi_{\theta}} \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \right] \\
&\approx \sum_t \mathbb{E}_{s_t \sim \pi_{\theta}} \left[\mathbb{E}_{a_t \sim \pi_{\theta}} \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \right]
\end{aligned} \tag{2.43}$$

The third and last steps can be seen as importance sampling and neglecting state distribution mismatch respectively. Loosely stated, the last approximation assumes that the change in the state distribution induced by a small update to the policy parameters is negligible. This assumption is justified by theoretical guarantees imposing an upper bound to the distribution chance provided by Kakade et al. [78]. This suggests that one can reliably optimise the conservative policy iteration surrogate objective [78]:

$$J^{CPI}(\theta') = \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \hat{A}_t^{\pi_\theta} \right] \tag{2.44}$$

However, this approximation is only valid in a local neighbourhood, requiring a carefully chosen step size to avoid instability. In TRPO, this is achieved by maximising $J^{CPI}(\theta')$ under a hard constraint on the Kullback-Liebler divergence between the old and the new policy. However, as this is computationally expensive, the proximal policy optimisation (PPO) algorithm refines this by integrating the constraint into the objective function by redefining the objective function to:

$$\begin{aligned}
&J^{CLIP}(\theta') = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t^{\pi_\theta}, \text{clip}_\epsilon \left(r_t(\theta) \right) \hat{A}_t^{\pi_\theta} \right) \right] \\
&\text{clip}_\epsilon(x) = \text{clip}\left(x, 1-\epsilon, 1+\epsilon\right)
\end{aligned} \tag{2.45}$$

where $r_t(\theta)$ is a shorthand notation for the probability ratio $\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$. The truncation of the probability ratio is motivated by a need to restrict $r_t(\theta)$ from moving outside of the interval $[1-\epsilon, 1+\epsilon]$. Also, the expectation is taken over the minimum of the clipped and unclipped objective, implying that the overall objective function is a lower bound of the original objective function $J^{CPI}(\theta')$. At each training iteration, the advantage estimates are computed over batches of trajectories collected from N_A concurrent actors, each executing the current policy π_θ for T timesteps. Afterwards, a stochastic gradient descent (SGD) update using the *Adam* optimiser [82] of minibatch size N_{MB} is performed for N_E epochs.

The PPO algorithm strikes a balance between ease of implementation and data efficiency, and is likely to perform well in a wide range of continuous environ-

Algorithm 2.1 Proximal Policy Optimisation

```
for iteration = $1, 2, \ldots$ do
    for actor = $1, 2, \ldots N$ do
        For $T$ time-steps, execute policy $\pi_\theta$.
        Compute advantage estimates $\hat{A}_1, \ldots \hat{A}_T$
    end for
    for epoch = $1, 2, \ldots N_E$ do
        Obtain mini batch of $N_{MB}$ samples from the $N_A T$ simulated time-steps.
        Perform SGD update from minibatch $(\mathbf{X}_{MB}, \mathbf{Y}_{MB})$.
        $\theta \leftarrow \theta'$
    end for
end for
```

ments without extensive hyperparameter tuning [81]. Sensitivity to hyperparameter choices is a frequently encountered problem for policy gradient methods [83, 84], and given the computation time required to train and test agents in a collision avoidance environment, this could be a detrimental bottleneck in our research.

2.3.5 Tools and libraries

The Python library **OpenAI Gym** [85] was created to standardise the benchmarks used in RL research. It provides an easy-to-use framework for creating RL environments in which custom RL agents can be deployed and trained with minimal overhead.

`stable-baselines3` [86], another Python package, provides a large set of state-of-the-art parallelizable RL algorithms compatible with the OpenAI gym framework, including PPO. The algorithms are based on the original versions found in OpenAI Baselines [87], but `stable-baselines` provides several improvements, including algorithm standardisation and exhaustive documentation.

Chapter 3

Machine learning for control

Machine learning for control (also known as intelligent control) is useful when we lack a model for the target system or if the existing models are unsuitable for control design purposes.

In marine systems, autonomy offers surface vehicles the opportunity to improve transportation efficiency while reducing greenhouse emissions. However, for safe and reliable autonomous surface vehicles (ASVs), effective path planning is a prerequisite which should cater to the two critical tasks of path following and collision avoidance (COLAV).

Learning-based control approaches are gaining popularity today, particularly for highly complex, nonlinear, and stochastic systems where standard control design methods do not always perform satisfactorily [88, 89, 90]. Learning-based methods can be used to discover new actions and patterns and adapt to the environment to yield better performance. RL is an area of ML of particular interest for control applications, such as the guidance of surface vessels. Fundamentally, RL is concerned with estimating the optimal behaviour for an agent in an unknown, potentially partly unobservable environment, relying on trial-and-error-like approaches to iteratively approximate the behaviour policy that maximises the agent's expected long-time reward in the environment. The field of RL has seen rapid development over the last few years, leading to many impressive achievements, such as playing chess and various other games at a level that is not only exceedingly superhuman but also overshadows previous artificial intelligence (AI) approaches by a wide margin [91, 92, 93].

Despite the vast amount of literature on the topic and the numerous different approaches, of which only a small subset has been mentioned here, it appears that, when applied to vehicles with nonholonomic and real-time constraints such as autonomous surface vehicles, no existing method is without drawbacks, whether it is unrealistic assumptions about the vessel dynamics (if not an outright neglect thereof), problems with scalability in terms of environment complexity (including the degrees of freedom, the number of obstacles as well as their shapes and their velocities), excessive computation time requirements in general, unrealistic assumptions of availability of measurements, the disregard for desirable output path properties such as continuity, smoothness, feasibility or even safety, an incompatibility with external environmental forces, a lack of determinism (which may or may not be deemed problematic), stability issues due to singularities or local minima leading to sub-optimal guidance strategies [94, 95, 96].

The remainder of this chapter briefly overviews issues within marine vessels' autonomy. Section 3.2 provides a short theoretical overview of the modelling aspects of ships, and Section 3.1 reviews the literature on path planning and collision avoidance. Section 3.3 shows how RL can be used to train an agent to perform collision avoidance using an interactive simulation. A challenge with systems like this is that it is difficult to guarantee performance. Our RL ship performs well and can tune its behaviour online. However, there are still situations during the training and test phase where it fails to avoid collisions. Safe RL is a field of research that attempts to address this. One approach is to add an auxiliary failsafe that monitors whether the agent is about to violate a constraint and "corrects" the input to the system in a way guaranteed to be safe (although it might be conservative). Section 3.4 presents a predictive safety filter for an autonomous ship. In this chapter, we will explore an application of RL to the control of an autonomous ship. Section 3.3 then shows how this model is used to create a simulation environment with simulated sensors and how a RL agent can be trained by interacting with such a model. Additionally, the agent is given insight into the reward function during training, allowing us to tune its behaviour during operation. The results show that the agent performs well on average but suffers occasional collisions. As an additional failsafe, in Section 3.4, we design a predictive safety filter that guarantees safety during operation.

3.1 Literature Review: Maritime Collision Avoidance

A rich set of studies on automatic maritime collision avoidance exist today, and recent review articles can be found that summarise the majority of state-of-the-art methods [97, 98]. A common way of structuring the COLAV system is using a hierarchy of planners [99, 94, 100, 101]. Eriksen et al. [101] divide the COLAV problem into three levels separated by their timescale: (a) High-level planning, (b) Mid-level planning, (c) Low-level planning. High-level planners are also known as *deliberate* methods, and mid/low-level planners are sometimes called *reactive* methods [102]. A high-level planner generates a trajectory or path to the final destination, considering static obstacle data from, e.g. electronic navigation charts. This problem corresponds to finding a series of waypoints η_i to follow in marine navigation. Because of the larger timescales (minutes/hours/days), the dynamics of the ship are often simplified, reducing this to a path-finding problem. Grid-based or lattice-based methods typically discretise the map geometry [103] or partition it into, e.g. Voronoi cells [104, 105, 106]. These can then be treated as graphs, and algorithms such as A* search [107] can be used to find shortest/cheapest paths [108]. Randomised sampling methods that explore the space using random steps have also been successfully applied [109, 110].

Mid-level COLAV planning algorithms try to avoid static and dynamic obstacles near the ownship. Dynamic obstacles are typically detected and tracked online using the onboard exteroceptive sensors and Automatic Identification System (AIS) data. At this level and timescale (seconds/minutes), the planner should consider the vessel's dynamics to suggest feasible manoeuvres for the ship to execute. Because there may be multiple obstacles, the suggested manoeuvre should guarantee safety at all time steps considered in its horizon, which is usually formulated as an optimal control problem [111, 112, 113, 114, 115, 116]. The computational requirements of such an approach can quickly grow with the complexity of the problem representation. These methods often have to make assumptions or simplifications to keep the problem tractable. Another approach is to approximate the problem and attempt to find suboptimal solutions that are "good enough". Scenario-based MPC relies on sampling to efficiently generate solution candidates and selects the best-performing candidate. Tengesdal et al. [117] develop a method that considers multiple possible own-ship and target-ship manoeuvres at a series of decision points in time.

At the lowest level (seconds or milliseconds), the planner must be able to react quickly to unexpected situations. Low-level planners are also referred to as *reactive* COLAV due to the short time scales involved. Such situations can arise when a nearby vessel makes a sudden and dangerous manoeuvre or loses control of the vessel. Higher-level planners can also fall back to reactive COLAV if there is high uncertainty regarding the positions of nearby ships due to sensor malfunctions. Classic examples include Potential Field (PF) methods [118, 119, 120, 121], dynamic window methods [122, 123, 124] and Velocity Obstacle methods [125, 126, 127].

No method is without drawbacks, and many existing works make unrealistic assumptions about the environment, the ship dynamics, availability of measurements or suffer from numerical issues [94, 95, 96]. Lower-level planners rely only on data collected from the immediate environment and are susceptible to local minima that can lead a vehicle to a dead end [94]. Higher-level planners are generally more likely to suggest a course that leads a ship to the intended goal, but they require more computation than lower-level methods. Unforeseen obstacles or incomplete/uncertain mapping data can necessitate frequent replanning, However, the problem of optimal path planning amid multiple obstacles is provably NP-hard [128], making it difficult to perform real-time replanning when computational resources are limited [129].

A natural way to mitigate these weaknesses is to combine multiple methods in a hierarchy. For example, a typical configuration is to switch a low-level planner (e.g. the potential field method) whenever the high-level planner fails (e.g. the surroundings do not match the charts). Such *hybrid* architectures are intended to combine the strengths of reactive and deliberate approaches and have gained traction in recent years [130, 101]. The approach presented in Section 3.4 is somewhat related to this; the existence of some a priori known nominal path is assumed, but strictly following the path will invariantly lead to collisions with obstacles. Unlike other approaches, there is, however, no switching mechanism that activates some reactive fallback algorithm in dangerous situations.

3.2 Modelling and control of ships

The models used in this thesis are based on the Robot-Inspired Model for marine craft presented by Fossen [37]. Readers interested in a more detailed and rigorous explanation of the principles behind marine craft modelling are referred to this reference.

3.2.1 Reference frames

We will primarily use a geographic reference frame to describe the position of ships, which can be seen as a tangent plane to the Earth at some reference point. The advantage of this is that the equations of motion are simpler to express in a Cartesian coordinate system. The North-East-Down (NED) reference frame is denoted $\{n\} = (x_n, y_n, z_n)$, where the axes point North, East and Down (i.e. towards the centre of Earth) respectively. The velocity of the ship is expressed w.r.t. the BODY frame $\{b\} = (x_b, y_b, z_b)$, which has its origin at some point o_b along the centre line of the ship, and the axes point towards the front of the ship, starboard, and downwards respectively.

3.2.2 State variables

The state of a marine vessel is represented by 12 generalised coordinates (6 for pose, 6 for velocities), as per Society of Naval Architects and Marine Engineers (SNAME) conventions [131]. The pose and generalised velocity of the ship are denoted as $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$, respectively, and have the following elements:

$$\begin{aligned}\boldsymbol{\eta} &\triangleq [x^n \quad y^n \quad z^n \quad \phi \quad \theta \quad \psi]^\top \\ \boldsymbol{\nu} &\triangleq [u \quad v \quad w \quad p \quad q \quad r]^\top\end{aligned} \tag{3.1}$$

where the $[x^n \quad y^n \quad z^n]$ are the position w.r.t. $\{n\}$, and the Euler angles $[\phi \quad \theta \quad \psi]$ are the *roll*, *pitch*, and *yaw* of the ship. The linear velocities $[u \quad v \quad w]$ are called *surge*, *sway*, and *heave* respectively, while the angular velocities $[p \quad q \quad r]$ are called the *roll-rate*, *pitch-rate*, and *yaw-rate*.

3.2.3 Dynamics

Assumption 3.2.1 (Calm sea)**.** There is no ocean current, no wind and no waves and thus no external disturbances to the vessel.

Twelve coupled, first-order, nonlinear ordinary differential equations describe the vessel dynamics. In the absence of ocean currents, waves and wind, these can be expressed in a compact matrix-vector form as:

$$\begin{aligned}\dot{\boldsymbol{\eta}} &= \mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{B}\mathbf{f} &= \mathbf{M}_{\mathbf{RB}}\dot{\boldsymbol{\nu}} + \mathbf{C}_{\mathbf{RB}}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) \quad \text{(rigid-body, hydrostatic)} \\ &+ \mathbf{M}_{\mathbf{A}}\dot{\boldsymbol{\nu}} + \mathbf{C}_{\mathbf{A}}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} \quad \text{(hydrodynamic)}\end{aligned} \tag{3.2}$$

Here, $\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta})$ is the transformation matrix from the body frame $\{b\}$ to the NED reference frame $\{n\}$. $\mathbf{M}_{\mathbf{RB}}$ and $\mathbf{M}_{\mathbf{A}}$ are the mass matrices representing rigid-body mass and added mass, respectively. Analogously, $\mathbf{C}_{\mathbf{RB}}(\boldsymbol{\nu})$ and $\mathbf{C}_{\mathbf{A}}(\boldsymbol{\nu})$ are matrices incorporating centripetal and Coriolis effects. Finally, $\mathbf{D}(\boldsymbol{\nu})$ is the damping matrix, $\mathbf{g}(\boldsymbol{\eta})$ contains the restoring forces and moments resulting from gravity and buoyancy, $\mathbf{B}$ is the actuator configuration matrix and $\mathbf{f}$ is the vector of control inputs.

3.2.4 3-DOF manoeuvring model

This section outlines the ASV assumptions and the resulting 3-degrees of freedom (DOF) model.

Assumption 3.2.2 (State space restriction)**.** The vessel is always located on the surface, and thus there is no heave motion. Also, there is no pitching or rolling motion.

This assumption implies that the state variables z^n, ϕ, θ, w, p, q are all zero. Thus, we are left with the three generalised coordinates x^n, y^n and ψ and the body-frame velocities u, v and r. In this case, the transformation matrix $\mathbf{J}_{\mathbf{\Theta}}(\boldsymbol{\eta})$ is reduced to a basic rotation matrix $\mathbf{R}_{z,\psi}$ for a rotation of ψ around the z_n-axis as defined by

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Furthermore, since restoring forces are unimportant for 3- DOF manoeuvring [132], we set $\mathbf{g}(\boldsymbol{\eta}) = \mathbf{0}$. Also, by combining the corresponding rigid body and added mass terms associated such that $\mathbf{M} = \mathbf{M}_{\mathbf{RB}} + \mathbf{M}_{\mathbf{B}}$ and $\mathbf{C}(\boldsymbol{\nu}) = \mathbf{C}_{\mathbf{RB}}(\boldsymbol{\nu}) + \mathbf{C}_{\mathbf{A}}(\boldsymbol{\nu})$, we obtain the simpler 3- DOF state-space model:

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{R}_{z,\psi}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} &= \mathbf{B}\mathbf{f} \end{aligned} \tag{3.3}$$

where $\boldsymbol{\eta} \triangleq [x^n, y^n, \psi]^\top$ and $\boldsymbol{\nu} \triangleq [u, v, r]^\top$ and each matrix is 3x3.

Assumption 3.2.3 (Vessel symmetry)**.** The vessel is port-starboard symmetric.

Assumption 3.2.4 (Origin at the centerline)**.** The body-fixed reference frame $\{b\}$ is centred somewhere at the longitudinal centerline passing through the vessel's centre of gravity.

Assumption 3.2.5 (Sway-underactuation)**.** There is no force input in sway, so the only control inputs are the surge thrust T_u and the yaw moment T_r.

Assumptions 3.2.3 and 3.2.4, commonly found in manoeuvring theory applications, justify a sparser structure of the system matrices, where some non-diagonal elements are set to zero. Also, from Assumption 3.2.5 we have that $\mathbf{f} \triangleq [T_u, T_r]^\top$. The matrices and their numerical values are obtained from [132], where the model parameters were estimated experimentally for CyberShip II, a 1:70 scale replica of a supply ship, in a marine control laboratory.

3.3 Deep reinforcement learning for path following and collision avoidance

This section explores how recent advances in RL can be applied to the guidance and control of ASVs. Specifically, the goal is to train a RL agent to follow a predefined path while avoiding any obstacles that it may encounter. This goal is accomplished by training the agent in a simulated environment of randomly generated waypoints and obstacles. Two types of measurements are made available to the agent: (i) A set of virtual distance sensors for detecting nearby obstacles and (ii) Some values related to the relative position and orientation of the path. Multiple RL methods are tested using the implementations from the `stable-baselines` project [86].

For simplicity, the scope is limited to static, circular obstacles, although the work can easily be extended beyond these simplifications. The resulting interplay between the environment (including the dynamics of the vessel itself) and the agent is illustrated in Fig. 3.1. Although the agent is trained in simulation, thereby requiring a model of the vessel in question, RL methods may also be applied to real-world systems directly. Paper E and Section 3.4 explore how this might be done safely.

As these methods develop, direct application of RL will become more and more feasible, enabling intelligent and self-improving controllers for autonomous systems. Refer to Section 2.3 for an overview of basic RL concepts.

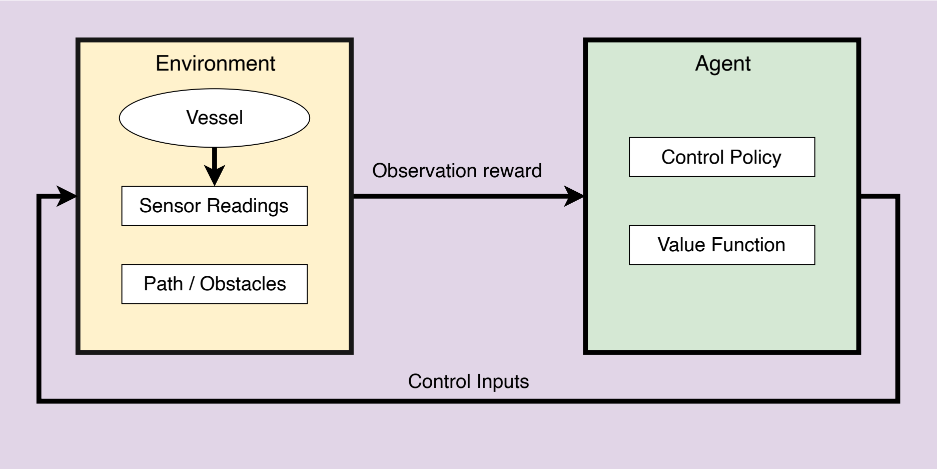


Figure 3.1: Block diagram illustrating the interaction between the environment and the RL agent.

3.3.1 Environment setup

The environment is an ocean surface filled with obstacles and a known path for the agent to follow. The vessel dynamics (see Section 3.2.3) are considered part of the environment, as it is outside the agent's control. It is also critical that the training environments are diverse so that the trained agent can generalise to unseen obstacle landscapes, i.e. real-world deployments. A lack of diversity can lead to the agents overfitting the training conditions. For instance, if all obstacles are located very close to the path during training, the agent may learn to turn whenever it sees an obstacle, even when it is not in the ship's way. Also, if the obstacle density is too low, it is unlikely that the agent would perform well in a high-obstacle-density environment.

The procedure outlined in Algorithm 3.1 randomly generates new, independent training environments that meet the diversity requirements. Some randomly sampled environments generated from this algorithm can be seen in Fig. 3.2. Achieving good performance within these environments (i.e. adhering to the planned path while avoiding collisions) necessitates a nontrivial guidance algorithm.

Algorithm 3.1 Generate path with obstacles

```
Require:
  Number of obstacles $N_o \in \mathbb{N}_0$
  Number of path waypoints $N_w \in \mathbb{N}_0$
  Path length $L_p \in \mathbb{N}_0$
  Mean obstacle radius $\mu_r \in \mathbb{R}^+$
  Obstacle displacement distance standard deviation $\sigma_d \in \mathbb{R}^+$
  procedure GENERATEPATHCOLAVENVIRONMENT($N_o$, $N_w$, $L_p$, $\mu_r$, $\sigma_d$)
      Draw $\theta_{start}$ from Uniform$(0, 2\pi)$
      Path origin $\mathbf{p}_{start} \leftarrow 0.5 L_p \left[\cos(\theta_{start}), \sin(\theta_{start})\right]^\top$
      Goal position $\mathbf{p}_{end} \leftarrow -\mathbf{p}_{start}$
      Generate $N_w$ random waypoints between $\mathbf{p}_{start}$ and $\mathbf{p}_{end}$.
      Create smooth arc length parameterised path $\mathbf{p}_p(\bar{\omega}) = [x_p(\bar{\omega}), y_p(\bar{\omega})]^\top$ using 1D Piecewise Cubic Hermite Interpolator (PCHIP) provided by Python library SciPy [133].
      repeat
          Draw arclength $\bar{\omega}_{obst}$ from Uniform$(0.1L_p, 0.9L_p)$.
          Draw obstacle displacement distance $d_{obst}$ from $\mathcal{N}(0, \sigma_d^2)$
          Path angle $\gamma_{obst} \leftarrow \mathrm{atan2}\left(\mathbf{p}_p{}'(\bar{\omega}_{obst})_2, \mathbf{p}_p{}'(\bar{\omega}_{obst})_1\right)$
          Obstacle position $\mathbf{p}_{obst} \leftarrow \mathbf{p}_p(\bar{\omega}_{obst}) + d_{obst}[\cos(\gamma_{obst} - \frac{\pi}{2}), \sin(\gamma_{obst} - \frac{\pi}{2})]^\top$
          Draw obstacle radius $r_{obst}$ from Poisson$(\mu_r)$.
          Add obstacle $(\mathbf{p}_{obst}, r_{obst})$ to environment
      until $N_0$ obstacles are created
  end procedure
```

In the current work the values of $N_o = 20$, $N_w = \mathcal{U}(2, 5)$, $L_p = 400$, $\mu_r = 30$, $\sigma_d = 150$ (where $\mathcal{U}$ is the uniform distribition) were used.

Agent

Although the *agent*, within the context of RL, can be considered to be the vessel itself, it is more accurate to look at it as the guidance mechanism controlling the vessel, as its operation is limited to outputting the control signals that steer the

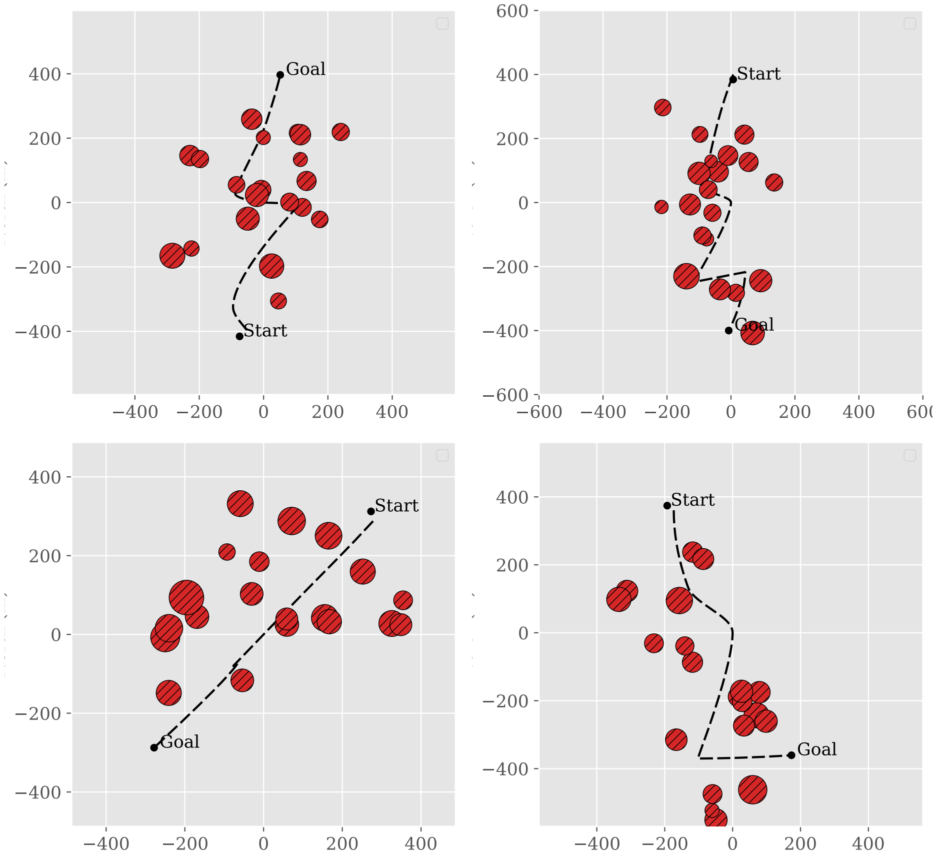


Figure 3.2: Four random samples of the stochastically generated path following scenario. Note that the scenario difficulty is highly varying.

vessel's actuators. As discussed in Section 3.2.4, the available control signals are the surge thrust T_u, driving the vessel forward, and the yaw moment T_r, inducing a change in the vessel's heading. The RL agent's action, which it will output at each simulated time-step, is then defined as the vector $a = [T_u, T_r]^\top$. Specifically, the action network, which we train by applying the PPO algorithm described in Section 2.3.4, will output the control signals following a forward pass of the current observation vector through the nodes of the neural network. Also, the value network is trained simultaneously, facilitating estimation of the state value function $V(s)$, which is used for GAE as described in Section 2.3.3.

Deciding what constitutes a state s is of utmost importance; the information provided to the agent must be of sufficient fidelity for it to make rational guidance decisions, especially as the agent will be purely reactive, i.e. not be able to let previous observations influence the current action. At the same time, by including too many features in the state definition, we risk over-parameterisation within the NNs, which can lead to poor performance and excessive training time requirements [60]. Thus, a compromise must be reached, ensuring a sufficiently low-dimensional observation vector while providing a sufficiently rich observation of the current environment. Having separate observation features representing path-following performance and obstacle closeness is a natural choice.

Path following

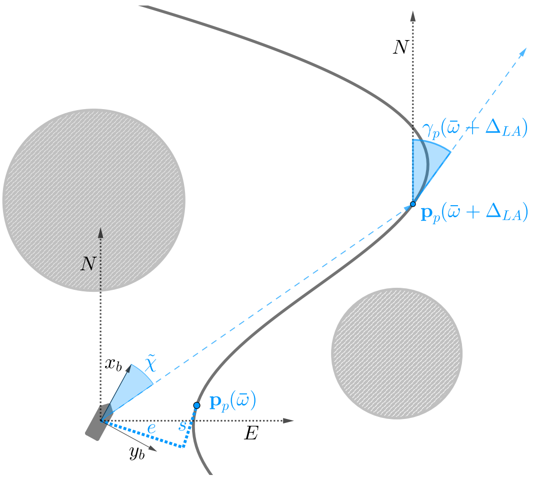

N
$\gamma_p(\bar{\varpi}+\Delta_{LA})$
$\mathbf{p}_p(\bar{\varpi}+\Delta_{LA})$
N
x_b
$\tilde{\chi}$
$\mathbf{p}_p(\bar{\varpi})$
e
s
E
y_b

Figure 3.3: Illustration of the distances and angles for path following, namely the Cross-Track Error (CTE) e, Along-Track Error (ATE) s, heading error $\tilde{\chi}$, path reference point $\mathbf{p}_p(\bar{\varpi})$, look-ahead point $\mathbf{p}_p(\bar{\varpi}+\Delta_{LA})$ and look-ahead path tangential angle $\gamma_p(\bar{\varpi}+\Delta_{LA})$.

The agent needs to know how the vessel's current position and if the heading is aligned with the desired path. A few concepts often used for guidance can help formalise this, illustrated in Fig. 3.3. First, we formally define the desired path as the one-dimensional manifold given by:

$$\mathcal{P} \triangleq \left\{\mathbf{p} \in \mathbb{R}^2 \;\middle|\; \mathbf{p} = \mathbf{p}_p(\bar{\varpi}) \; \forall \, \bar{\varpi} \in \mathbb{R}^+\right\} \tag{3.4}$$

Accordingly, for any given $\bar{\varpi}$, we can define a local path reference frame $\{p\}$ centered at $\mathbf{p}_p(\bar{\varpi})$ whose x-axis has been rotated by the angle:

$$\gamma_p(\bar{\varpi}) \triangleq \operatorname{atan2}\left(y_p'(\bar{\varpi}), x_p'(\bar{\varpi})\right) \tag{3.5}$$

relative to the inertial NED -frame. Next, we consider the so-called look-ahead point $\mathbf{p}_p(\bar{\omega} + \Delta_{LA})$, where $\Delta_{LA} > 0$ is the look-ahead distance. In traditional path-following, lookahead-based steering, i.e. setting the look-ahead point direction as the desired course angle, is a commonly used guidance principle [37]. Based on the look-ahead point, we define the *course* error, i.e. the course change needed for the vessel to navigate straight towards the look-ahead point, as:

$$\tilde{\chi}(t) \triangleq \operatorname{atan2}\left(\frac{y_p(\bar{\omega} + \Delta_{LA}) - y_p(\bar{\omega})}{x_p(\bar{\omega} + \Delta_{LA}) - x_p(\bar{\omega})}\right) - \chi(t) \tag{3.6}$$

where $\chi(t)$ is the vessel's current heading as defined in Section 3.2.2. As presented by Breivik et al. [134], given the current vessel position $\mathbf{p}(t)$ we can define the error vector $\boldsymbol{\epsilon}(t) \triangleq [s(t), e(t)]^{\top} \in \mathbb{R}^2$, containing the *along-track* error $s(t)$ and the *cross-track* error $e(t)$ at time t:

$$\boldsymbol{\epsilon}(t) = \mathbf{R}_{z,-\gamma_p(\bar{\omega})}\left(\mathbf{p}(t) - \mathbf{p}_p(\bar{\omega})\right) \tag{3.7}$$

Using Newton's method, a natural approach for updating the path variable $\bar{\omega}$ is to repeatedly calculate the value that yields the closest distance between the path and the vessel. Newton's method only guarantees a local optimum, which prevents sudden path variable jumps given that the previous path variable value is used as the initial guess [135]. Another approach is to update the path variable according to the differential equation:

$$\dot{\bar{\omega}} = \sqrt{u^2 + v^2} \cos \tilde{\chi}(t) - \gamma_{\hat{\omega}} s(t) \tag{3.8}$$

where the ATE coefficient $\gamma_{\hat{\omega}} > 0$ ensures that the absolute ATE $|s(t)|$ will decrease. This method is computationally faster, so we use it in our Python implementation. More specifically in the current work $\gamma_{\hat{\omega}} = 0.05$ and $\Delta_{LA} = 100m$.

Obstacle detection

Using rangefinder sensors as the basis for obstacle avoidance is a natural choice. A reactive navigation system applied to a real-world vessel typically would entail either such a solution or a camera-based one. Given the availability of standard rangefinder sensors such as lidar, radar or sonar, this approach should enable a relatively straightforward transition from the simulated environment to a real one.

In the setup used, $N = 225$ sensors with a total visual span of $S_s = \frac{4\pi}{3}$ radians (240 degrees) are arranged in the trivial manner illustrated in Fig. 3.4. The sensors

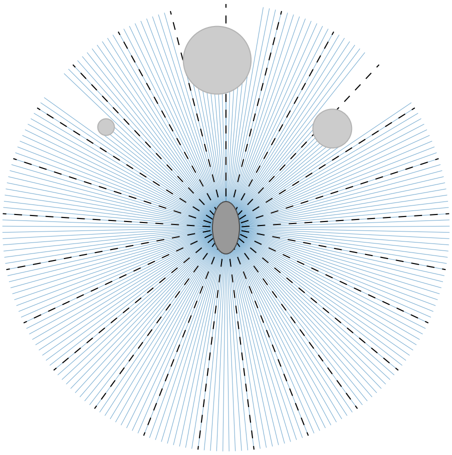


Figure 3.4: Illustration of the distance sensor arrangement. The $N = 225$ sensors are grouped into $d = 25$ sectors, where they are pooled into a single value.

are assumed to have a range of $S_r = 150$ meters, which was deemed sufficient given the relatively small size of the vessel.

The trade-off between computation speed and sensor resolution must be considered w.r.t. the number of sensors. In the experiments conducted in this research project, 225 sensors were chosen, even if it is likely that a much lower number of sensors would yield similar performance. Regarding the visual span, it could be argued that providing 180 degree vision would be sufficient to achieve good collision avoidance, given the precondition of static obstacles. However, to avoid sub-optimal performance due to a restrictive sensor suite configuration, the conservative choice of having a 240 degree vision was made.

Even if, in theory, a sufficiently large neural network is capable of representing any function with any degree of accuracy, including mappings from sensor readings to collision-avoiding steering manoeuvres in our case, there are no guarantees for neither the feasibility of the required network size nor the convergence of the optimisation algorithm used for training to the optimal network weights [60]. Thus, forcing the action network to output the control signal based on 225 sensor readings (as well as the features intended for path-following) is unlikely to be a viable approach, given the complexity required for any mapping between the full sensor suite to the steering signal.

Instead, we propose three approaches for transforming the sensor readings into a reduced observation space. As illustrated in Fig. 3.4, this involves partitioning the sensor suite into d disjoint sensor sets, hereafter referred to as *sectors*. First, we define the sensor *density* n as the number of sensors contained by one sector: $n \triangleq \frac{N}{d}$

Each sector is made up of neighbouring sensors, so we can formally define the k^{th} sector, which we denote by $\mathcal{S}_k$, as:

$$\mathcal{S}_k \triangleq \left\{x_{(k-1)n+1}, \ldots, x_{kn}\right\} \tag{3.9}$$

where x_i refers to the i^{th} sensor measurement according to a counter-clockwise indexing direction. This partitioning, which assumes that N is a multiple of d, is illustrated in Fig. 3.4.

Based on partitioning the sensor suites into sectors, we then seek to reduce the dimensionality of our observation vector. Instead of including each sensor measurement x_i in it, we provide a single scalar feature for each sector $\mathcal{S}_k$, effectively summarising the local sensor readings within the sector. The resulting dimensionality reduction is significant; instead of having N sensor measurements in the observation vector, we now have only d features. What remains is the exact computation procedure by which a single scalar is outputted based on the current sensor readings within each sector.

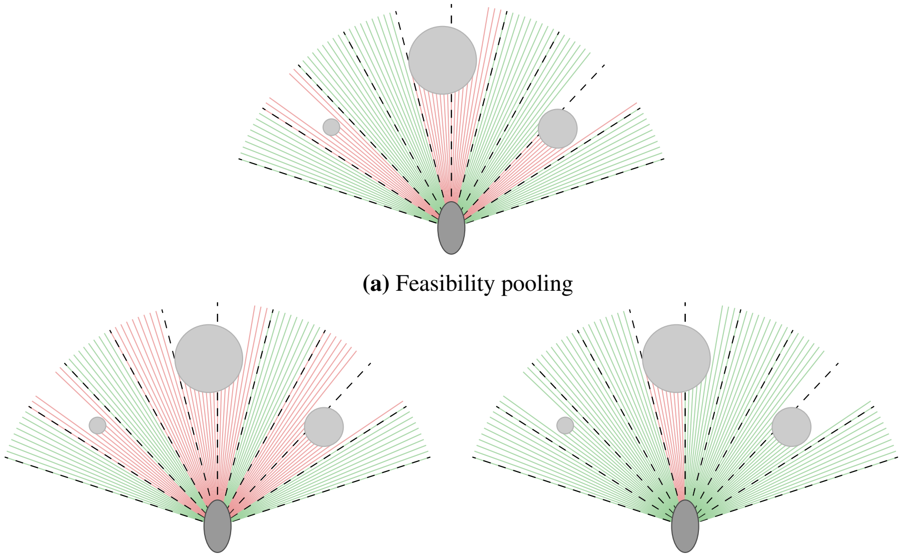
Always returning the minimum sensor reading within the sector, referred to as *min pooling*, i.e. outputting the shortest measured obstacle distance within the sector, is a natural approach which yields a conservative and thereby safe observation vector. However, as seen in Fig. 3.5(b), this approach might be overly restrictive in specific obstacle scenarios, where feasible passings between obstacles are inappropriately overlooked. However, even if the opposite approach (*max pooling*) solves this problem, it is easy to see that this can also lead to dangerous strategies when small obstacles are ignored (see Fig. 3.5(c)).

We introduce a new pooling method to alleviate the problems associated with min and max pooling. The new method computes the maximum feasible travel distance within the sector, taking into account the location of the obstacle sensor readings as well as the width of the vessel. This computation involves iterating over the distance sensor readings in ascending order and checking whether it is feasible for the vessel to advance any further. Algorithm 3.2 provides a pseudocode implementation of this algorithm.

Algorithm 3.2 has a runtime complexity of $\mathcal{O}(dn^2)$ when executed on the entire sensor suite. Therefore, the feasibility pooling approach is asymptotically slower than simple max or min pooling, which are $\mathcal{O}(dn)$. However, Fig. 3.6(a) shows that this additional computation for $n = 9$ is negligible compared to calculating the

Algorithm 3.2 Feasibility pooling for rangefinder

```
Require:
  Vessel width $W \in \mathbb{R}^+$
  Total number of sensors $N \in \mathbb{N}$
  Total sensor span $S_s \in [0, 2\pi]$
  Sensor rangefinder measurements for current sector $\mathbf{x} = \{x_1, \ldots, x_n\}$
  function FEASPOOL($\mathbf{x}$)
      Angle between neighboring sensors $\theta \leftarrow \frac{S_s}{N-1}$
      Initialise $\mathcal{I}$ to be the indices of $\mathbf{x}$ sorted in ascending order according to the measurements $x_i$
      for $i \in \mathcal{I}$ do
          Arc-length $d_i \leftarrow \theta x_i$
          Opening-width $y \leftarrow d_i/2$
          Opening was found $s_i \leftarrow false$
          for $j \leftarrow 0$ to $n$ do
              if $x_j > x_i$ then
                  $y \leftarrow y + d_i$
                  if $y > W$ then
                      $s_i \leftarrow true$
                      break
                  end if
              else
                  $y \leftarrow y + d_i/2$
                  if $y > W$ then
                      $s_i \leftarrow true$
                      break
                  end if
                  $y \leftarrow 0$
              end if
          end for
          if $s_i$ is $false$ then return $x_i$
          end if
      end for
  end function
```



(a) Feasibility pooling

(b) Min pooling

(c) Max pooling

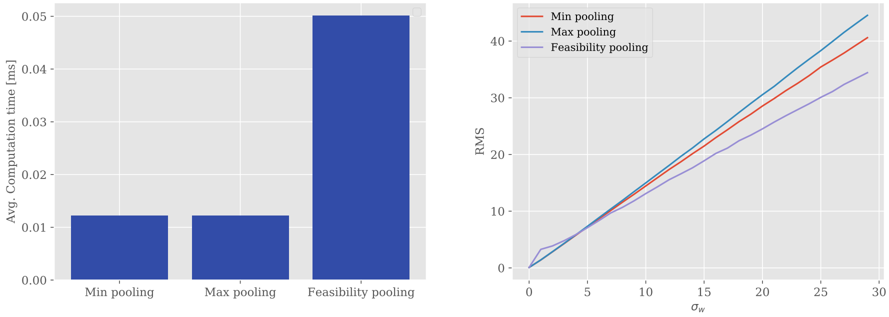
Figure 3.5: Pooling techniques for sensor dimensionality reduction. For the sectors coloured green, the maximum distance S_r was outputted. Min-pooling yields an overly restrictive observation vector, effectively blocking most travel directions. On the other hand, max-pooling is overly optimistic, potentially leading to dangerous situations. Feasibility pooling strikes a balance between min and max pooling.

interception points between the rangefinder rays and the obstacles.

Table 3.1: Rangefinder configuration

Parameter	Description	Value
U_{max}	Maximum vessel speed	2 m/s
W	Vessel width	4 m
N	Number of sensors	225
S_s	Total visual span of sensors	240°
S_r	Maximum rangefinder distance	150 m
d	Number of sensor sectors	25

Another interesting aspect to consider when comparing the pooling methods is the sensitivity to sensor noise. A compelling metric for this is the degree to which the pooling output differs from the original noise-free output when usually distributed noise with standard deviation σ_w is applied to the sensors. Specifically, we report the root mean squared error between the original pooling outputs and the noise-affected measurements. The results for $\sigma_w \in \{1, \ldots, 30\}$ are presented in Fig. 3.6(b), showing that the proposed feasibility pooling method is slightly more robust than the other variants.



(a) Average per-sector computation time for pooling methods when $n = 9$

(b) Robustness metric for pooling methods for $\sigma_w \in \{1, \ldots, 30\}$

Figure 3.6: Computational time and robustness of the different pooling approaches. The noise-affected measurements were clipped at zero to avoid negative values.

3.3.2 Rewards

Any RL agent is motivated by the pursuit of maximum reward. Ideally, the agent should receive its reward at the end of the episode after reaching the goal position or colliding with an obstacle. However, such a reward function is sparse, leaving the agent with a near-impossible learning task. This issue can be alleviated by designing a continuous reward signal that guides the agent to better performance.

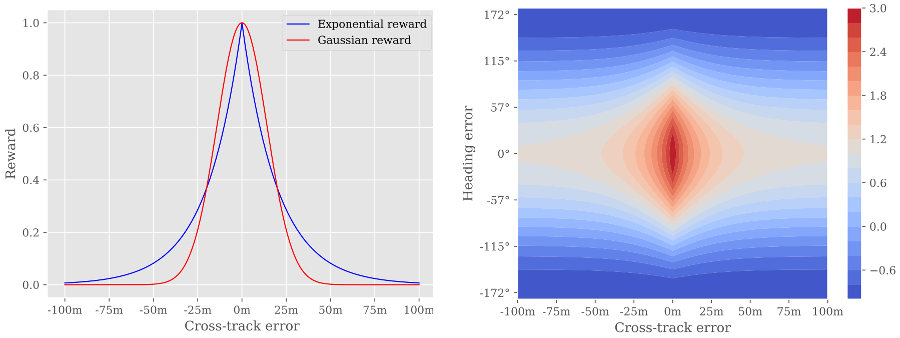
Given the dual nature of the objective, which is to follow the path while avoiding obstacles, rewarding the agent separately for its performance in these two domains is natural. Thus, we introduce the reward terms $r_{pf}(t)$ and $r_{oa}(t)$, being the reward components at time t representing the path-following and the obstacle-avoiding performance, respectively. Also, we introduce the weighting coefficient $\lambda \in [0, 1]$ to regulate the trade-off between the two competing objectives, leading to the following reward function:

$$r(t) = \lambda r_{pf}(t) + (1 - \lambda) r_{oa}(t) \tag{3.10}$$

Path following performance

A reasonable approach to incentivise adherence to the desired path is to reward the agent for minimising the absolute CTE $e(t)$. Martinsen [135] uses a Gaussian reward function centred at $e(t) = 0$ with some reasonable standard deviation σ_e for this purpose. However, based on Fig. 3.7(a), we argue that the exponential

$e^{-\gamma_e|y_e(t)|}$ has slightly more reasonable characteristics for this purpose due to its fatter tails, thus rewarding the agent for a slight improvement to an unsatisfactory location.



(a) Cross-section of the path-following reward assuming path-tangential full-speed motion

(b) Path-following reward function assuming full-speed motion

Figure 3.7: Cross-section and level curves for the path-following reward function for $\gamma_e = 0.05$

However, these measures are not enough to encourage the agent to progress along the path. The velocity component along the desired course is multiplied by $\sqrt{u^2 + v^2} \cos \tilde{\chi}(t)$, which penalises the agent for moving backwards, and no reward if it moves perpendicularly to the path. However, this reward function is zero if the agent stands still or the course error is $\pm 90°$, no matter what the CTE is. Similarly, when the CTE grows large, it receives no reward regardless of the speed or course error. To address this, we add constant multiplier terms 1, which yields the following reward function for path-following:

$$r_{pf}(t) = -1 + \left(\frac{\sqrt{u^2+v^2}}{U_{max}} \cos \tilde{\chi}(t) + 1 \right) \left(e^{-\gamma_e|y_e(t)|} + 1 \right) \tag{3.11}$$

where U_{max} is the maximum vessel speed. Note that, for added flexibility, it is possible to replace the 1 multipliers with optimisable coefficients. However, for parametric simplicity, we decide to use 1.

Obstacle avoidance performance

In order to encourage obstacle-avoiding behaviour, penalising the agent for the *closeness* of nearby obstacles in a strictly increasing manner seems natural. Having access to the sensor measurements outlined in Section 3.3.1 at each timestep, we use these as surrogates for obstacle distances through which the agent is penalised. By noting that the severity of obstacle closeness intuitively does not increase linearly with distance, but instead increases in some more or less exponential manner and that the severity of obstacle closeness depends on the orientation of the vessel with regards to the obstacle in such a manner that obstacles located behind the vessel are of much lower importance than obstacles that are right in front of the vessel, is it easy to see that the term $(1+|\gamma_\theta\theta_i|)^{-1}(\gamma_x \max(x_i, \epsilon_x)^2)^{-1}$, where θ_i is the vessel-relative angle of sensor i such that a forward-pointing sensor has angle 0, exhibits the desirable properties for penalising the vessel based on the i^{th} sensor reading. This reward function is plotted in Fig. 3.8.

In order to cancel the dependency on the specific sensor suite configuration, i.e. the number of sensors and their vessel-relative angles, that arises when this penalty term is summed over all sensors, we use a weighted average to define our obstacle-avoidance reward function such that:

$$r_{oa}(t) = -\frac{\sum_{i=1}^{N}(1+|\gamma_\theta\theta_i|)^{-1}(\gamma_x \max(x_i, \epsilon_x)^2)^{-1}}{\sum_{i=1}^{N}(1+|\gamma_\theta\theta_i|)^{-1}} \tag{3.12}$$

where $\epsilon_x > 0$ is a small constant removing the singularity at $x_i = 0$.

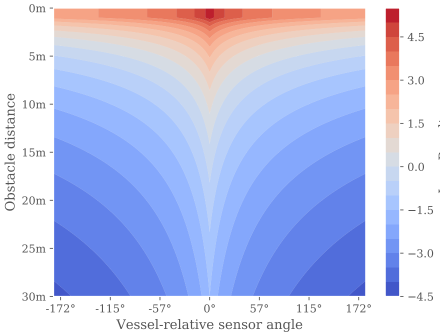


Figure 3.8: Obstacle closeness penalty as a function of vessel-relative sensor angle and obstacle distance, imposing a maximum penalty for obstacles in front of the vessel.

Total reward

In order to discourage the agent from standing still at a safe location, we impose a constant living penalty $r_{exists} < 0$ to the overall reward function. A simple way of setting this parameter is to assume that, given a total absence of nearby obstacles and perfect vessel alignment with the path, the agent should receive a zero reward when moving at a specific slow speed $\alpha_r U_{max}$, where $\alpha_r \in (0,1)$ is a constant parameter. This condition is encoded using the following function:

$$\begin{aligned} &r_{exists} + \lambda\left(\left(\frac{\alpha_r U_{max}}{U_{max}} + 1\right)(1+1) - 1\right) = 0 \\ &r_{exists} = -\lambda(2\alpha_r + 1) \end{aligned} \tag{3.13}$$

Also, in the interest of having bounded rewards, we enforce a lower bound activated upon collisions by defining the total reward:

$$r(t) = \begin{cases} (1-\lambda)\, r_{collision} & \text{(if collision)} \\ \lambda r_{pf}(t) + (1-\lambda)\, r_{oa}(t) + r_{exists} & \text{(otherwise)} \end{cases} \tag{3.14}$$

Deciding the optimal value for the trade-off parameter λ is nontrivial. This issue is closely connected to the fundamental challenge tackled in this project, namely how to avoid obstacles without deviating unnecessarily from the desired trajectory. Thus, we initialise it randomly at each environment reset by sampling it from a probability distribution. In order to familiarise the agent with different degrees of radical collision avoidance strategies ($\lambda \to 0$), which is helpful in dead-end scenarios where the correct behaviour is to ignore the desire for path adherence in order to escape the situation, we sample $\log_{10} \lambda$ from a gamma distribution such that:

$$-\log_{10} \lambda \sim \mathit{Gamma}(\alpha_\lambda, \beta_\lambda) \tag{3.15}$$

To let the agent base its guidance strategy on the current λ, we include $\log_{10} \lambda$ as an additional observation feature. The reward parameters were set as $\alpha_\lambda = 1.0$, $\beta_\lambda = 2.0$, $\gamma_e = 0.05$, $\gamma_\theta = 4.0$, $\gamma_x = 0.005$, $\epsilon_x = 1.0m$, $\alpha_r = 0.1$, $r_{collision} = -2000$.

The complete observation vector, which in the context of RL represents the state s, contains features representing the position and orientation of the vessel regarding the path and the pooled sensor readings and the logarithm of the current trade-off parameter λ.

3.3.3 Methodology

Training

The RL agent is trained using the PPO algorithm (see Algorithm 2.1) implemented in the Python library `stable-baselines` [86], with the hyperparameters given by $\gamma = 0.999$, $T = 1024$, $N_A = 8$, $K = 10^6$, $\eta = 0.0002$, $N_{MB} = 32$, $\lambda = 0.95$, $c_1 = 0.5$, $c_2 = 0.01$, $\epsilon = 0.2$. The action and value function networks were implemented as fully-connected NNs, both using the $\tanh(.)$ activation function and consisting of two hidden layers with 64 nodes. We simulate the vessel dynamics

Table 3.2: Observation vector s at timestep t.

Observation feature	Definition
Surge velocity	$u^{(t)}$
Sway velocity	$v^{(t)}$
Yaw rate	$r^{(t)}$
Look-ahead course error	$\gamma_p(\bar{\omega}^{(t)} + \Delta_{LA}) - \chi^{(t)}$
Course error	$\tilde{\chi}^{(t)}$
CTE	$e^{(t)}$
Reward trade-off parameter	$\log_{10} \lambda^{(t)}$
Obstacle closeness, first sector	$1 - \frac{1}{S_r}\text{FeasPool}(\mathbf{x} = \{x_1, \dots x_d\})$
$\vdots$	$\vdots$
Obstacle closeness, last sector	$1 - \frac{1}{S_r}\text{FeasPool}(\mathbf{x} = \{x_{N-d}, \dots x_N\})$

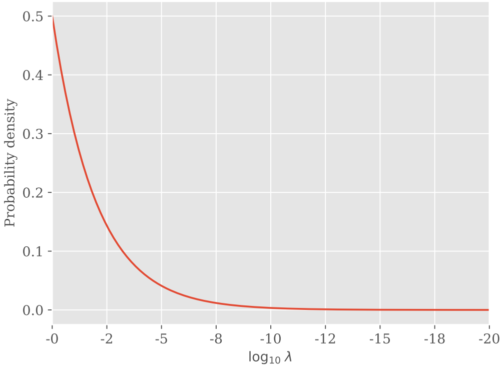


Figure 3.9: Gamma-distribution with parameters $\alpha_\lambda = 1$, $\beta_\lambda = 2$ from which $-\log_{10}\lambda$ is drawn.

using the fifth-order Runge-Kutta-Fahlberg method [136] using the timestep $\Delta t = 0.1s$. Whenever the vessel either reaches the goal $\mathbf{p}_{end}$, collides with an obstacle or reaches a cumulative negative reward exceeding -5000, the environment is reset according to Algorithm 3.1.

Evaluation

We analyse the agent's performance based on quantitative and qualitative testing. Evaluating how the value of the reward trade-off parameter λ, which is fed to the agent as an observation feature, influences the guidance behaviour is of particular interest. A wide range of values was tested, including both radical path adherence (i.e. $\lambda = 1$) and various shades of radical obstacle avoidance strategies (i.e. $\lambda \to 0$). The exact values can be found in Table 3.3.

Quantitative testing

In order to obtain statistically significant evidence for the guidance ability of the trained agent, we simulate the agent's behaviour in 100 random environments generated stochastically according to Algorithm 3.1. We then report the performance criteria regarding success rate, average CTE, and average episode length. In the current context, the success rate is defined as the percentage of episodes in which the agent reached the goal, average CTE is defined as the average deviation from the path in metres, and the average episode length is expressed in seconds.

Qualitative testing

In addition to the statistical evaluation, we observe the agents' behaviour in the test scenarios shown in Fig. 3.10.

Comparison with alternative RL algorithms

In order to assess the performance of the PPO algorithm on this guidance problem, we train the agent using several other frequently cited model-free policy gradient algorithms, a class of RL algorithms known for excelling at continuous control tasks [84]. Deep Deterministic Policy Gradient (DDPG) [70], Actor Critic using Kronecker-Factored Trust Region (ACKTR) [137] and Asynchronous Advantage Actor Critic (A3C) [138] are all available in the `stable-baselines` library. Their quantitative test results will be included as benchmarks for the performance of the PPO agent.

3.3.4 Results and Discussions

This chapter presents the test results obtained from training and testing the agent and discusses the findings.

Training process

We train the agent for 3903 episodes, corresponding to more than 5 million simulated time-steps of length $\Delta t = 0.1s$. At this point, all the metrics used for monitoring the training progress had stabilised. The training process ran eight parallel simulation environments for faster convergence and took approximately 48 hours on an Intel Core i7-8550U processor.

Test results

As outlined, each value of λ was tested for 100 episodes, all of which took place in randomly generated path following environments according to Algorithm 3.1.

Table 3.3: Quantitative test results obtained from 100 episode simulations per agent.

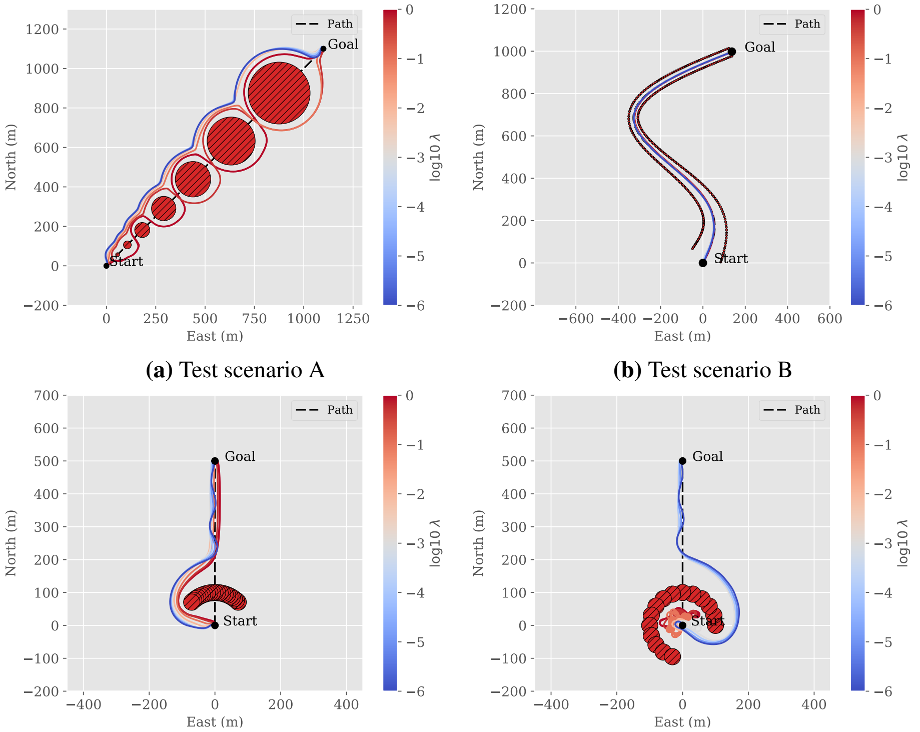
Agent	λ	Success Rate	Avg. CTE	Avg. Ep. Length
1	1	97%	34.92 m	1001 s
2	9×10^{-1}	97%	36.56 m	1028 s
3	5×10^{-1}	99%	38.15 m	1024 s
4	1×10^{-1}	100%	49.13 m	1077 s
5	1×10^{-2}	100%	63.95 m	1062 s
6	1×10^{-3}	100%	68.36 m	1238 s
7	1×10^{-4}	100%	72.99 m	1480 s
8	1×10^{-5}	100%	70.40 m	1469 s
9	1×10^{-6}	100%	70.51 m	1212 s

A larger sample size is beneficial for quantitative evaluation, but in the interest of time, 100 test episodes for each λ value was a reasonable compromise. Calculating the interception points between the rangefinder rays and the obstacles is the most computationally expensive part of the simulation. Thus, the simulation can be made orders of magnitude faster by lowering the sampling rate of the sensors. However, we decided to perform the testing without any restrictions on the sensor suite. The observed test results are displayed in Table 3.3.

Additionally, we simulated each agent in the four outlined qualitative test scenarios. Except for scenario B, in which all agents chose more or less the same trajectory, the other scenarios reflect the differences between the agents. The agents' trajectories in each test scenario are plotted in Fig. 3.10.

The PPO agent was superior to the other RL tested algorithms. PPO was also the only method that completed the task to an acceptable degree. A possible reason is that only the default set of hyper-parameters in the `stable-baselines` package were tested for the other RL algorithms. The A3C agent was the least competent, mindlessly guiding the vessel in an arbitrary direction until a collision occurred. The ACKTR agent appears to master the path-following task but frequently crashes. The DDPG agent rarely collides but does not follow the path and often goes in circles. All four algorithms are compared in Fig. 3.11, where the trained agents are simulated in a randomly generated scenario. The figure illustrates the superior performance of the PPO agent.

These results indicate that a reactive RL agent can become proficient at the combined path-following / COLAV task after being trained using the state-of-the-art PPO algorithm. Before conducting any experiments, we assumed that decreasing λ and thus decreasing the degree to which the agent would prioritise path adherence over



(a) Test scenario A

(b) Test scenario B

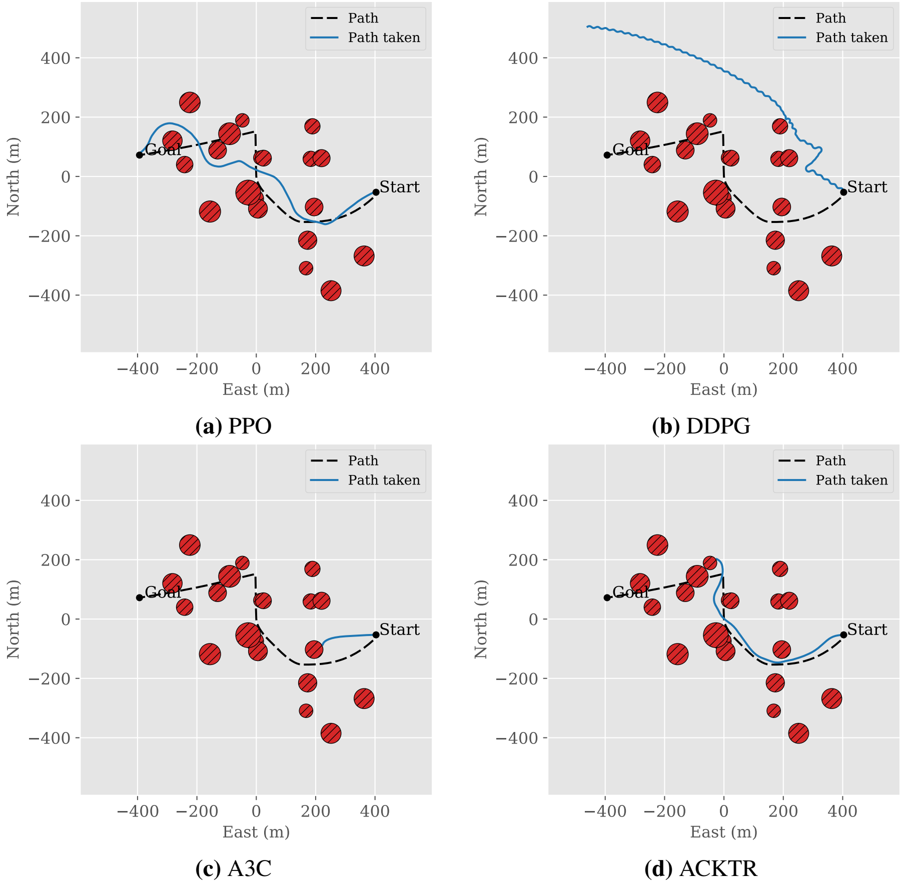
(c) Test scenario C

(d) Test scenario D

Figure 3.10: Agent trajectories in qualitative test scenarios when λ parameter is varied. The behaviour in terms of collision avoidance is significantly modulated.

collision avoidance would lead to a higher success rate. Also, we expected this performance increase to come at the expense of the agent's ability to follow the path, leading to an increase in the average CTE. The results show a clear and reliable trend, supporting our hypothesis. As seen in Table 3.3, the collision avoidance rate stabilises at 100% when λ is sufficiently small. Fig. 3.12, which features two episodes extracted from the training process, clearly illustrates why a small λ will lead to a lower collision rate and cause a significant worsening in path following performance.

From plotting the test metrics against λ, it becomes clear that the trends can be described mathematically by simple parametric functions of λ. After deciding on suitable parameterisations, we use the Levenberg-Marquardt curve-fit method provided by the Python library SciPy [133] to obtain a non-linear least squares estimate for the model parameters. The fitted models for our evaluation metrics can be visualised in Figs. 3.12(a) and 3.12(b). The fitted parametric models allow us to



(a) PPO

(b) DDPG

(c) A3C

(d) ACKTR

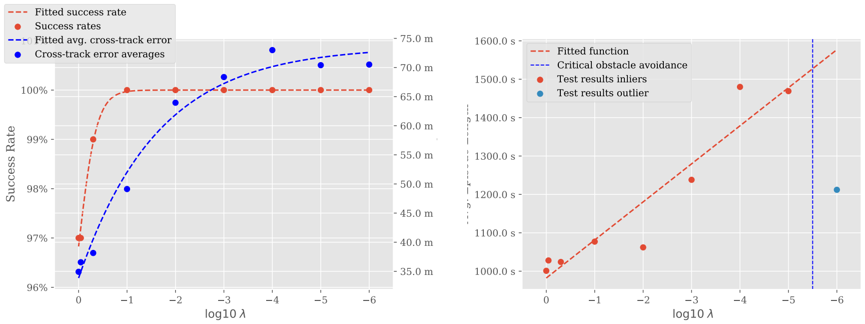
Figure 3.11: Comparison of agent trajectories in randomly generated scenarios for different RL algorithms. All agents were given $\lambda = 1$. Only the PPO agent managed to reach the goal.

generalise the observed results to unseen values of λ.

3.3.5 Discussion

These results demonstrate that RL is a viable approach to the challenging dual-objective problem of controlling a vessel to follow a path given by a priori known waypoints while avoiding obstacles. More specifically, we have shown that the state-of-the-art PPO algorithm converges to a policy that yields intelligent guidance behaviour under the presence of non-moving obstacles surrounding and blocking the desired path.

Engineering the agent's observation vector and the reward function involved design-



(a) Empirical success rates and avg. CTE

(b) Empirical average episode length

Figure 3.12: Empirical success rates. (a) The agent's empirical success rate and avg. CTE fitted to $\hat{f}(\lambda) = a + \frac{1-a}{1+\lambda^b}$ and $\hat{f}(\lambda) = a + b\lambda^{-c}$, respectively. The non-linear least squares estimate for the success rate model parameters is $a = 0.937,\ b = 5.364$, whereas the estimate for the average CTE model parameters is $a = 73.6,\ b = -35.8,\ c = -0.265$. (b) The agent's empirical average episode length fitted to $\hat{f}(\lambda) = a - b\log_{10}\lambda$. The non-linear least squares estimate for the model parameters is $a = 982,\ b = 99.1$. The point marked as an outlier was excluded from the regression. For small λ, the agent will avoid entire clusters of obstacles instead of individual obstacles. Thus, the log-linear model will only be valid until a certain point; this threshold is labelled "critical obstacle avoidance".

ing and implementing several novel ideas, including the Feasibility Pooling algorithm for intelligent real-time sensor suite dimensionality reduction. By augmenting the agent's observation vector by the reward trade-off parameter λ, and thus enabling the agent to adapt to changes in its reward function, we have demonstrated through experiments that the agent is capable of adjusting its guidance strategy (i.e. its preference of path-adherence as opposed to collision avoidance) based on the λ value that is fed to its observation vector.

Even in challenging test environments with many obstacles, the agent's success rate is $> 90\%$ when λ is chosen in favour of path-following and close to 100% when a more defensive strategy is chosen. It is worth mentioning that here, we only studied the impact of λ on the agent's performance. However, we currently lack a procedure for optimising this parameter, e.g. by analysing AIS data. This problem is left as future work.

3.4 Constraint satisfaction using a safety filter

As seen in Section 3.3, it is challenging to guarantee that ML controllers will always respect constraints, especially during the initial training phase. Besides safety considerations, a system might have physical constraints that must be respected. For example, an autonomous ship with azimuthal thrusters must avoid all collisions while keeping in mind the maximum output and turning rate of its thrusters. This issue limits the applicability of RL methods to many real-world systems.

A natural solution to this problem is to use a *predictive safety filter* [139]. This auxiliary system component detects when the controlled system is headed towards an unsafe state. It immediately falls back to a control policy known to be safe, as a driving instructor might intervene during a lesson. The defining characteristic of a predictive safety filter is that it attempts to find a minimal modification to the control inputs w.r.t. the constraint set over a finite number of future time steps.

This separation of concerns yields a modular approach, where the learning-based component can be freely designed and optimised while the safety filter guarantees constraint satisfaction. The advantages are two-fold. First, the predictive safety filter is a simpler optimisation problem than a predictive controller that simultaneously optimises performance and safety. Secondly, the learning-based component can be trained using more complex, sparser cost functions without compromising the convergence of the safety filter, which can yield higher performance after training. In maritime systems, the predictive safety filter is comparable to COLAV systems for autonomous ships.

The predictive safety filter is formulated as an optimisation problem over multiple time steps [139] and is, therefore, most comparable to a mid-level COLAV algorithm. The novelty of this work is the implementation and validation of a safety filter on a maritime vessel that can act as a safety harness around, e.g. learning-based planning algorithms higher up in the planning hierarchy.

The work demonstrates the potential for such methods to enable safer navigation and is illustrated for a passenger ferry voyaging in a narrow water canal. The contributions of this article are thus:

- The implementation of a predictive safety filter on a passenger ferry platform for both anti-grounding and ship collision avoidance.
- Validation of the proposed method for a variety of scenarios in simulation

The article is structured as follows. Relevant modelling of the own-ship platform is presented in Section 3.4.1. The safety filter is detailed in Section 3.4.2. Results and their discussions are presented in Section 3.4.3, and finally, Section 3.4.4 concludes the current work.

3.4.1 Modelling

This section considers milliAmpere 1, a small passenger ferry prototype for urban environments. It aims to be a safe, flexible, and on-demand replacement for bridges

[140]. The ferry is small and manoeuvrable, with two azimuthal thrusters mounted on the underside, which allow it to navigate crowded waterways with other ships.

The equations of motion are standard and are given in Example 2.1.2. Pedersen presents a more in-depth derivation of the model and experimental identification of the parameters specific to the milliAmpere ferry [38]. The following sections present the thruster dynamics of the ferry and a simple Line-of-Sight (LOS) controller for waypoint tracking.

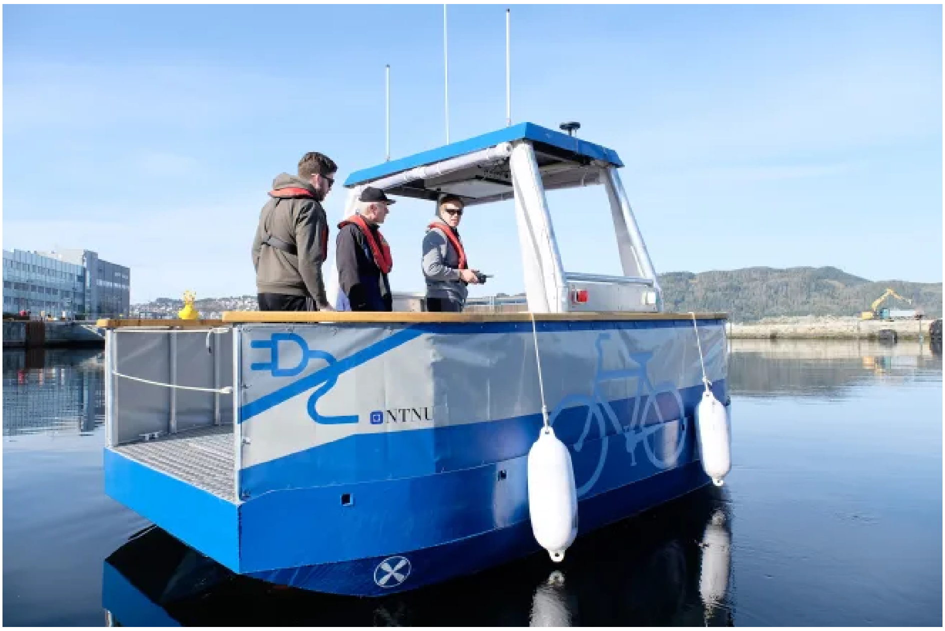


Figure 3.13: The Milliampere ferry

where the pose of the ship is $\boldsymbol{\eta} = [x \quad y \quad \psi]^\top$, the velocity is $\boldsymbol{\nu} = [u \quad v \quad r]^\top$, and the rigid body mass $\mathbf{M}$, Coriolis matrix $\mathbf{C}(\boldsymbol{\nu})$ and damping matrix $\mathbf{D}(\boldsymbol{\nu})$ are all 3×3 matrices. Furthermore, the kinetics can be written as:

Azimuthal thruster dynamics and control allocation

The ferry has two azimuthal thrusters that rotate freely, as shown in Fig. 3.14. These angles are $\boldsymbol{\alpha}$, the motor speeds (in RPM) as $\boldsymbol{\omega}$, and the resulting net thrusts as $\mathbf{f}$. We use the subscript i to refer to properties of an individual thruster, e.g. α_i is the angle of the ith thruster. From Fig. 3.14, we see that the net force and moment on the ship are:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_m \end{bmatrix} = \mathbf{T}(\boldsymbol{\alpha})\mathbf{f} = \begin{bmatrix} \cos\alpha_1 & \cos\alpha_2 \\ \sin\alpha_1 & \sin\alpha_2 \\ \ell_1 \sin\alpha_1 & \ell_2 \sin\alpha_2 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}. \tag{3.16}$$

Choosing the thrust $\mathbf{f}$ to achieve some force $\boldsymbol{\tau}$ is known as the *control allocation* problem. The simplest approach is to extend $\mathbf{T}(\boldsymbol{\alpha})$ by decomposing the thrust vectors into their x and y components:

$$\begin{aligned} \boldsymbol{\tau} =& \mathbf{T}_e \mathbf{f}_e \\ =& \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & \ell_1 & 0 & \ell_2 \end{bmatrix} \begin{bmatrix} f_{1,x} \\ f_{1,y} \\ f_{2,x} \\ f_{2,y} \end{bmatrix} \end{aligned} \tag{3.17}$$

thereby yielding the linear transformation $\mathbf{T}_e$. Appropriate values for $\mathbf{f}$ and $\boldsymbol{\alpha}$ can be found by taking the pseudoinverse of $\mathbf{T}_e$. We denote the control allocation mapping as:

$$(\boldsymbol{\alpha}, \mathbf{f}) = \mathbf{T}^{-1}(\boldsymbol{\tau}) \quad \text{s.t.} \quad \begin{cases} \mathbf{f}_e = \mathbf{T}_e^{\dagger} \boldsymbol{\tau} \\ f_i = \sqrt{f_{i,x}^2 + f_{i,y}^2} & i \in \{1, 2\} \\ \alpha_i = \operatorname{atan2}(f_{i,y}, f_{i,x}) & i \in \{1, 2\} \end{cases} \tag{3.18}$$

This approach does not consider actuator constraints and can yield infeasible control sequences. In this case, the control allocation problem $\mathbf{T}^{-1}(\boldsymbol{\tau})$ is formulated as a constrained Nonlinear Program (NLP) [37]. Control feasibility is implicitly handled by the predictive safety filter framework when actuator constraints are included in the formulation. The thrust f_i is related to the motor RPM ω_i by the following invertible function:

$$\mathbf{f} = \mathbf{L}(\boldsymbol{\omega}) \tag{3.19}$$

This mapping was determined experimentally and modelled as an invertible polynomial [38]. The relationship between the desired motor RPM $\boldsymbol{\omega}_d$ and the actual value $\boldsymbol{\omega}$ is modelled as a proportional control law with gain K_{ω_i} corresponding to thruster i:

$$\dot{\omega}_i = K_{\omega_i}(\omega_{d,i} - \omega_i) \tag{3.20}$$

Finally, the azimuthal thrusters have a constant turning rate K_{α_i}, which is approximated with the help of the function:

$$\lambda(\beta;\epsilon) = \frac{\mathrm{ssa}(\beta)}{\sqrt{\mathrm{ssa}(\beta)^2+\epsilon}} \tag{3.21}$$

where $\mathrm{ssa}(\cdot)$ is the smallest signed angle function. Given the desired angles $\boldsymbol{\alpha}_d$ and the actual angles $\boldsymbol{\alpha}$, the turning rate for the ith thruster is then:

$$\dot{\alpha}_i = K_{\alpha_i}\lambda\left(\alpha_{d,i} - \alpha_i; \epsilon_i\right) \tag{3.22}$$

where K_{α_i}, ϵ_i, K_{ω_i} are thruster-specific parameters. We refer to [38] for the precise values.

Full dynamics

The full dynamics of the ferry can now be written as follows:

$$\begin{aligned}
\dot{\boldsymbol{\eta}} &= \mathbf{R}(\psi)\boldsymbol{\nu} \\
\dot{\boldsymbol{\nu}} &= \mathbf{M}^{-1}\left(\mathbf{T}(\boldsymbol{\alpha})\,\mathbf{L}(\boldsymbol{\omega}) - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right) \\
\dot{\boldsymbol{\omega}} &= \mathbf{K}_{\boldsymbol{\omega}}(\boldsymbol{\omega}_d - \boldsymbol{\omega}) \\
\dot{\alpha}_i &= K_{\alpha_i}\lambda\left(\alpha_{d,i} - \alpha_i; \epsilon_i\right)
\end{aligned} \tag{3.23}$$

The inputs to this model are the desired thruster angles $\boldsymbol{\alpha}_d$ and thrust vector $\boldsymbol{\omega}_d$, which are typically computed from a desired force $\boldsymbol{\tau}_d$ using Eq. (3.18). However, as we shall see later, it is simpler to skip this step in formulating the predictive safety filter and select $\boldsymbol{\alpha}_d$ and $\boldsymbol{\omega}_d$ as decision variables.

Naive controller

In order to validate the safety filter, we use a simple LOS guidance law controller that tracks a path specified by a sequence of waypoints. When the ownship comes within a specified radius of the current target waypoint, the reference is switched to the next waypoint in the sequence [37]. The LOS guidance law is defined as:

$$\psi_d = \pi_p - \arctan\left(\frac{y_e}{\Delta}\right) \tag{3.24}$$

where π_p is the angle of the vector from the previous waypoint to the current target waypoint defined in the North-East (NE)-frame, Δ is the *look-ahead distance*, and y_e is the *cross-track error* [37]. The desired force on the ship is then defined as:

$$\boldsymbol{\tau}_d = \begin{bmatrix} \tau_{x,d} \\ \tau_{y,d} \\ \tau_{m,d} \end{bmatrix} = \begin{bmatrix} T_{x,d} \\ 0 \\ K_p(\psi_d - \psi) \end{bmatrix} \tag{3.25}$$

The $\boldsymbol{\tau}_d$ vector is a constant desired force $T_{x,d}$ in the forward ship direction, zero desired force laterally, and a desired moment proportional to the difference between the desired heading and actual heading. Finally, the desired force vector is mapped to control inputs $(\mathbf{f}_d, \boldsymbol{\alpha}_d)$ by:

$$\begin{aligned} (\mathbf{f}_d, \boldsymbol{\alpha}_d) &= \mathbf{T}^{-1}(\boldsymbol{\tau}_d) \\ (\boldsymbol{\alpha}, \boldsymbol{\omega}) &= \big(\boldsymbol{\alpha}_d, \mathbf{L}^{-1}(\mathbf{f}_d)\big) \end{aligned} \tag{3.26}$$

These control inputs are likely unsafe or infeasible and are passed on to the safety filter for evaluation and modification.

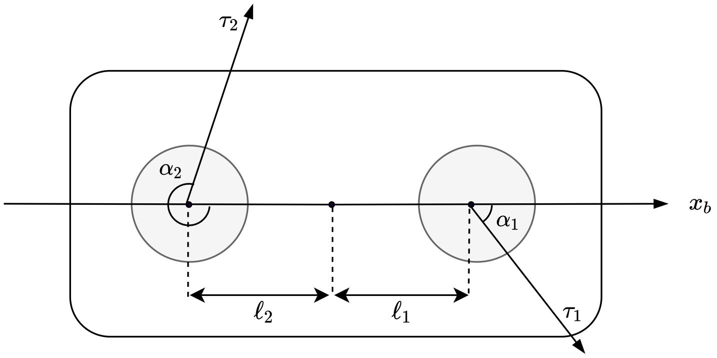


Figure 3.14: Actuators on the milliAmpere ferry: Two azimuthal thrusters can rotate freely.

3.4.2 Safety filter

The predictive safety filter is formulated as an optimisation problem constrained by the ferry dynamics, state and actuator limits, and anti-collision conditions. The objective of the problem is to find a minimal perturbation $\boldsymbol{\delta}$ to the input $\boldsymbol{\tau}$ such that the safety requirements are satisfied. The modified input $\bar{\boldsymbol{\tau}}$ is then passed to the system. Fig. 3.15 shows how the safety filter interacts with an idealised guidance and navigation system. Note that we assume perfect knowledge of the state of the ship and obstacles; we leave the handling of uncertainty and robust constraint satisfaction as future work.

The system dynamics given by Eq. (3.23) are discretised using an explicit Runge-Kutta method of order 1 with constant time-step h. The inputs and states at each

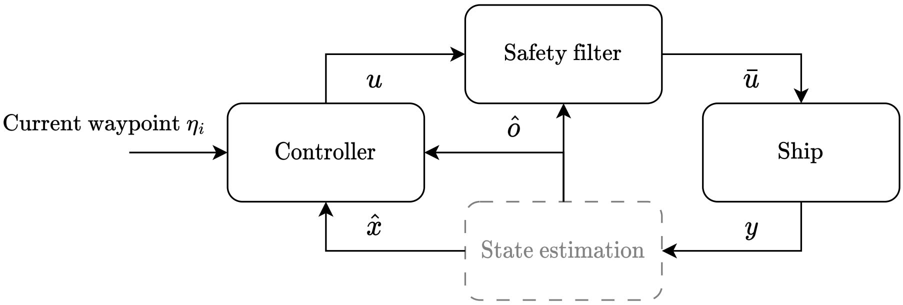


Figure 3.15: Interactions between safety filter and typical guidance system. State estimation is not included in this work.

time step are taken as decision variables, also known as direct multiple shooting. In the following, we use subscript k to refer to the time step, i for a vector element, and j for the jth obstacle. The full NLP is written as: :

$$\begin{aligned}
\min_{\boldsymbol{\eta}_k,\boldsymbol{\nu}_k,\boldsymbol{\alpha}_k,\boldsymbol{\omega}_k\boldsymbol{\delta}_k} \quad & \sum_{k=1}^{N} \gamma_\alpha^2\|\boldsymbol{\delta}_{\alpha,k}\|^2 + \gamma_\omega^2\|\boldsymbol{\delta}_{\omega,k}\|^2 \\
\text{s.t.} & \alpha_{lb} \leq \alpha_{i,k} \leq \alpha_{ub} \quad \forall i,k \\
& \omega_{lb} \leq \omega_{i,k} \leq \omega_{ub} \quad \forall i,k \\
& \bar{\boldsymbol{\alpha}}_k = \boldsymbol{\alpha}_k + \boldsymbol{\delta}_{\alpha,k} \\
& \bar{\boldsymbol{\omega}}_k = \boldsymbol{\omega}_k + \boldsymbol{\delta}_{\omega,k} \\
& \boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + h\mathbf{R}(\psi)\boldsymbol{\nu}_k \\
& \boldsymbol{\nu}_{k+1} = \boldsymbol{\nu}_k + h\mathbf{M}^{-1}[\boldsymbol{\tau}(\bar{\boldsymbol{\alpha}}_k,\bar{\boldsymbol{\omega}}_k) - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu}] \\
& |\bar{\alpha}_{i,k+1} - \bar{\alpha}_{i,k}| \leq \Delta\alpha \quad \forall i,k \\
& |\bar{\omega}_{i,k+1} - \bar{\omega}_{i,k}| \leq \Delta\omega \quad \forall i,k \\
& (\mathbf{A}\mathbf{p}_k - \mathbf{b}) + d \leq 0 \quad \forall k \\
& g_j(\mathbf{p}_k, \mathbf{o}_{j,k}) > 0 \quad \forall j,k \\
& \mathbf{o}_{j,k+1} = \mathbf{o}_{j,k} + \mathbf{v}_j h \quad \forall j,k
\end{aligned} \tag{3.27}$$

The position $[x \quad y]$ of the own-ship at time-step k is denoted $\mathbf{p}_k$, and d is the safe radius. The perturbed inputs $\bar{\boldsymbol{\alpha}}$ and $\bar{\boldsymbol{\omega}}$ are bounded by $(\alpha_{lb},\alpha_{ub})$ and $(\omega_{lb},\omega_{ub})$ respectively, and rate-limited by the constants $\Delta\alpha$ and $\Delta\omega$ respectively. The rate-limiting constraints were chosen instead of directly modelling the actuator dynamics given in Eq. (3.23). This step simplifies the constraint formulation while maintaining a sufficiently accurate approximation, particularly for $\boldsymbol{\alpha}$.

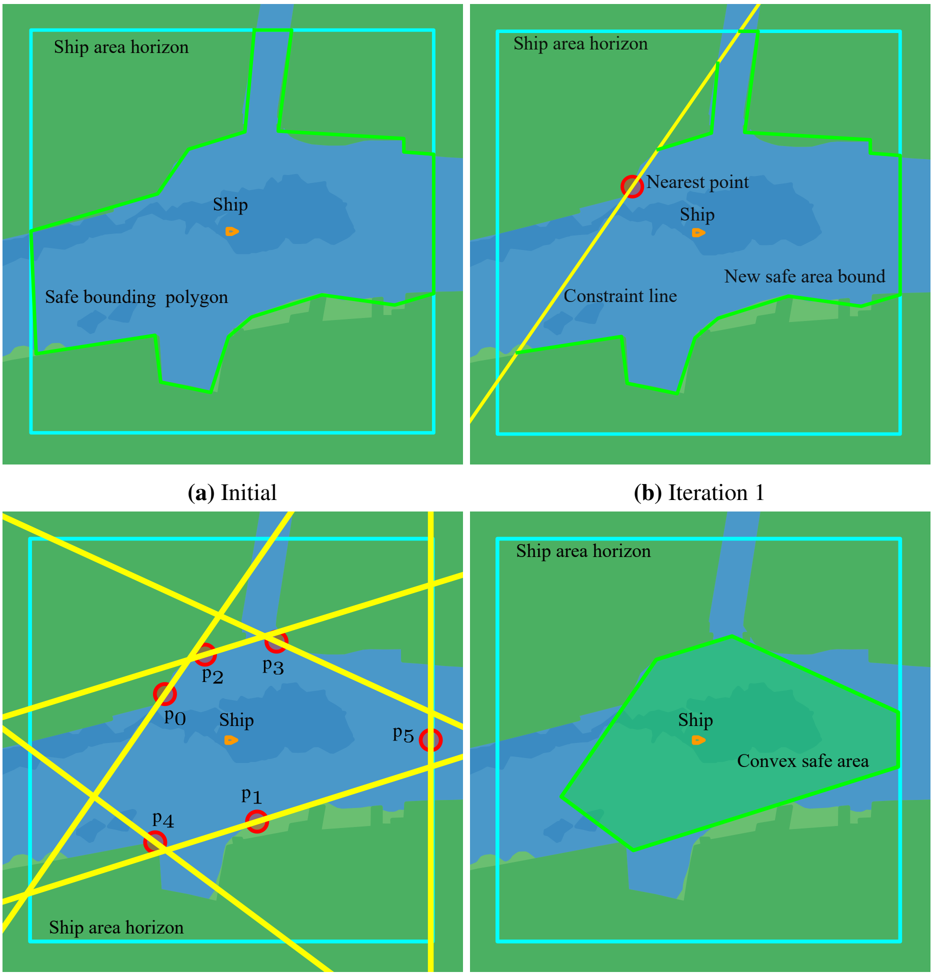
Anti-grounding is achieved by defining a safe water region using the linear constraint set defined by $\mathbf{A}$ and $\mathbf{b}$, regularly updated as the vessel advances along the path. Section 3.4.2 describes how $\mathbf{A}$ and $\mathbf{b}$ are computed. In this formulation, both static and dynamic obstacles are modelled as ellipses centred at the positions $\mathbf{o}_{j,k} = [x_{obs} \;\; y_{obs}]$, where k again refers to the time-step and j is the obstacle index. The unsafe ellipses are represented as signed distance functions $g_j(\cdot)$, which measure the distance between the safe radius of the own-ship and the surface of the jth ellipse, and are defined precisely in Section 3.4.2. If the safe radius and the ellipse intersect, the distance functions return a negative value. Dynamic obstacle movement is modelled with a constant velocity $\mathbf{v}_j$, which updates the position $\mathbf{o}_{j,k}$ at each time step. Further implementation details and parameter values can be found in Section 3.4.2.

Anti-grounding constraint representation

In the vicinity of land, the area where the ship can safely navigate within a given timespan is generally a non-convex set. This constraint is challenging to model in an optimisation problem. Instead, a convex subset of this area is identified online from cartographic data. We refer to this area as the Convex Safe Set (CSS). The procedure is based on the algorithm presented by Bitar et al. [141] and is summarised in Algorithm 3.3. Our implementation uses the Shapely python package [142]. The estimated CSS is updated every 10 seconds during the simulation. The advantages of this approach are that it is relatively cheap to compute the CSS, it can be represented as a linear constraint set, and it can be pre-computed at regular points along a nominal trajectory if needed. The disadvantage is that a CSS can be overly conservative, especially when the vessel is near either land or some obstacle.

Algorithm 3.3 Inner convex safe set estimation

```
$p \leftarrow$ Current position of ship
$H \leftarrow$ Rectangle with center at $p$ and width,height = $D_{max}$
$S \leftarrow$ Polygonal representation of safe region around $p$, extracted from map data
$S' \leftarrow H \cap S$
$B \leftarrow$ Boundary($S'$)
$C \leftarrow$ Empty table for storing constraints
while $B \neq Empty$ do
    $p_n \leftarrow$ Nearest point on $B$, as seen from $p$
    $C' \leftarrow$ Constraint line orthogonal to $(p_n - p)$, with mid-point at $p_n$
    Remove segments of $B$ that are outside of constraint line $C'$, as seen from $p$
    Add constraint line $C'$ to table $C$
end while
return $C$
```



(a) Initial

(b) Iteration 1

(c) Final iteration

(d) Final CSS

Figure 3.16: Algorithm for computing convex safe set

Obstacle constraint representation

Each obstacle is represented as an oriented ellipse, which provides more design flexibility than a circular representation and more closely matches the profile of a typical vessel. In theory, it is possible to use Algorithm 3.3 to compute a safe set without obstacles. However, requiring the safe set to be convex can be conservative, e.g. when the vessel must perform a sharp turn.

Fig. 3.17 illustrates a situation where the convex safe set works poorly. Here the vessel must navigate between two obstacles (denoted obs_1 and obs_2), and the nominal trajectory is, in fact, safe. However, due to the approach angle, no possible convex set contains the nominal trajectory. The safety filter, therefore, corrects course and takes an evasive manoeuvre instead. It can be argued that this issue resolves itself when the ship passes obs$_1$, and a CSS that contains the nominal path can then be found. However, depending on the vessel's speed, it may already be too late to correct course again and avoid obs$_2$. In the best case, the vessel may take the turn later than desirable, which breaks Rules 8 and 16 of Convention on the International Regulations for Preventing Collisions at Sea (COLREGS) (i.e. manoeuvres must be made or signalled in ample time).

By representing the obstacles directly using additional constraints, the predictive safety filter can plan much more effectively. We show experimentally in Section 3.4.3 that this problem is still very tractable despite the added complexity.

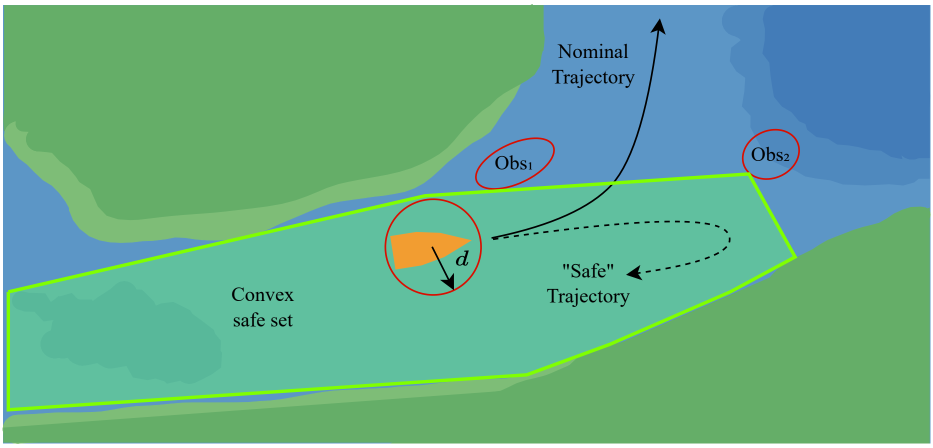


Figure 3.17: Illustration of a convex safe set that also takes obstacles into account. Despite the nominal trajectory being safe, the safe set's convexity requirement is overly conservative.

We define the parameters for the jth obstacle as the tuple $(\mathbf{o}_j, \mathbf{v}_j, a_{obs,j}, b_{obs,j}, \theta_{obs,j})$, where $\mathbf{o}_j$ is the coordinate vector for the centre of the obstacle, $\mathbf{v}_j$ is the velocity of the obstacle, $a_{obs,j}$ and $b_{obs,j}$ is $1/2$ the length of the ellipse in its semi-major and semi-minor axes respectively, and θ_{obs} denotes the angle between the coordinate-frame x-axis and the semi-major axis of the elliptical obstacle. The formula for an elliptical disk can be written as:

$$E(\boldsymbol{x}, a, b) \leq 0 \tag{3.28}$$

where

$$E(\boldsymbol{x}, a, b) = \frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} - 1 \tag{3.29}$$

The constraint function $g_j(\mathbf{p}, \mathbf{o})$ for the jth obstacle is defined as:

$$g_j(\mathbf{p}, \mathbf{o}) = E\left[\boldsymbol{R}(\theta_{obs})(\mathbf{p} - \mathbf{o}),\ a_{obs} + d,\ b_{obs} + d\right] \tag{3.30}$$

where $\boldsymbol{R}(\theta_{obs})$ is the rotation matrix:

$$\boldsymbol{R}(\theta_{obs}) = \begin{bmatrix} \cos\theta_{obs} & -\sin\theta_{obs} \\ \sin\theta_{obs} & \cos\theta_{obs} \end{bmatrix} \tag{3.31}$$

Eq. (3.30) can be expanded as:

$$\begin{aligned} g_j(\mathbf{p}, \mathbf{o}) = & \frac{\left(\cos\theta_{obs}(x - x_{obs}) - \sin\theta_{obs}(y - y_{obs})\right)^2}{(a_{obs} + d)^2} \\ & + \frac{\left(\sin\theta_{obs}(x - x_{obs}) + \cos\theta_{obs}(y - y_{obs})\right)^2}{(b_{obs} + d)^2} - 1 \end{aligned} \tag{3.32}$$

To take into account the dynamic obstacle motion over the prediction horizon, we include a movement constraint on the form $\mathbf{o}_{i,k+1} = \mathbf{o}_{i,k} + \mathbf{v}_i h$, where the linear velocity of the ith obstacle is denoted $\mathbf{v}_i$. Straight-line obstacle motion is thus assumed, which is here deemed reasonable as we do not consider longer time horizons.

Implementation

The `CasADi` symbolic framework is used to encode the resulting optimisation scheme efficiently [143], which is then solved using the open-source `IPOPT` software [144]. A time-step of $h = 0.5s$ yielded sufficiently accurate state predictions for the relatively slow dynamics of the ship. Furthermore, a prediction horizon of $N = 30$ was selected because it satisfactorily balanced performance and computational complexity.

The optimal control problem has $N(n_x + n_u) = 30(6+4) = 300$ decision variables to be computed for each iteration of the optimisation algorithm. All experiments were run on a consumer-grade laptop. The cost parameters were chosen as follows:

$$\begin{aligned} \gamma_\alpha^2 &= \frac{1}{(\alpha_{ub} - \alpha_{lb})^2} \\ \gamma_\omega^2 &= \frac{10}{(\omega_{ub} - \omega_{lb})^2} \end{aligned} \tag{3.33}$$

Due to the relatively high cost of perturbing $\boldsymbol{\omega}$, the safety filter prioritises turning the ship by modifying $\boldsymbol{\alpha}$, rather than slowing down by setting $\bar{\boldsymbol{\omega}} \approx \mathbf{0}$. Table 3.4 shows the parameters used in the experiments.

Table 3.4: Safety filter parameters

Parameter	Value	Description
Δ	$100\,\mathrm{m}$	Lookahead distance
K_ψ	200	Heading gain
$T_{x,d}$	$350\,\mathrm{N}$	Constant forward force
N	30	Horizon length
h	$0.5\,\mathrm{s}$	time-step
$\Delta\alpha$	$0.5\,\mathrm{rad}$	Rate limit ($\boldsymbol{\alpha}$)
$\Delta\omega$	$0.875\,\mathrm{krpm}$	Rate limit ($\boldsymbol{\omega}$)
α_{lb}	$-\pi$ [rad]	Lower bound ($\boldsymbol{\alpha}$)
α_{ub}	π [rad]	Upper bound ($\boldsymbol{\alpha}$)
ω_{lb}	$-4\,\mathrm{krpm}$	Lower bound ($\boldsymbol{\omega}$)
ω_{ub}	$4\,\mathrm{krpm}$	Upper bound ($\boldsymbol{\omega}$)
d	$5\,\mathrm{m}$	Own-ship safe radius
γ_α	$0.159\,\mathrm{rad}^{-1}$	Perturbation cost ($\boldsymbol{\alpha}$)
γ_ω	$0.39\,\mathrm{krpm}^{-1}$	Perturbation cost ($\boldsymbol{\omega}$)

3.4.3 Results and discussions

Realistic scenarios in the Trondheim canal were constructed using the `seacharts` library for Python [145]. All scenarios were designed in the Trondheim canal, as shown in Fig. 3.18, and can be summarised as:

(a) Two wide barriers blocking the canal

(b) Planned path cuts through the land

(c) Curved barrier forcing the ship to backtrack

(d) Case (c) with a longer prediction horizon

(e) Single incoming ship

(f) Two incoming ships

Fig. 3.19 shows how the safety filter corrects unsafe control inputs to avoid collisions in each test case. While we will discuss the results of each case individually, additional focus has been placed on cases (c) and (d), which demonstrate the

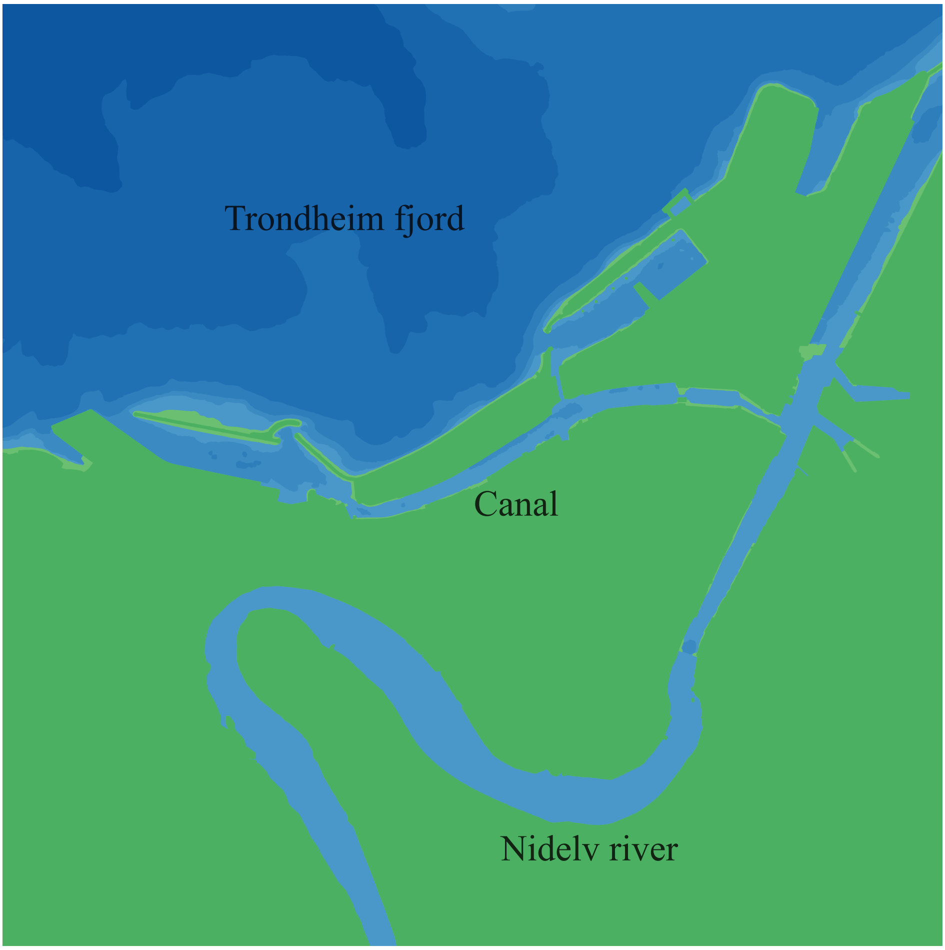
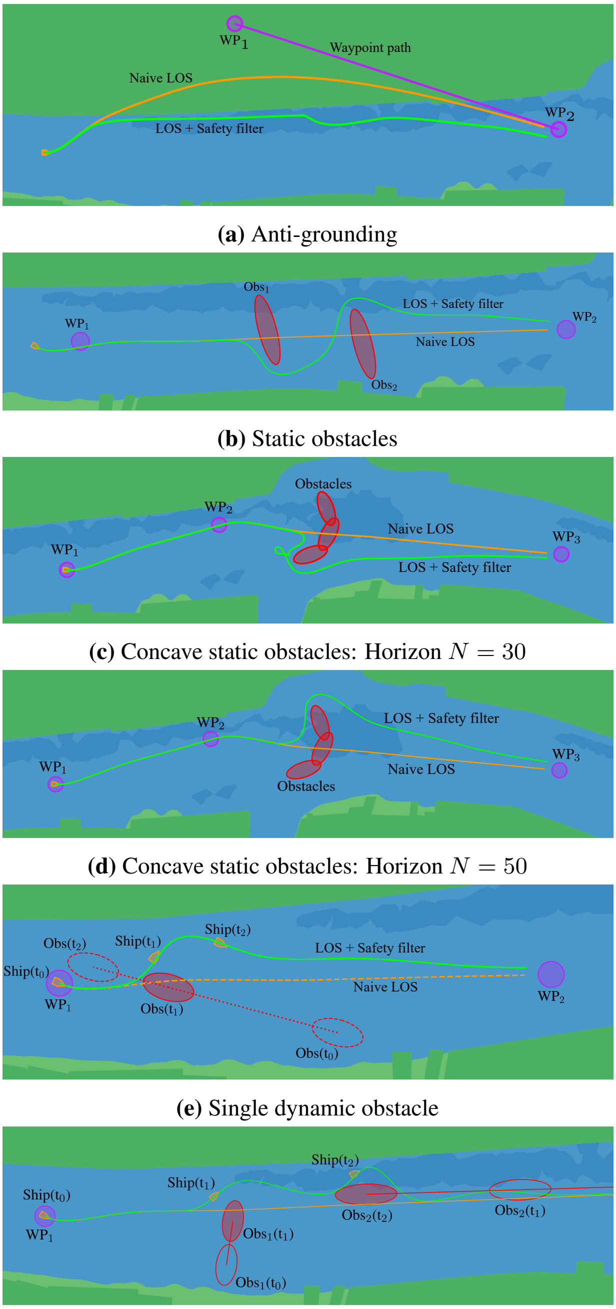


Figure 3.18: Simplified navigational chart of Trondheim area

consequences of handling goal fulfilment and safety separately. The actual perturbations done to the azimuthal angles in cases (c) and (d) are shown in Figs. 3.20 and 3.21 respectively, and the computation time throughout each case is plotted in Fig. 3.23. The motor speed perturbations were relatively small due to the high cost placed on them and were therefore not included.

Fig. 3.19(a) shows that the system can perform effective anti-grounding even when the reference waypoint is infeasible. Notably, a kink is introduced into the otherwise smooth trajectory. This quirk is due to a small outcrop of land further along the path, which yields an overly conservative anti-grounding safe set. Upon investigation, it was found that this type of behaviour can also occur when there is a tight chokepoint



(a) Anti-grounding

(b) Static obstacles

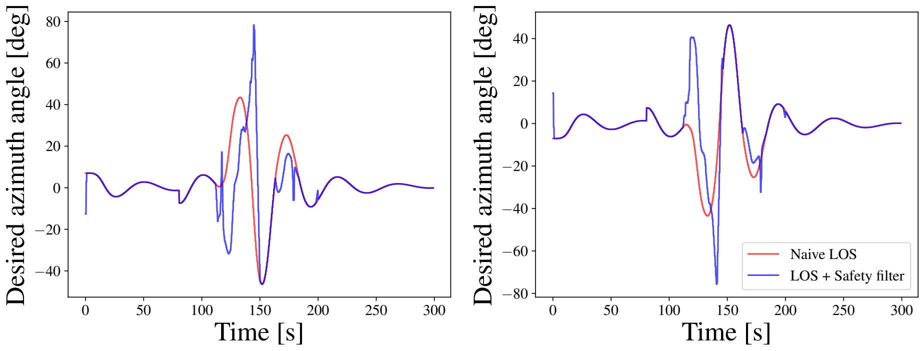
(c) Concave static obstacles: Horizon $N = 30$

(d) Concave static obstacles: Horizon $N = 50$

(e) Single dynamic obstacle

(f) Multiple dynamic obstacles

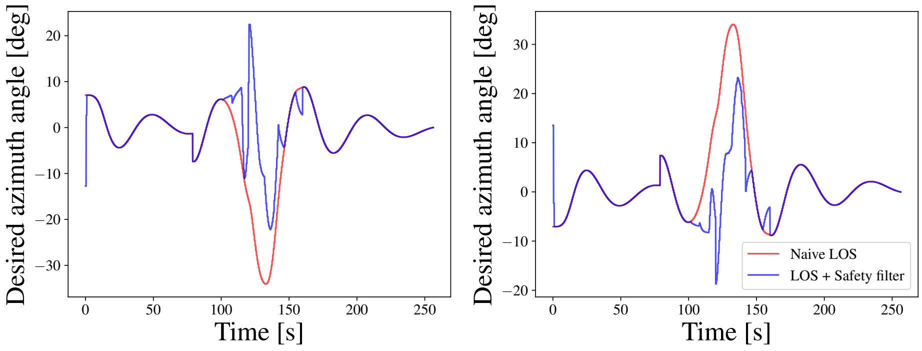
Figure 3.19: Overview of results for all test cases



(a) α_1

(b) α_2

Figure 3.20: Azimuth angle control input modification for case (c)



(a) α_1

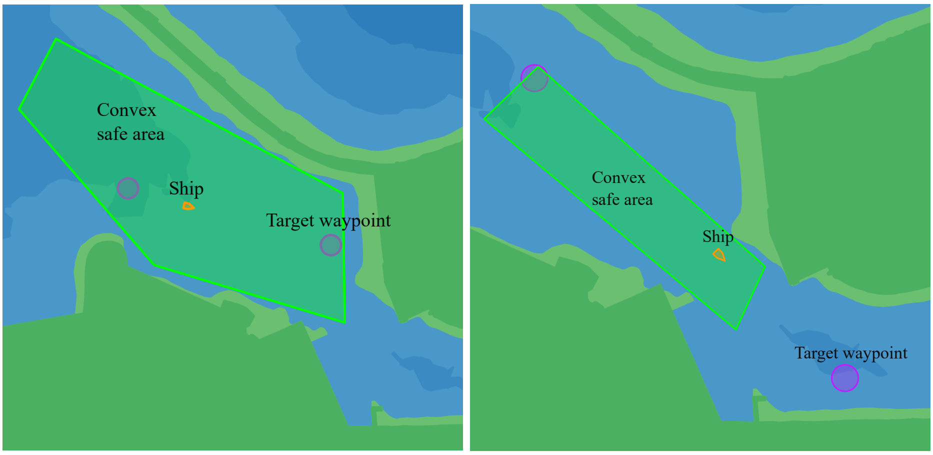
(b) α_2

Figure 3.21: Azimuth angle control input modification for case (d)

in the canal, as seen in Fig. 3.22.

Fig. 3.19(b) shows that the safety filter can avoid large obstacles by turning early. This scenario represents the worst-case where the ownship needs to move from one canal bank to the other in relatively little time.

Figs. 3.19(c) and 3.19(d) show how the behaviour of the safety filter for a planning horizon of 30 s and 50 s respectively. In the first case, the concavity of the barrier forces the ownship to backtrack. In this situation, the safety filter has to force the ownship to move in the opposite direction of the desired path. A shorter planning horizon causes the ownship to turn quite late, causing an aggressive turning manoeuvre. The looping behaviour can be explained by the fact that the safety filter no longer activates when the ownship moves away, such that the



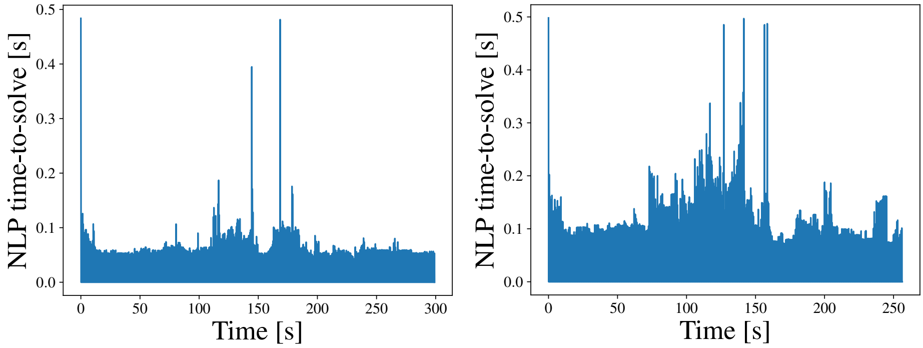
(a) Open area

(b) Chokepoint

Figure 3.22: Comparison of best -and worst-case performance for convex safe set estimation algorithm

naive controller moves the ownship towards the barrier again. This issue is easily mitigated by increasing the planning horizon, as shown in Fig. 3.19(d).

The longer planning incurs a higher solve-time for the safety filter, which occasionally reaches the threshold of computation time. While this might cause the solver to return a sub-optimal solution, in practice, the solve-times quickly decay due to a warm-start strategy being used (see Fig. 3.23(b)).



(a) Case (c)

(b) Case (d)

Figure 3.23: Solve time throughout the trajectory for Cases (c) and (d)

Fig. 3.19(e) shows how the safety filter can also easily handle a large moving obstacle. The positions of the ownship and the dynamic obstacle are shown at three different time steps for clarity. The safety filter can also avoid multiple dynamic obstacles, as shown in Fig. 3.19(f). Again, the positions of the ownship and obstacles are plotted multiple times. These two test cases highlight that the safety filter does not follow the COLREGS [146]. Specifically, the ownship should give way to the right (rules 14 and 15), with a manoeuvre initiated in ample time to signal its intention to the other ships (rule 8). The simple objective function in Eq. (3.27) does not capture these considerations. Instead, the role of the safety filter is to serve as a "last line of defence" for a learning-based algorithm trained to manoeuvre correctly in traffic situations.

3.4.4 Discussion

Self-improving systems that can automatically learn from experience and optimise their performance are increasingly becoming a reality. In practice, it is challenging to guarantee safe operation due to the learning subsystems without significantly restricting their model class. In order to retain this flexibility, other systems that robustly guarantee safety are essential tools in designing control systems that incorporate learning components.

The predictive safety filter framework proposed by Wabersich et al. is adapted to the domain of autonomous collision avoidance for ships [139]. The filter activates when the control system proposes a potentially unsafe trajectory, and it computes a minimal adjustment of the input in order to satisfy the constraints.

Anti-grounding is achieved via the computation of a convex safe set from cartographic data using the method proposed by Bitar et al. [141]. In addition, static and dynamic obstacles are modelled as ellipses, and the corresponding distance functions are used as constraints in formulating the predictive safety filter.

The predictive safety filter was implemented using open-source software and is shown to be feasible for real-time applications ($< 1\,\mathrm{Hz}$), despite the nonlinear obstacle constraints. The performance was not optimised further in this work, but this can be achieved without significant effort by using the `acados` library [147] to compile the solver code or simply by using faster hardware.

The limitations of the approach include (i) Sub-optimal behaviour when the planning horizon is too short, (ii) Conservative, safe set estimation that performs poorly in narrow canals, (iii) Static obstacles are modelled with constant velocities (iv) No handling of uncertainties (v) The safe trajectories do not follow traffic rules according to the COLREGS [146]. Points (ii-v) are of particular interest and will be the subject of future work.

Chapter 4

Modelling and system identification

Many real-world phenomena can be modelled as differential equations, which allow us to predict the change in the state of the system over time. The evolution of the weather, progression of chemical reactions, the spread of diseases, and dynamics of vehicles can all be modelled as dynamical systems. These equations are often derived from first principles, an approach we refer to as PBM. Through careful observation, we can develop theories to describe and understand the underlying system. This understanding is condensed into mathematical equations, which are either solved directly or discretised via some numerical algorithm to make predictions about the system.

Despite the considerable effort involved in developing most models, in the end, they must be adjusted and tuned so that their predictions match the observations we make. In this sense, all models are data-driven, and our prior knowledge only serves to provide the appropriate structure. Since most of our effort is placed into the design of this structure, it would be immensely useful if we could develop algorithms that automate this process. This is the intuition behind DDM, where we try to develop models directly from data (see Fig. 4.2(b)). This approach has gained popularity with the rapid progress in machine learning and the massive increase in the amount of available data. Indeed, DDMs offers enormous flexibility and can provide remarkably accurate predictions with relatively little computation, even when the underlying data-generating process is not understood. However, it is well known that these models do not generalise well, meaning that they often fail when faced with data that is not well represented by the training data.

In applications where large amounts of high-dimensional data are available (or can be generated on demand), DNNs have been successfully applied to problems once considered impossible to solve computationally. For example, [91] combine DNNs and Monte-Carlo tree search to create an agent that plays Go at superhuman levels. This approach is not limited to boardgames; [148] use a variant of this method to greatly improve on protein-structure prediction, a very difficult problem from computational chemistry. Within the realm of dynamical systems, [149] use physics-incorporated convolutional recurrent NNs for dynamical systems forecasting and source identification. They are also often used in RL to represent a value function or a model for some dynamical system. The reason behind this success is that NNs are universal function approximators. More importantly, they can be efficiently evaluated on parallel hardware such as a graphics processing unit (GPU) and are trained using SGD. This means that they can be scaled to very large problems. However, they require a lot of data to achieve good performance and avoid overfitting. One hypothesis for this is that NNs are typically over-parameterised and therefore require many steps to adjust all of the parameters. Overtraining on the same limited dataset will cause the model to overfit the training data and perform poorly on unseen data. While over-parameterisation has been found to aid convergence during training [150], it also introduces redundant information into the weights. Furthermore, there is a lack of robust theory for the analysis of properties such as stability and robustness, and practitioners often have to fall back on empirical testing to assure the safety of their models. Despite these advantages, there remain some challenges before these models can find their way into high stake or safety-critical applications.

For problems where data is scarce or expensive to procure, these drawbacks mean that a middle ground between PBM and DDM is sought, with a preference for more transparent linear methods that can yield more insight into the underlying phenomena. For example, [151] use multivariate statistical methods to develop a model of the internal state of an aluminium smelting process. We call this approach HAM, although many other terms have been coined in the literature, such as Informed Machine Learning [152], Scientific Machine Learning (SciML) [153], and Structured Learning [154]. Models following this paradigm are developed at the intersection of PBM, DDM and Big-Data (see Fig. 4.3).

A natural way to merge DDMs and PBMs is to treat them as modules and connect them in some configuration. For example, Fig. 4.1 shows how NNs can be seen as dense computational graphs, which can correspond to equations.

$$\begin{aligned}\dot{\theta} &= \omega \\ \dot{\omega} &= -\sin\theta - \omega\end{aligned}$$

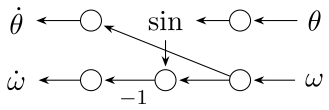


Figure 4.1: Correspondence between equation-based models and computational graphs

4.1 Literature review

Both [155] and [156] provide comprehensive overviews of techniques for integrating DDM with PBM. Many of the hybridisation techniques fall into the following categories: (i) Embedding PBMs inside NNs, (ii) Model order reduction, (iii) Physics-based regularisation terms, (iv) Data-driven equation discovery, (v) Error correction approaches, and (vi) Sanity check mechanisms using PBMs.

4.1.1 Structural methods

The most straightforward class of methods is to simply embed a PBM into a differentiable framework such as PyTorch [63]. For example, recent work developed a differentiable convex optimisation solver that can be used as a module in a NN [157]. In related work, [158] propose the differentiable physics engine, a rigid body simulator that can be embedded into a NN. They demonstrate that it is possible to learn a mapping from visual data to the positions and velocities of objects, which are then updated using the simulator. The same has been done for linear complementarity problems, which has been used to create a differentiable physics simulator with analytical gradients [158]. Similar ideas have been used to simulate a structural dynamics problem by designing a hybrid recurrent neural network (RNN) that contains an implicit numerical integrator [159]. This approach can serve as a powerful inductive bias for machine learning problems, allowing the specification of structure and constraints. However, PBM methods are often iterative, which increases computational costs relative to a standard NN during both training and inference. The advantage of these approaches is that they are usually quite data-efficient. A disadvantage is that these embedded PBMs are often iterative methods, making both inference and training more expensive. Furthermore, iterative methods can be difficult to train using SGD methods due to vanishing/exploding gradients when the loop is unrolled.

4.1.2 Model order reduction methods

Reduced order models (ROMs) are a successful and widely adopted methodology [160]. A ROM method typically projects complex partial differential equations onto a lower dimensional space based on the singular value decomposition of the offline high-fidelity simulation data. This yields a set of ODEs that can be efficiently solved [161]. ROMs have been used to accelerate high-fidelity numerical solvers by several orders of magnitude [162]. However, ROMs tend to become unstable in the presence of unknown/unresolved complex physics. Recent research by Pawar et al. has shown how unknown and hidden physics within a ROM framework can be accounted for using DNNs [163, 164]. Despite these benefits, ROMs require full knowledge of the original equation before they can be applied.

4.1.3 Physics-based regularisation terms

Instead of encoding prior knowledge to produce increasingly complex models, inductive biases can be introduced into the training method itself. The physics-informed neural network (PINN) treats a NN as the solution $\mathbf{x}$ of a PDE [165], e.g. $\mathcal{L}\mathbf{x} = \mathbf{f}(\mathbf{x})$, where $\mathcal{L}$ is a linear differential operator such that $\mathcal{L}\mathbf{x}$ represents any linear combination of derivatives of $\mathbf{x}$. Every term of $\mathcal{L}\mathbf{x}$ can be computed using automatic differentiation for a selection of sample points, and the network will converge to the true solution $\mathbf{x}$ when optimised with the cost function $(\mathcal{L}\mathbf{x} - \mathbf{f}(\mathbf{x}))^2$. This penalty term can be introduced as a soft constraint for models that are additionally trained on measurement data. PINNs can be used to solve problems such as heat transfer, as was done by Zobeiry et al. [166] for parts in a manufacturing process. The PINN approach has also been extended to control applications by framing the method in a state-space setting [167]. Shen et al. create a model for classifying bearing health by training a NN on physics-based features and regularising the model using the output from a physics-based threshold model [168]. In practice, optimising such complex cost functions turns out to be quite challenging [169].

4.1.4 Data-driven equation discovery

Data-driven equation discovery methods attempt to describe data by constructing equations from a dictionary of function primitives [170]. These methods are beneficial when the data are described by hidden or partially known physics. The solutions to this problem are sparse, and many methods employ feature selection based on l_1 regularisation or gene expression programming to find parsimonious solutions [171, 172]. Other works optimise this search by trying to discover symmetries in the data [173]. A notable work is SinDy [174], which uses compressed sensing to approximate data using a sparse library of functions. This is useful when the learned model needs to be interpretable and human-readable. This approach is arguably useful for scientific discovery as well, as it produces equations that can be further analysed and combined with existing theories. These approaches have only been shown to work for relatively low-dimensional examples and require significant computational time. One of the limitations of this class of method is that, in the case of sparse regression, additional features are required to be handcrafted, while in the case of symbolic regression, the resulting models can often be unstable and prone to overfitting.

Deep symbolic regression approaches ([175] and [176]) treat a NN itself as an expression tree and optimise it directly to obtain a closed-form equation, where the neurons in each layer have different activation functions representing the library of allowed functions. While this can quickly fit higher dimensional data, like standard NNs, it tends to overfit the data. While deep symbolic regression can, in theory, express arbitrary compositions of the allowed functions, not all of these functions are relevant, and their presence can induce overfitting. A related concept, called physics-guided neural network (PGNN) ([177, 178, 31]), mitigates this somewhat by only using features that appear in existing PBMs, along with standard activation functions such as `ReLU` to retain the universal approximation capabilities of the network. These functions act as a store of prior knowledge that the network can utilise while still modelling the unknown physics as a black box. Similarly to the PBM embedding approach, complicated features can increase the computational cost.

4.1.5 Error correction and sanity check mechanisms

Corrective source term approach (CoSTA) is a method that explicitly addresses the problem of unknown physics [179]. This is done by augmenting the governing equations of a PBM with a DNN-generated corrective source term that takes into account the remaining unknown/neglected physics. One added benefit of the CoSTA approach is that the physical laws can be used to keep a sanity check on the predictions of the DNN used, i.e. checking conservation laws. A similar approach has also been used to model unresolved physics in turbulent flows [180, 181]. However, even these approaches assume a specific structure for at least the known part of the equation.

4.1.6 Challenges

While many HAM approaches have seen some success, they suffer from various issues, such as increased computational cost for training and inference, inconsistent training convergence, and overfitting. See ([182, 183, 184, 185]) for more in-depth reviews of this field. In this work, the efficacy of the CoSTA approach is investigated, where the output of a discretised PBM is corrected by a DDM trained on the error of the base model. This approach is a natural way to use existing models. For example, [186] used compressed sensing to recover the residual of a PBM from sparse measurements, which is used to improve state estimates using a Kalman filter. CoSTA also has theoretical justifications [179], as it is possible to correct a variety of model errors in this way. Blakseth et al. apply CoSTA to multiple heat transfer problems and show that CoSTA model has an inbuilt sanity check mechanism. [179, 187]

Data-driven modelling using neural networks

NNs are dense models with many parameters. The largest networks in use today often have more parameters than the amount of available data to train them on. For example, the widely publicised GPT-3 model has 175 billion parameters [25]. Because of this, avoiding overfitting and getting deep learning models to generalise is an important topic in deep learning. Methods that accomplish this are generically referred to as *regularisation* [60]. Examples of such methods include weight decay [188], dropout [189], and batch normalisation [190], all of which are essential tools in ensuring a low generalisation error for these models. In recent years, more and more research has shifted towards sparse architectures with significantly fewer non-zero trainable parameters than their dense counterparts [191]. There are many reasons for this. First of all, sparser networks are much cheaper to store and evaluate, which is vital for practitioners wishing to deploy their models on lower-cost hardware [192]. Secondly, recent work shows a tantalising hint that sparse models may generalise better than their dense counterparts. In their seminal work, [193] show with high probability that randomly initialised dense NNs contain subnetworks that can improve generalisation compared to the dense networks.

Many regularisation methods can be expressed as a penalty function $R(\mathbf{w})$ that operates on the parameters θ of the network. The total loss function $C(\mathbf{x}_i, \mathbf{y}_i, \theta)$ used for training the network can then be written as:

$$C(\mathbf{x}_i, \mathbf{y}_i, \theta) = L\left(\mathbf{y}_i, \mathcal{N}(\mathbf{x}_i; \theta)\right) + \lambda R(\mathbf{w}) \tag{4.1}$$

where the set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ is the training dataset, $L(\cdot,\cdot)$ is the *loss function* and $\lambda \in \mathbb{R}^+$ serves to trade-off $L(\cdot,\cdot)$ and $R(\cdot)$.

The standard choice of loss function $L(\cdot,\cdot)$ for regression tasks is the Mean squared error (MSE):

$$L(\mathbf{x}_i, \mathbf{y}_i) = (\mathbf{x}_i - \mathbf{y}_i)^2 \tag{4.2}$$

In the training process, the total cost function $C(\cdot,\cdot)$ is minimized to find optimal values of the parameters:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left\{ \frac{1}{N} \sum_{i=1}^{N} C(\mathbf{x}_i, \mathbf{y}_i, \theta) \right\}. \tag{4.3}$$

The most straightforward way to penalise non-sparse θ is the ℓ_0 norm, often referred to as the sparsity norm:

$$R_{\ell_0}(\mathbf{w}) = ||\mathbf{w}||_0 = \sum_i \begin{cases} 1 & w_i \neq 0, \\ 0 & w_i = 0. \end{cases} \tag{4.4}$$

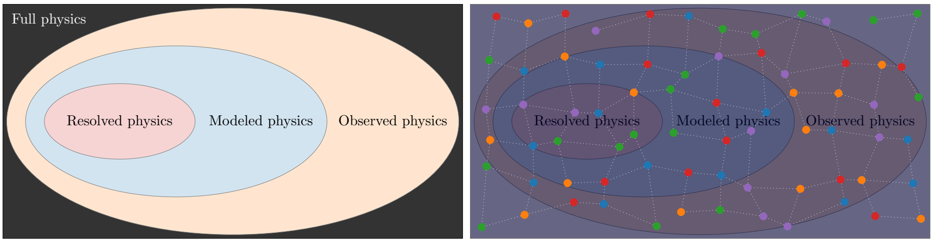
It is clear that $\ell_0(\theta)$ returns the number of nonzero parameters. It has been shown that adding this regularisation term can yield unique solutions for over-determined linear systems, which is the basis of compressed sensing [194]. However, $\ell_0(\theta)$ is non-differentiable, making it unsuitable for gradient descent optimisation. In fact, [195] show that this optimisation problem is NP-hard [195]. Instead, we can utilise the ℓ_1 norm, which is a convex relaxation of the ℓ_0 norm and is given by:

$$R_{\ell_1}(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_i |w_i| \tag{4.5}$$

The ℓ_1 norm sometimes does not reduce the weights to zero, but rather to very small magnitudes. In this case, we can apply a threshold to the weights and set all weights below this threshold to zero. This method is known as *magnitude pruning* and is the simplest of a family of pruning methods [191]. Despite its simplicity, it can reduce the computation complexity of a NN while maintaining the performance of the model [196].

4.2 Physics guided neural networks for modelling of non-linear dynamics

From the previous discussion, it is clear that almost all the HAM approaches discussed above require information about the structure of the equation representing the physics, which is not always available. We often have a very simplistic understanding of a system. For example, we can have some understanding of the diurnal variation of solar radiation but not of its influences on atmospheric flow. [177] proposed a physics-guided machine learning (PGML) approach to exploit prior knowledge of this kind. The idea behind the PGML approach is to inject partial knowledge into one of the layers within a DNN to guide the training process. The



(a) Physics-based modelling **(b)** Data-driven modelling

Figure 4.2: Building a model requires assumptions and simplifications. The result is that of the physics we can observe and understand; only a small part of the physics can be described using models, and even less can be numerically simulated. In contrast, large datasets can cover the full space, enabling general ML models to provide predictions in the absence of understanding or models.

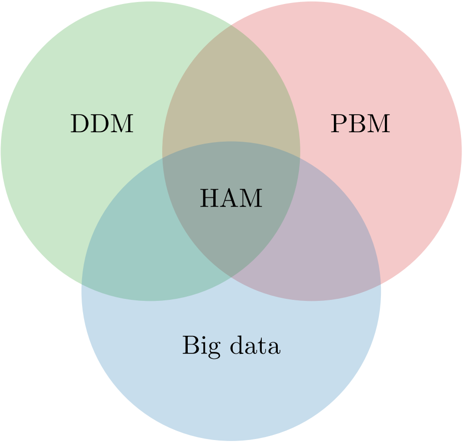


Figure 4.3: Hybrid analysis and modelling at the intersection of PBM, DDM and Big data.

partial knowledge can, for example, come from a simplistic model or an empirical law [178, 197].

This work extends the PGML concept to modelling nonlinear dynamical systems. Since we limit the model space to NNs, we call the approach PGNN. Through a series of experiments involving a variety of equations representing nonlinear dynamical systems like Lotka-Volterra, Duffing, Van der Pol, Lorenz, and Henon-Heiles equations, we attempt to answer the following questions:

- What are the effects of knowledge injection on training convergence?
- How does the accuracy/performance change with the choice of injection

layer?

- Is there any correlation between model uncertainty and knowledge injection?

The proposed method's brief background and rationale are given in Section 4.2.1. Section 4.2.2 details the selected dynamical systems considered in our study. Finally, the results are discussed in Section 4.2.3, and conclusions and recommendations for future work are made in Section 4.2.4.

4.2.1 Physics-guided neural networks

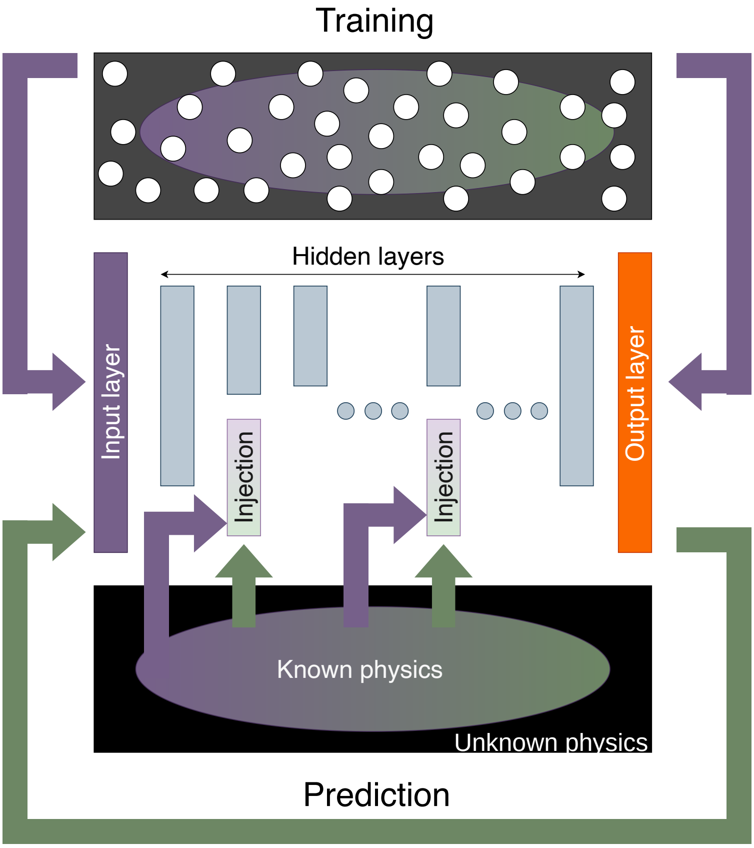


Figure 4.4: PGNN framework: The purple arrows correspond to the training phase, the green arrows correspond to the prediction phase, and the white circles represent the data.

The basic idea behind PGNN is to generalise the Principal Component Regression (PCR). In PCR, instead of directly regressing the dependent variable on the explanatory variables, the latent variables derived from the explanatory variables after applying Principal Component Analysis (PCA) are used as regressors. Replacing the high dimensional explanatory variables (containing redundancy) with lower dimensional latent variables as the input to the regression model can significantly reduce the complexity of the regressors and make them more robust. However, two problems are associated with PCR: (i) the latent variables computed using PCA can only be a linear combination of the explanatory variables, (ii) The regression task is decoupled from the latent variable computation.

In the PGNN approach, both the computation of the latent variables and the regression are combined within a neural network framework with bottleneck layers representing the latent variable layers. Additionally, the latent variables can be supplemented with additional features (partial knowledge) to improve the accuracy and reduce the uncertainty of the trained PGNN model. If the additional features were combined with the explanatory features as input to the DNN, chances are high that they would get corrupted during the training process.

We now present the rationale behind the PGNN approach to modelling nonlinear dynamics. Many studies have recently used deep learning to model the spatiotemporal dynamics of high-dimensional systems [198, 199, 163]. Given some dynamical system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$, a NN can be trained directly on the mapping $\mathbf{x}(\cdot)$ by sampling repeatedly from the system. After training, the network can then be numerically integrated in order to perform predictions on the future states of the system, e.g. by computing the forward Euler step $\mathbf{x}_{k+1} = \mathbf{x}_k + h\,\hat{\mathbf{f}}(\mathbf{x}_k)$. Consider a dataset generated by a more general dynamical system:

$$\mathcal{L}\mathbf{x} = \mathbf{f}\left(\mathbf{g}(\mathbf{x}), \mathbf{h}(\mathbf{x})\right) \tag{4.6}$$

where $\mathcal{L}$ is a linear differential operator, and $\mathbf{f}(\cdot)$, $\mathbf{g}(\cdot)$ and $\mathbf{h}(\cdot)$ are functions of the state. Now, the following scenarios can arise:

1. Eq. (4.6) is fully known meaning that the operator $\mathcal{L}$, and the functions $\mathbf{f}(\cdot)$, $\mathbf{g}(\cdot)$ and $\mathbf{h}(\cdot)$ are precisely known
2. The operator $\mathcal{L}$ is known but one or two of the functions $\mathbf{f}(\cdot)$, $\mathbf{g}(\cdot)$ and $\mathbf{h}(\cdot)$ are unknown
3. The operator $\mathcal{L}$ is known, but the functions $\mathbf{f}(\cdot)$, $\mathbf{g}(\cdot)$ and $\mathbf{h}(\cdot)$ are all unknown
4. The operator $\mathcal{L}$ and the functions $\mathbf{f}(\cdot)$, $\mathbf{g}(\cdot)$ and $\mathbf{h}(\cdot)$ are all unknown

A purely PBM approach using the known equations can be applied to the first scenario. The only advantage of DDM over PBM is possibly superior computational performance, which may enable real-time applications. In the second and third scenarios, while the problem can, in theory, be solved entirely using DDM, it would be unwise to ignore the known part completely. Incorporating them into the DDM may simplify the learning task and improve generalisation. In the fourth scenario, PBM is impossible, and it is necessary to model the process entirely from data.

Assume now that only $\mathbf{h}(\mathbf{x})$ is known in Eq. (4.6). Then Eq. (4.6) can be learned using a PGNN with an $\mathbf{h}(\mathbf{x})$ injected at a hidden layer of the neural network, as shown in Fig. 4.4. By stacking known features into an intermediate layer, they can be utilised more effectively. The significance of which layer is used for injection is unknown.

Here, we briefly explain the architecture of the NN and PGNN. A neural network consists of several layers with a predefined number of neurons. Each neuron has a weighted connection to all neurons in the previous layer and a bias term, which amounts to an affine transformation:

$$\mathbf{z}^l = \mathbf{W}^l \boldsymbol{\chi}^{l-1} + \mathbf{b}^l \tag{4.7}$$

where $\boldsymbol{\chi}^{l-1}$ is the output of the $(l-1)^{\text{th}}$ layer, $\mathbf{W}^l$ is the matrix of weights representing the incoming connection strengths the l^{th} layer, and $\mathbf{b}^l$ is the bias vector. For notational simplicity we define $\boldsymbol{\chi}^0 = \mathbf{x}$. The transformed input is then passed through a node's activation function ζ, which is some nonlinear function. The introduction of this nonlinearity prevents the chain of affine transformations from simplifying and allows the neural network to learn highly complex relations between the input and output. The output of the l^{th} layer can be written as:

$$\boldsymbol{\chi}^l = \zeta(\mathbf{z}^l), \quad \boldsymbol{\chi}^0 = \mathbf{x} \tag{4.8}$$

where ζ is the activation function. Some possible choices are the ReLU, tanh, and sigmoid activation functions. We refer the reader to [60] for a complete overview. If there are L layers between the input and the output in a neural network, then the output of the neural network can be represented as follows:

$$\dot{\mathbf{x}} = \zeta_L\left(\mathbf{W}^L, \mathbf{b}^L, \ldots, \zeta_2\left(\mathbf{W}^2, \mathbf{b}^2, \zeta_1(\mathbf{W}^1, \mathbf{b}^1, \mathbf{x})\right)\right) \tag{4.9}$$

where $\mathbf{x}$ and $\dot{\mathbf{x}}$ are the independent and dependent variables of the system, respectively. The above equation can also be written as:

$$\dot{\mathbf{x}} = \zeta_L(\cdot;\boldsymbol{\theta}_L) \circ \cdots \circ \zeta_2(\cdot;\boldsymbol{\theta}_2) \circ \zeta_1(\mathbf{x};\boldsymbol{\theta}_1) \tag{4.10}$$

where θ represents the weights and biases of the corresponding neural network layer. For the PGNN framework, the information from the known part of the system is injected into an intermediate layer of the neural network as follows:

$$\dot{\mathbf{x}} = \zeta_L(\cdot; \theta_L) \circ \cdots \circ \underbrace{\mathcal{C}\left(\zeta_i(\cdot; \theta_i), h(\mathbf{x})\right)}_{\text{Known function injection}} \circ \cdots \circ \zeta_1(\mathbf{x}; \theta_1) \tag{4.11}$$

where $\mathcal{C}(\cdot, \cdot)$ represents the concatenation operation and the available information about the system, i.e. $h(\mathbf{x})$, is injected at ith layer. However, the choice of this layer is significant, and there is currently no way to know beforehand which layer will yield the best results. Therefore, we apply knowledge injection at each layer and compare the results.

4.2.2 Methodology

We performed experiments on five nonlinear dynamical systems (see Examples 2.1.4 to 2.1.8) to test the applicability of the PGNN approach. These systems were chosen because of the variety of nonlinear phenomena that they exhibit. For each system, suitable injection terms are identified. The same NN architecture (3 hidden layers) was used in all cases to reduce the number of experiments. The functions were then injected with the following configurations: no injection, injection in the first layer, second layer, and third layer. Then, ten models were trained on the data for each injection configuration to estimate the model uncertainty.

Dynamical systems and injection terms

The dynamical systems used for these experiments are presented in more detail in Examples 2.1.4 to 2.1.8.

The Lotka-Volterra system is presented in Example 2.1.4 and is also known as the “predator-prey model”. The equation’s only nonlinear term is xy, which we will use as an injection term.

The Duffing oscillator is shown in Example 2.1.5, exhibiting complicated oscillatory behaviour. The x^3 term is used for knowledge injection, and we also reuse the $\cos(\omega t)$ term to see if providing redundant features has any effect.

The Van der Pol system can be found in Example 2.1.6 and tends to a limit cycle. The x^2y term is relatively complex, and we select this for knowledge injection.

The Lorenz system (see Example 2.1.7) is a famous chaotic system with solutions that represent a butterfly (possibly the origin of the term “The Butterfly Effect”). The terms xy and xz are natural candidates for injection, although, for brevity, only xy is tested.

The Henon-Heiles system was initially developed to describe stellar motion around the centre of a galaxy. More details can be found in Example 2.1.8. This system has several features that can be injected. We try xy and y^2, with x^2 omitted because it appears in the equation similarly to y^2.

Data generation and pre-processing

For each of the five dynamical systems, we generated training data for the NNs and a test set to judge if the trained model can generalise to previously unseen states. A set of initial conditions x_0 was manually chosen for each system, and the corresponding trajectories were generated using the RKF45 solver with adaptive time-stepping until a final time T. We used the implementation in the SciPy software stack [133], which is based on the Dormand-Prince pair of formulas [200]. The resulting data were then interpolated to generate a regular time series with timestep h. The time derivative at each data point was estimated as the forward difference $\left(f(x^+) - f(x)\right)/h$. The datasets can then be described as a list of pairs $\mathcal{D} = \{(\mathbf{x}_k, \mathbf{y}_k)\}$, where $\mathbf{x}_k$ and $\mathbf{y}_k$ are the kth state and time derivative respectively. A validation set was constructed by reserving 20% of the data. The validation set is not used to train the models but to evaluate the models' generalisation performance on unseen data during training. The initial conditions and other parameters used for each system are provided in Table 4.1. The test trajectory was generated from the last initial condition for each system, as discussed in Section 4.2.3. The total numbers of training, validation, and test data for each system are given in Table 4.2.

Neural network architectures and training

The same network architecture was used in all cases for better comparison. The networks were given three hidden layers with 32, 64, and 32 neurons, respectively. This architecture was found to have a sufficiently high capacity to model all systems and is small enough to avoid overfitting. The injection was performed by concatenating the injection term to the selected hidden layer. Each model ensemble consisted of ten NNs, which yielded decent uncertainty estimates. The models were implemented in TensorFlow [64] and trained using the ADAM optimiser [82] with default parameters. The models were trained on batches of 32 samples at a time (this number is known as the batch size) for a total of 100 epochs. An epoch is the number of batch iterations after which the model will have trained on all data within the training set. We describe each batch as a set of indices $\mathcal{B} \subset \mathcal{N}$ that correspond to data in $\mathcal{D}$.

Since this is a regression problem, the mean-squared error (MSE) was utilised as a loss function. The loss for the training batch $\mathcal{B}$ is then:

Table 4.1: Parameters and initial conditions used to generate the datasets. The training set was constructed by simulating each system with the initial conditions (IC) shown below using Runge-Kutta-Fehlberg method (RKF45) with adaptive timestep until time T, and then estimating the pairs $(\mathbf{x}(t), \dot{\mathbf{x}}(t))$ at regular time intervals of length h. 20% of these pairs were reserved for the validation set. The test set was generated from a different initial condition, as shown below.

System	h	T	Test IC	Train & Val. IC
Lotka-Volterra	$0.05\,\mathrm{s}$	$200\,\mathrm{s}$	$(5, 1)$	$(2, 1)$ $(10, 1)$ $(12, 1)$ $(15, 1)$ $(20, 1)$ $(22, 1)$ $(25, 1)$
Duffing	$0.05\,\mathrm{s}$	$200\,\mathrm{s}$	$(1, 0.5)$	$(1, 1)$ $(0, 1)$ $($-$1, 1)$ $(1,$-$1)$ $(0,$-$1)$ $($-$1,$-$1)$
Van der Pol	$0.005\,\mathrm{s}$	$20\,\mathrm{s}$	$(2,$-$5)$	$(0, 6)$ $(0, -2)$ $($-$1, 2)$ $(1,$-$4)$ $(0, 0.1)$ $(1, 3)$ $($-$2, 5)$
Lorenz	$0.005\,\mathrm{s}$	$25\,\mathrm{s}$	$(1, 1,$-$5)$	$(1, 1, 1)$ $(5, 1, 1)$ $(1, 5, 1)$ $(1, 1, 5)$ $($-$5, 1, 1)$ $(1,$-$5, 1)$
Henon-Heiles	$0.05\,\mathrm{s}$	$100\,\mathrm{s}$	$($-$0.325, 0.4, 0, 0)$	$(0.1, 0.5, 0, 0)$ $(0.3, 0.4, 0, 0)$ $($-$0.35, 0.4, 0, 0)$ $(0.3,$-$0.1, 0, 0)$

Table 4.2: Size of the training and validation datasets for each system. The total number of data points can be computed from Table 4.1 as $T/h \times$ (number of ICs), and taking 20% of the total as the validation set.

System	Training data	Validation data	Test data
Lotka-Volterra	22400	5600	4000
Duffing	19200	4800	4000
Van der Pol	22400	5600	4000
Lorenz	24000	6000	5000
Henon-Heiles	6400	1600	2000

$$L_{MSE}(\mathcal{B};\theta) = \sum_{k\in\mathcal{B}} \|\mathbf{y}_k - \hat{f}(\mathbf{x}_k;\theta)\| \tag{4.12}$$

where $(\mathbf{x}_k, \mathbf{y}_k)$ is the kth pair in the dataset $\mathcal{D}$, as described in Section 4.2.2. Regularisation methods such as weight decay are used during training to prevent overfitting. We did not encounter any overfitting issues, so we omitted regularisation to reduce the number of comparisons.

Model evaluation

It was found that simply reporting the MSE on the test set did not clearly show how the models performed. Therefore, we chose to report the model performance as the MSE between a rolling forecast and the test trajectory, which we refer to as the rolling forecast mean squared error (RFMSE). A trajectory is generated from an initial condition during the rolling forecast stage. The forecast state at the next time step is predicted using a forward Euler step from the previous state, starting from the given initial condition. The timesteps showed in Table 4.1 were used. The resulting trajectories of the model ensemble were then compared to the actual trajectory of the system. We believe that reporting the RFMSE more accurately reflects the actual use case of these models and makes it easier to qualitatively see how knowledge injection can affect the predictive accuracy and model uncertainty within each model class.

4.2.3 Results and discussion

In this section, we report the performance of the ensembles in terms of their training/validation loss, as well as the RFMSE on the test trajectory (see Section 4.2.2). The model uncertainty within each model class is shown using 95% confidence bounds around the average predictions. We also report the mean training and validation loss for each ensemble. The loss signals of all models were smoothed using an exponential moving average filter using a weight of 0.2 before being averaged. The smoothing improves the clarity of the plots and allows us to compare overall trends between ensembles as well as the stability of the training. First, an overview of the results is presented, and then we provide a more detailed look at the best-performing injection term within each model class.

Overview of results

The RFMSE for each model class is visualised in Fig. 4.5. Note that the data has been normalised due to the different scales of each test set, such that a value of 1 represents the top performer for each system. Additionally, because the predictions of the models blow up in some cases, we compute the RFMSE on a shorter time interval: [25s, 70s, 2.5s, 2.5s, 15s] for the Lotka-Volterra, Duffing, Lorenz, Van der Pol, and Henon-Heiles systems respectively. For all systems, the best-performing ensemble on average was a PGNN, often by a significant margin. The choice of injection layer appears to be a significant factor, although, at this stage, the data shows no conclusive pattern. This result is surprising, as all of the nonlinear terms show up as additive terms in the equations, and there does not appear to be a good reason for the difference. Methods such as layer-wise relevance propagation [201] could be adopted to interpret the impact of the choice of layer for knowledge injection, and we consider it as part of our future work.

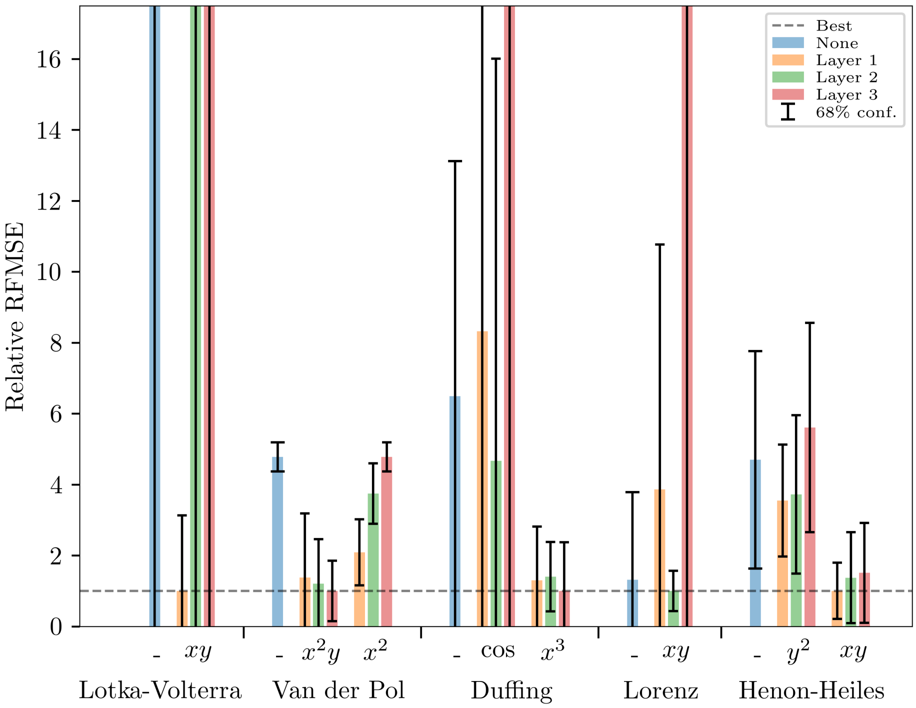
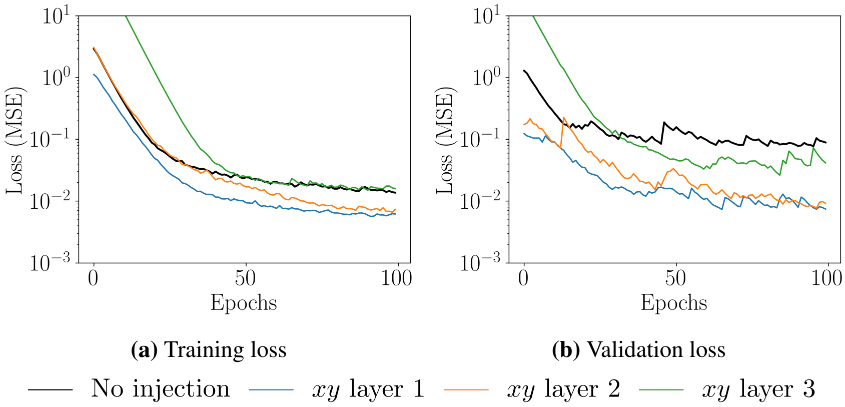


Figure 4.5: The relative RFMSE for all model ensembles across different systems. The values for each system have been divided by the minimum RFMSE in each group for better comparison, such that the top performers for each system have a value of 1. Note that in some cases, the rolling forecasts have diverged. Because of this, we do not compute the RFMSE on the full trajectory. Instead we use the final times [25s, 70s, 2.5s, 2.5s, 15s] for each system respectively. The 68% confidence intervals shown here were chosen to improve clarity while allowing for a comparison between models.

Lotka Volterra system

Fig. 4.6 shows that injecting the xy term in any layer caused the networks to reach a lower validation loss more quickly. This improvement was greatest when the injection was placed in the first hidden layer. Fig. 4.7 shows the mean rolling forecast of the ensembles with a 95% confidence interval. Injection in the first layer significantly improves the accuracy and uncertainty of the forecast. However, the forecast quickly blows up for the models with second and third-layer injections, with the latter being especially pronounced.



(a) Training loss

(b) Validation loss

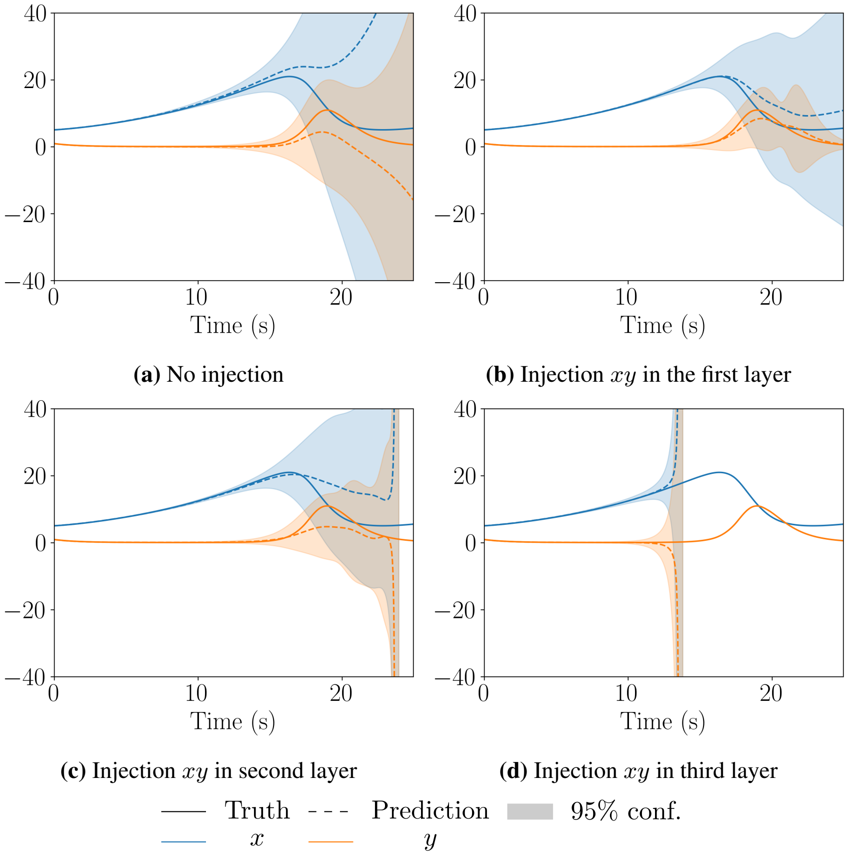
Figure 4.6: Comparison of the training and validation loss for the Lotka-Volterra system for different injection configurations.

Duffing system

Fig. 4.8 compares the training and validation loss of the models injected with x^3 and $\cos(\omega t)$ terms. Both training and validation loss are significantly improved with the x^3 injection, while $\cos(\omega t)$ appears to have little effect.

The predicted trajectories for x^3 models can be seen in Fig. 4.9, while the $\cos(\omega t)$ models have been omitted for brevity. Note that the figure shows a time segment from 75s–100s to highlight the differences between the models. We observe that knowledge injection improves the accuracy and model uncertainty in all cases, and all ensembles perform similarly.

It is interesting that although $\cos(\omega t)$ is available as an input to the network (due to the parameterisation described in Example 2.1.5), injecting the same term changes the RFMSE significantly, despite being redundant information. However, this is not reflected in the training and validation loss, where the baseline model and the models injected with $\cos(\omega t)$ appear to have nearly identical training characteristics. When forecasting over more extended periods, we found that the $\cos(\omega t)$ models



(a) No injection

(b) Injection xy in the first layer

(c) Injection xy in second layer

(d) Injection xy in third layer

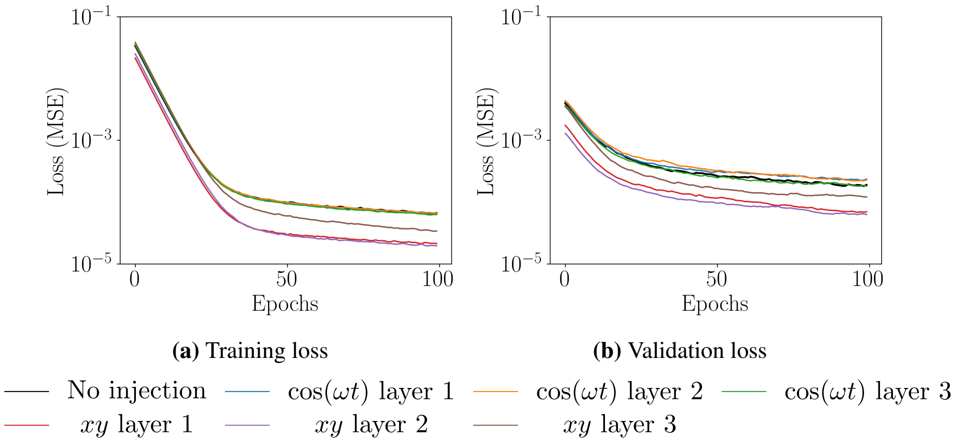
Figure 4.7: Rolling forecast for the Lotka-Volterra system with and without injection at different layers. The best results are achieved through knowledge injection in the first layer. Injecting into the second and third layers leads to more blowups.

yielded unstable predictions with a higher blow-up rate. The other models tended to decay instead.

Van der Pol system

For this system, the functions x^2y and x^2 were injected in all three layers. Fig. 4.10 shows how the x^2y injected models improve validation loss by 1-2 orders of magnitude better than the model without injection.

This improvement can be understood by inspecting Fig. 4.11, which shows a rolling forecast on the test trajectory near a fast transient. The figure shows that the models without injection fail to capture the transient and converge to the slow dynamics



(a) Training loss

(b) Validation loss

Figure 4.8: Comparison of the training loss for the Duffing oscillator for different injection configurations.

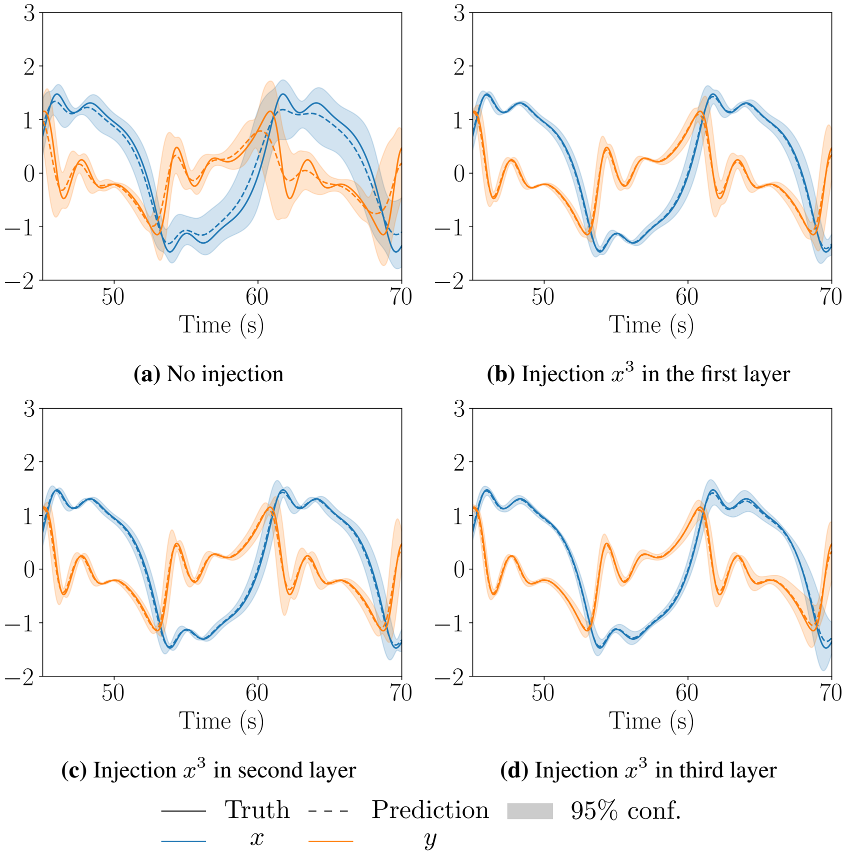
afterwards. The long duration of the slow dynamics likely leads to a significant build-up of error throughout the trajectory.

Surprisingly, the ensemble without knowledge injection exhibits very low model uncertainty. The fast and slow dynamics of the Van der Pol oscillator might again explain this. The dataset is likely unbalanced, dominated by slower varying states due to the longer duration of the slow dynamics. This imbalance could cause poor performance on the fast transients, which can be seen as relatively rare events.

Fig. 4.11 shows that this is greatly improved through knowledge injection. All injected models track the transient more closely, reach the correct value for the slow dynamics, and the actual trajectory is within the 95% confidence intervals for all model classes. The model uncertainty naturally increases at the transient and appears to shrink to zero when the slow dynamics begin.

Lorenz System

The function xy was used for knowledge injection. Fig. 4.12 clearly shows that knowledge injection improved the training convergence and final validation loss over 100 epochs. Fig. 4.13 shows that the ensemble without injection stays close to the actual trajectory but oscillates out of phase with the ground truth. Injection in the first and second layers seems to reduce this lag, and the second layer injection performs marginally better with lower model uncertainty. The predictions from the ensemble with third-layer injections quickly diverge.



(a) No injection

(b) Injection x^3 in the first layer

(c) Injection x^3 in second layer

(d) Injection x^3 in third layer

Figure 4.9: Rolling forecast for the Duffing oscillator with and without injection at different layers. A small time segment from 45s–70s from the forecast is shown here to highlight the differences between the models.

Henon–Heiles system

All models performed similarly for this system, as shown for the xy injection in Fig. 4.15. Fig. 4.14 shows that the injected models converge slightly faster and reach an overall lower validation loss.

4.2.4 Discussion

The physics-guided neural network PGNN framework was applied to a set of five different dynamical systems represented by first and second-order non-linear ordinary differential equations. Three systems had two suitable injection terms, which were also investigated for eight system and injection terms combinations.

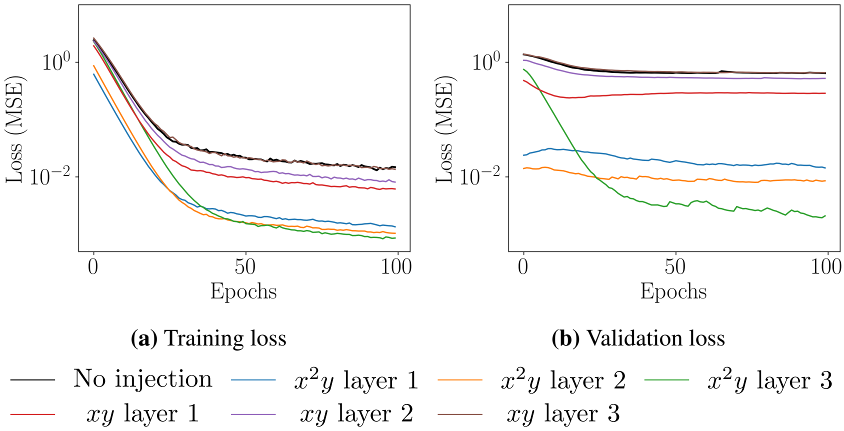
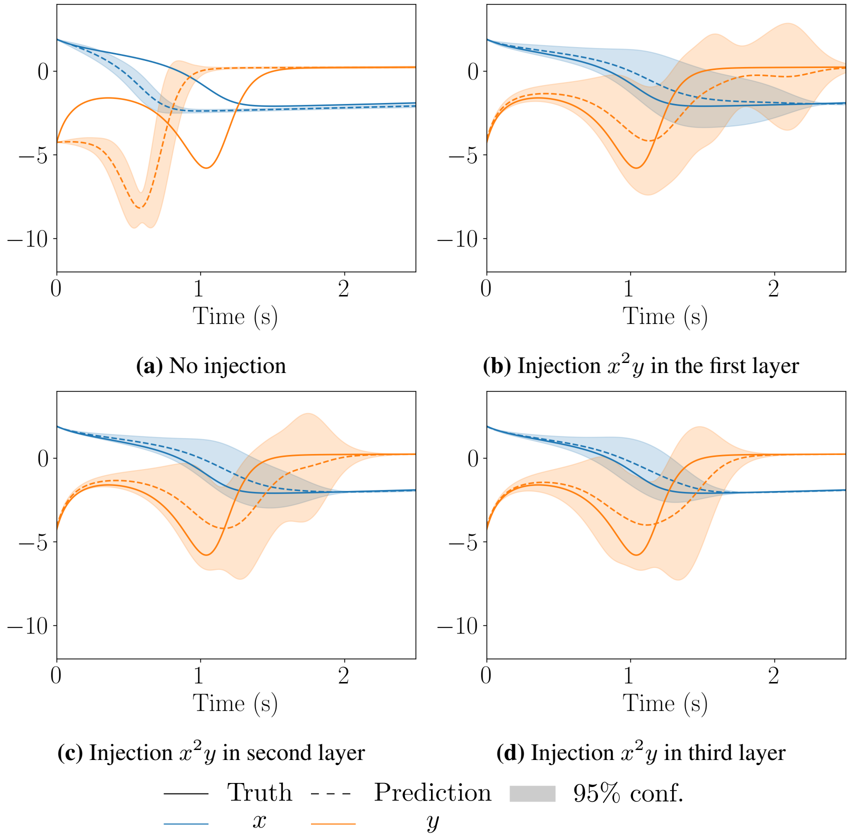


Figure 4.10: Comparison of the training and validation loss for the Van der Pol oscillator for different injection configurations.

All possible injection layers were evaluated and compared. The main conclusions from the work are as follows:

- Knowledge injection can accelerate training and lead to better convergence. However, knowledge injection does not guarantee performance improvement.
- Accuracy of the models can generally be improved through knowledge injection. Knowledge injection helped the models capture the fast transients for the Van der Pol system, which were relatively underrepresented in the dataset. However, the improvements were insignificant for some problems, e.g. the Henon-Heiles system.
- The study remains inconclusive regarding the impact of knowledge injection on model uncertainty. We see a shrinkage in the uncertainty for the Duffing oscillator, but for Van der Pol, we see an increase. However, it should be stressed that lower uncertainty with poor prediction would not be desirable.

The first limitation of this work is that the choice of injection layer was found to impact performance significantly, and the results do not show how such a choice could be made a priori. A second limitation is that we do not propose a method to identify relevant information to inject into the network's hidden layers. Instead, we tried all combinations of injection layers and terms and selected the best performers. However, this will scale poorly for network architectures with more hidden layers or multiple injection terms. Hyperparameter search algorithms (for example, genetic



(a) No injection

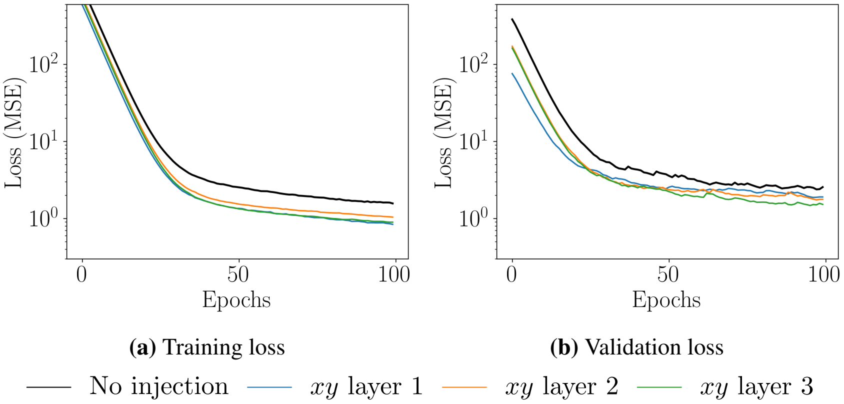
(b) Injection x^2y in the first layer

(c) Injection x^2y in second layer

(d) Injection x^2y in third layer

Figure 4.11: Rolling forecast for the Van der Pol oscillator with and without injection at different layers.

algorithms due to the discrete optimisation variable) may be helpful tools for choosing an effective injection layer. However, this still involves training multiple models. A more straightforward solution to both problems is to make all injection features available to all layers via skip connections. However, preliminary results in this direction have shown that this does not reach the same level of performance as the best single injection layer. Combining this approach with sparsifying regularisation may bias the network towards selecting the best injection layer and term. The role of the injection layer could be elucidated by testing the PGNN approach with deeper architectures and recurrent NNs. Another method that could help identify suitable injection terms is symbolic regression based on gene expression programming. By running a large ensemble on the data and selecting the most frequently appearing terms, it might be possible to collect good injection terms.



(a) Training loss

(b) Validation loss

Figure 4.12: Comparison of the training and validation loss for the Lorenz system for different injection configurations.

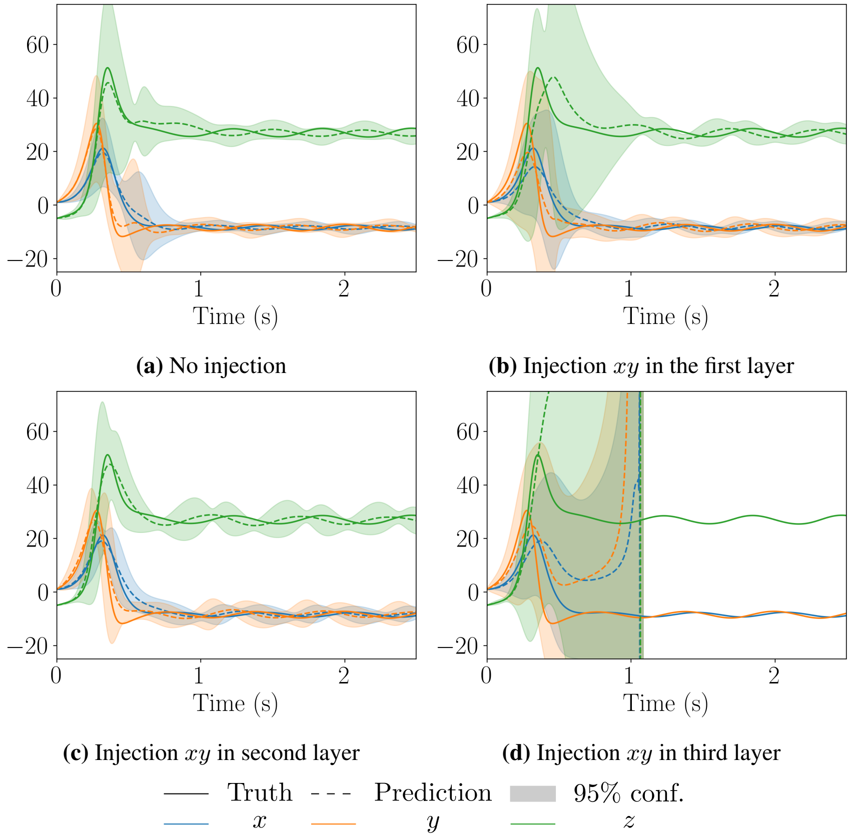
Extensions of PGNNs may help model more complex systems with rich dynamics and environmental interactions. Significant assumptions are typically made about the nature of environmental forces, and practitioners often defer to the data, e.g. when modelling the average wind force on marine vessels [37, 202, 203]. There is already much work where more advanced ML techniques such as RL are applied to these systems [30]. The improved training characteristics and low overhead of PGNNs may prove helpful in these RL contexts, where data efficiency is relatively poor. Furthermore, the learned weighting of explicit features arguably makes the models more interpretable, often desirable in safety-critical contexts. However, more work is needed in this direction. PGNNs may also require fewer parameters than conventional DNNs to model the same data, which could enable the use of existing robustness verification algorithms [204], vastly improving confidence in these systems during deployment.

The attractiveness of prior knowledge injection is that it generalises two of the most common hybridisation methods: input feature engineering and output error correction. By injecting arbitrary features into a neural network's intermediate layers, we open the proverbial black box and recognise its potential as a general-purpose feature learner and feature selector powered by stochastic optimisation.

4.3 Corrective source-term approach

In this work, we extend and apply CoSTA to correct a misspecified PBM of a complex aluminium extraction process simulation. The main contributions are:

- An extension of CoSTA to multidimensional problems: The previous works



(a) No injection

(b) Injection xy in the first layer

(c) Injection xy in second layer

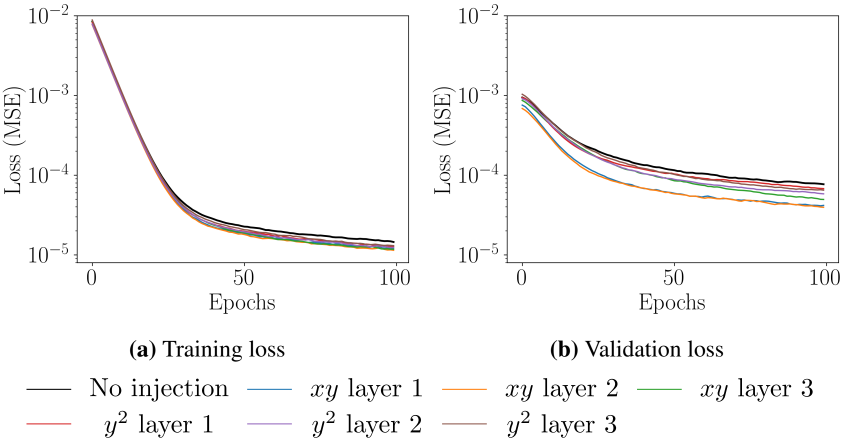
(d) Injection xy in third layer

Figure 4.13: Rolling forecast for the Lorenz system without and with injection at different layers. Here the injection is most effective at the second layer, while the third layer injection causes the predictions to blow up.

utilising CoSTA were limited to modelling a single state temperature in either one or two-dimensional heat transfer. The current application of the aluminium extraction process involves eight states.

- A successful application of CoSTA to a system with external control inputs: None of the previous work involved any control inputs. In the current work, five inputs are used to excite the system.

- A successful application of CoSTA to a system with complex coupling between different states and inputs: The complex system considered here involves eight states and five inputs which form a set of eight ordinary equations which are highly coupled. The previous works involving heat



(a) Training loss

(b) Validation loss

Figure 4.14: Comparison of the training and validation loss for the Henon Heiles system with different injection configurations.

transfer involved only one partial differential equation hence the potential of CoSTA to couple problems was never evaluated earlier.

This section is structured as follows. Section 4.3.2 presents an ablated version of the aluminium electrolysis plant from Example 2.1.3. Section 4.3.1 We then outline the methodology of the work in Section 4.3.3, namely how the data was generated, how the models were trained, and how they were evaluated. In Section 4.3.4, we present the results and give a detailed discussion about the behaviour of the process and the models. We then summarise our findings and outline future work in Section 4.3.5.

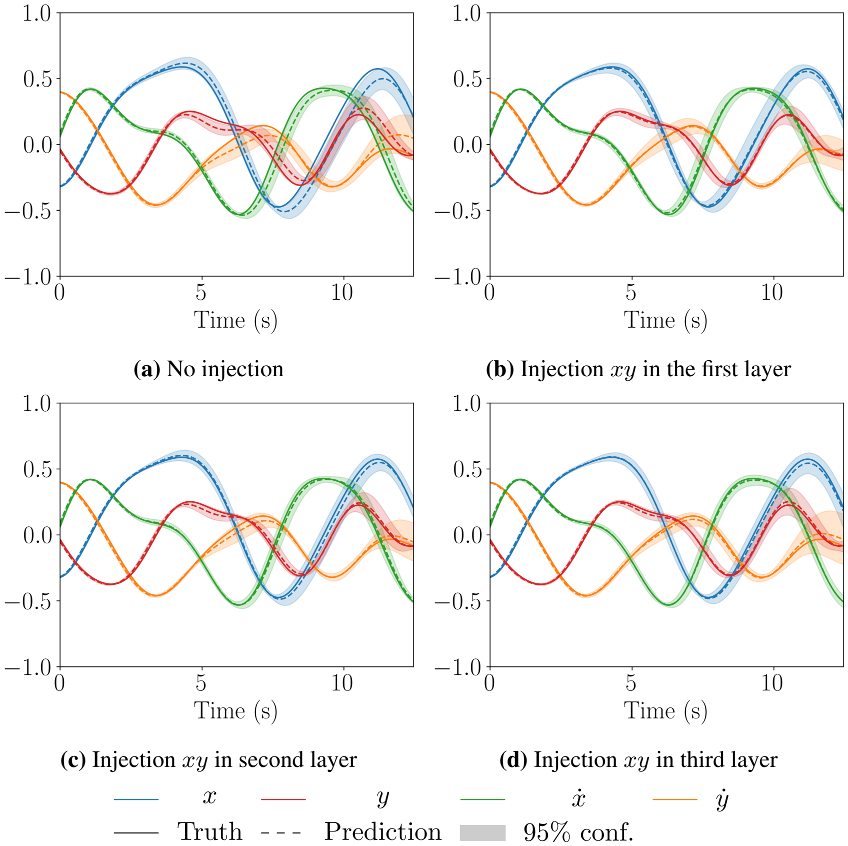
In order to investigate the applicability of CoSTA to engineering applications, we perform a case study on an aluminium extraction process using the Hall-Héroult process. In the following sections, we describe the underlying PBM for this system, the fundamentals of NNs, and the CoSTA approach to HAM.

4.3.1 Corrective source term approach

This section outlines the CoSTA approach, illustrated in Fig. 4.16. Suppose we want to solve the following general problem:

$$\mathcal{L}\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{4.13}$$

where $\mathcal{L}$ is a differential operator, $\mathbf{x}$ is the unknown state of the system that we wish to compute, and $\mathbf{f}(\cdot, \cdot)$ is a source term that depends on the state $\mathbf{x}$ and external inputs $\mathbf{u}(t)$.



(a) No injection

(b) Injection xy in the first layer

(c) Injection xy in second layer

(d) Injection xy in third layer

Figure 4.15: Rolling forecast for the Henon-Heiles system without and with xy injection at different layers. Injecting functions appear to aid the learning of this system significantly. The best results are achieved by into the last hidden layer.

Assume now that we have a PBM designed to predict $\mathbf{x}$, and let $\tilde{\mathbf{x}}$ denote the PBMs prediction of the actual solution $\mathbf{x}$. If $\tilde{\mathbf{x}} \neq \mathbf{x}$, there is some error in the PBM, and this error must stem from at least one of the following misspecifications in the model:

1. Incorrect $\mathbf{f}$ in Eq. (4.13), replaced by $\tilde{\mathbf{f}}$.
2. Incorrect $\mathcal{L}$ in Eq. (4.13), replaced by $\widetilde{\mathcal{L}}$.
3. A combination of the above.
4. Discretisation of $\mathcal{L}$, replaced by $\mathcal{L}_{\mathcal{D}}$[1].

[1] Derived using, for example, finite differences. This step is necessary when Eq. (4.13) lacks

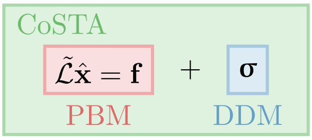


Figure 4.16: CoSTA combines PBM and DDM into a unified model by adding a NN -generated corrective source term to the governing equation of the PBM.

Note that case 4 is also mathematically equivalent to misspecifying $\mathcal{L}$. For example, $\frac{\partial}{\partial t}$ could be approximated using a forward finite difference. We can write this using the difference operator Δ_h, such that h is the time step and $\frac{1}{h}\Delta_h f(t) = (f(t+h) - f(t))/h$. We can therefore limit our discussion to Cases 1 and 2 without loss of generality.

Suppose now that the PBM -predicted solution $\tilde{\mathbf{x}}$ is given as the solution of the following system:

$$\tilde{\mathcal{L}}\tilde{\mathbf{x}} = \tilde{\mathbf{f}} \tag{4.14}$$

This formulation encompasses both Case 1 ($\tilde{\mathcal{L}} = \mathcal{L}$ and $\tilde{\mathbf{f}} \neq \mathbf{f}$), Case 2 ($\tilde{\mathcal{L}} \neq \mathcal{L}$ and $\tilde{\mathbf{f}} = \mathbf{f}$), and combinations thereof (for $\tilde{\mathcal{L}} \neq \mathcal{L}$ and $\tilde{\mathbf{f}} \neq \mathbf{f}$). Furthermore, suppose we modify the system above by adding a source term $\hat{\boldsymbol{\sigma}}$ to Eq. (4.14) and let the solution of the modified system be denoted $\hat{\hat{\mathbf{x}}}$. Then, the modified system reads

$$\tilde{\mathcal{L}}\hat{\hat{\mathbf{x}}} = \tilde{\mathbf{f}} + \hat{\boldsymbol{\sigma}} \tag{4.15}$$

and the following theorem holds.

Theorem 4.3.1. Let $\hat{\hat{\mathbf{x}}}$ be a solution of Eq. (4.15), and let $\mathbf{x}$ be a solution of Eq. (4.13). Then, for both operators $\tilde{\mathcal{L}}$, $\mathcal{L}$ and both functions $\mathbf{f}$, $\tilde{\mathbf{f}}$, such that $\hat{\hat{\mathbf{x}}}$ and $\mathbf{x}$ are uniquely defined, there exists a function $\boldsymbol{\sigma}$ such that $\hat{\hat{\mathbf{x}}} = \mathbf{x}$.

Proof. Define the residual $\boldsymbol{\sigma}$ of the PBMs governing Eq. (4.14) as:

$$\boldsymbol{\sigma} = \tilde{\mathcal{L}}\mathbf{x} - \tilde{\mathbf{f}}. \tag{4.16}$$

Instead of defining the residual in terms of the approximate solution (e.g. as is done in truncation error analysis [205]), we define $\boldsymbol{\sigma}$ by inserting the solution into

analytical solutions, which is almost always the case.

Eq. (4.13). If we set $\hat{\boldsymbol{\sigma}} = \boldsymbol{\sigma}$ in Eq. (4.15), we then obtain:

$$\widetilde{\mathcal{L}}\hat{\hat{\mathbf{x}}} = \tilde{\mathbf{f}} + \hat{\boldsymbol{\sigma}} \tag{4.17}$$

$$= \tilde{\mathbf{f}} + \widetilde{\mathcal{L}}\mathbf{x} - \tilde{\mathbf{f}} \tag{4.18}$$

$$= \widetilde{\mathcal{L}}\mathbf{x} \tag{4.19}$$

$$\implies \quad \hat{\hat{\mathbf{x}}} = \mathbf{x} + \mathbf{c} \tag{4.20}$$

where $\mathbf{c}$ is a function of independent variables. We can eliminate $\mathbf{c}$ by setting appropriate boundary conditions. □

This proof shows that we can always find a corrective source term $\hat{\boldsymbol{\sigma}}$ that compensates for any error in the PBMs governing Eq. (4.14) such that the solution $\hat{\hat{\mathbf{x}}}$ of the modified governing Eq. (4.15) is equal to the actual solution $\mathbf{x}$. This observation is the principal theoretical justification of CoSTA.

4.3.2 Ablated physics-based model for aluminium extraction

We test the approach using the aluminium electrolysis system presented in Example 2.1.3. As previously discussed, we are interested in modelling scenarios where the PBM does not capture the complete underlying physics of the system. This case is illustrated in Fig. 4.2(a), where the black background represents unobservable physical phenomena not adequately explained by available theory. The orange ellipse represents physics ignored due to assumptions. The red ellipse corresponds to resolved physics after solving PBM numerically, while the blue ellipse corresponds to the modelled physics.

The model presented in Eq. (2.11) makes some simplifications compared to the actual process of aluminium electrolysis. Firstly, we only model the heat transfer through the side walls, assuming that the heat flow through the top and bottom of the plant is negligible in comparison. The model may thus overestimate the internal temperatures, and the required power input through the line current u_2 may be slightly lower than in practice. Secondly, the spatial variations of the state variables are not considered. Instead, only the average values of the states, such as the side ledge temperature, or cumulative values, such as the mass of the side ledge x_1, are computed. Routine operations such as the alumina feeding and anode replacement disturb the local thermal balance and cause local thermal imbalances [206]. Modelling these local variations would require knowledge of, e.g. the mass transfer inside the cell due to the flow patterns, velocity fields in the bath, and current distribution. These phenomena (corresponding to the orange ellipse of Fig. 4.2(a)) are challenging to model and measure and are therefore omitted to reduce complexity.

For this case study, we use simulation data generated from Eq. (2.11) to validate the CoSTA method. To that end, we simplify further: we ignore Eq. (2.12a) and set the liquidus temperature g_1 to a constant.

$$g_{1,PBM} = 968°C \tag{4.21}$$

We refer to the resulting model as the *ablated PBM* . This choice was made because the model is sensitive to errors in g_1. Inspecting Eq. (2.11) shows that the ablated PBM will incorrectly predict the evolution of $[x_1, x_4, x_6, x_7, x_8]$. As we will see later in Section 4.3.4 and Fig. 4.20, this can lead to errors of roughly $5°$C in g_1, and 500kg in the side ledge mass x_1 (a relative error of 10%). The case study aims to develop a DDM to correct the ablated PBM using measurement data sampled from the actual model.

4.3.3 Method and experimental setup

In this section, we explain how we generated the data, how the data was divided into training, validation and test sets, and how the models were evaluated.

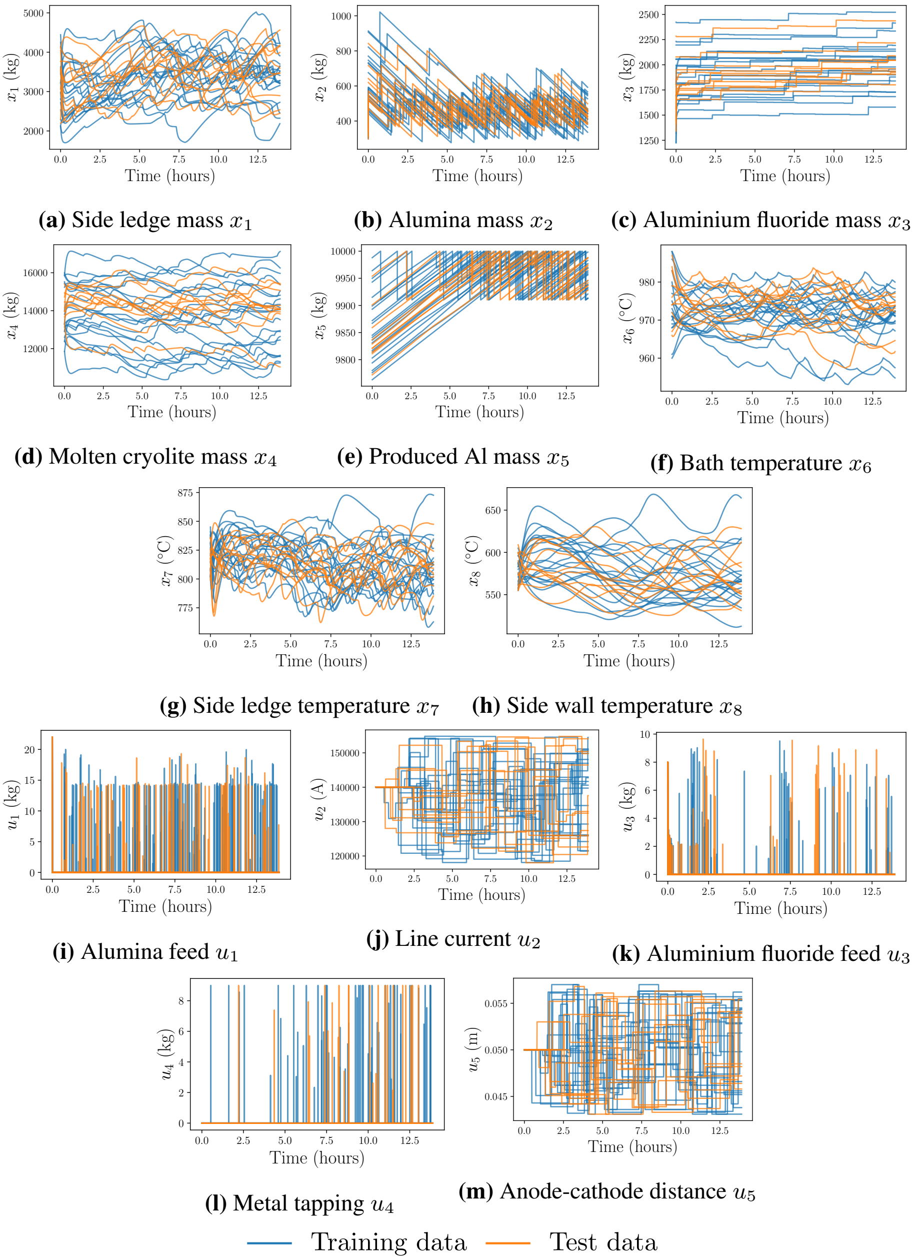
Data generation and preprocessing

The dynamical system data is generated by integrating the set of non-linear ODEs in Eq. (2.11) representing the system dynamics using the fourth-order numerical integrator Runge-Kutta 4 (RK4) with a fixed timestep $\Delta T = 10s$. One time-series simulation starts at an initial time t_0 with a set of initial conditions $\mathbf{x}(t_0)$ and last until a final time $T = 5000 \times \Delta T$. For the slow dynamics of the aluminium process, a sampling time of $10s$ turns out to be sufficiently fast with negligible integration errors. Higher sampling frequencies would lead to unnecessary high computational time and large amounts of simulation data. The initial conditions for each trajectory were uniformly sampled from the ranges shown in Table 4.3. Each simulation generates a set of trajectories with 8 states and 5 inputs. Forty simulated trajectories are used for training the models, and 100 simulated trajectories are used as the test set. This relatively large number of test cases was chosen to allow us to explore the statistics of how the model performs.

Fig. 4.17 shows the time series evolution of the entire training set and test set. The training set trajectories are blue, while the test set trajectories are orange. The figures show that the ranges of the training and test sets are similar, indicating that models are evaluated on interpolation cases in the test set.

Estimation of the time derivative

The ODEs in Eq. (2.11) are time-invariant. This means that at time $k + 1$, $\dot{\mathbf{x}}_{k+1}$ in general only depends on the current state and input $(\mathbf{x}_k, \mathbf{u}_k)$ at time k. In other



(a) Side ledge mass x_1

(b) Alumina mass x_2

(c) Aluminium fluoride mass x_3

(d) Molten cryolite mass x_4

(e) Produced Al mass x_5

(f) Bath temperature x_6

(g) Side ledge temperature x_7

(h) Side wall temperature x_8

(i) Alumina feed u_1

(j) Line current u_2

(k) Aluminium fluoride feed u_3

(l) Metal tapping u_4

(m) Anode-cathode distance u_5

Figure 4.17: Training and test set trajectories of the system states. Only ten random sample test trajectories are shown here for clarity.

words, the system has the Markov property. Therefore, the datasets are listed in pairs $\mathcal{D} = \{(\mathbf{x}_k, \mathbf{y}_k)\} = \{(\mathbf{x}_k, \mathbf{u}_k), \dot{\mathbf{x}}_k\}$. The Markov assumption does not always hold in practice, and the state vector must therefore be augmented with additional information, i.e. lookback states from previous time steps. Takens' Theorem gives an upper bound on the number of necessary lookback states [207]. The time derivatives at time k are estimated as the forward difference $\dot{\mathbf{x}}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k)/h$, where h is the time step. A value of $h = 10\text{s}$ is used. This numerical derivative induces a discretisation error. However, since the dynamics of the aluminium electrolysis is slow, this error is considered negligible. Because the systems are driven by an input signal $\mathbf{u}$, we must choose a value for $\mathbf{u}_k$ at each time step. This choice will significantly affect the variation in the dataset.

Table 4.3: Initial conditions for system variables. For x_2 and x_3, concentrations c_{x_2} and c_{x_3} are given.

Variable	Initial condition interval
x_1	$[2060,\ 4460]$
c_{x_2}	$[0.02,\ 0.05]$
c_{x_3}	$[0.09,\ 0.13]$
x_4	$[11500,\ 16000]$
x_5	$[9550,\ 10600]$
x_6	$[940,\ 990]$
x_7	$[790,\ 850]$
x_8	$[555,\ 610]$

Input signal generation

While machine learning models are beneficial for function approximation and interpolating data, they do not always extrapolate correctly. The quality and variety of the training data are significant factors and must therefore cover the intended operational space of the system. Here, the operational space means the region of the state space in which the system operates, meaning state and input vectors $[\mathbf{x}^\top, \mathbf{u}^\top]^\top$ observed over time. The data should capture the different nonlinear trends of the system covered by the operational space. For systems *without exogenous inputs*, variation can only be induced by simulating the system with different initial conditions $\mathbf{x}(t_0)$. The initial conditions are generated similarly for systems *with exogenous inputs*. Moreover, the input vector $\mathbf{u}$ will excite the system dynamics. The aluminium process has a feedback controller that ensures safe and prescribed operation. However, operational data from a controlled, stable process is generally characterised by a low degree of variation, which is insufficient for effective system identification. A well-known convergence criterion for identifying linear time-invariant systems is persistency of excitation (PE). A signal $\mathbf{x}(t_k)$ is PE of order L if all sub-sequences $[\mathbf{x}(t_k), \ldots, \mathbf{x}(t_k + L)]$ span the space of all possible sub-sequences of length L that the system is capable of generating. While the PE criterion is not directly applicable to nonlinear systems, sufficient coverage of the dynamics is required for successful system identification [4, 208].

We add random perturbations to the control inputs to push the system out of its standard operating conditions. In general, each control input i is given by:

$$u_i = \text{Deterministic term + Random term} \tag{4.22}$$

The control inputs u_1, u_3 and u_4 are impulses. The random term is zero for these control inputs when the deterministic term is zero. The deterministic term is a proportional controller. The control inputs u_2 and u_5 are always nonzero. These control inputs have constant deterministic and random terms that change periodically. The random term stays constant for ΔT_{rand} seconds before changing to a new randomly determined constant.

Choosing the period ΔT_{rand} balances different objectives. On the one hand, increasing the period ΔT_{rand} is desirable to stabilise and evolve to reveal the system dynamics under the given conditions. On the other hand, it is desirable to test the systems under many different operational conditions. By empirically testing different periods ΔT_{rand} and seeing how the dynamics evolve in simulation, it turns out that setting $\Delta T_{rand} = 30\Delta T$ is a fair compromise between the two. In this study, we generate the random disturbances using the Amplitude-modulated Pseudo-Random Binary Signal (APRBS) method [209]. Table 4.4 gives the numerical values of the deterministic term of the control input, the interval of values for the random terms, and the duration ΔT_{rand} of how long the random term is constant before either becoming zero (u_1, u_3, u_4) or changing to a new randomly chosen value (u_2, u_5).

Table 4.4: Equations used to control the aluminium process

Input	Deterministic term	Random term interval	ΔT_{rand}
u_1	$3 \cdot 10^4(0.023$ - $c_{x_2})$	[-2.0, 2.0]	ΔT
u_2	$1.4 \cdot 10^4$	[-$7 \cdot 10^3, 7 \cdot 10^3$]	$30 \cdot \Delta T$
u_3	$1.3 \cdot 10^4(0.105$ - $c_{x_3})$	[-0.5, 0.5]	ΔT
u_4	$2(x_5$ - $10^4)$	[-2.0, 2.0]	ΔT
u_5	0.05	[-0.015, 0.015]	$30 \cdot \Delta T$

Training

The models were trained on the training set using the total-loss function shown in Eq. (4.1), where the loss function $L(\cdot,\cdot)$ is the MSE as shown in Eq. (4.2). Four different model types were compared:

- Dense NN
- Sparse NN
- PBM + Dense NN
- PBM + Sparse NN

The dense networks were trained with $\lambda = 0$, and sparse networks with $\lambda = 10^{-4}$. The architecture of all networks was $[13, 20, 20, 20, 20, 8]$ (13 inputs, eight outputs, four hidden layers with 20 neurons each) . The `ReLU` activation function was used for all layers except the output layer, which had no activation function. The same architecture was used for all networks for a fairer comparison. All models were trained for 100 epochs (an epoch is defined as one complete pass over the dataset). The ADAM optimiser [82] was used with the following default parameters: Initial learning rate $\eta = 10^3$, Gradient forgetting factor $\beta_1 = 0.9$, and Gradient second-moment forgetting factor $\beta_2 = 0.999$.

Performance metrics

This work will focus on long-term forecast error as a performance measure. The initial condition $\mathbf{x}(t_0)$ is given to the models. Then the consecutive n time steps of the states are estimated $\{\hat{\mathbf{x}}(t_1),\ ...,\ \hat{\mathbf{x}}(t_n)\}$, refered to as a *rolling forecast*. The model estimates the time derivatives of the states $d\hat{\mathbf{x}}_i/dt$ based on the current state $\mathbf{x}(t_i)$ and control inputs $\mathbf{u}(t_i)$ and initial conditions $\mathbf{x}_0 = \mathbf{x}(t_0)$, or the estimate of the current state variables $\hat{\mathbf{x}}(t_i)$ if $t > t_0$:

$$\frac{d\hat{\mathbf{x}}(t_i)}{dt} = \begin{cases} \mathcal{N}\left(\hat{\mathbf{x}}(t_i),\ \mathbf{u}(t_i)\right), & \text{if } t_i > t_0 \\ \mathcal{N}\left(\mathbf{x}_0(t_i),\ \mathbf{u}(t_i)\right), & \text{if } t_i = t_0 \end{cases} \tag{4.23}$$

Then, the next state estimate $\mathbf{x}(t_{i+1})$ is calculated as:

$$\hat{\mathbf{x}}(t_{i+1}) = \hat{\mathbf{x}}(t_i) + \frac{d\hat{\mathbf{x}}(t_i)}{dt} \cdot \Delta T \tag{4.24}$$

The rolling forecast can be computed for each state x_i for one set of test trajectories $\mathcal{S}_{test}$. However, presenting the rolling forecast of multiple test sets would render the interpretation difficult. By introducing a measure called Average Normalised Rolling Forecast Mean Squared Error (AN-RFMSE) that compresses the information about model performance, the models can quickly be evaluated on many test examples. The AN-RFMSE is a scalar defined as:

$$\text{AN-RFMSE} = \frac{1}{p}\sum_{i=1}^{p}\frac{1}{n}\sum_{j=1}^{n}\left(\frac{\hat{x}_i(t_j) - x_i(t_j)}{\mathrm{std}(x_i)}\right)^2, \tag{4.25}$$

where $\hat{x}_i(t_j)$ is the model estimate of the simulated state variable x_i at time step t_j, $\mathrm{std}(x_i)$ is the standard deviation of variable x_i in the training set $\mathcal{S}_{train}$, $p = 8$ is the number of state variables and n is the number of time steps the normalised rolling forecast MSE is averaged over. Hence, for every model $\mathcal{N}_j$ and every test set time series $\mathcal{S}_{test}(i)$, there is a corresponding AN-RFMSE.

4.3.4 Results and discussion

Ten instances of the 4 model types were trained on the same dataset for uncertainty quantification. Only one instance of the ablated PBM was used as defined in Section 4.3.3. All model instances were evaluated on 100 test trajectories, yielding 4100 data points. Some of the model forecasts were found to blow up. We set a threshold where a blow-up is defined as when the final predicted state's normalised MSE exceeds 3. Fig. 4.18 shows a violin plot of the AN-RFMSE for all model types, without the blow-ups. The AN-RFMSE is shown at three different times to demonstrate all model types' short-term, medium-term, and long-term performance. Fig. 4.19 shows the frequency of blow-ups for each model type. These results show that on average, all DDM and CoSTA models have a lower RFMSE than the ablated PBM in the short and medium term. However, we still observe that all DDM and CoSTA models experience some blow-ups in the long term, which the PBM model does not. The dense DDM fared the worst, as 27.3% of the forecasts were found to result in blow-ups in the long term. The sparse DDM marginally improves on the RFMSE, but we found that the blow-up rate was significantly reduced in the long term compared to the dense DDM. Both dense and sparse CoSTA models were significantly more accurate than the DDM models. The sparse CoSTA had similar accuracy to the dense CoSTA models in the short and medium term. However, the sparse CoSTA model had no blow-ups in the short and medium term and had half the blow-up rate of the Sparse DDM in the long term. These experiments demonstrate that CoSTA can reliably correct misspecified PBMs and improves predictive stability compared to end-to-end learning. The base PBM does not exhibit any blow-up issues, implying that the blow-ups can be attributed to using NNs. If long-term forecasts are required (> 3000 timesteps), we recommend

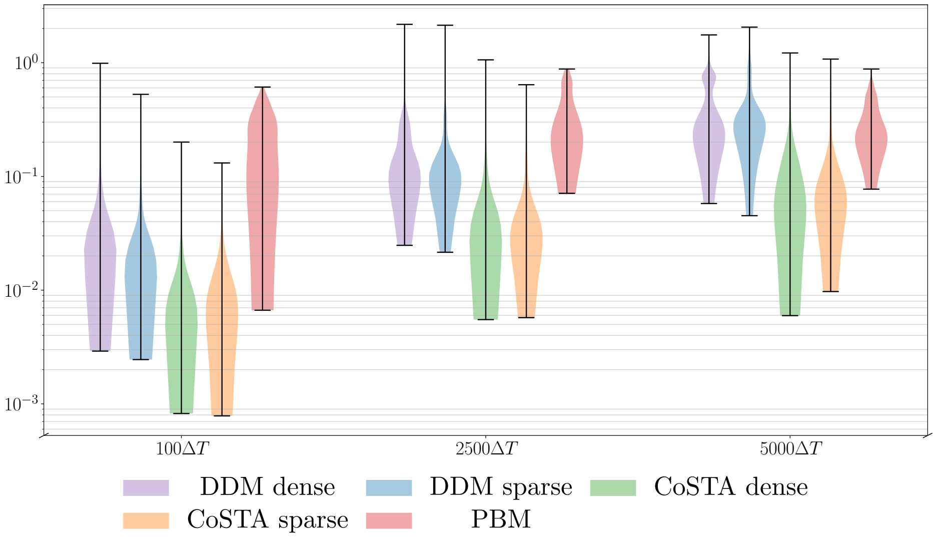


Figure 4.18: Violin-plot of the AN-RFMSE for all model types for 100 different initial conditions and inputs signals. The width of the bar reflects the distribution of the data points, and the error bars represent the range of the data. The error is shown after three different times to compare the short, medium, and long-term performance. We trained ten different instances for each model type for statistical significance. We see that CoSTA improves the predictive accuracy over the whole trajectory. Introducing sparse regularisation appears to improve performance for DDM. However, it only appears to affect CoSTA models in the long term, where sparse CoSTA appears to have less variance.

combining the CoSTA approach with a sanity check mechanism to detect potential blow-ups.

Fig. 4.20 shows the mean predictions for each model type for a representative test trajectory, along with a 99.7% confidence interval to show the spread of the predictions from the ten instances of each model type. For better clarity, only the sparse models are shown due to their superior performance compared to their dense counterparts. Before discussing the differences between the models, we will describe the system's dynamics and how the incorrect PBM behaves in comparison.

First, note that all variables are non-negative, as they reflect different physical quantities in the system, i.e. mass, temperature, and current. Inspecting Eq. (2.11), we see that the states x_2, x_3, and x_5 are linearly dependent on u_1, u_2, u_3, and u_4. We refer to these as the *linear states*, and the rest as the *nonlinear states*.

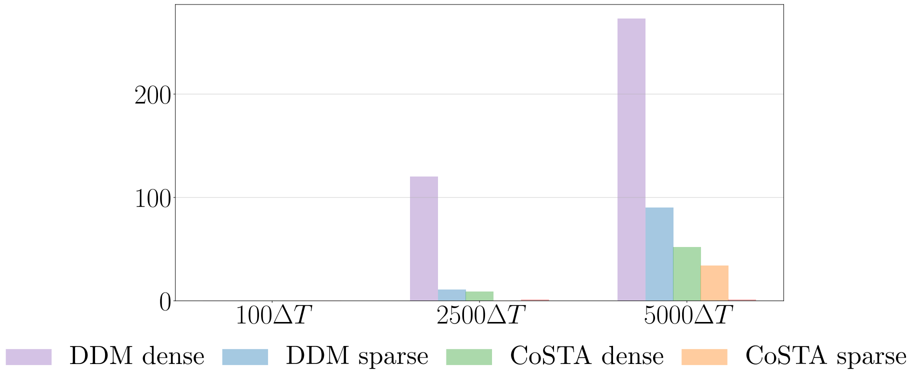


Figure 4.19: Bar chart of the number of times model estimates blow up and diverges. The plot contains 100 initial conditions and input signals for all model types. The number of blow-ups was counted after three different times to compare the performance in the short, medium, and long term. We trained ten different instances for each model type for statistical significance. We see that applying CoSTA can increase the predictive stability in the long term. That is, the number of blow-ups for CoSTA models is far less than for DDM. However, PBM does not suffer from significantly fewer blow-ups than CoSTA.

Liquidus temperature

Fig. 4.20(i) shows the true liquidus temperature g_1 (in black) and the constant PBM estimate of the liquidus temperature (in red dotted line). The liquidus temperature g_1, which is the temperature at which the bath solidifies, is determined by the chemical composition of the bath. That is, g_1 is determined by the mass ratios between x_2, x_3, and x_4. The fact that PBM assumes g_1 to be constant induces modelling errors for the PBM.

Mass of side ledge

Fig. 4.20(a) shows the mass x_1 of frozen cryolite ($Na_3\,Al\,F_6$) which makes up the side ledge. The solidification rate $\dot{x}_1$ is proportional to the heat transfer Q_{liq-sl} through the side ledge ($Q_{liq-sl} \sim \left(\frac{g_1 - x_7}{x_1}\right)$) minus the heat transfer $Q_{bath-liq}$ between the side ledge and the bath ($Q_{bath-liq} \sim (x_6 - g_1)$). The solidification rate $\dot{x}_1$ is dependent on the value of g_1, and therefore the PBM incorrectly predicts the mass rate $\dot{x}_1$. In Fig. 4.20(a), we see that the PBM modelling error for x_1 starts to increase after approximately one hour. At the same time, the actual liquidus temperature g_1 drifts away from the constant PBM estimate of g_1, see Fig. 4.20(i). As we can see, the PBM overestimates g_1. Therefore, the PBM will also overestimate the heat transfer out of the side ledge, leading to an overestimate of the amount of cryolite that freezes and hence an overestimate of the increase in side ledge mass. However, this modelling error is limited by the effect that an increased side ledge mass (and therefore increased side ledge thickness) leads to better isolation. Thus, the PBM estimate of the heat transfer through the side ledge Q_{liq-sl} is inversely proportional to the x_1 estimate, and the modelling error of x_1 reaches a steady state for a constant modelling error in g_1. In addition to modelling errors due to errors in the g_1 estimate, modelling errors of x_6 and x_7 propagate as modelling errors in $\dot{x}_1$.

Both the mean of DDM and the mean of CoSTA models appear to predict the response of x_1 correctly. However, both model classes show a growing spread. While the error spread of both model classes appears to grow over time, the DDM error grows roughly twice as fast. Furthermore, both CoSTA and DDM show some cases where the error bound becomes significantly large, meaning that one or more models fail. These cases appear more frequently for the DDM models, and the errors are more significant than for the CoSTA models. Figs. 4.20(f) and 4.20(g) shows that these error peaks often coincide with the peaks in the bath temperature x_6 and the side ledge temperature x_7.

Mass of alumina

Fig. 4.20(b) shows the mass x_2 of aluminium in the bath. Eq. (2.11) shows that $\dot{x}_2$ (mass rate of Al_2O_3) is proportional to u_1 (Al_2O_3 feed), and negatively proportional to u_2. Fig. 4.20(b) shows that this yields a saw-tooth response that rises as u_1 spikes, and decays with a rate determined by u_2. This state has no dependence on g_1 nor other states that depend on g_1. Therefore the PBM (and CoSTA predict this state with no error. On the other hand, the spread of the DDM models grows over time, with the mean error eventually becoming significant.

Mass of aluminium fluoride

The x_3 state (mass of $Al\,F_3$) acts as an accumulator, rising when $Al\,F_3$ is added to the process (u_3 spikes), and falling when Al_2O_3 is added to the process (u_1 spikes). The latter is caused by impurities ($Na_2\,O$) in the Alumina ($Al_2\,O_3$) reacting with $Al\,F_3$, generating cryolite ($Na_3\,Al\,F_6$). The latter effect is relatively small, as seen in Fig. 4.20(c). Despite this, the DDM models these decreases correctly. However, the DDM models become less and less accurate as time passes. The PBM and CoSTA model x_3 with no error.

Mass of molten cryolite

The state x_4 represents the mass of molten cryolite in the bath, where $\dot{x}_4 = k_5 u_1 - \dot{x}_1$. The first term represents additional cryolite generated by reactions between impurities in the added alumina (u_1) and AlF_3 (x_3). The second term describes how the cryolite can freeze (x_1) on the side ledge, which can melt again as the side-ledge temperature x_7 increases. As seen in Fig. 4.20(d), the response of x_4, therefore, mirrors that of x_1, with relatively small upturns when alumina is added (u_1). Inspecting Fig. 4.20a, we see that the models have identical behaviour. Incorrectly estimating x_4 causes some issues. The mass ratio c_{x_2} (see Table 2.1) is essential in terms of determining the cell voltage U_{cell}. A forecasting error of x_4 will propagate as a forecasting error of c_{x_2}, leading to inaccurate estimates of the cell voltage U_{cell}. This point is elaborated when discussing the bath temperature x_6.

Mass of produced metal

The state x_5 has linear dynamics with a saw-tooth characteristic, growing at a rate proportional to the line current (u_2) and falling when metal is tapped (u_4 spikes). Looking at Fig. 4.20(e), the DDM models have similar error dynamics to the other linear states, while the PBM and CoSTA models have virtually no error.

Temperature of the bath

The bath temperature x_6 has several possible sources of PBM modelling errors. As discussed earlier, since the PBM overestimates the side ledge thickness due to a modelling error of g_1, it follows that the PBM overestimates the thermal insulation of the side ledge and the bath temperature, as the heat transfer out of the bath is underestimated. In Fig. 4.20(f), we see this overestimate of x_6 provided by the PBM after approximately one hour, simultaneously as the PBM starts to overestimate the side ledge mass x_1.

Furthermore, the change in bath temperature $\dot{x}_6$ is determined by the energy balance in the bath. The energy balance in the bath consists of several components, namely the electrochemical power P_{el} which adds energy to the system, the heat transfer from the bath to the side ledge $Q_{bath-sl}$ which transports energy out of the bath, and the energy $E_{tc,liq}$ required to break inter-particle forces in the frozen cryolite liquidus temperature. The electrochemical power $P_{el} = U_{cell} \cdot u_2$ is the product of the cell voltage U_{cell} and the line current u_2. The cell voltage is given by $U_{cell} = \left(g_5 + \frac{u_2 u_5}{2620 g_2}\right)$, where g_5 is the bubble voltage drop, and $\frac{u_2 u_5}{2620 g_2}$ is the voltage drop due to electrical resistance in the bath. The bubble voltage-drop g_5 increases exponentially when the operation gets close to an anode effect. Anode effects occur when the mass ratio of alumina - c_{x_2} is reduced to the critical mass ratio of alumina $c_{x_2,crit} \sim 2$. Anode effects can explain the error peaks in the x_6 estimate, which are most present for the DDM models. Fig. 4.20(f) shows the peaks of the error band for the DDM happen simultaneously with overestimates of x_4 (see Fig. 4.20(d)), indicating that the DDM wrongly predict anode effects in these cases.

Moreover, the voltage drop due to electrical resistance is given by $\frac{u_2 u_5}{2620 g_2}$, where u_2 is the line current, u_5 is the anode-cathode distance, $2620[m^2]$ is the total surface of the anodes and g_2 is the electrical conductivity. Within reasonable operational conditions, $\frac{1}{g_2}$ can be approximated as a function that increases linearly with the increasing mass ratio of alumina c_{x_2}. The modelling error in x_4 can therefore propagate to x_6. After approximately eight hours, the error bound of CoSTA models shows that one of the models overestimates x_6, followed by an underestimate of x_6. A possible explanation is that the CoSTA model first erroneously predicts the anode effect. The underestimate of x_6 that instantaneously follows can be caused by an underestimate of c_{x_2} that is lower than $c_{x_2,crit}$, which leads to negative P_{el} values in the model.

Temperature of the side ledge

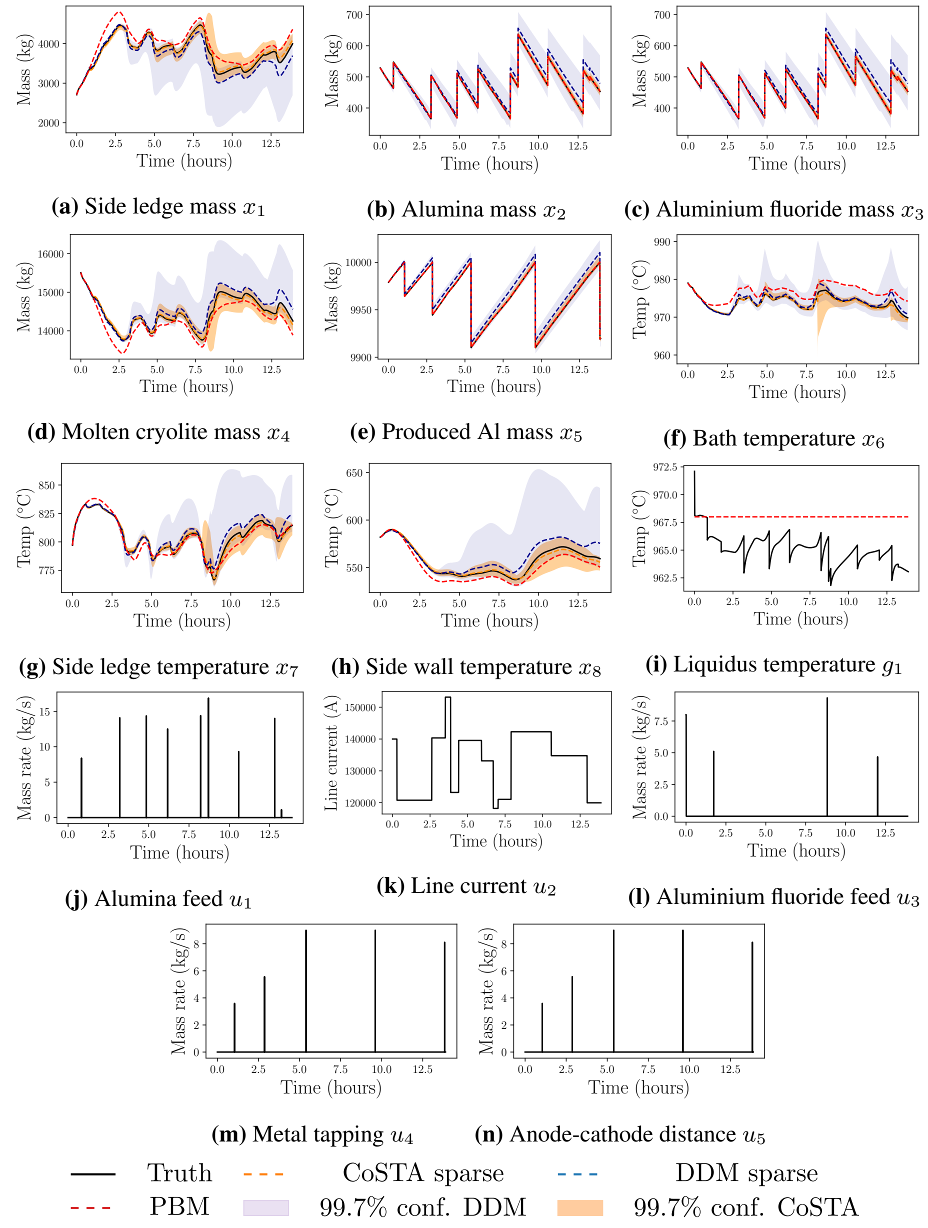
The change in temperature of the side ledge $\dot{x}_7$ is determined by the heat balance in the side ledge. That is, the heat transfer from the bath to the side ledge Q_{liq-sl}, the heat transfer from the side ledge to the side wall $Q_{sl-wall}$, and the energy $E_{tc,sol}$ required to heat frozen side ledge to liquidus temperature from side ledge temperature. The change of side ledge temperature depends on the side ledge thickness x_1, the bath temperature x_6, the side ledge temperature x_7, the wall temperature x_8 and the liquidus temperature g_1. As argued above, for the PBM modelling errors in $x_1,\ x_6,\ x_7, x_8$, and g_1 will propagate as modelling errors in the side ledge temperature change $\dot{x}_7$. For the DDM and CoSTA models, the error bounds for the modelling errors of x_7 shown in Fig. 4.20(g) are mainly growing simultaneously with error spikes in x_6, presumably caused by erroneously predicted anode effects, as explained above.

Temperature in the wall

Fig. 4.20h shows that the temperature of the side wall x_8 changes according to the heat transfer from the side ledge to the wall $Q_{sl-wall}$, and the heat transfer from the wall to the ambient Q_{wall-0}. Changes in the wall temperature $\dot{x}_8$ depend on the side ledge temperature x_7, the wall temperature x_8, and the side ledge thickness x_1. PBM modelling errors of these states at time k propagate as modelling errors in the side wall temperature x_8 in the next time step, $k+1$. Hence, the PBM will, with correct inputs, always model the correct $\dot{x}_8$ since the PBM model of $\dot{x}_8$ is equal to the simulator.

4.3.5 Conclusions and future work

In this work, we presented a recently developed approach in modelling called the Corrective Source Term Approach CoSTA. CoSTA belongs to a family of HAM tools where PBM and DDM are combined to exploit the best of both approaches



(a) Side ledge mass x_1

(b) Alumina mass x_2

(c) Aluminium fluoride mass x_3

(d) Molten cryolite mass x_4

(e) Produced Al mass x_5

(f) Bath temperature x_6

(g) Side ledge temperature x_7

(h) Side wall temperature x_8

(i) Liquidus temperature g_1

(j) Alumina feed u_1

(k) Line current u_2

(l) Aluminium fluoride feed u_3

(m) Metal tapping u_4

(n) Anode-cathode distance u_5

Figure 4.20: Rolling forecast of a representative test trajectory. 10 CoSTA models with sparse corrective NNs, 10 DDMs consisting of sparse NN models, as well as a PBM, are predicting the test set trajectories given the initial conditions and the input vector at any given time.

while eliminating their weaknesses. The method was applied to model an aluminium extraction process governed by very complex physics. First, ground-truth data were collected using a detailed high-fidelity simulation. Then, an ablated model was created by setting an internal variable of the simulator to a constant. Finally, the ablated model was supplemented with a corrective source term modelled using a NN that compensated for the ignored physics. The main conclusions from the study are as follows:

- CoSTA, in all the scenarios investigated, could correct for the ignored physics and was consistently more accurate than both the PBM and DDM over a reasonably long time horizon.
- CoSTA was consistently more stable and consistent in predictions compared to pure DDM.
- Regularizing the networks using ℓ_1 weight decay was effective in improving model stability in both DDM and CoSTA.

One significant benefit of the CoSTA approach is that it can maximise the utilisation of domain knowledge, only using black-box DDM to handle physics that are not well known. Although it remains to be investigated in future work, it can be expected that much simpler models will be sufficient for modelling the corrective source terms. These source terms can then be investigated to achieve additional insight giving more confidence in the model. Even if it is not possible to interpret the source terms, it should still be possible to place bounds on their outputs using domain knowledge. These bounds can then be used as an inbuilt sanity check mechanism. For example, since we know the amount of energy put into the system, the source terms for the energy equation will be bounded, so any NN -generated source term violating this bound can be confidently rejected, making the models more attractive for high-stake applications like the one considered here. Another topic worth investigating is the robustness of the method to noise.

4.4 Sparse neural networks with skip-connections for nonlinear system identification

Recent research has found that using sparser networks may be the key to training models that can generalise across many situations. One hypothesis is that NNs are typically over-parameterised and require many training iterations to adjust all parameters. However, overtraining the models on the same limited dataset will lead to overfitting and poor performance on unseen data. While over-parameterisation has been found to aid convergence during training [150], it also introduces redundant information into the weights. In particular, [193] showed empirically that for any dense architecture, there is a high probability that a sparse subnetwork will train faster and generalise better than the dense network. This statement is known as the Lottery Ticket Hypothesis. Many sparsification methods can be seen as attempts to extract such a “winning lottery ticket” from an initially dense network. There have been numerous advances in this field, and we refer to [191] for a recent and comprehensive review. The well-known ℓ_1 regularisation is used to induce weight sparsity in the model.

Another challenge related to using NNs is the choice of architecture and hyper-parameters. Typical networks have multiple layers which are densely connected, although this can vary between domains. Choosing an appropriate architecture involves trial and error to improve performance and avoid overfitting. It is commonly understood that the early layers of a neural network significantly impact the overall performance. However, deep networks often suffer from the vanishing or exploding gradient problem, which prevents effective training of these early parameters [60]. Skip-connections were initially proposed to circumvent this by introducing a shorter path between the early layers and the output [62]. The method enabled the training of significantly deeper networks, but [210] also demonstrated that they may help improve training convergence.

In dynamical systems and control, models are often developed with a purpose, i.e. designing a control system or state observer. Crucially, we are interested in the behaviour and performance of the controlled system in terms of objectives such as energy efficiency or yield. Accordingly, the model does not need to be perfectly accurate for the entire state space so long as the resulting closed-loop performance is sufficient. This methodology is sometimes called identification for control (I4C). If high-frequency measurements from the system are available, only the short-term behaviour of the model is essential since any drift out of the operational space is quickly corrected. However, if measurements are rarely available, such as in the aluminium electrolysis process that we consider, the long-term model behaviour and open-loop stability become much more critical. Stable long-term predictions are also crucial for effective decision-making.

In this work, we investigate the effects of adding skip connections and ℓ_1 regularisation on the accuracy and stability of these models for short, medium, and long horizons. We address the following questions:

- How do skip connections affect the stability and generalisation error of NNs trained on high-dimensional nonlinear dynamical systems?
- How does sparsity affect stability and generalisation error for NNs with skip connections when modelling nonlinear dynamics?

- How does the amount of training data affect NNs with skip connections compared to NNs without skip connections?

We make the following contributions:

- We perform a black box system identification of an aluminium electrolysis cell using different NN architectures.
- We demonstrate that the accuracy and open-loop stability of the resulting models is greatly improved by using ℓ_1 weight regularisation and incorporating skip connections into the architecture.
- This advantage is consistent across datasets of varying sizes.

We evaluate NNs for nonlinear system identification by first training them on synthetic data generated from a known PBM. The model we will use is presented in Example 2.1.3, which describes the internal dynamics of an aluminium electrolysis cell based on the Hall-Héroult process. Fig. 2.1 shows a diagram of the electrolysis cell. Traditional PBMs of such systems are generally constructed by studying the mass/energy balance of the chemical reactions.

4.4.1 Deep neural network with skip connections

A NN with L layers can be compactly written as an alternating composition of affine transformations $\mathbf{W}\mathbf{z} + \mathbf{b}$ and nonlinear activation functions $\boldsymbol{\sigma} : \mathbb{R}^n \mapsto \mathbb{R}^n$:

$$\begin{aligned} \mathcal{N}(\mathbf{z}) &= \mathcal{N}_L \circ \cdots \circ \mathcal{N}_2 \circ \mathcal{N}_1 \\ \mathcal{N}_i(\mathbf{z}) &= \boldsymbol{\sigma}^{[i]}(\mathbf{W}^{[i]}\mathbf{z} + \mathbf{b}_i), \end{aligned} \tag{4.26}$$

where the activation function $\boldsymbol{\sigma}^{[i]}$, weight matrix $\mathbf{W}^{[i]}$, and bias vector $\mathbf{b}_i$ correspond to the ith layer of the network. The universal approximation property of NNs makes them attractive as a flexible model class when many data are available. The representation capacity is generally understood to increase with depth and width (the number of neurons in each layer). However, early attempts to train deep networks found them challenging to optimise using backpropagation due to the vanishing/exploding gradients problem. One of the major developments that enabled researchers to train deep NNs with many layers is the *skip connection*. A skip connection is simply an additional inter-layer connection that bypasses some of the layers of the network. This "shortcut" provides alternate pathways through which the loss can be backpropagated to the early layers of the NN, which helps stabilise the gradients. In this work, we utilise a modified DenseNet architecture proposed by Huang et al. [211], where the outputs of earlier layers are concatenated to all the consecutive layers. We simplify the structure such that the model only contains skip-connections from the input layer to all consecutive layers. We call this architecture InputSkip, which has reduced complexity compared to DenseNet. This design is motivated by the fact that the output of each layer (including the final output) becomes a sum of both a linear and a nonlinear transformation of the initial input $\mathbf{x}$. Hence, the skip connections from the input layer to consecutive layers facilitate the reuse of the input features for modelling different linear and nonlinear relationships more independently.

4.4.2 Method and setup

In this section, we present all the details of data generation, preprocessing, and the methods required to reproduce the work. The steps can be briefly summarised as follows:

- Use Eq. (2.11) with random initial conditions to generate 140 trajectories with 5000 timesteps each. Set aside 40 for training and 100 for testing. Construct three datasets by selecting 10,20, and 40 trajectories, respectively.
- For each model class and dataset, train ten instances on the training data.
- Repeat all experiments with ℓ_1 regularisation, see loss function in Eq. (4.27).
- Use trained models to generate predicted trajectories along the test set and compare them to the 100 test trajectories.

Data generation

Eq. (2.11) was discretised using the RK4 scheme with a fixed timestep $h = 10\,\mathrm{s}$ and numerically integrated on the interval $[0, 5000h]$. We used uniformly randomly sampled initial conditions from the intervals shown in Table 4.5 to generate 140 unique trajectories. We set aside 40 trajectories for training and 100 of the trajectories as a test set. The 40 training trajectories were used to create three datasets of varying sizes (small, medium, large), namely 10, 20, and 40 trajectories. The datasets contained 50000, 100000, and 200000 individual data points.

Eq. (2.11) also depends on the input signal $\mathbf{u}$. In practice, this is given by a deterministic control policy $\mathbf{u} = \boldsymbol{\pi}(\mathbf{x})$ that stabilises the system and keeps the state $\mathbf{x}$ within some region of the state space that is suitable for safe operation. However, data collected using the deterministic policy was insufficient to successfully train our models because the controlled trajectories showed minimal variation after some time, despite having different initial conditions. This lack of diversity in the dataset resulted in models that could not generalise to unseen states, which frequently arose during evaluation. To inject more variety into the data and sample states $\mathbf{x}$ outside of the standard operational area, we used a stochastic controller

$$\boldsymbol{\pi}_s(\mathbf{x}) = \boldsymbol{\pi}(\mathbf{x}) + \mathbf{r}(t)$$

that introduced random perturbations $\mathbf{r}(t)$ to the input. These perturbations were randomly sampled using the APRBS method [209, 208].

In system identification, it is typical to optimise the model to estimate the function $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. However, this is not feasible for Eq. (2.11) because the inputs $\mathbf{u}$ are not differentiable. Instead, we discretise the trajectories using the forward Euler difference and use this as the regression variable:

$$\mathbf{y}_k = \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{h}$$

The datasets are then constructed as sets of the pairs $([\mathbf{x}_k, \mathbf{u}_k], \mathbf{y}_k)$.

Table 4.5: Initial conditions intervals for $\mathbf{x}$

Variable	Initial condition interval
x_1	$[2060,\ 4460]$
c_{x2}	$[0.02,\ 0.05]$
c_{x3}	$[0.09,\ 0.13]$
x_4	$[11500,\ 16000]$
x_5	$[9550,\ 10600]$
x_6	$[940,\ 990]$
x_7	$[790,\ 850]$
x_8	$[555,\ 610]$

Training setup

We optimise the models by minimising the following loss function using stochastic gradient descent:

$$\mathbf{J}(\mathcal{B}, \theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\mathbf{y_i} - \mathcal{N}(\mathbf{x}_i, \mathbf{u}_i))^2 + \lambda \sum_{j=1}^{L} |\mathbf{W}^{[j]}| \tag{4.27}$$

where $\mathcal{B}$ is a *batch* of randomly sampled subset of indices from the dataset, L is the number of layers of the NN, and λ is the regularisation parameter. This loss function is the sum of the MSE of the model $\mathcal{N}$ w.r.t. the regression variables $\mathbf{y}$, and the ℓ_1 norm of the connection weight matrices $\mathbf{W}^{[i]}$ in all layers. We used a batch size of $|\mathcal{B}| = 128$. We used the popular ADAM solver [82] with default parameters to minimise Eq. (4.27).

Evaluation of model accuracy

As previously mentioned, we are interested in evaluating the long-term predictive accuracy of the models. The model $\mathcal{N}(\mathbf{x}, \mathbf{u})$ is used to generate an estimated trajectory from the initial condition $\mathbf{x}(t_0)$, using the following recurrence relation:

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_k + h\mathcal{N}(\hat{\mathbf{x}}_k, \mathbf{u}_k) \tag{4.28}$$

where $\hat{\mathbf{x}}_0 = \mathbf{x}_0$. Note that the input signal $\mathbf{u}_k$ is replayed directly from the test trajectory. Borrowing a term from the field of time-series analysis, we refer to this as a *rolling forecast*. To evaluate the accuracy of a model over multiple trajectories, we define the AN-RFMSE:

$$\text{AN-RFMSE} = \frac{1}{p}\sum_{i=1}^{p}\frac{1}{n}\sum_{j=1}^{n}\left(\frac{\hat{x}_i(t_j) - x_i(t_j)}{\operatorname{std}(x_i)}\right)^2, \tag{4.29}$$

where $\hat{x}_i(t_j)$ is the model estimate of the simulated state variable x_i at time step t_j, $\operatorname{std}(x_i)$ is the standard deviation of variable x_i in the training set $\mathcal{S}_{train}$, $p = 8$ is the number of state variables and n is the number of time steps being averaged over.

Evaluation of model stability

A symptom of model instability is that its predictions can *blow-up*, which is characterised by a rapid (often exponential) increase in prediction error. We define a blow-up as the point where the normalised mean absolute error for all system states exceeds three (this corresponds to standard deviations). We detect this as follows:

$$\max_{j<n}\left[\frac{1}{p}\sum_{i=1}^{p}\left(\frac{|\hat{x}_i(t_j) - x_i(t_j)|}{\operatorname{std}(x_i)}\right)\right] > 3 \tag{4.30}$$

where $p = 8$ is again the number of state variables and n is the number of time steps to consider. This estimate is conservative. However, this does not lead to a significant underestimation of the number of blow-ups. Once a model starts to drift rapidly, the normalised error rapidly exceeds three standard deviations.

4.4.3 Results and discussions

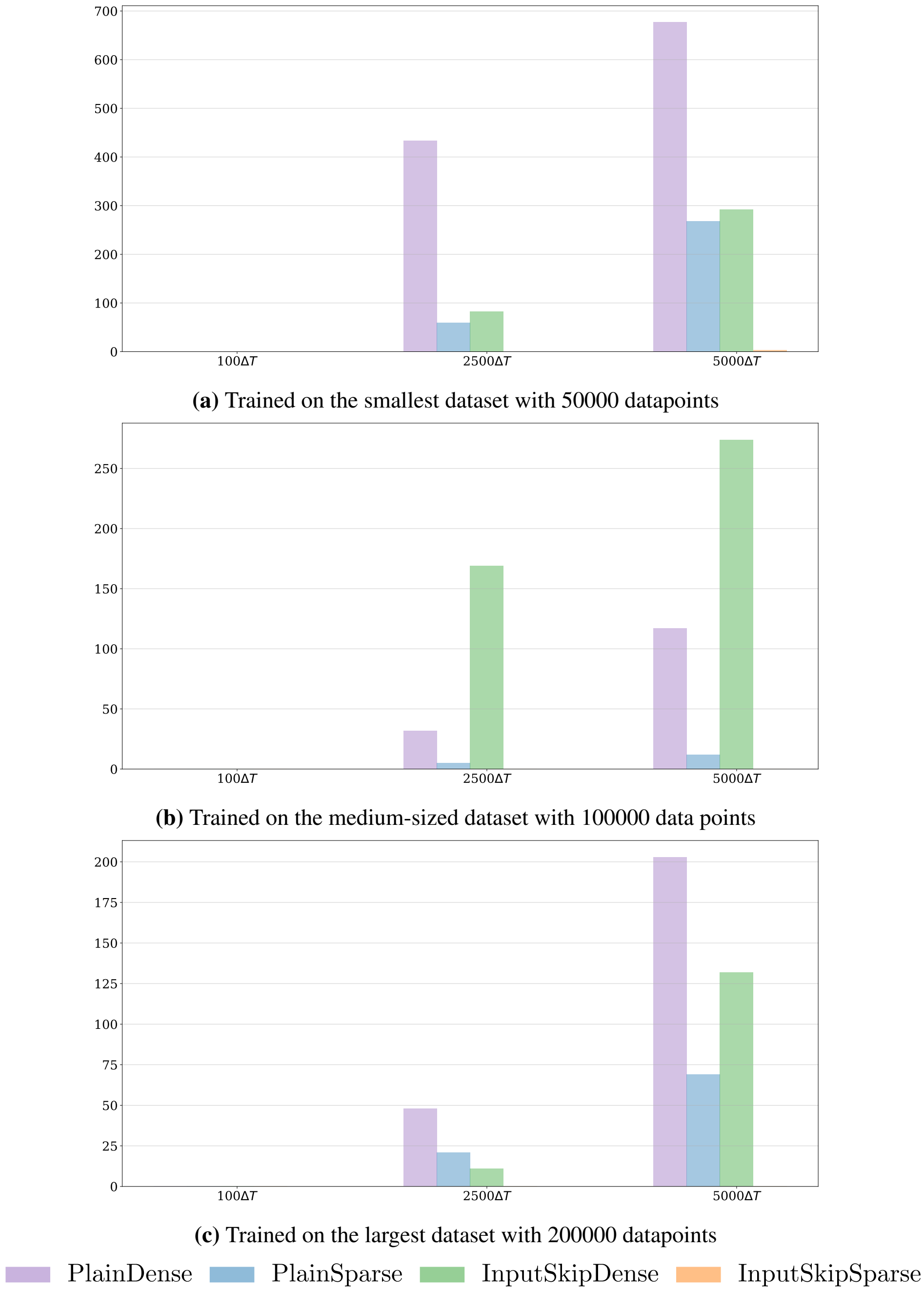
We characterise the different model classes (PlainDense, PlainSparse, InputSkip-Dense, InputSkipSparse) by estimating their blow-up frequencies and their RFMSE on the validation data. The blow-up frequency is an interesting measure since it can indicate how stable the model is in practice.

We perform a Monte Carlo analysis by training ten instances of each model class and evaluating these on 100 trajectories randomly generated using the actual model, yielding 1000 data points for each model class. We repeat the experiments for three different dataset sizes to study the data efficiency of the models.

Fig. 4.21 presents the total number of blow-ups recorded within each model class after $100h$, $2000h$, and $5000h$ (short, medium, and long term respectively). For simplicity, blow-ups were detected by thresholding the computed variance of a predicted trajectory and manually inspected. It is clear that for short time horizons, all the models exhibit robust behaviour independently of the size of the training datasets. However, for medium and long time horizons, PlainDense, PlainSparse, and InputSkipDense architectures exhibit a significant number of blow-ups and, therefore, instability. Fig. 4.21(a) - Fig. 4.21(c) show that PlainDense is generally the most unstable, with up to 67% of all trajectories resulting in a blow-up. For the smallest amount of training data (Fig. 4.21(a)) PlainSparse and InputSkipDense have similar blow-up frequencies. The stability of InputSkipDense and PlainDense improves with more training data, measured by the number of blow-ups. However, both models are more unstable relative to PlainSparse, which also shows improved stability for the larger datasets.

In comparison, almost no blow-ups are recorded using the InputSkipSparse architecture, even for the small training dataset. In Fig. 4.19, the orange bars corresponding to the blow-up frequency of InputSkipSparse models are not visible for any training sets due to the significantly lower number of blow-ups. For InputSkipSparse models trained on the smallest dataset, only 3 out of 1000 possible blow-ups were reported for the longest horizon. Apart from that, no blow-ups were reported for the InputSkipSparse models. Only a few blow-ups were recorded after $5000h$ in the medium term.

Fig. 4.22 presents a violin plot of the accuracy of each model class, expressed in terms of RFMSE over different time horizons. Only the plot for the smallest dataset (50000 points) is shown because the results are similar. A larger width of the violin indicates a higher density of that given RFMSE value, while the error bars show the minimum and maximum recorded RFMSE values. The model estimates that blew up (see Fig. 4.21) are omitted. In this way, we estimate the generalisation performance of the models only within their regions of stability. Note that the violin plots for model classes with many blow-ups are made using fewer samples and can be seen as slightly "cherry-picked". Nonetheless, the InputSkipSparse architecture consistently yields more accurate results, up to an order of magnitude better than the others in the long term.



(a) Trained on the smallest dataset with 50000 datapoints

(b) Trained on the medium-sized dataset with 100000 data points

(c) Trained on the largest dataset with 200000 datapoints

Figure 4.21: Divergence plot: Number of trajectories that blow up over different time horizons. The total number of trajectories is 1000, so the values can be read as a permille.

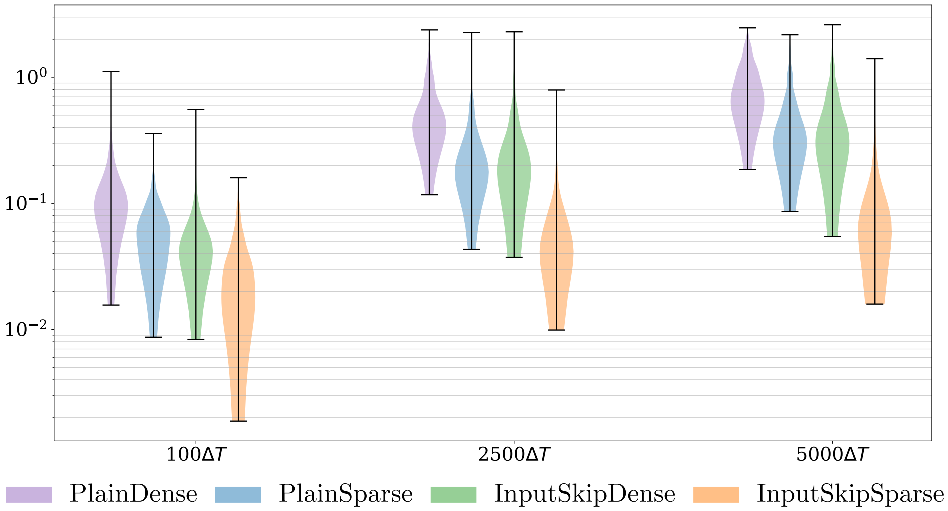
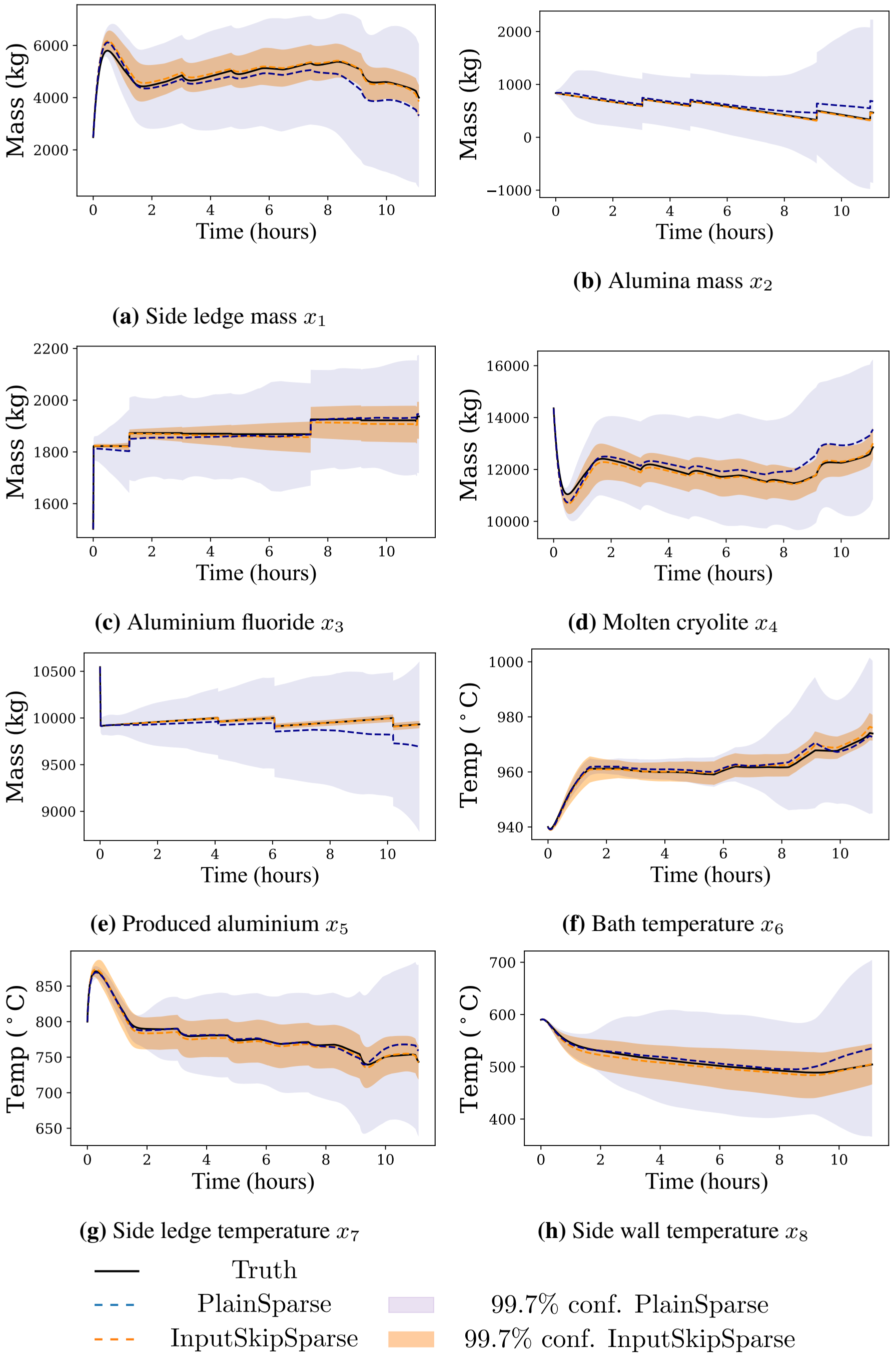


Figure 4.22: Model accuracy expressed in terms of RFMSE over different horizons. Ten models of each model type (PlainDense, PlainSparse, InputSkipDense, InputSkipSparse) are trained on the smallest dataset of 50000 data points. The model estimates that blow up (see Fig. 4.21) are excluded. The error bars for each model type represent 95% confidence intervals. Sparse models with skip connections (InputSkipSparse) are consistently more accurate than sparse and dense models without skip connections.

4.4.4 Discussion

This work compared the performance of two different model structures trained with and without sparsity promoting ℓ_1 regularisation. The two model types are a standard NN and a more specialised architecture that includes skip-connections from the input layer to all consecutive layers. Four model structures were tested: PlainDense, PlainSparse, InputSkipDense, and InputSkipSparse. The main conclusions of the article are as follows:

- NNs with skip connections are more stable for predictions over long time horizons compared to standard NNs. Furthermore, the accuracy of NNs with skip connections is consistently higher for all forecasting horizons.
- Sparsity-promoting ℓ_1 regularisation improves the stability of all models tested. This improvement was more apparent for models with the InputSkip architecture than the standard models.
- The InputSkipSparse showed satisfactory stability characteristics even when the amount of training data was restricted, suggesting that this architecture is more suitable for system identification tasks than the standard NN structure.



(a) Side ledge mass x_1

(b) Alumina mass x_2

(c) Aluminium fluoride x_3

(d) Molten cryolite x_4

(e) Produced aluminium x_5

(f) Bath temperature x_6

(g) Side ledge temperature x_7

(h) Side wall temperature x_8

Figure 4.23: Rolling forecast of a representative test trajectory

The case study shows that both sparsity-promoting regularisation and skip connections can result in more stable NN models for system identification tasks while requiring fewer data and improving their multi-step generalisation for both short, medium and long prediction horizons. Despite the encouraging performance of the sparse-skip networks, we can not guarantee similar performance for noisy data. We have only investigated the use of synthetic data devoid of any noise, although such a study will be an interesting line of future work. This case study also has relevance beyond the current setup. In more realistic situations, we often have a partial understanding of the system we wish to model (see Eq. (2.11)) and only wish to use data-driven methods to correct a PBM when it disagrees with the observations (e.g. due to a faulty assumption). As shown in Section 4.3, combining PBMs and data-driven methods in this way also has the potential to inject instability into the system. Finding new ways to improve or guarantee out-of-sample behaviour for data-driven methods is paramount to improving the safety of these systems.

Chapter 5

Verification

DNNs are still regarded as “black box” models, and few guarantees can be made about their behaviour. The idea of adversarial attacks has exposed that many existing DNNs models have very low robustness. It has been shown that by changing the input minimally in a targeted way, DNNs can be tricked into giving erroneous output. Such attacks are sometimes limited to a single pixel [212]. One way to address this is via *adversarial training*, where adversarial examples are identified and included in the training process. While this has been shown to improve the robustness of the model, it does not guarantee that other examples do not exist. This issue is critical when applying DNN to safety-critical systems like robotic surgery or autonomous cars, making it challenging to safely and responsibly apply these models in a system identification or control context. There is a clear need for practical methods to verify these models by checking and certifying properties such as robustness and stability, motivating recent research into the verification of DNNs.

It is well known that NNs with PWA activation functions are themselves PWA, with their domains consisting of a vast number of linear regions. If we can find ways to compute these linear regions, we can apply the many existing tools to analyse and control the PWA systems. Prior work on this topic has focused on counting the number of linear regions rather than obtaining explicit PWA representations.

A wealth of literature exists on PWA systems, particularly in modelling and control. For example, the explicit solution to the linear MPC problem is a PWA function, and there are schemes for using PWA models in the optimisation loop [213]. Furthermore, methods exist for verifying the stability of PWA systems and stabilising them [214]. Positive invariant sets can be constructed for PWA systems by analysing the possible transitions between the linear regions of the system [215]. Thus, by decomposing a DNN into its PWA representation, these established methods can be used to obtain concrete stability results for a large family of NNs.

Algorithms have been developed to provide such robustness guarantees for piecewise linear networks w.r.t. some perturbation with a bounded norm. Liu et al. [204] give a good overview of such methods. These bounds are often computed by formulating the network as a set of constraints in a mixed integer optimisation problem [216, 217, 218]. However, the problem is computationally expensive and has been shown by Katz et al. to be NP-complete [219]. This fact means that even relatively small networks can take several hours to verify [218].

There is significant research interest in reducing the computational demands of verification. A relatively simple approach is to reduce the size of the neural network, which can be done (i) before training by adjusting the architecture of the network (e.g. neural architecture search), (ii) after training (using *model compression/knowledge distillation* methods), or (iii) during training (by pruning neurons or weights). Methods have been developed to identify all neurons and layers with a constant influence over the relevant domain and prune them [220, 221]. Both papers report that training the original network with ℓ_1 weight decay improved compression results significantly.

In Section 5.1, we propose a novel method to compute the PWA representation of any fully connected neural network with rectified linear unit activations.

In Section 5.2, we use this method to visualise the linear regions of a neural network during training and see how common types of regularisation can influence them. This investigation provides a more intuitive and geometric perspective on weight sparsity, the dying neuron problem, and the mechanism by which standard weight decay tends to induce either dead or constant-effect neurons. We also construct a pathological example where sparsity-inducing regularisation leads to worse generalisation.

An obstacle to scaling this approach to larger models is that the number of regions grows exponentially with the depth and input dimension of the network. However, most of the linear regions of the networks appear to provide little additional information. Taking advantage of this fact, Section 5.3 describes a method to compute an approximate PWA representation of a neural network and demonstrates its use on a nonlinear system identification benchmark based on data from an F-16 jet. The severity of the approximation can be easily controlled via a single parameter. This tool makes applying the numerous analysis and control design methods available for PWA systems possible.

5.1 Computing linear regions of piecewise affine neural network

This section presents an algorithm that can convert any neural network using fully connected layers and ReLU activations into its exact PWA representation that can be visualised and analysed, giving an insight into the inner workings of the network. Existing linear programming (LP) methods (specifically the MPT toolbox for MATLAB© [222]) for working with polyhedral sets and hyperplane arrangements were adapted for this purpose. The approach can also be extended to any linear/affine layer (convolutional layers, batch normalisation), as well as any PWA activation function (Leaky ReLU, maxout).

5.1.1 PWA functions

A PWA function $\mathbf{p} : \mathbb{R}^n \mapsto \mathbb{R}^m$ with N pieces can be written as:

$$\mathbf{p}(\mathbf{x}) = \begin{cases} \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 & \forall \mathbf{x} \in \Omega_1 \\ \vdots & \vdots \\ \mathbf{W}_N\mathbf{x} + \mathbf{b}_N & \forall \mathbf{x} \in \Omega_N \end{cases} \tag{5.1}$$

Each $(\mathbf{W}_i, \mathbf{b}_i)$ is associated with some region $\Omega_i \subset \mathbb{R}^n$. We refer to each case of Eq. (5.1) as $\mathbf{p}_i(\mathbf{x})$. All Ω_i are disjoint, implying that $\mathbf{p}$ is single-valued and only one $\mathbf{p}_i(\mathbf{x})$ is active at one time. This definition does not restrict $\mathbf{p}$ to be continuous; however, this is assumed in most cases.

Note that any individual affine transformation can also be written as a piecewise linear (PWL) function by lifting it to homogeneous coordinates:

$$\begin{bmatrix} \mathbf{p}_i(\mathbf{x}) \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{W}_i & \mathbf{b}_i \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \quad \forall \mathbf{x} \in \Omega_i \tag{5.2}$$

This identity allows chains of affine transformations to be written more compactly as a series of matrix multiplications, which we will use to simplify the notation of this chapter.

5.1.2 PWA neural networks and activation patterns

Neural networks consisting of linear/affine layers and continuous PWA activation functions are themselves PWA and continuous [223] (see Eq. (5.1)). This section shows this non-rigorously, starting with a simple scalar example, defining activation patterns and linear regions, and presenting relevant notation.

This section makes heavy use of the indexing notation presented in Section 2.2, where the superscript $\cdot^{(n)}$ is used to associate a symbol to the neuron with index n, while $\cdot^{[k]}$ refers to the kth layer of the network.

First, consider a NN with L layers each consisting of a single neuron:

$$\begin{aligned} \mathcal{N}(x) &= \left(\sigma^{[L]} \circ f^{[L]} \circ \cdots \circ \sigma^{[1]} \circ f^{[1]}\right)(\mathbf{x}) \\ f^{[k]}(x) &= w^{[k]}x + b^{[k]} \end{aligned} \tag{5.3}$$

Let the activation function for all neurons be the following continuous PWA function:

$$\sigma^{(n)}(z) = \begin{cases} a_1 z, & z \geq 0 \\ a_2 z, & z < 0 \end{cases} \tag{5.4}$$

where n is the index of the neuron. An offset term is omitted without loss of generality because any such term could be cancelled by an equal offset in the bias vector of the previous fully connected layer. This function divides its input space into two intervals, namely $(-\infty, 0)$ and $[0, \infty)$. Instead of writing down the whole network as a chain of operations, we can write it recursively and expand each case:

$$\mathcal{N}(x) = \begin{cases} a_1 z^{[L]} \\ \textbf{for } z^{[L]} \geq 0 \\ \\ a_2 z^{[L]} \\ \textbf{for } z^{[L]} < 0 \end{cases} = \begin{cases} a_1^2 w^{[L]} z^{[L-1]} + a_1 b^{[L]}, \\ \textbf{for } z^{[L]} \geq 0 \textbf{ and } z^{[L-1]} \geq 0 \\ \\ a_1 a_2 w^{[L]} z^{[L-1]} + a_1 b^{[L]}, \\ \textbf{for } z^{[L]} \geq 0 \textbf{ and } z^{[L-1]} < 0 \\ \\ a_1 a_2 w^{[L]} z^{[L-1]} + a_1 b^{[L]}, \\ \textbf{for } z^{[L]} < 0 \textbf{ and } z^{[L-1]} \geq 0 \\ \\ a_2^2 w^{[L]} z^{[L-1]} + a_1 b^{[L]}, \\ \textbf{for } z^{[L]} < 0 \textbf{ and } z^{[L-1]} < 0 \end{cases} \tag{5.5}$$

It is easy to see that all cases are affine transformations and will remain so after subsequent expansions. We can therefore conclude that $\mathcal{N}(x)$ is a PWA function. Fully expanding the expression leads to 2^L different cases, where each case is associated with some set of inputs x that results in a specific configuration of $\{\mathrm{sgn}(z^{[k]})\}_{k=1}^{L}$.

More generally, every $z^{[k]}$ must lie in one and only one of the intervals that the activation function $\sigma(z)$ is defined on, in this case, $(-\infty, 0)$ and $[0, \infty)$. This "active interval" is referred to as the **activation** of a neuron, while the set of activations of all neurons in a network is called the **activation pattern** [218], denoted by π. The set of all inputs x that result in a specific activation pattern is called a **linear region**.

For this simple example, we can introduce a shorthand to refer to the intervals of Eq. (5.4), namely $\{-\} = (-\infty, 0)$ and $\{+\} = [0, \infty)$. An activation pattern could now be written as $\pi = (-, -, +, \dots, -)$, where the position of the sign in the tuple corresponds to the layer of the neuron. However, when considering NNs with more complex architectures, it is more convenient to explicitly assign each neuron some index n and associate this with either $\{+\}$ or $\{-\}$. An activation pattern, according to this definition, has the form:

$$\pi = \{(1, a_1), \dots, (n, a_n)\}, \quad a_i \in \{+, -\} \tag{5.6}$$

Defining activation patterns in this way permits the specification of *partial activation patterns*, where only a subset of all neurons are given fixed activations. Now, given some activation pattern π, we can compute the corresponding affine transformation simply by replacing each $\sigma^{(n)}$ with the linear transformation specified by π and simplifying the resulting expression. This operation applied to the nth neuron of the network (see Eq. (5.4)) is given the following notation:

$$\sigma^{(n \mid \pi)}(x) = \begin{cases} a_1 x & \text{if } (n, +) \in \pi \\ a_2 x & \text{if } (n, -) \in \pi \\ \sigma^{(n)}(x) & \text{otherwise} \end{cases} \tag{5.7}$$

Applying this operation to a layer of neurons is written as:

$$\boldsymbol{\sigma}^{[k \mid \pi]}(x) = \begin{bmatrix} \sigma^{(n_1 \mid \pi)}(x) \\ \sigma^{(n_2 \mid \pi)}(x) \\ \sigma^{(n_3 \mid \pi)}(x) \\ \vdots \end{bmatrix} \quad \forall \text{ neurons } n_i \in \text{ layer } k \tag{5.8}$$

We also extend the notation to Eq. (5.3) and all subnetworks as follows:

$$\mathcal{N}^{[L \mid \pi]}(x) = \left(\sigma^{[L \mid \pi]} \circ f^{[L]} \circ \cdots \circ \sigma^{[1 \mid \pi]} \circ f^{[1]}\right)(x) \tag{5.9}$$

The discussion so far has been limited to the scalar network described by Eq. (5.3). Now consider a network with L fully connected layers. We use Eq. (5.2) to express the network as a series of alternating matrix multiplications and applications of $\boldsymbol{\sigma}(\mathbf{z})$. The operation performed by a fully connected layer (without activation) is:

$$\begin{bmatrix} \mathbf{z}^{[k]} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{W}^{[k]} & \mathbf{b}^{[k]} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}^{[k-1]} \\ 1 \end{bmatrix} = \mathbf{T}^{[k]} \begin{bmatrix} \mathbf{x}^{[k-1]} \\ 1 \end{bmatrix} \tag{5.10}$$

With a slight abuse of notation, we treat the linear transformation $\mathbf{T}^{[k]}$ as a function and use the composition operator $\circ$ to represent matrix multiplications such as $(\mathbf{T}^{[2]} \circ \mathbf{T}^{[1]})\mathbf{x} = \mathbf{T}^{[2]}\mathbf{T}^{[1]}\mathbf{x}$. The network can then be written as:

$$\begin{bmatrix} \mathcal{N}(\mathbf{x}) \\ 1 \end{bmatrix} = \left(\boldsymbol{\sigma}^{[n]} \circ \mathbf{T}^{[n]} \circ \cdots \circ \boldsymbol{\sigma}^{[1]} \circ \mathbf{T}^{[1]}\right) \left(\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}\right) \tag{5.11}$$

As mentioned before, $\boldsymbol{\sigma}^{[k]}(\mathbf{z})$ applies $\sigma^{[k]}$ to $\mathbf{z}$ element-wise. Eq. (5.7) then becomes a diagonal matrix:

$$\boldsymbol{\sigma}^{[k \mid \pi]}(\mathbf{z}) = \operatorname{diag}\left\{\sigma^{[k \mid \pi]}(z_1), \sigma^{[k \mid \pi]}(z_2), \ldots, \sigma^{[k \mid \pi]}(z_n)\right\} \tag{5.12}$$

It is now easy to see how the network in Eq. (5.11) simplifies into a linear transformation when subject to some activation pattern π. However, only a subset of all possible activation patterns can be triggered by some input to the network, as demonstrated in the next section using some simple examples.

5.1.3 Linear regions of a simple piecewise affine neural network (PWANN)

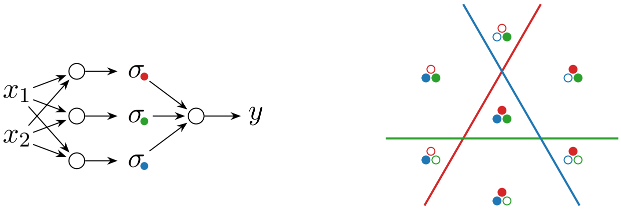


Figure 5.1: Illustration of the linear regions of a single-layered network. Filled and hollow circles represent active/inactive ReLUs of the same colour.

Consider a neural network with two inputs and one hidden layer with three nodes and ReLU activation, as shown in Fig. 5.1. The general form of the network in homogeneous coordinates is:

$$\begin{bmatrix} \mathcal{N}(\mathbf{x}) \\ 1 \end{bmatrix} = \boldsymbol{\sigma}\left(\mathbf{T}\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}\right) = \boldsymbol{\sigma}\left(\begin{bmatrix} \mathbf{W} & \mathbf{b} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}\right) \tag{5.13}$$

Each row of the parameter matrix $\mathbf{T}$ corresponds to a single neuron. Eq. (5.13) can then be written as:

$$\mathcal{N}(\mathbf{x}) = \begin{bmatrix} \sigma\left(\mathbf{w}_1^{[1]}\mathbf{x} + b_1^{[1]}\right) \\ \sigma\left(\mathbf{w}_2^{[1]}\mathbf{x} + b_2^{[1]}\right) \\ \sigma\left(\mathbf{w}_3^{[1]}\mathbf{x} + b_3^{[1]}\right) \end{bmatrix} \tag{5.14}$$

The vector $\mathbf{w}_i^{[1]}$ represents the ith row of $\mathbf{W}^{[1]}$, which corresponds to the incoming connection weights of a single neuron. Each neuron has two activation states given by the complementary half-spaces $\mathbf{wx} + b < 0$) and $\mathbf{wx} + b \geq 0$, respectively. The boundary between these two modes is given by $\mathbf{wx} + b = 0$, which defines a line. In the general case, this boundary will be a hyperplane in $\mathbb{R}^n$ when there are n inputs, as shown in Fig. 5.2.

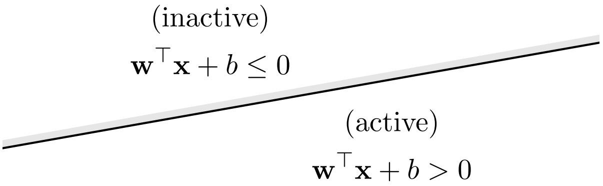
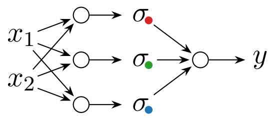

(inactive)
$\mathbf{w}^\top\mathbf{x} + b \leq 0$
(active)
$\mathbf{w}^\top\mathbf{x} + b > 0$

Figure 5.2: Each node with ReLU activation has two modes: one where it is active and one where it is inactive. Boundaries are therefore drawn, with a shaded side representing the inactive side.

Superimposing these boundaries yields a *hyperplane arrangement*, which defines a set of polyhedral regions $\mathcal{P}_i$ in the input space, each corresponding to a different activation pattern π_i. Fig. 5.1 illustrates some example regions, where the activation states of each neuron are represented using colour instead of symbols such as $\{+\}$ and $\{-\}$.

The ReLU activation function is equivalent to Eq. (5.4) with $a_1 = 1$ and $a_2 = 0$, implying that an inactive neuron is equivalent to setting the corresponding row in the $\mathbf{T}$ matrix to zero. Fig. 5.3 lists all possible linear regions for this example. Note that there is one activation pattern that is not feasible, namely $\pi = \{(1,-),(2,-),(3,-)\}$.



$$\overset{\bullet}{\bullet\bullet} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{w}_1^{(1)} \\ \mathbf{w}_2^{(1)} \\ \mathbf{w}_3^{(1)} \end{bmatrix} \mathbf{x} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \right) + b^{(2)}$$

$$\overset{\circ}{\bullet\bullet} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{0} \\ \mathbf{w}_2^{(1)} \\ \mathbf{w}_3^{(1)} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \right) + b^{(2)}$$

$$\overset{\bullet}{\bullet\circ} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{w}_1^{(1)} \\ \mathbf{0} \\ \mathbf{w}_3^{(1)} \end{bmatrix} \mathbf{x} + \begin{bmatrix} b_1^{(1)} \\ 0 \\ b_3^{(1)} \end{bmatrix} \right) + b^{(2)}$$

$$\overset{\bullet}{\circ\bullet} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{w}_1^{(1)} \\ \mathbf{w}_2^{(1)} \\ \mathbf{0} \end{bmatrix} \mathbf{x} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ 0 \end{bmatrix} \right) + b^{(2)}$$

$$\overset{\bullet}{\circ\circ} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{w}_1^{(1)} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \mathbf{x} + \begin{bmatrix} b_1^{(1)} \\ 0 \\ 0 \end{bmatrix} \right) + b^{(2)}$$

$$\overset{\circ}{\circ\bullet} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{0} \\ \mathbf{w}_2^{(1)} \\ \mathbf{0} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ b_2^{(1)} \\ 0 \end{bmatrix} \right) + b^{(2)}$$

$$\overset{\circ}{\bullet\circ} \longrightarrow y = \mathbf{w}^{(2)} \left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{w}_3^{(1)} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ b_3^{(1)} \end{bmatrix} \right) + b^{(2)}$$

Figure 5.3: The complete PWA representation for the simple network in Fig. 5.1. For clarity, the activation states of each ReLU (denoted σ) have been colour-coded, where a filled/hollow dot corresponds to activity/inactivity, respectively. Each piece computes a copy of this transformation, with some rows set to zero.

We add a layer with a single neuron to the network, as shown in Fig. 5.4. The input space of the second layer now consists of multiple regions, each defined by a unique activation pattern π_i, within which the output of the first layer is an affine transformation given by $\mathcal{N}^{[1\,|\,\pi_i]}$. The output of the second layer is then:

$$\mathcal{N}^{[2\,|\,\pi_i]}(\mathbf{x}) = \sigma^{[2]}\left(\mathbf{w}^{[2]}\mathcal{N}^{[1\,|\,\pi_i]}\mathbf{x} + b^{[2]}\right) \tag{5.15}$$

The switching boundary of the second layer is then given by $\mathbf{w}^{[2]}\mathcal{N}^{[1\,|\,\pi_i]}\mathbf{x} + b^{[2]} = 0$. As the input $\mathbf{x}$ moves between regions, the activation pattern of the first layer will abruptly change. Therefore, the switching boundary of the second layer is continuous but appears to "bend" whenever a neuron of the previous layer changes its activation, as seen in Fig. 5.4. This phenomenon is generally seen whenever a neuron boundary crosses any boundary of a previous layer, as illustrated in Fig. 5.5.

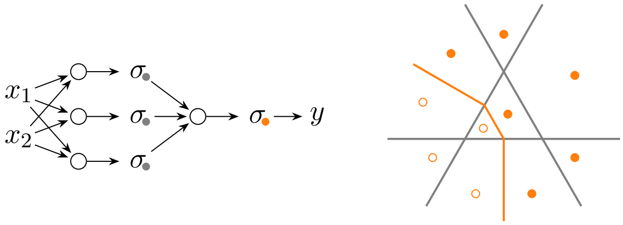


Figure 5.4: Illustration of the linear regions in Fig. 5.1, after adding a layer with one neuron. The input to the second layer varies with the activation pattern of the first layer, resulting in a piecewise continuous boundary where each piece is a distinct hyperplane.

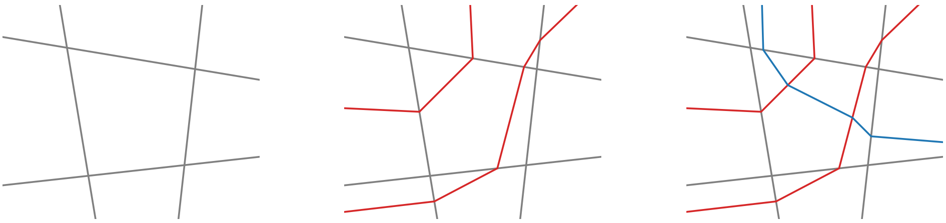


Figure 5.5: Diagram of the switching boundaries of successive layers. The boundaries of each neuron "bend" at the boundaries of previous layers.

So far, we have seen that PWA activation functions induce a hyperplane arrangement at each layer, where the arrangement's cells are the network's linear regions. Furthermore, neurons in previous layers modify the switching behaviour of neurons.

5.1.4 Representing the linear regions

The previous examples demonstrated how the PWA representation might be obtained when the activation pattern π_i is known and describe the structure of the linear regions. What now remains is to explicitly compute the regions, which are polyhedra [224, 225]. The most practical approach is to define the regions using the hyperplanes themselves, known as the H-representation, where the region is defined as the intersection of the half-spaces defined by the hyperplanes [226]. If the bounding hyperplanes have indices $\mathcal{H} = \{1, 2, \ldots, n\}$, then the polyhedron $\mathcal{P}$ can be written as:

$$\mathcal{P} = \{\, \mathbf{x} \mid \mathbf{w}_i^\top \mathbf{x} + b_i \geq 0, \ \forall i \in \mathcal{H} \,\} \tag{5.16}$$

Alternatively, this can be written as the matrix inequality:

$$\mathbf{H}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} = \begin{bmatrix}\mathbf{w}_1^\top & b_1\\ \vdots & \vdots\\ \mathbf{w}_n^\top & b_n\end{bmatrix}\begin{bmatrix}\mathbf{x}\\1\end{bmatrix} \geq 0 \tag{5.17}$$

In particular, the matrix representation makes it easy to quickly test whether a point is contained within the region, to compute an internal point by finding the Chebyshev centre and can be used to check for intersections between polytopes of different dimensions [227, 222]. A drawback is that there may be redundant constraints, which can slow down later operations. Identifying and removing the redundant constraints is also generally expensive, as this involves solving an LP for each hyperplane. However, heuristics exist to reduce this number [228].

Finding the regions defined by the neuron boundaries of a layer with PWA activation is equivalent to finding the regions of a hyperplane arrangement. A compact and rigorous discussion of hyperplane arrangements is given by Stanley [229]. Zavslavsky's Theorem gives an upper bound on the maximal number of regions for n hyperplanes in $\mathbb{R}^d$ [230].

$$\sum_{j=0}^{d}\binom{n}{j} \tag{5.18}$$

The number of regions increases quickly with d and n. A surface plot of Eq. (5.18) is given in Fig. 5.6. The regions may be found by iteratively bisecting a growing collection of regions, as illustrated in Fig. 5.7. If the bisecting hyperplane is given by

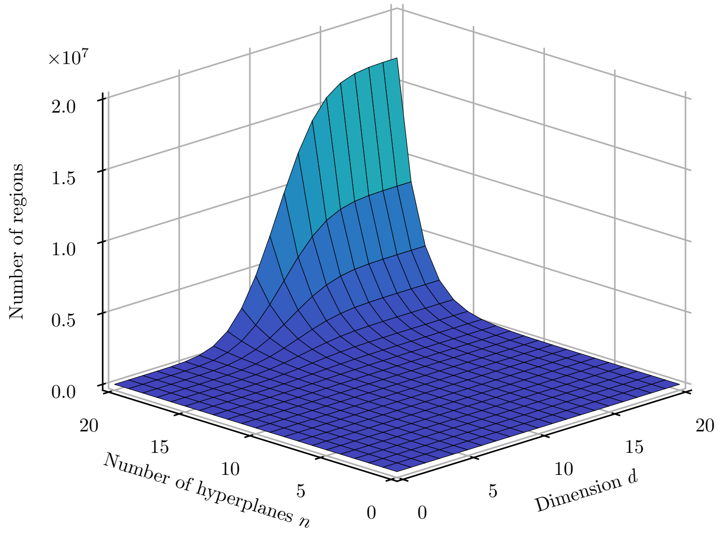


Figure 5.6: Growth of Zavslavsky's upper bound for the number of regions in a hyperplane arrangement.

$\mathbf{w}_{\text{bi}}^{\top}\mathbf{x} + b_{\text{bi}} = 0$, then the H-representations of the positive and negative half-spaces are given by:

$$\begin{aligned} \mathbf{h} &= \begin{bmatrix} \mathbf{w}_{\text{bi}} & b_{\text{bi}} \end{bmatrix} \quad \text{(positive half-space)} \\ \text{-}\mathbf{h} &= \begin{bmatrix} \text{-}\mathbf{w}_{\text{bi}} & \text{-}\ b_{\text{bi}} \end{bmatrix} \quad \text{(negative half-space)} \end{aligned} \tag{5.19}$$

Then, assuming that the current region has the H-representation $\mathbf{H} = \begin{bmatrix}\mathbf{W} & \mathbf{b}\end{bmatrix}$, the new H-representations of the two subregions will be:

$$\begin{aligned} \mathbf{H}^{+} &= \begin{bmatrix} \mathbf{W} & \mathbf{b} \\ \mathbf{w}_{\text{bi}}^{\top} & b_{\text{bi}} \end{bmatrix} \\ \mathbf{H}^{-} &= \begin{bmatrix} \mathbf{W} & \mathbf{b} \\ \text{-}\mathbf{w}_{\text{bi}}^{\top} & \text{-}b_{\text{bi}} \end{bmatrix} \end{aligned} \tag{5.20}$$

The regions $\mathbf{H}^{+}$ and $\mathbf{H}^{-}$ can be understood as the intersections between $\mathbf{H}$ and the positive/negative half-spaces on either side of the bisecting hyperplane. Therefore the superscript corresponds with the neuron activation associated with the hyperplane. If the hyperplane bisects the region, neither $\mathbf{H}^{+}$ nor $\mathbf{H}^{-}$ are empty sets. If the hyperplane does not bisect the region, then exactly one of the subregions is an empty set. Whether the matrix $\mathbf{H}$ represents an empty set requires solving a phase I

Algorithm 5.1 Add hyperplane $\mathbf{h}$ to region tree $\mathcal{T}$

```
function AddToTree(node $\mathcal{T}$, hyperplane $\mathbf{h}$, index $n$)
    $(\mathbf{H}, \mathcal{C}, \pi) \leftarrow \mathcal{T}$    ▷ Extract region, children, and activation
    $\mathbf{H}^+, \mathbf{H}^- \leftarrow (\mathbf{H} \cap \mathbf{h}), (\mathbf{H} \cap -\mathbf{h})$    ▷ Eq. (5.20)
    if $\mathbf{H}^+ = \emptyset$ then    ▷ $\mathbf{H}^+ = \emptyset$ and $\mathbf{H}^- \neq \emptyset$
        $\pi \leftarrow \pi \cap \{(n, -)\}$
    else if $\mathbf{H}^- = \emptyset$ then    ▷ $\mathbf{H}^+ \neq \emptyset$ and $\mathbf{H}^- = \emptyset$
        $\pi \leftarrow \pi \cap \{(n, +)\}$
    else    ▷ Hyperplane intersects with region $\mathbf{H}$
        if $\mathcal{C} = \emptyset$ then
            $\mathcal{C} \leftarrow \{(\mathbf{H}^+, \emptyset, \{(n, +)\}), (\mathbf{H}^-, \emptyset, \{(n, -)\})\}$    ▷ Add children
        else
            for child $\mathcal{T}_c \in \mathcal{C}$ do
                AddToTree($\mathcal{T}_c$, $\mathbf{h}$, $n$)    ▷ Recurse over children of node
            end for
        end if
    end if
end function
```

LP. The hyperplanes can then be considered individually, checking for intersections with all the regions found so far and bisecting when there is an intersection.

A naive implementation would replace each region once it has been bisected, maintaining a flat set of regions. However, the search space can be reduced significantly by retaining all regions and subregions in a binary tree, which we refer to as the *region tree* $\mathcal{T}$. Intuitively, if a hyperplane does not intersect a region, it will not intersect any of its subregions, allowing us to rule out all descendants of that region. The number of pruned regions increases with the depth of the tree.

The ith node of the region tree is represented as the tuple $\mathcal{T}_i = (\mathcal{P}_i, \mathcal{C}_i)$, where $\mathcal{P}_i$ is the associated region and $\mathcal{C}_i$ is the set of children of the node. Then, for every hyperplane $\mathbf{h}$ in the arrangement $\mathcal{H}$, we attempt to add $\mathbf{h}$ to the tree by recursively checking for intersections. Algorithm 5.1 shows the pseudocode for this procedure.

In addition to identifying which regions are feasible, we must keep track of each node's activation pattern π_i. Fortunately, this information can be extracted from the node feasibility check. Whenever a neuron obtains a particular activation ($+$ or $-$), this must also hold for all descendant nodes. The activation pattern at each node can then be recovered by traversing the tree depth-first. The pseudocode is shown in Algorithm 5.2.

Algorithm 5.2 Depth-first traversal of region tree to find activation patterns

```
function TRAVERSE(node $\mathcal{T}$, activation pattern $\pi$)
    $(\mathbf{H}, \mathcal{C}, \pi_{\text{partial}}) \leftarrow \mathcal{T}$
    for each $(n, a) \in \pi_{\text{partial}}$ do
        $\pi[n] \leftarrow a$
    end for
    for each child $\mathcal{T}_c \in \mathcal{C}$ do
        TRAVERSE($\mathcal{T}$, $\pi$)
    end for
    if $\mathcal{C} = \emptyset$ then
        yield $(\mathcal{T}, \pi)$
    end if
end function
```

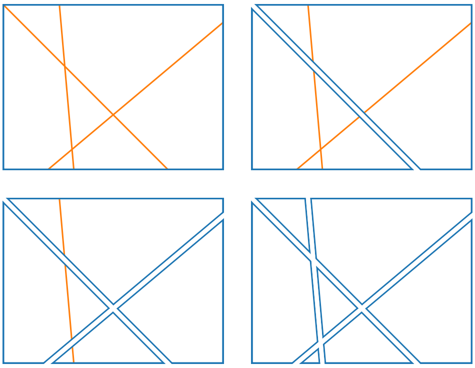


Figure 5.7: Illustration of a procedure for finding the regions of a hyperplane arrangement. Each hyperplane is considered and is used to bisect the previously found regions by adding it to their H-representations. At each iteration, searching all previously found regions for intersections is necessary. The search space can be significantly reduced by checking the parent regions first and storing the regions in a binary tree structure, adding new nodes every time a hyperplane bisects a region.

5.1.5 Algorithm

As shown in the examples, a neural network can be converted to its PWA representation in an iterative fashion, starting at the input layer. Using Algorithm 5.1, we can associate each region of a hyperplane arrangement as the leaf node of a binary tree, where the remaining nodes represent the ancestor regions. Now we must compute the network output within each region and handle the switching behaviour across layers.

Algorithm 5.3 Compute linear regions of network $\mathcal{N}$

```
function GETLINEARREGIONS(network $\mathcal{N}$, initial region $\mathcal{P}_0$)
    $\mathcal{T}_0 \leftarrow \{\mathcal{P}_0, \emptyset, \emptyset\}$                    ▷ Initialise region tree
    $\mathcal{W} \leftarrow \{(\mathcal{T}_0, \mathbf{I})\}$                       ▷ Initialise working set
    $n \leftarrow 0$, $k \leftarrow 1$                  ▷ Initialise neuron and layer indices
    for each layer $(\boldsymbol{\sigma}, \mathbf{W}, \mathbf{b}) \in \mathcal{N}$ do
        for each $(\mathcal{T}, \mathbf{P}) \in \mathcal{W}$ do
            for each row $[\mathbf{w}, b]$ of $[\mathbf{W}, \mathbf{b}]$ do
                $\mathbf{h} \leftarrow [\mathbf{w}, b]$
                ADDTOTREE($\mathcal{T}$, $\mathbf{Ph}$, $n$)
                $n \leftarrow n + 1$
            end for
        end for
        $\mathcal{W} \leftarrow \emptyset$
        leaf nodes $\mathcal{A} \leftarrow$ TRAVERSE($\mathcal{T}_0$)
        for each $(\mathcal{T}_i, \pi_i) \in \mathcal{A}$ do
            Compute $\mathbf{P}_i = \mathcal{N}^{[k \mid \pi_i]}$                                ▷ Eq. (5.12)
            $\mathcal{W} \leftarrow \mathcal{W} \cap \{(\mathcal{T}_i, \mathbf{P}_i)\}$
        end for
        $k \leftarrow k + 1$
    end for
end function
```

In Algorithm 5.3 the nodes $\mathcal{T}_i$ with a known linear transformation $\mathbf{P}_i$ are stored in the *working set* $\mathcal{W}$. Every element in $\mathcal{W}$ is a tuple of the form $(\mathcal{T}_i, \mathbf{P}_i)$, where $\mathcal{T}_i$ is a polyhedral region and $\mathbf{P}_i$ is a matrix that defines the affine transformation computed within that region. In practice, $\mathbf{P}_i$ might be implemented as an additional attribute on $\mathcal{T}_i$ or in a hash table indexed by the corresponding activation pattern π_i. The neural network $\mathcal{N}$ is represented as the tuple $(\boldsymbol{\sigma}, \mathbf{W}, \mathbf{b})$, corresponding to Eq. (2.20).

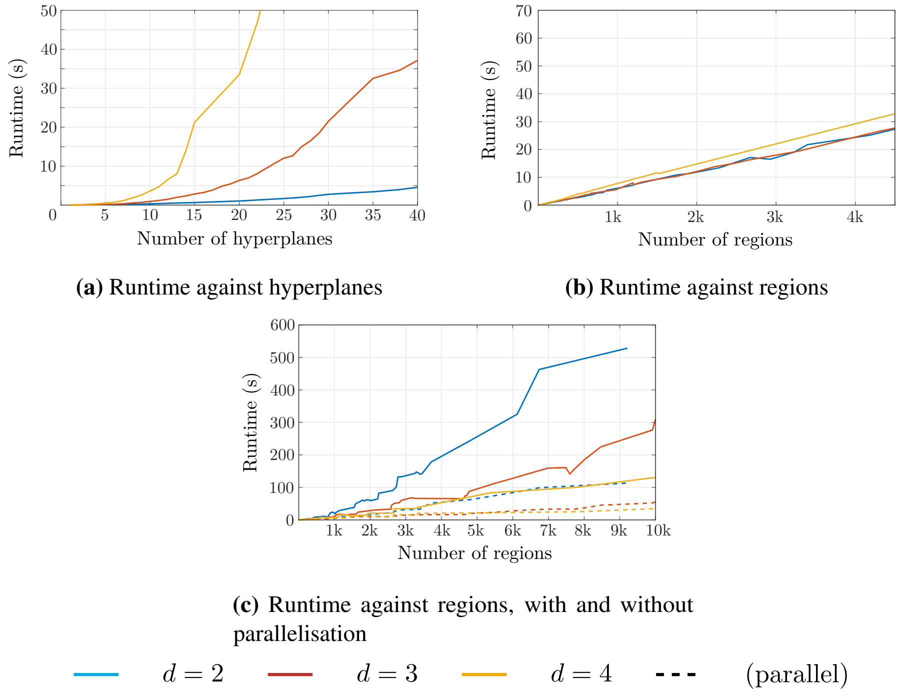
As the size of the working set increases after processing each layer, it is clear that the worst-case performance of the algorithm is highly dependent on the total depth of the network. However, it is not clear how quickly the working set will grow. For example, some regions in the working set may be intersected multiple times by the node boundaries in the next layer, while others will not. Despite this, the problem is inherently parallelisable. When parsing a network layer, the hyperplane arrangement problem is solved separately for each region in the working set, allowing for significant speedups when many cores are available.

5.1.6 Results

All runtimes were measured using a machine with a 6-core, 3.5 GHz processor and 16 GB of RAM. The polyhedral computations described in previous sections were performed using the MPT toolbox for MATLAB©. The results for Algorithm 5.3 in terms of the number of hyperplanes have been presented together in Fig. 5.8(a). The runtimes for Algorithm 5.3 are also presented regarding the number of regions in Fig. 5.8(b), showing that the runtime is roughly proportional to the number of regions found. The effect of increasing the input dimension (and thus the size of the required LPs) is almost negligible in comparison, suggesting that it is the high number of calls to the LP solver rather than the size of the LPs that dominates the time complexity of Algorithm 5.3. As the LPs are relatively small, choosing an LP solver with a low amount of presolving might yield significant improvements. Our implementation used the default LP solver included with MATLAB© (`linprog`), which is often outperformed by other solvers.

The runtime of the main algorithm was measured with and without parallelisation on the available six cores. The runtime as a function of the number of regions of the final network is shown in Fig. 5.8(c). Networks with an input dimension of up to four were processed as the number of regions quickly exploded, and the runtimes became intractable for networks with larger input dimensions. The runtime increases exponentially with the size of the network. Parallelisation was very effective, with the performance increasing by a factor approaching the number of cores used (i.e. six cores). The per-region cost decreases with the input dimension, suggesting an efficiency gain when increasing the input dimension. However, the corresponding points on the lines represent networks of very different sizes for any given number of regions. In theory, networks with two inputs and three hidden layers with ten neurons might have similar region counts as networks with four inputs and two hidden layers with five neurons. However, the small network will likely take longer to process because it has an additional hidden layer and more neurons.

As previously mentioned, PWA functions are widely used to represent complex dynamical systems. NNs are not as commonly used due to the difficulty of reasoning about their behaviour. However, it is possible to train a neural network on dynamical data and then retrieve its PWA form. The algorithm is now applied to a neural network with two inputs. Each output is plotted separately as a surface, and the linear network regions are plotted in the plane. The neural network was given two hidden layers, with 15 and 5 neurons, respectively. The network was trained on the dynamics described by Example 2.1.1, using the following parameter values: g = 9.81, m = 1, L = 5, and d = 0.1. This can be reformulated as a system of first order ODE where $x_1 = \theta$ and $x_2 = \dot{\theta}$:



(a) Runtime against hyperplanes

(b) Runtime against regions

(c) Runtime against regions, with and without parallelisation

Figure 5.8: Plots of Algorithm 5.3's runtime against the number of hyperplanes and found regions. Increasing the input dimension d significantly increases the runtime due to the increased number of regions.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{L} \sin x_1 - \frac{d}{m} x_2 \end{bmatrix} \tag{5.21}$$

A training dataset was created by sampling $\mathbf{x} = [\theta, \dot{\theta}]$ 50000 times from the continuous uniform distribution $\mathcal{U}(-\pi, \pi)$ and the normal distribution $\mathcal{N}(0, 5)$ respectively, creating a sample of states $\mathbf{x} = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^\top$. The corresponding $\dot{\mathbf{x}}$ was then found through Eq. (5.21). Then the neural network was trained on the data using the Adam (derived from "adaptive moment estimation") optimiser with a learning rate of 0.003 for 50 epochs, finally achieving a root mean square error (RMSE) of $1.615 \cdot 10^{-4}$. The true and learned dynamics were then simulated using the MATLAB© function `ode45`. Fig. 5.9 compares the two. The NN's complete PWA form is shown in Fig. 5.10 as a pair of surface plots, along with the 116 linear regions.

Interestingly, the linear regions show a concentration of horizontal boundaries

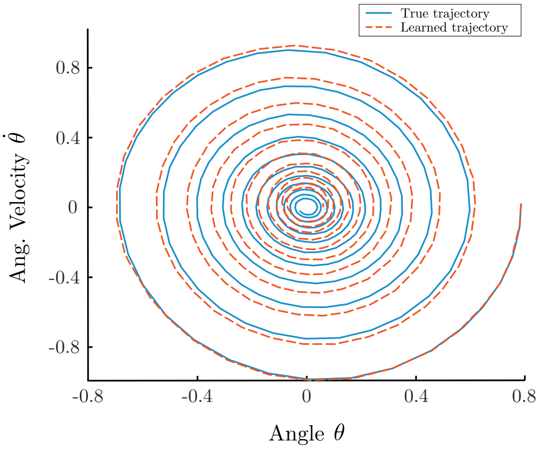
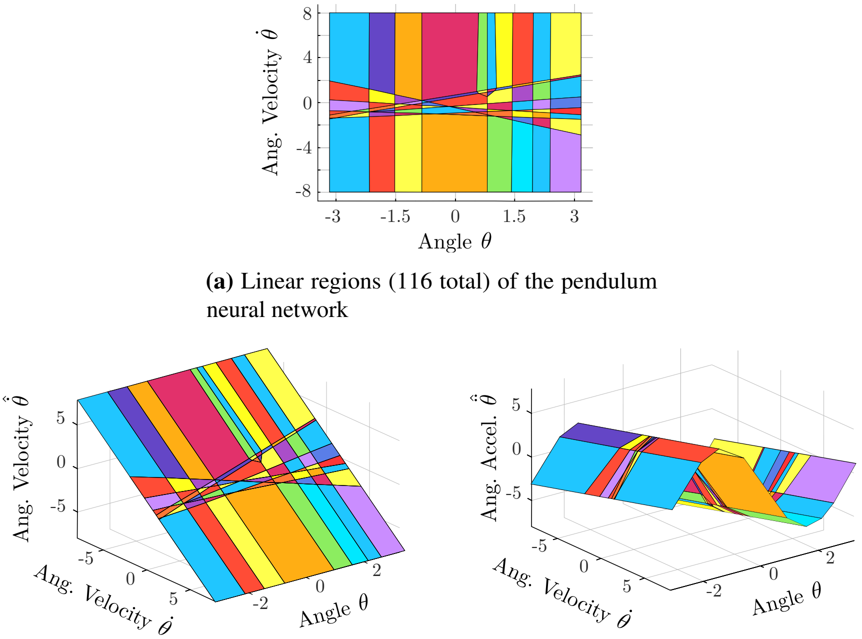


Figure 5.9: True and simulated trajectory using the neural network with $\theta_0 = \frac{\pi}{4}$. The network displays some asymmetries in its trajectory, suggesting that the learned pendulum would swing slightly higher on one side. It also appears to converge slightly off-centre of the origin because the neural network does not assume energy conservation.

around $\dot{\theta} = 0$, along with several vertical boundaries. Because the network is locally linear, the boundaries determine any changes in gradient. It is, therefore, likely that the concentration of horizontal boundaries serves to give the two outputs a constant slope in the $\dot{\theta}$ direction (see Figs. 5.10(b) and 5.10(c)). Likewise, the vertical boundaries form large sheets arranged in a sinusoidal shape that approximates Eq. (2.2). It is interesting to see such a structure emerging during the training process. However, there are still some irregularities related to the large number of small regions between closely packed boundaries. These small regions are numerous but highly redundant as they do not contribute significantly to the shape of the output.

Analysing the stability of such a representation can be challenging because it requires keeping track of all possible transitions between regions. For example, this is done in energy-based methods [214]. It may therefore be desirable to take steps to simplify the PWA representation either during or after the training process by merging boundaries that appear redundant or by introducing new boundaries. Adding a form of regularisation that forces similar connection weights for neurons in the same layer to converge together could help reduce the effective region count. The architecture of a network could then be simplified by merging neurons with very similar weights. Likewise, if the network performs poorly in a particular region of the state space, neurons can be split in two, introducing additional boundaries.



(a) Linear regions (116 total) of the pendulum neural network

(b) First output of the network: $x_1 \approx \dot{\theta}$

(c) Second output of the network: $x_2 \approx \ddot{\theta}$

Figure 5.10: The complete PWA form of the NN that was trained to imitate a pendulum. The linear regions appear to have arranged themselves in patterns supporting the output's shape.

5.1.7 Related work

Studies of the linear regions of NNs started with the need to understand how *expressive*[1] these networks are, and how this changes with the network architecture [225, 224, 231]. Expressivity is often measured using the Vapnik-Chervonenkis (VC) dimension [232], and tight bounds have been found for the VC dimension of PWA NNs [233]. Heuristics for the expressivity of PWA networks have also been developed [234]. Empirical evidence strongly suggests that increasing the depth of a network has a more significant impact on expressivity than increasing the width of existing layers [223, 235]. Serra et al. present upper and lower bounds on the maximum number of regions that improve on previous results, along with a mixed-integer formulation from which the regions can be counted by enumerating the integer solutions [218]. They established that for a network with input dimension d, number of hidden layers L, each with n nodes and ReLU activation, the asymptotic

[1] A more expressive network can compute more complex, rich functions.

bounds for the maximal number of regions are:

$$\begin{aligned} \text{Lower:} \quad & \Omega\Big((\tfrac{n}{d})^{(L-1)d} n^d\Big) \\ \text{Upper:} \quad & \mathcal{O}(n^{dL}) \end{aligned} \tag{5.22}$$

This upper bound is exponential in both d and L. Unfortunately, the most useful NNs are those with large input dimension d and many hidden layers L. The number of linear regions of such a network is enormous. It is likely due to this that there have been a limited number of studies into identifying these regions. There have been studies on approximating nonlinear NNs with PWA functions [236]. Conversely, work has been done on the inverse problem of representing PWA functions more compactly as NNs [237].

Chapter 12 of [238] shows how feedforward, convolutional, and recurrent NNs using ReLU or perceptron activation functions can be explicitly rewritten as a binary regression tree with hyperplane splits. This transformation is shown in Fig. 5.17 for a feedforward neural network with an activation function that shows a single transition at $c = 0$. This brute-force approach yields trees of depth N with 2^N leaf nodes, where N is the number of neurons in the network. Bertsimas et al. [238] show that much shallower regression trees with depths between $4 - 8$ yield comparable performance to 2-layer NNs with 256 neurons in each layer on some regression tasks, hinting at the idea that some of the inner complexity of the network is redundant.

5.1.8 Discussion

The previous sections presented an algorithm to obtain the PWA representation of a neural network using ReLU activation functions. Results demonstrating conversions of randomly initialised NNs with up to four dimensions and three layers were reported, the largest of which had 31835 linear regions. A parallelised version of the algorithm could perform this conversion on a standard desktop computer in around a minute. With more computational resources and further algorithmic optimisations, it is clear that much larger networks will be able to be converted.

The examples demonstrated the algorithm for networks with fully connected layers and ReLU activations only. However, the approach may be generalised to any linear layer and arbitrary PWA activation functions (for example, leaky ReLU). The family of linear layers includes convolutional layers, normalisation layers, and networks with more complex branching architectures, which encompass most architectures in use today.

The input dimension of the network is a significant source of complexity, limiting this approach to networks with fewer inputs. Alternatively, the method may be applied to a subset of the network's inputs, i.e. finding the linear regions in an input subspace. Using the method together with dimensionality reduction techniques is an approach that shows great promise for the study of complex systems that resist analysis.

5.2 Regularising Piecewise Affine Neural Networks

An issue with Algorithm 5.3 is that it does not scale well to realistically sized NNs. However, it is common to employ some form of regularisation during training, e.g. the sparsity inducing ℓ_1 regularisation used in Section 4.4. It stands to reason that this must also influence the linear regions of the network being trained and possibly reduce the total number. Xiao et al. [239] report that robustness verification algorithms run faster on networks that are suitably regularised during training. Specifically, *weight sparsity* and *ReLU stability* were found to be the most significant factors, where ReLU stable neurons are defined to have a constant effect within a certain perturbation radius of the input $\mathbf{x}$. The adversarial robustness of the network is also related to the minimum distance between the input $\mathbf{x}$ and the decision boundary or the nearest linear region boundary [240].

Despite the observations that standard forms of regularisation appear to be useful for training easily verifiable, compressible, and robust models, there is still a lack of understanding of how regularisation affects the linear regions of a neural network. This section aims to visualise and quantify this relationship by plotting the linear regions and reporting their size and number during training, with the hope that this will yield greater insight into the inner workings of these networks. The results of applying various regularisation strategies to a neural network are presented and discussed in Section 5.2.2 before we conclude with a short note on the implications of the work in Section 5.2.3.

5.2.1 Experiments

This section describes the setup of the experiments that were performed. First, we discuss the investigated regularisers, then the network architecture used in all the experiments is explained, and finally, the dataset and the training procedure are presented.

Regularisers

Regularisation has been defined as "any modification to a learning algorithm that aims to reduce generalisation error" [60], which can include techniques such as weight decay, dropout, dataset augmentation, adding noise, parameter tying, batch normalisation, weight normalisation, and early stopping, among many others. To limit the scope of the work, we only consider ℓ_1 and ℓ_2 weight decay, weight normalisation (with weight decay), and dropout. ℓ_1 and ℓ_2 weight decay are standard techniques that penalise the 1 and 2 norms of the network parameters, respectively, scaled by a parameter λ. We do not apply weight decay to the bias of the network. Weight normalisation is a reparameterisation of the connection weights s.t. $\|\mathbf{w}\| = g$ proposed by Salimans et al. [241]:

$$\mathbf{w} = g\frac{\mathbf{v}}{\|\mathbf{v}\|} \tag{5.23}$$

Salimans et al. show that weight normalisation has similar properties to batch normalisation, with the added effect of separating the "direction" of the connection weights (in a vector sense) from their magnitude. The output of the neuron (with input $\mathbf{x}$ to the layer) is $\sigma(g\ \mathbf{vx}/\ \|\mathbf{v}\| + \mathbf{b})$. The smallest distance between the origin and the activation boundary of the ith neuron in the first hidden layer is $d_i = b_i/g_i$. The scalar g now modulates the strength of the output of the neuron, but increasing g will also move the neuron boundary closer to the origin, and vice versa.

Network architecture

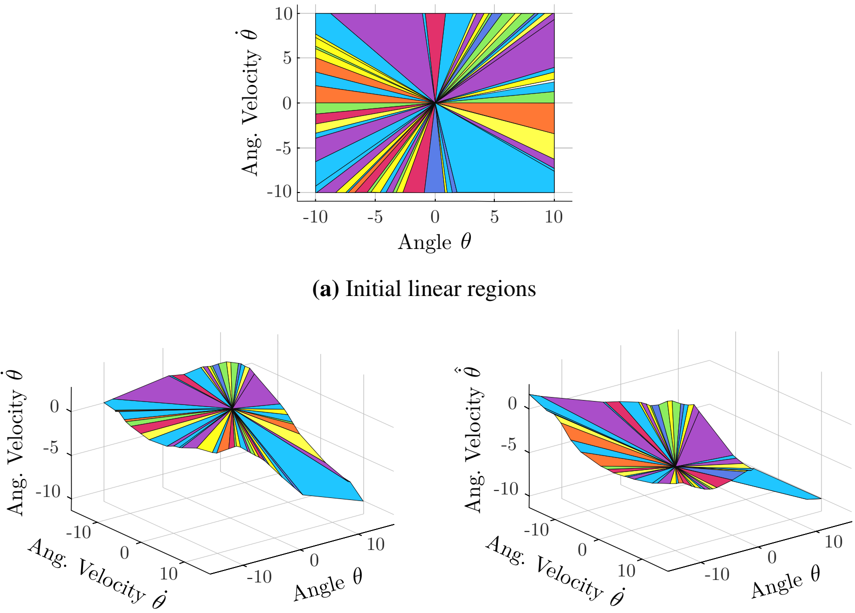
Despite the availability of algorithms, the full PWA representation can only be found for relatively small NNs due to the typically large number of linear regions. The network's size is further constrained by the need to visualise the regions at many points during the training process. The architecture was therefore kept relatively small, with two inputs/outputs and two hidden layers with 20 and 10 neurons, respectively. Reducing the size of the network below this resulted in inconsistent training convergence. Both hidden layers were given ReLU activation, as this is the most popular piecewise linear activation function. The same neural network architecture was used for all the experiments. The initial weights of the network were set to be the same in all the experiments, with zero initial bias. A visualisation of the linear regions and the two outputs of the network is shown in Fig. 5.11.

Dataset

A simple synthetic dataset for a damped pendulum (Example 2.1.1 with unit parameters) was chosen, which can be represented by the following ODEs:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin x_1 - x_2\end{bmatrix} \tag{5.24}$$

Note that when the pendulum is dropped from the topmost position ($x_1 = \pi$), the maximum angular velocity is bounded by $|\dot{x}_1| = |x_2| < 0.87$. A dataset with



(a) Initial linear regions

(b) First output

(c) Second output

Figure 5.11: The randomly initialised network used in all experiments. Note that the network has no initial bias, and therefore all neuron boundaries initially intersect the origin.

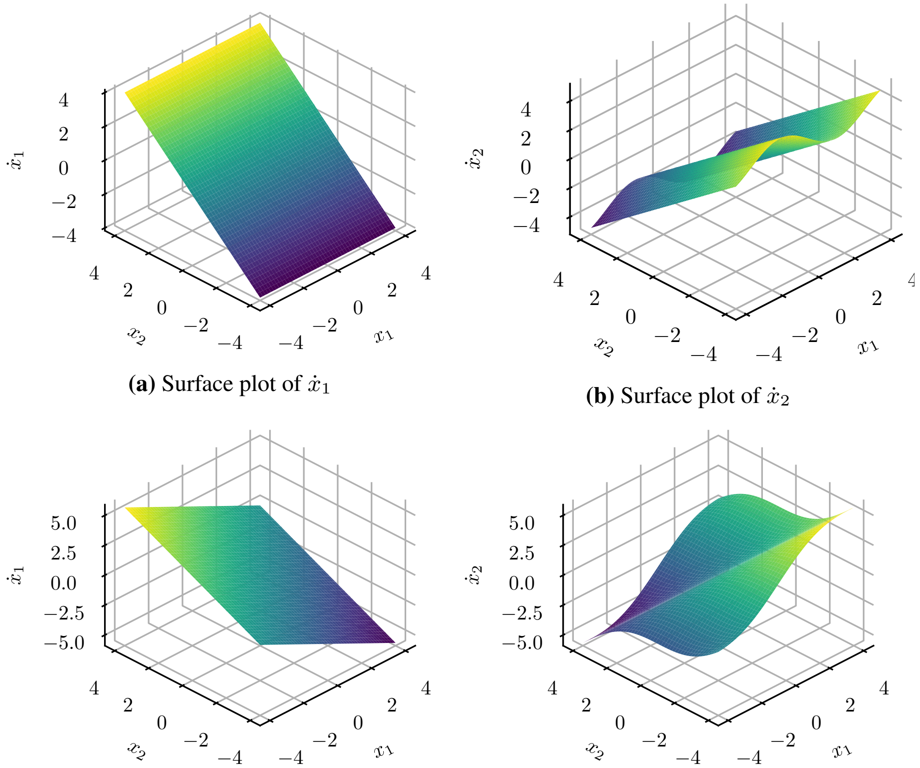
a reasonable range of values can therefore be sampled from an input space of $[-\pi, \pi] \times [-\pi, \pi]$, which has the advantage (for visualisation purposes) of being square. The training set was generated by uniformly sampling 10000 points from the chosen input space, while 20% of the points were reserved for validation. The surface described by Eq. (5.24) is shown in Figs. 5.12(a) and 5.12(b). While running the experiments, the regularisers were observed to display a tendency to align the neuron boundaries with the axes. This kind of alignment is a useful bias for the pendulum example. To see why, note that the first variable $\dot{x}_1$ is linear and has no curvature. The Jacobian of $\dot{x}_2$ is:

$$\mathbf{J}_2 = \begin{bmatrix} 0 & 0 \\ \sin x_1 & 0 \end{bmatrix} \tag{5.25}$$

We see that the surface only curves in the x_1 direction. The pendulum model Eq. (5.24) is therefore well described by networks with neuron boundaries parallel to the y-axis. In order to test the regularisers on an example where this is not true, a new dataset was generated by setting $x_1 \leftarrow \frac{1}{\sqrt{2}}(x_1 + x_2)$ and $x_2 \leftarrow \frac{1}{\sqrt{2}}(x_1 - x_2)$:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}}(x_2 - x_1) \\ -\sin \frac{1}{\sqrt{2}}(-x_1 - x_2) - \frac{1}{\sqrt{2}}(x_2 - x_1) \end{bmatrix} \tag{5.26}$$

This corresponds to the dynamics of Eq. (5.24), but rotated $\pi/4\,\mathrm{rad}$ about the z-axis (see Figs. 5.12(c) and 5.12(d)).



(a) Surface plot of $\dot{x}_1$

(b) Surface plot of $\dot{x}_2$

(c) Rotated $\dot{x}_1$

(d) Rotated $\dot{x}_2$

Figure 5.12: (a) and (b) are the two outputs of the normal pendulum dynamics, while (c) and (d) show the rotated outputs. The rotated outputs have a curvature that is not aligned with the axes and should therefore alter the behaviour of sparsity-inducing regularisers.

Training

Each experiment was run for 200 epochs, with a batch size of 40 (250 iterations per epoch), using the ADAM optimiser with an initial learning rate of $\eta = 0.001$ and exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In each experiment, the linear regions after 40 epochs (10000 batch iterations) are shown, and the epoch that performed the best overall on the validation set is reported. The total number of regions and regions overlapping with the dataset's domain are shown throughout the training process. However, due to computer precision issues, the regions could only be found on a 1000×1000 square, so the total number of regions may be more than reported.

5.2.2 Results

The results are presented in the following order: standard ℓ_1 and ℓ_2 weight decay, weight normalisation with weight decay, dropout with and without weight normalisation, and weight decay and dropout on the rotated dataset. The results of each experiment are summarised in 4 plots, which are as follows. First, the linear regions after 40 epochs of training (10000 batch iterations) are plotted, followed by the regions of the model with the lowest validation loss. The regions are plotted on the domain $[-10, 10] \times [-10, 10]$ to give a better overview. Note that this is larger than the domain of the dataset, which is $[-\pi, \pi] \times [-\pi, \pi]$. Then the total number of regions (along with the number of regions that overlap with the dataset domain, any regions outside this can be considered constant influence regions) are plotted over the training process. Finally, the log area of the regions is presented.

Weight decay

Both ℓ_1 and ℓ_2 penalties were tested with various penalty parameters λ. Weight decay was only applied to the connection weights, not the bias values. The results for $\lambda = 0.001$ and $\lambda = 0.0001$ are shown in Fig. 5.13. The network trained without regularisation showed a roughly constant number of regions within the dataset domain while the total number of regions increased. The distribution of region areas also spread out. ℓ_1 and ℓ_2 weight decay strongly align the neuron boundaries with the axes.

Both regularisers significantly reduced the number of regions within the dataset's domain and the total number of regions. ℓ_1 weight decay was much more aggressive in this respect, as many neuron boundaries appear to have been pushed away from the origin. These observations are supported by the area plots, which show a strong and sudden increase in the area of the regions. This increase occurs around the same time as a sharp fall in the total number of regions, which is likely an artefact of the method used to calculate the regions, which limited the computations to a 1000×1000 square. If any neuron boundaries were pushed beyond this, the number of reported regions would appear to drop. These observations can be explained by inspecting the output of the ith neuron in the first layer:

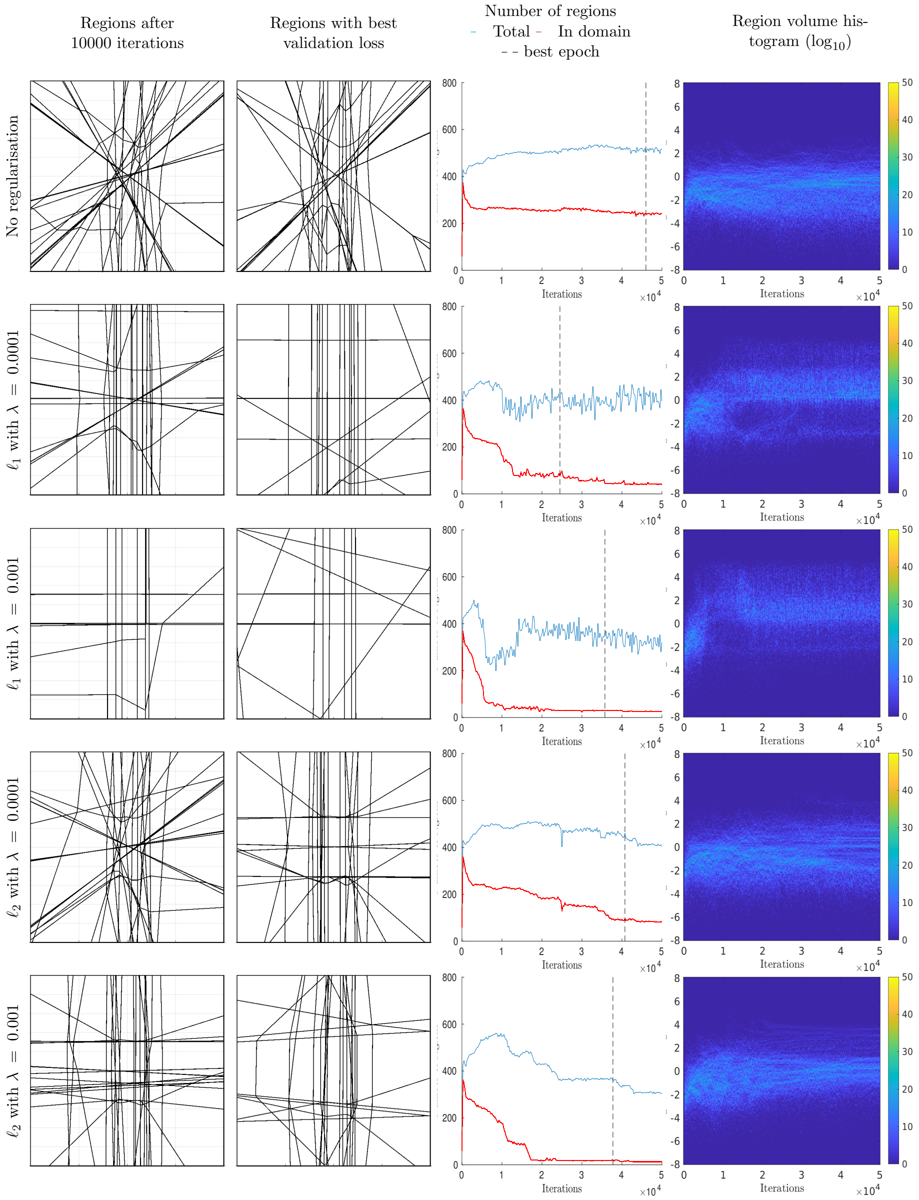


Figure 5.13: Linear regions of a neural network after applying ℓ_1 and ℓ_2 weight decay with different penalty parameters λ. The number of regions overlapping with the dataset domain and the total number of regions are shown. Finally, the log area of the regions is presented throughout the training process.

$$\sigma(z) = \sigma\left(\mathbf{w}_i^{[1]}\mathbf{x} + b_i^{[1]}\right) \tag{5.27}$$

where $z = 0$ represents the switching hyperplane of this ReLU neuron. The smallest distance between the origin and the boundary of the ith neuron in the first hidden layer is then:

$$d_i^{[1]} = \left|b_i^{[1]}\right| \left\|\mathbf{w}_i^{[1]}\right\|_2^{-1} \tag{5.28}$$

This distance for a neuron in the second layer will depend on the first layer's activation pattern π. For simplicity, we assume that all 1st layer neurons are active. The origin-boundary distance for the ith second layer neuron is then:

$$d_i^{[2]} = \left|\mathbf{w}_i^{[2]}\mathbf{b}^{[1]} + b_i^{[2]}\right| \left\|\mathbf{w}_i^{[2]}\mathbf{W}^{[1]}\right\|^{-1} \tag{5.29}$$

Forcing the connection weights $\mathbf{W}^{[1]}$ and $\mathbf{W}^{[2]}$ to be small while keeping the biases $\mathbf{b}^{[1]}$ and $\mathbf{b}^{[2]}$ unchanged will also increase this distance.

Weight normalisation with weight decay

Fig. 5.14 shows that weight normalisation appears to slow down the previously observed effects of weight decay, as the number of regions within the domain decreased more slowly. Furthermore, even without weight decay, the number of regions within the dataset domain after 200 epochs appears to be less than in Fig. 5.13, although the evolution appears fairly similar. The total number of regions still increased similarly. The axes alignment is observed in all cases, although ℓ_1 weight decay appears to enhance this.

Interestingly, the neuron boundaries were not pushed away from the origin as quickly as previously observed. Training with the weaker ℓ_2 weight decay was almost indistinguishable from unregularised training. These results support the findings of van Laarhoven, who showed that both batch and weight normalisation counteract ℓ_2 weight decay [242].

Dropout

Dropout layers were added to the first and second hidden layers. The small size of the network meant that increasing the dropout rate above 0.4 resulted in training convergence issues. Results for dropout rates 0.1 and 0.4 are shown in Fig. 5.15, with and without weight-normalised layers. Again, we see that the neuron boundaries quickly align with the axes.

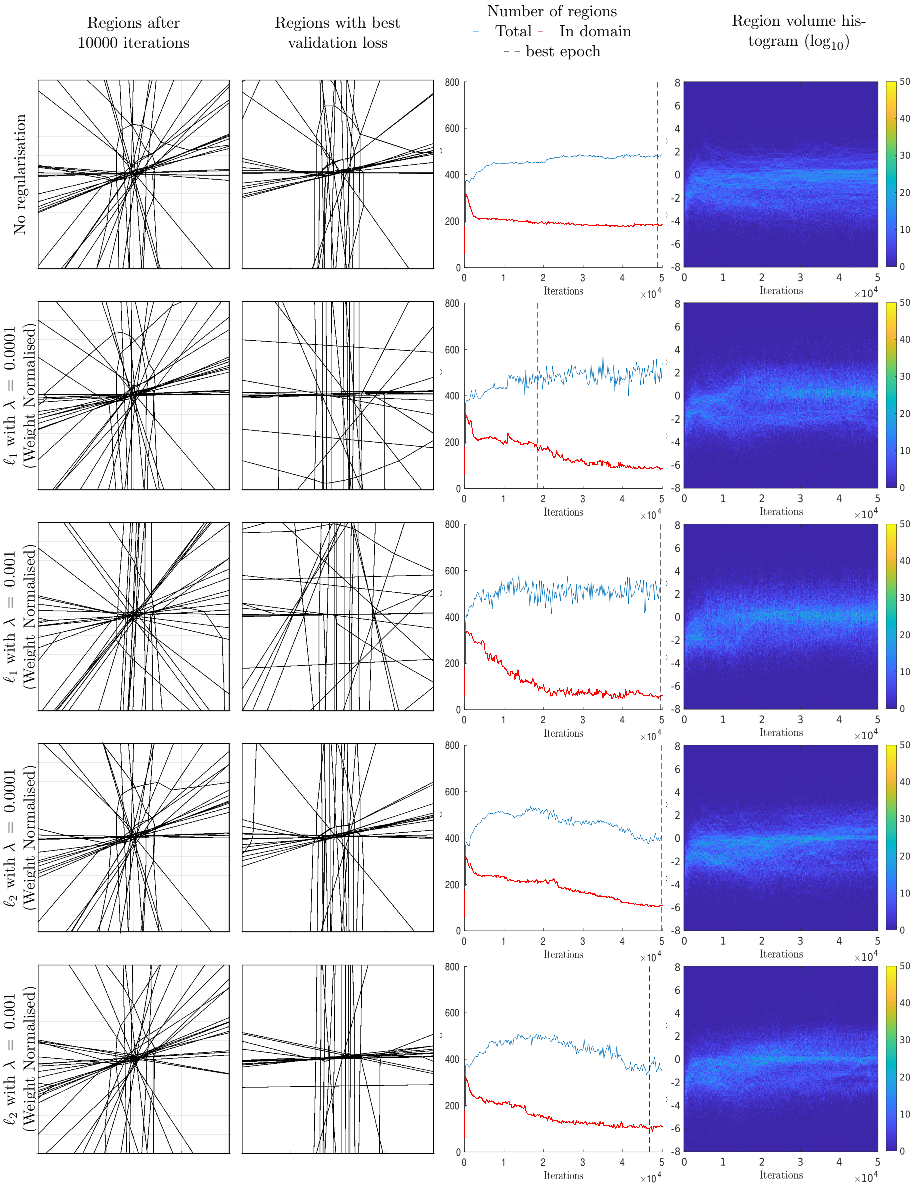


Figure 5.14: The number of linear regions after weight normalisation was applied to the fully connected layers. ℓ_1 and ℓ_2 weight decay were applied with the same decay parameters. As before, the regions are plotted on the domain $[-10, 10] \times [-10, 10]$.

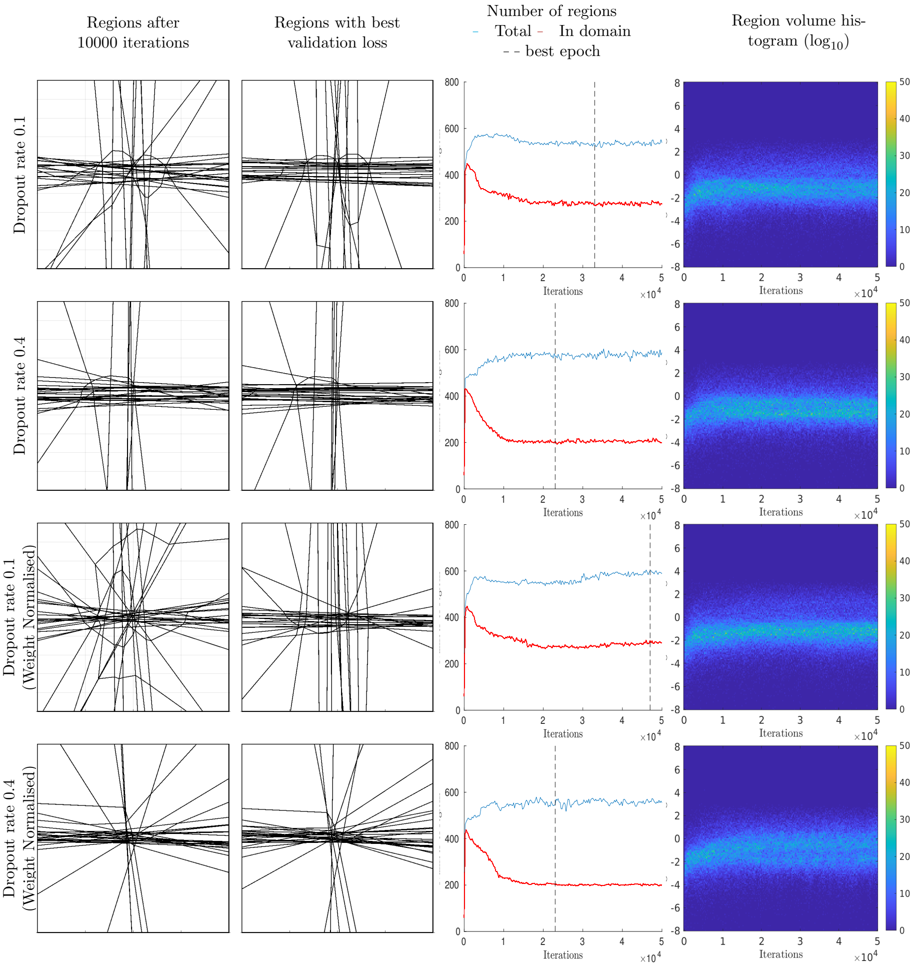


Figure 5.15: The effect of dropout layers on the linear regions of the network. The small size of the network meant that increasing the dropout beyond roughly 0.4 resulted in convergence issues during training. As before, the regions are plotted on the domain $[-10, 10] \times [-10, 10]$.

A common feature is a large cluster of horizontal boundaries, resulting in more regions than in previous results. The dataset is linear in this direction (see Fig. 5.12), suggesting that these neurons are highly redundant and can be replaced with a single linear function.

Rotated dataset

A good model for this new dataset should have very similar neuron boundaries to the models seen in Fig. 5.13, but rotated $\pi/4$ degrees. In this case, the “optimal” weights will not be sparse, so the training objective and the weight decay regularisation penalty are now in conflict. Fig. 5.16 shows that the evolution of the number of regions and their size was similar to the results with the original dataset but slower. For example, the same transition in region size is seen for ℓ_1 weight decay, but around 10000 iterations (40 epochs) later than in Fig. 5.13. Additionally, after 200 epochs, ℓ_2 weight decay resulted in 100 more regions inside the data domain relative to Fig. 5.13, an increase of 50%.

Overall the boundaries of the weight-decayed networks intersect more with each other, exhibiting less collinearity than previous results. The number of regions should therefore increase to the increased number of intersections. In contrast, the network with dropout layers has a strikingly similar distribution of regions that strongly resembles the corresponding result in Fig. 5.15, with a rotation of $\pi/4$.

5.2.3 Discussion

There is much to be learned from studying the piecewise linear structure of NNs. This work has some important takeaways.

First, weight decay tends to push neuron boundaries away from the origin, inducing dead or stable neurons with ReLU activations. This effect may be desirable when the model is compressed or verified after training. The results show that ℓ_1 regularisation is particularly effective at introducing neuron stability, while weight normalisation was observed to slow down this effect significantly.

When weight decay is strong enough to induce weight sparsity (or near-sparsity), this has the effect of aligning neuron boundaries with the axes of the input space. This alignment yields networks with fewer linear regions due to fewer intersections between neuron boundaries, which is desirable for robustness verification and model compression. However, this was observed to cause slower training convergence on data with curvature in other directions. Instead of inducing weight sparsity, it may be interesting to induce collinearity in the connection weight matrix.

Dropout induced collinearity in the neuron boundaries and appeared invariant to the dataset’s rotation, yielding a model with similar linear regions in both cases.

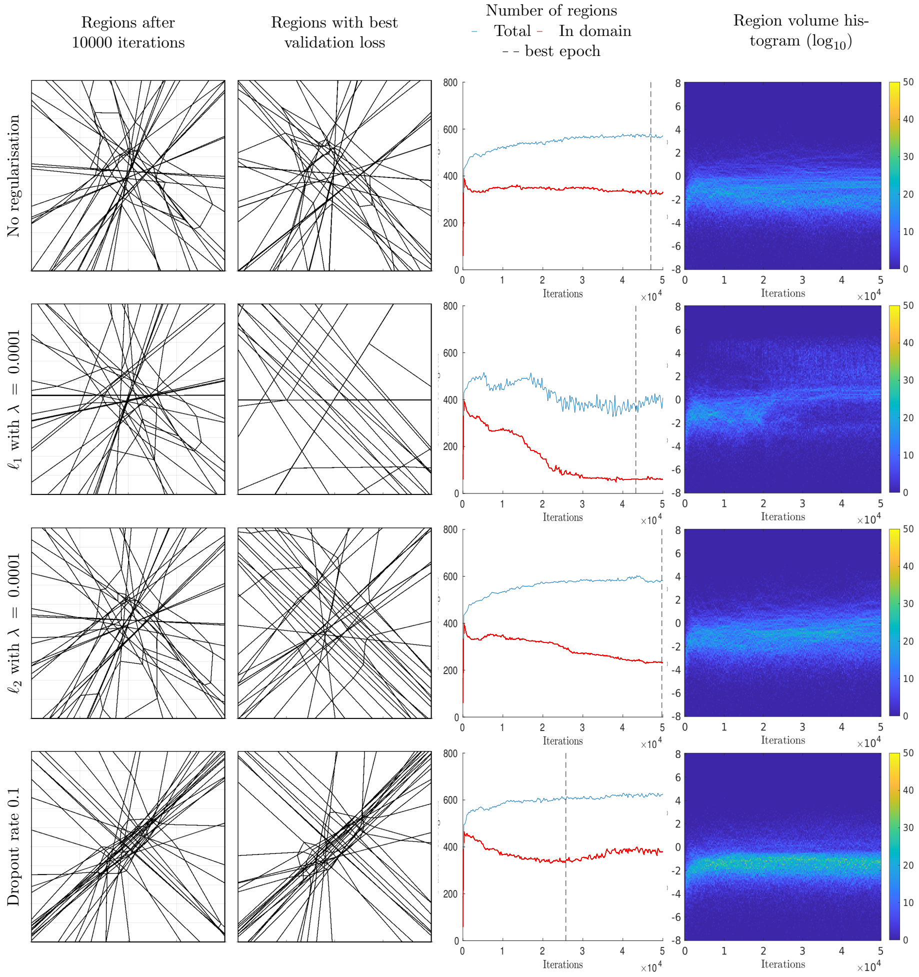


Figure 5.16: The number, size, and visualisations of the linear regions of various networks after training on the rotated dataset. As before, the regions are plotted on the domain $[-10, 10] \times [-10, 10]$.

However, it introduced significant redundancy by clustering parallel neuron boundaries, resulting in a network with significantly more linear regions than if weight decay were used. The small distance between data points and neuron boundaries may also cause robustness issues [240]. However, this effect was only observed in the direction where the dataset was linear.

5.3 Approximating Piecewise Affine Neural Networks

Despite the apparent ability of regularisation to reduce the number of regions, the decrease is insufficient to make NN verification methods practical. Instead, we may make more progress by approximating the problem. The literature supports this, as there are innumerable papers describing how NNs can be compressed effectively without losing performance (see Marino et al. [243] for a recent survey). Indeed, research shows that much of the performance of large NNs can be distilled down to a much smaller sub-network [193]. This section has two contributions: (i) An empirical investigation into the efficacy of ℓ_1 regularisation and weight pruning for reducing the number of linear regions. (ii) A novel method to approximately decompose a neural network into a PWA function.

The problem can be formulated as follows: Given a neural network $\mathcal{N}(\mathbf{x})$ trained on the dataset $\mathcal{D}$ with data $(\mathbf{y}_k, \mathbf{x}_k) \in \mathcal{D}$, $\mathbf{y}_k \in \mathbb{R}^{n_y}$, $\mathbf{x}_k \in \mathbb{R}^{n_x}$, compute a PWA function that closely approximates $\mathcal{N}(\mathbf{x})$ while maintaining a sufficiently low number of regions.

5.3.1 Related work

Tøndel et al. [244] present a method to efficiently compute PWA functions as regression trees with hyperplane splits, also known as *oblique regression trees*. The early days of neural network research have produced many methods that attempted to extract rules or decision trees from networks. A survey by Andrews et al. [245] categorises the rule extraction methods as either *decompositional* (utilise the internal structure of the network), *pedagogical* (treat the network as a black box), or *eclectic* (a mixture of both). An example of a pedagogical approach is ANN-DT [246], where a tree is grown by selectively sampling the output of a neural network. CRED [247] is a decompositional approach proposed that applies the regression tree method C4.5 [248] to a single hidden layer network, obtaining intervals of possible values for each hidden neuron for a given output. Zilke et al. extend the CRED approach to multilayer networks by constructing rules for each layer based on the output of the previous layer [249]. These methods differ from the exact decompositional methods in that they only allow a single variable to branch at a time.

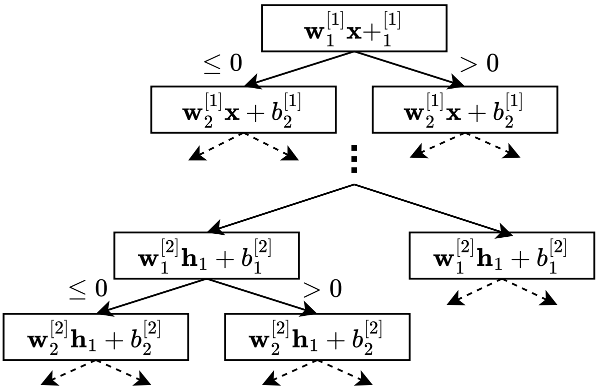

$\mathbf{w}_1^{[1]}\mathbf{x}+_1^{[1]}$
≤ 0
> 0
$\mathbf{w}_2^{[1]}\mathbf{x} + b_2^{[1]}$
$\mathbf{w}_2^{[1]}\mathbf{x} + b_2^{[1]}$
$\vdots$
$\mathbf{w}_1^{[2]}\mathbf{h}_1 + b_1^{[2]}$
$\mathbf{w}_1^{[2]}\mathbf{h}_1 + b_1^{[2]}$
≤ 0
> 0
$\mathbf{w}_2^{[2]}\mathbf{h}_1 + b_2^{[2]}$
$\mathbf{w}_2^{[2]}\mathbf{h}_1 + b_2^{[2]}$

Figure 5.17: Interpreting a neural network with ReLU activation functions as a tree by branching once per neuron. Each node defines a convex set of points defined by the constraints of its ancestors. Note that many infeasible branches can be pruned, e.g. a branch with constraints $x_1 \leq 1$ and $x_1 \geq 2$.

5.3.2 Proposed method

Our method aims to construct a linear regression tree as shown in Fig. 5.17 that approximates NNs with the PWA activation function given in Eq. (5.4). Recall from Section 5.1.2 that $\sigma(z)$ computes either $a_1 z$ or $a_2 z$, depending on whether $z > 0$ or not. For a single input $\mathbf{x}$, this results in a distinct activation for every neuron in the network. The output of the neuron will transition between these states at $z = 0$. We define this as the *neuron boundary*. For the ith neuron in the kth layer, the corresponding neuron boundary is:

$$z_i^{[k]} = \mathbf{w}_i^{[k]}\mathbf{f}^{[k-1]} + b_i^{[k]} = 0 \tag{5.30}$$

If a neuron boundary intersects a region $\mathcal{P}$, then it splits $\mathcal{P}$ into two new affine pieces determined by the constraints $z_i^{[k]} < 0$ and $z_i^{[k]} \geq 0$. If not, the neuron has a purely affine output within $\mathcal{P}$. Section 5.3.2 shows how Eq. (5.30) can be simplified to:

$$\mathbf{h}\begin{bmatrix}\mathbf{x}\\-1\end{bmatrix} = \mathbf{0} \tag{5.31}$$

This makes it easy to determine whether to split a region. The core idea of the algorithm is to build a tree of these splits by iterating through each neuron, determining which leaf nodes to split by computing Eq. (5.30) and assessing its feasibility

Algorithm 5.4 PWA approximation

```
function ApproximateNet(initial region $\mathbf{H}_0$, network $\mathcal{N}$)
    Tree root $\mathcal{T}_0 \leftarrow (\mathbb{R}^d, \emptyset)$
    for neuron $\mathcal{N}_k \in \mathcal{N}$ do
        $\mathcal{T}_0 \leftarrow$ search($\mathcal{T}_0, \mathcal{N}_k$, $\mathbf{H}_0$)
    end for
    return $\mathcal{T}_0$
end function

function search($\mathcal{N}_k$)
    if truncated($\mathcal{T}_i$) then
        Do nothing
    else if StopCriterion($\mathcal{T}_i$) then
        $\mathbf{h} \leftarrow$ GetConstraint($\mathcal{T}_i$)
        ApproxOutput($\mathcal{N}_k$, $\mathcal{N}$, $\mathbf{H}$)
    else if HasChildren($\mathcal{T}_i$) then
        $\mathcal{L}, \mathcal{R} \leftarrow$ left($\mathcal{T}_i$), right($\mathcal{T}_i$)
        $\mathbf{h} \leftarrow$ GetConstraint($\mathcal{T}_i$)
        search($\mathcal{L}$, $\mathcal{N}_k$, $\mathbf{H} \cap \mathbf{h}$)
        search($\mathcal{R}$, $\mathcal{N}_k$, $\mathbf{H} \cap -\mathbf{h}$)
    else
        $\mathbf{h} \leftarrow$ NeuronBoundary($\mathcal{N}_k$)
        if $\exists\mathbf{x}$ s.t. $\mathbf{H}\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] \leq 0$ and $\mathbf{h}\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] = 0$ then
            SetConstraint($\mathcal{T}_i$, $\mathbf{h}$)
            Left($\mathcal{T}_i$) $\leftarrow$ new node
            Right($\mathcal{T}_i$) $\leftarrow$ new node
        end if
    end if
end function
```

w.r.t. the constraints of the current branch, and locally approximating $\mathcal{N}(\mathbf{x})$ when a stopping criterion has been met. The procedure is summarised below, and the pseudocode for the full procedure is presented in Algorithm 5.4. The method can be summarised as follows:

1. Initialise the tree with a node $\mathcal{T}_0$ with an initial constraint set $\mathbf{H}_0\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] \leq 0$ that defines the domain of the dataset $\mathcal{D}$.
2. Iterate through the neurons in each layer. For each neuron, $\mathcal{N}_i$, traverse the tree $\mathcal{T}_0$ and accumulate the node constraints on the current branch into

the matrix $\mathbf{H}$. At each leaf node $\mathcal{T}_j$, compute the neuron boundary $\mathbf{h}$. If $\mathbf{h} \cap \mathbf{H}$ is feasible, split $\mathcal{T}_j$ into two children with constraints $\mathbf{h}\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] < 0$ and $\mathbf{h}\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] \geq 0$. See Section 5.3.2.

3. The tree expansion can be truncated when a node meets some *stopping criterion*. See Section 5.3.2.
4. After all neurons have been processed, the affine function computed at each leaf node must be found, and the output for truncated nodes must be *approximated*. See Section 5.3.2.

The algorithm was implemented using the MPT toolbox [222] for MATLAB© to perform the polyhedral computations. MOSEK [250] was used as an optimisation backend.

Checking branch feasibility

As the algorithm traverses the tree to the node $\mathcal{T}_i$, it accumulates the constraints of the parents into the matrix $\mathbf{H}$. The feasibility of these constraints could be evaluated by solving a phase I LP:

$$\begin{aligned}\mathbf{x}^* &= \min_{\mathbf{x},\mathbf{z}} \ \mathbf{1z}\\ \text{s.t.}\mathbf{H}\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] - \mathbf{z} &\leq 0\end{aligned} \tag{5.32}$$

with initial values $\mathbf{x} = \mathbf{0}$ and $\mathbf{z} = \mathbf{H}[\mathbf{0}^\top 1]^\top$. If a solution $\mathbf{z} = 0$ is found, the constraint set $\mathbf{H}$ is feasible. Alternatively, the problem can be transformed via Farka's lemma. However, modern solvers are good at checking feasibility, and a 'dummy' objective function of $\mathbf{0}$ can be used.

Stopping criteria

This work uses a stopping criterion based on the number of data samples within the region $\mathcal{P}$. If $N = |\mathcal{D}|$ and $n_p = |\{\mathbf{x} \mid \mathbf{x} \in \mathcal{D}, \mathbf{H}\left[\begin{smallmatrix}\mathbf{x}\\-1\end{smallmatrix}\right] \leq 0\}|$, we define the stopping criterion as:

$$n_p/N = r_d < r_d^{\min} \tag{5.33}$$

where $r_d^{\min}$ is referred to as the *data ratio*. This metric has the advantage of being relatively efficient to compute for a node $\mathcal{T}_j$ (check $\mathbf{HX} \geq 0$ for all data $\mathbf{X}$ contained by parent node), and naturally adapts the resolution of the approximation to match the relative amount of available data. A disadvantage is that it requires access to the original training data of the network.

What did not work The following alternative stopping criteria were considered:

1. Node reaches maximum depth d_{max}
2. Local region $\mathcal{P}$ has volume less than V_{min}
3. Local region $\mathcal{P}$ has diameter less than D_{min}

Criterion (1) is simple to implement and requires negligible computation. However, it yielded a large spread in region sizes (in terms of volume and diameter), which appeared to cause overfitting where there were small regions and underfitting where there were large regions. Criteria (2) and (3) solve this by measuring the size directly. However, despite recent work in this area (e.g. Emiris et al. [251]), estimating these properties is extremely expensive for high-dimensional polytopes. Another issue shared by (1),(2), and (3) is that they can yield regions containing no elements of $\mathcal{D}$, which can complicate the approximation step.

Approximation schemes

If a node has not been truncated, it represents one of the actual linear regions $\mathcal{P}$ of $\mathcal{N}$ and the local affine function can be determined via Eq. (5.12). If the node was truncated, the output must be approximated instead. The following approach was chosen: sample some points $\mathbf{x}_i \in \mathcal{P}$, compute the corresponding outputs $\mathbf{y}_i = \mathcal{N}(\mathbf{x}_i)$, and perform a local linear regression via least squares:

$$\min_{\mathbf{A},\mathbf{a}} \sum_i (\mathbf{A}\mathbf{x_i} + \mathbf{a} - \mathbf{y}_i)^2 \tag{5.34}$$

The samples $\mathbf{x}_i$ were taken directly from $\mathcal{D}$, which worked well with the stopping criterion and neatly avoided the problem of out-of-distribution sampling that occurs with other methods (see “What did not work”). A drawback with this method is that no approximation can be made if there is no data in some region. Because the stopping criterion is based on the data ratio, this was never found to be a problem during our experiments.

What did not work The following alternative sampling strategies were also considered.

1. Normally distributed $\mathbf{x}_i$ around interior point $\mathbf{x}^*$.
2. Boundary points of $\mathcal{P}$ were sampled via random raycasting from the interior point $\mathbf{x}^*$.

Raycasting from $\mathbf{x}^*$ is performed by solving:

$$\mathbf{x}_i = \quad \mathbf{x}^* + \mathbf{r}_i t_i \tag{5.35}$$

where

$$\begin{aligned} t_i &= \min_j t \\ \text{s.t.}' t &\geq 0, \quad t = -\frac{\mathbf{h}_j \left[\begin{smallmatrix} \mathbf{x} \\ -1 \end{smallmatrix} \right]}{\mathbf{h}_j \left[\begin{smallmatrix} \mathbf{r}_i \\ 0 \end{smallmatrix} \right]} \end{aligned}$$

where $\mathbf{h}_j$ is the jth row of $\mathbf{H}$. Both approaches require an interior point $\mathbf{x}^*$, found by computing the Chebyshev centre after the feasibility check Eq. (5.32). Approach 2 only samples points on the boundary of $\mathcal{P}_k$ and was considered in an attempt to minimise the boundary discontinuity between adjacent regions. Both of these sampling methods were found to produce inferior results. We hypothesise that this is because they rely on samples that may lie outside $\mathcal{D}$'s data distribution.

5.3.3 Experiments

Algorithm 5.4 was tested on NNs trained on a nonlinear system identification benchmark based on the ground vibrations of an F-16 jet. The role of ℓ_1 regularisation and weight pruning on the number of regions was investigated using the approach described by Xiao et al. [239]. Algorithm 5.4 was applied with no stopping criterion in order to obtain the exact PWA representation where possible (a time budget of 1 hour was used). The best-performing model was approximated using Algorithm 5.4 with a range of values for $r_d^{\min}$ as a stopping criterion. The benchmark dataset was generated by attaching a shaker under the aircraft's right wing and measuring the resulting vibrations at three different sites [252]. Accelerometer readings were aggregated into three 400 Hz signals representing the acceleration at three distinct points: the excitation site (a_1), the wing surface (a_2), and a payload (a_3) shown in Fig. 5.18. The only inputs were the shaker force (F) and input voltage (V). We group the signals as $\mathbf{u}_k = (F(k), V(k))$ and $\mathbf{y}_k = (a_1(k), a_2(k), a_3(k))$, where k is the current timestep. The model performance was computed as the root mean squared error $\epsilon_{\mathcal{D}}$:

$$\epsilon_{\mathcal{D}} = \sqrt{\frac{1}{N} \sum_{k=1}^{N} \|\hat{\mathbf{y}}_k - \mathbf{y}_k\|^2} \tag{5.36}$$

Each variable of the data was normalised and differenced. Stationarity was checked using the augmented Dickey-Fuller test [253] and the Kwiatkowski-Phillips-Schmidt-Shin test [254]. Two regression datasets were constructed from the data:

- Dataset 1: Prediction variables; $\Delta\, a_1$, $\Delta\, a_2$, $\Delta\, a_3$ Regression variables: $\mathbf{u}_{k-1}$, $\mathbf{y}_{k-1}$ (5 variables)
- Dataset 2: Prediction variables; $\Delta\, a_1$, $\Delta\, a_2$, $\Delta\, a_3$ Regression variables: $\mathbf{u}_{k-1}$, $\mathbf{u}_{k-2}$, $\mathbf{y}_{k-1}$, $\mathbf{y}_{k-2}$ (10 variables)

Note that dataset 1 only includes first-order information, so we can expect the models to perform poorly here. However, this separation allows us to assess the effects of input dimension on the number of regions. Computing the exact PWA representation for models with more inputs was intractable. A linear regression model was fit to the data as a baseline. The chosen architecture had two hidden layers with 20 and 10 neurons, respectively, with ReLU activations. This choice was made to trade off sufficient model capacity and the tractability of computing the exact PWA representation, limiting our quantitative analyses to small NNs. The NN was trained using Tensorflow with the following loss function with ℓ_1 regularisation and the ADAM optimiser with default hyperparameters:

$$L(x) = \frac{1}{N}\sum_{k=1}^{N} \|\mathcal{N}(\mathbf{x}_k;\boldsymbol{\theta}) - \mathbf{y}_k\|_2^2 + \lambda_1 \sum_{i=1}^{2}\sum_{j=1}^{n(i)} \left\|\mathbf{w}_j^{[i]}\right\|_1 \tag{5.37}$$

where $N = |\mathcal{D}|$ is the number of data samples and $n(i)$ is the number of neurons in the ith layer. The weights $w_{k,i,j}$ were pruned by zeroing all weights below the threshold: $1 \cdot 10^{-3}$, as done by Xiao et al. [239].

5.3.4 Results

Table 5.1 and Table 5.2 show the results for datasets 1 and 2, respectively. The models trained on dataset 1 did not perform well due to the limited first-order inputs. They nonetheless demonstrate that ℓ_1 regularisation can drastically reduce the number of linear regions, although the effect on performance is unclear. For the model trained on dataset 2, ℓ_1 regularisation is observed to reduce the number of regions to practical levels for $\lambda_1 > 10^{-4}$. However, it also significantly degrades the performance of the NN. The computations were otherwise found to be intractable due to a sharp jump in complexity from $\lambda_1 = 5 \cdot 10^{-4}$ to $\lambda_1 = 10^{-4}$, which was consistent across several runs. Pruning using the threshold reported by Xiao et al. [239] appears to have little effect on both the performance and the number of regions. These results show that aggressive ℓ_1 regularisation only yields practical numbers of regions when the performance is worse than the linear model.

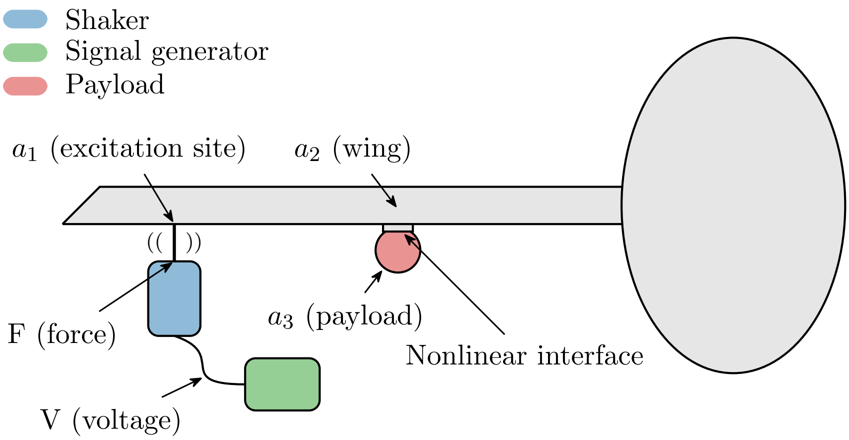


Figure 5.18: F-16 instrumentation and variables

Table 5.1: Comparison of models trained on Dataset 1. '-' entries did not finish within the time budget.

Model	Regions	Compute (s)	$\epsilon_{\mathcal{D}}$
(linear)	1	N/A	0.0785
$(\mathcal{N})$ $\lambda_1 = 0$	152388	1640	0.0779
+ Pruned 0/330	152388	1640	0.0779
$(\mathcal{N})$ $\lambda_1 = 10^{-5}$	181083	1960	0.0780
+ Pruned 1/330	181083	1950	0.0780
$(\mathcal{N})$ $\lambda_1 = 5 \cdot 10^{-5}$	126099	1370	0.0777
+ Pruned 19/330	126091	1370	0.0777
$(\mathcal{N})$ $\lambda_1 = 10^{-4}$	73740	787	**0.0751**
+ Pruned 53/330	73777	793	0.0751
$(\mathcal{N})$ $\lambda_1 = 5 \cdot 10^{-4}$	4280	47.5	0.0757
+ Pruned 185/330	4288	41.3	0.0757
$(\mathcal{N})$ $\lambda_1 = 10^{-3}$	773	9.83	0.0777
+ Pruned 220/330	772	8.465	0.0777
$(\mathcal{N})$ $\lambda_1 = 5 \cdot 10^{-3}$	8	1.25	0.0881
+ Pruned 300/330	8	0.852	0.0874
$(\mathcal{N})$ $\lambda_1 = 10^{-2}$	1	0.782	0.0759
+ Pruned 0/330	1	0.592	0.0759

Algorithm 5.4 was tested on the best-performing network on dataset 2, namely the network trained with $\lambda = 10^{-5}$. Table 5.3 shows the validation performance

Table 5.2: Comparison of models trained on Dataset 2. '-' entries did not finish within the time budget.

Model	Regions	Time (s)	$\epsilon_{\mathcal{D}}$
(linear)	1	N/A	$4.10 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 0$	-	-	$2.17 \cdot 10^{-2}$
+ Pruned 0/430	-	-	$2.17 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 10^{-5}$	-	-	$\mathbf{1.74 \cdot 10^{-2}}$
+ Pruned 21/430	-	-	$1.74 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 5 \cdot 10^{-5}$	-	-	$1.91 \cdot 10^{-2}$
+ Pruned 28/430	-	-	$1.92 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 10^{-4}$	-	-	$2.23 \cdot 10^{-2}$
+ Pruned 53/430	-	-	$2.23 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 5 \cdot 10^{-4}$	527	11.8	$3.37 \cdot 10^{-2}$
+ Pruned 296/430	526	10.2	$3.37 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 10^{-3}$	18	1.55	$4.08 \cdot 10^{-2}$
+ Pruned 362/430	18	1.09	$4.12 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 5 \cdot 10^{-3}$	30	2.41	$4.58 \cdot 10^{-2}$
+ Pruned 371/430	14	0.993	$5.94 \cdot 10^{-2}$
$(\mathcal{N})$ $\lambda_1 = 10^{-2}$	4	1.12	$6.44 \cdot 10^{-2}$
+ Pruned 380/430	4	8.06	$6.24 \cdot 10^{-2}$

Table 5.3: Algorithm 5.4 was applied to the network that performed best on dataset 2 using the data ratio as a stopping criterion.

$r_d^{\min}$	Regions	Time (s)	$\epsilon_{\mathcal{N}}$	$\epsilon_{\mathcal{D}}$
10%	61	50.5	$9.49 \cdot 10^{-3}$	$1.96 \cdot 10^{-2}$
5%	87	56.4	$8.93 \cdot 10^{-3}$	$1.95 \cdot 10^{-2}$
1%	285	77.8	$4.00 \cdot 10^{-3}$	$1.76 \cdot 10^{-2}$
0.5%	516	92.1	$2.23 \cdot 10^{-3}$	$1.75 \cdot 10^{-2}$
0.1%	1501	149	$7.24 \cdot 10^{-4}$	$1.74 \cdot 10^{-2}$
$\mathcal{N}$	-	-	0	$1.74 \cdot 10^{-2}$

of the resulting regression trees and the mean squared difference between the approximating tree and the model, denoted as $\epsilon_{\mathcal{N}}$. The approximation closely matches the network's output while keeping the number of regions and computation time practically low.

Fig. 5.19 compares the forecasting performance on dataset 2 of the linear regression model and the neural network trained with $\lambda_1 = 10^{-5}$. The linear model quickly

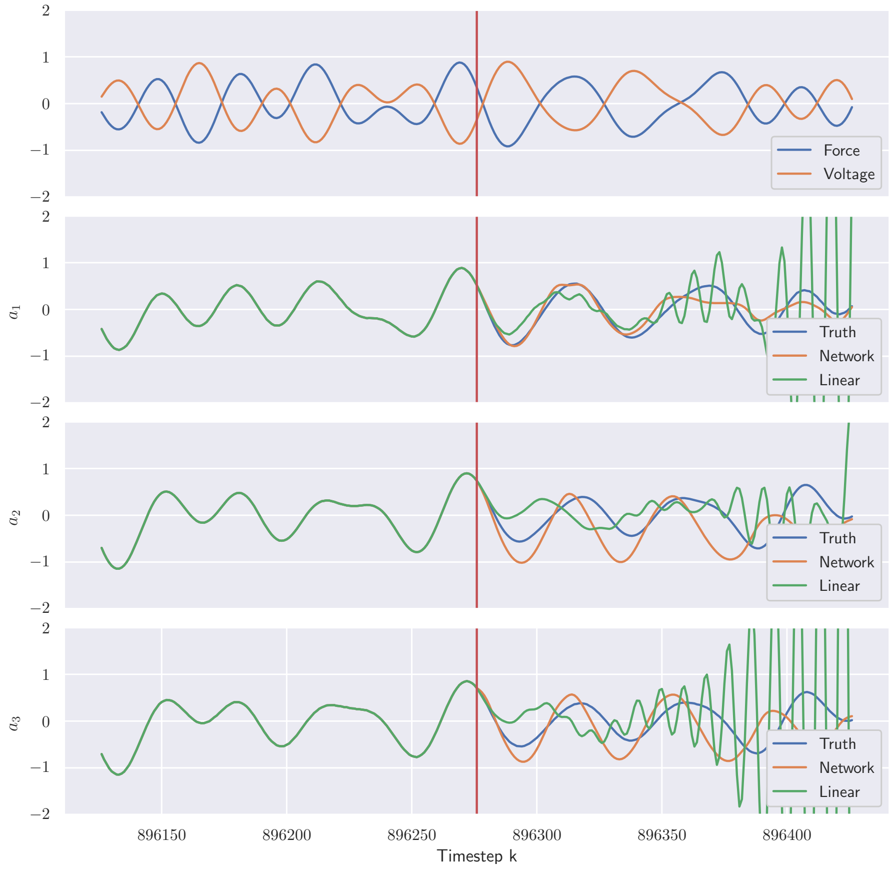


Figure 5.19: Forecast of the next 150 timesteps using linear regression and the neural network model trained with $\lambda_1 = 10^{-5}$. The forecast starts after the red line. The linear model quickly becomes unstable. The neural network appears to overshoot.

becomes unstable, while the neural network performs more consistently but tends to overshoot when forecasting. Fig. 5.20 compares the same forecast for the network and its PWA approximation computed using $r_d^{\min} = 0.1$. Despite being the coarsest approximation with only 61 regions, the approximation produces a similar forecast for all accelerations for the first 50 timesteps, with predictions for a_2 and a_3 later diverging slightly due to the accumulated error.

5.3.5 Discussion

A method for approximating NNs as a discontinuous PWA function $\mathbf{p}(\mathbf{x})$ was presented. The method was derived by reinterpreting existing exact PWA decomposition methods as a tree search that can be stopped early when an appropriate

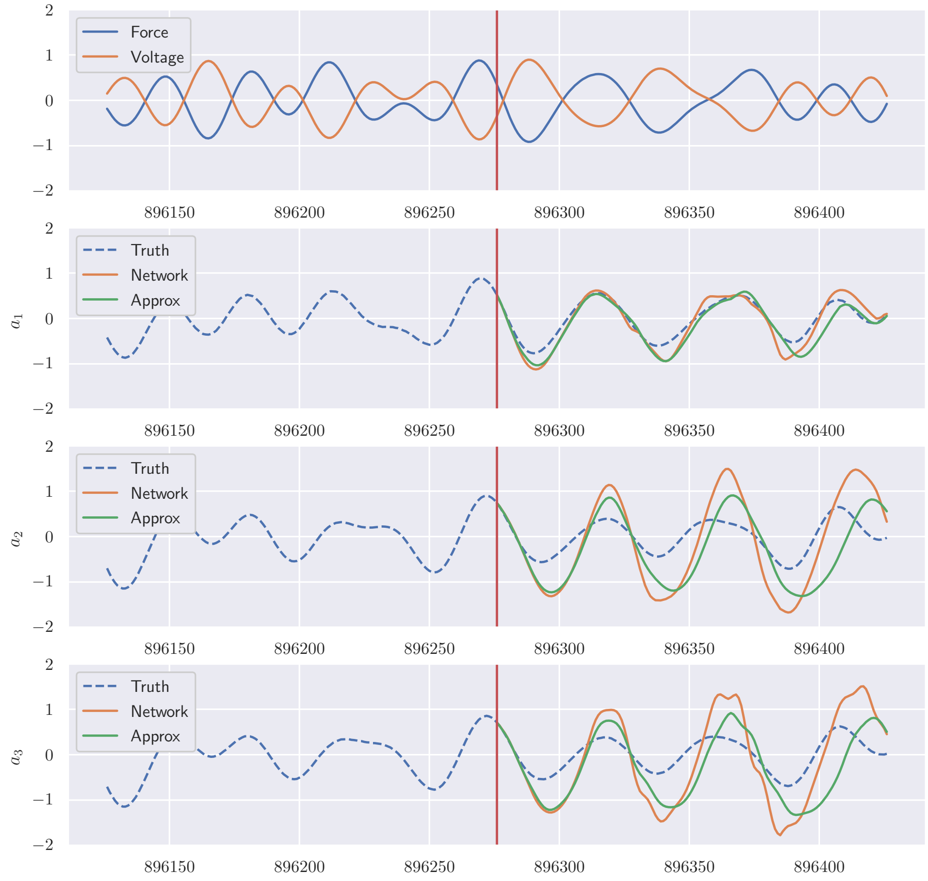


Figure 5.20: Forecast of the next 150 timesteps using the neural network model trained with $\lambda_1 = 10^{-5}$ and the PWA approximation with 61 regions computed using Algorithm 5.4. The forecast starts after the red line. The approximation is initially indistinguishable from the original network.

criterion is met. A suitable stopping criterion was proposed that allows users to control the trade-off between accuracy and model complexity via a single parameter r_d^{min}, which determines the minimum ratio of the dataset that any single affine piece of $\mathbf{p}(\mathbf{x})$ can enclose. As shown in Section 5.3.4, the approximation method is far more practical than the exact decompositional method, even when aggressive ℓ_1 regularisation and weight pruning are used during training. The proposed method has limitations: (i) The original training data is required. (ii) The approximation does not naturally extend to new data outside the identified linear regions. (iii) The resulting approximation is discontinuous. (iv) The method is potentially challenging to scale to networks with many inputs due to the complexity of solving Eq. (5.32) for high dimensional systems. Addressing these limitations would be an exciting line of future research. It is expected that the resulting approximations will be used to assist the analysis and interpretation of NNs and enable efficient control design in nonlinear system identification tasks.

Chapter 6

Conclusion

This thesis has investigated the use of machine learning (ML) techniques within three different contexts: (i) Control design, (ii) System identification, (iii) Verification. The following paragraphs summarise the findings and highlight potential research directions.

Chapter 3 investigated the use of reinforcement learning in the context of maritime vessels. Section 3.3 presents the work published in Paper A, where a Reinforcement learning (RL) agent was first trained to complete the conflicting tasks of path following and collision avoidance. Both tasks were described using hand-crafted reward functions, and the total reward was a weighted sum of the task-specific rewards using a trade-off parameter λ. However, choosing an optimal value of λ proved challenging, and for low values of λ, the collision avoidance rate dropped below 100%. To give the agent more "insight" into this tradeoff, λ was used as an additional input, and random values of λ were used during training. This additional insight into task priority allows the vessel's behaviour to be modulated during operation simply by varying λ, e.g. increasing the value when the vessel should behave more conservatively. A natural extension to this line of work could be to select a value of λ based on an estimate of the current collision risk. However, the results have shown that it is still challenging to guarantee 100% collision avoidance when using a RL agent. This difficulty should not discount the use of RL; autonomous vehicles are still an active research area, and RL can help identify effective control policies in complex scenarios or for challenging performance requirements. *Safe* RL is an emerging field that aims to improve the safety of RL. A promising framework for achieving this is the *predictive safety filter*, where a finite horizon optimal control problem (OCP) is solved to find a minimum modification of the input s.t. that the constraints are upheld. This auxiliary system can complement arbitrary controllers and is comparable to a mid-level collision avoidance (COLAV) system. Section 3.4 presents a predictive safety filter designed for the milliAmpere vessel, a small passenger ferry with two azimuthal thrusters. This contribution will be published in Paper E. The system was tested with a simple Line-of-Sight (LOS) controller for static and moving obstacles and was effective and efficient, running in real-time on a consumer laptop. The next steps in this line of research would be to make a robust safety filter with chance constraints and to test the system with a RL agent. It is an open question whether a safety filter can negatively affect the training of an RL agent. However, it seems likely that a safety filter can be used to generate additional reward signals, similar to how an intervention from a driving instructor can be a valuable lesson.

Chapter 4 looked at the use of neural networks (NNs) for modelling dynamical systems of the form presented in Section 2.1, with a focus on how prior knowledge can be utilised to enhance the generalisability of the models. Another significant benefit is that the necessary size and capacity of the NNs are reduced, making it easier to verify these models as discussed in Chapter 5. One way to include prior knowledge is to engineer features that are expected to be relevant for the ML task, which can be seen as using the outputs of existing models as additional input data. However, excessive feature engineering can lead to overfitting to spurious patterns in the data, reducing the generalisation capabilities of the model. To mitigate this, Paper B proposed the physics-guided neural network (PGNN) architecture, where features are *injected* into the intermediate layers of a NN (see Section 4.2). Synthetic data were generated from 5 different dynamical systems, presented in Examples 2.1.4 to 2.1.8, and PGNNs were trained to mimic the dynamics of these systems. This change in architecture was found to reduce overfitting and improve the models' generalisation and long-term predictive capability. However, a drawback of the method is that the injection layer is an additional hyperparameter that must be selected, and the optimal layer was found to vary significantly between systems. Prior knowledge in the form of data can be used to correct the error of an existing model, referred to as the Corrective source term approach (CoSTA) in this thesis. This approach is known as "boosting" in the ML literature. Boosting is useful when the underlying assumptions of a physics-based modelling (PBM) model are wrong or overly restrictive or if it is being adapted to a new domain. Section 4.3 applies this methodology to an ablated model of an aluminium electrolysis cell (see Example 2.1.3); this work will be published in Paper C. CoSTA was compared to a NN that was trained end-to-end on the same data, and the findings showed that CoSTA generalised better and was significantly more stable when making long-term predictions. While Data-driven modelling (DDM) models are reasonably good at interpolating between data points, they perform poorly at *extrapolation* beyond the dataset domain. If a NN follows a trajectory that leaves the dataset, undefined behaviour such as a blow-up can follow. Interestingly, it was found that this can be alleviated somewhat by using ℓ_1 weight decay. Following up on this observation, Section 4.4 repeated the experiments on the aluminium electrolysis cell using NNs with more complex architectures, different values of λ_1 for weight decay, and for datasets of different sizes. With statistical significance, combining skip-connections and ℓ_1 weight decay yielded models with significantly improved stability. Although researchers have been experimenting with NNs to model dynamical systems for decades, there is still little consensus on the best approach. Future work in this direction should focus on such models' possible failure modes (e.g. instability) and what kinds of inductive bias can be introduced to remedy these issues. At the time of writing, there is significant research interest in developing models that operate on continuous time data, e.g. neural differential equations [13] or "liquid" NNs [255, 256].

While Chapters 3 and 4 investigated various uses of ML methods and the resulting issues related to safety and stability, Chapter 5 represents ongoing work towards the verification of these systems by treating NNs as piecewise affine (PWA) systems. This conversion is exact when the only nonlinearities present in the networks are PWA functions, such that linear "piece" of the NN is associated with a region of the input space and a specific "activation pattern" of the nonlinearities. Section 5.1 presents a general algorithm that can compute the linear pieces of NNs and presents some runtime results. This work is available online as a preprint; see Paper F. The main challenge that prevents direct use of this algorithm is that the number of linear pieces grows exponentially with the input dimension and depth of the network, making it computationally expensive to identify all of them and analyse the resulting function. However, preliminary results showed that most networks exhibit large amounts of redundancy, and many small pieces can, in principle, be discarded without losing accuracy. This idea was investigated in Section 5.2 by experimenting with different types of regularisation and visualising the effects on the linear regions of a network trained to model a damped pendulum (see Example 2.1.1). The findings showed that ℓ_1 weight decay is particularly effective at reducing the number of regions within the data domain; a simple mechanism through which this can happen was proposed. Despite the encouraging results, significant reductions in the number of regions also come at the cost of reduced accuracy. Section 5.3.2 proposed a modification of the algorithm presented in Section 5.1 by introducing an approximation based on the number of data points present within each region. The approximate algorithm was then tested on a nonlinear systems benchmark based on the nonlinear wing vibrations of an F-16 jet; see Paper D for the published work. An advantage of the approach is that it is straightforward to implement, and the "resolution" of the approximation can be adjusted using a single parameter. However, access to the original dataset is required, which may not always be the case. Furthermore, the approximation depends on the order in which the NN neurons are visited. Choosing an order that minimises the number of linear pieces is an exciting line of future work. Existing methods for verifying NNs are motivated by the existence of adversarial examples and, therefore, primarily focused on achieving robustness certificates within the context of image processing. Instead, the algorithms presented in this thesis allow the analysis of NNs used to model *controlled dynamical systems*, e.g. by using existing tools for PWA systems such as robustness certificates and reachability analysis. This direction has substantial potential and will be the subject of future work.

Recent advances in ML have enabled practitioners to solve problems unimaginable just a few decades ago. This fact, coupled with the economic advantages and apparent universality of ML, means this field will only grow in the coming years. However, this does not mean existing theories and tools should be discarded in favour of automatic self-learning systems. Instead, more research needs to be done to ensure that these new tools are used correctly and safely in a provable manner. This thesis is a small step towards that vision, a future where learning systems are not the first (and last) resort but powerful tools that can be used to develop robust and adaptive systems and improve our understanding of the world around us.

Bibliography

[1] Andrej Karpathy. 'Software 2.0'. Medium. 2017.

[2] Karl Johan Åström and Richard M. Murray. 'Feedback Systems: An Introduction for Scientists and Engineers, Second Edition'. Princeton University Press, Feb. 2021. ISBN: 978-0-691-19398-4.

[3] L. Lamport. 'Proving the Correctness of Multiprocess Programs'. In: *IEEE Transactions on Software Engineering* SE-3.2 (Mar. 1977), pp. 125–143.

[4] Lennart Ljung. 'System Identification: Theory for the User'. Pearson, 1998. ISBN: 978-0-13-656695-3.

[5] Jonas Sjöberg et al. 'Nonlinear Black-Box Modeling in System Identification: A Unified Overview'. In: *Automatica* 31.12 (1995), pp. 1691–1724.

[6] Tingwu Wang et al. 'Benchmarking Model-Based Reinforcement Learning'. July 2019. arXiv: arXiv:1907.02057.

[7] David Luenberger. 'Observers for Multivariable Systems'. In: *IEEE Transactions on Automatic Control* 11.2 (1966), pp. 190–197.

[8] R. E. Kalman. 'A New Approach to Linear Filtering and Prediction Problems'. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45.

[9] Emanuel Todorov. 'General Duality between Optimal Control and Estimation'. In: *Proceedings of the Conference on Decision and Control*. IEEE, 2008, pp. 4286–4292.

[10] R. E. Kalman and R. S. Bucy. 'New Results in Linear Filtering and Prediction Theory'. In: *Journal of Basic Engineering* 83.1 (Mar. 1961), pp. 95–108.

[11] E.A. Wan and R. Van Der Merwe. 'The Unscented Kalman Filter for Nonlinear Estimation'. In: *Proceedings of the Adaptive Systems for Signal Processing, Communications, and Control Symposium*. IEEE, Oct. 2000, pp. 153–158.

[12] G. E. Hinton and R. R. Salakhutdinov. 'Reducing the Dimensionality of Data with Neural Networks'. In: *Science* 313.5786 (2006), pp. 504–507.

[13] Tian Qi Chen et al. 'Neural Ordinary Differential Equations'. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6571–6583.

[14] Danijar Hafner et al. 'Mastering Atari with Discrete World Models'. In: *International Conference on Learning Representations*. 2021.

[15] Danijar Hafner et al. 'Mastering Diverse Domains through World Models'. Jan. 2023. arXiv: arXiv:2301.04104.

[16] Edmund M. Clarke et al. 'Handbook of Model Checking'. Vol. 10. Springer, 2018.

[17] Hassan K. Khalil. 'Nonlinear Systems'. Third. Prentice-Hall. Pearson, 2002. ISBN: 0-13-067389-7.

[18] Chi-Tsong Chen. 'Linear System Theory and Design'. Oxford University Press, 2013. ISBN: 978-0-19-996454-3.

[19] James Blake Rawlings and David Q. Mayne. 'Model Predictive Control: Theory, Computation, and Design'. Second. Nob Hill Publishing, 2009. ISBN: 978-0-9759377-5-4.

[20] Stanley Bak, Hoang-Dung Tran and Taylor T. Johnson. 'Numerical Verification of Affine Systems with up to a Billion Dimensions'. In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019, pp. 23–32.

[21] Matthias Althoff, Goran Frehse and Antoine Girard. 'Set Propagation Techniques for Reachability Analysis'. In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.1 (2021).

[22] Sophie A. Gruenbacher et al. 'GoTube: Scalable Statistical Verification of Continuous Depth Models'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.6 (June 2022), pp. 6755–6764.

[23] Andrew D Selbst and Julia Powles. 'Meaningful Information and the Right to Explanation'. In: *International Data Privacy Law* 7.4 (Nov. 2017), pp. 233–242.

[24] Gareth James et al. 'An Introduction to Statistical Learning: With Applications in R'. Springer Publishing Company, Incorporated, 2014. ISBN: 1-4614-7137-0.

[25] Tom Brown et al. 'Language Models Are Few-Shot Learners'. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[26] Zachary C. Lipton. 'The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability Is Both Important and Slippery.' In: *Queue* 16.3 (June 2018), pp. 31–57.

[27] L. H. Gilpin et al. 'Explaining Explanations: An Overview of Interpretability of Machine Learning'. In: *Proceedings of the 5th International Conference on Data Science and Advanced Analytics*. IEEE, Oct. 2018, pp. 80–89.

[28] Bernease Herman. 'The Promise and Peril of Human Evaluation for Model Interpretability'. Oct. 2019. arXiv: arXiv:1711.07414.

[29] Finale Doshi-Velez and Been Kim. 'Towards A Rigorous Science of Interpretable Machine Learning'. Mar. 2017. arXiv: arXiv:1702.08608.

[30] Eivind Meyer et al. 'Taming an Autonomous Surface Vehicle for Path Following and Collision Avoidance Using Deep Reinforcement Learning'. In: *IEEE Access* 8 (2020), pp. 41466–41481.

[31] Haakon Robinson et al. 'Physics Guided Neural Networks for Modelling of Non-Linear Dynamics'. In: *Neural Networks* 154 (2022), pp. 333–345.

[32] Haakon Robinson et al. 'Deep Learning Assisted Physics-Based Modeling of Aluminum Extraction Process'. Conditional Acceptance: Engineering Applications of Artificial Intelligence. 2023.

[33] Haakon Robinson. 'Approximate Piecewise Affine Decomposition of Neural Networks'. In: *Proceedings of the 19th Symposium on System Identification (SYSID)*. Vol. 54. 2021, pp. 541–546.

[34] Aksel Vaaler et al. 'Safety Filter for Small Passenger Ferry'. Accepted: 42nd International Conference on Ocean, Offshore & Arctic Engineering. 2023.

[35] Haakon Robinson, Adil Rasheed and Omer San. 'Dissecting Deep Neural Networks'. Jan. 2020. arXiv: arXiv:1910.03879.

[36] Erlend Torje Berg Lundby et al. 'Sparse Neural Networks with Skip-Connections for Nonlinear System Identification'. Pending Review: CDC 2023. Jan. 2023. arXiv: arXiv:2301.00582.

[37] Thor Inge Fossen. 'Handbook of Marine Craft Hydrodynamics and Motion Control'. Second. Wiley, May 2021. ISBN: 978-1-119-99413-8.

[38] Anders Aglen Pedersen. 'Optimization Based System Identification for the milliAmpere Ferry'. MA thesis. NTNU, 2019.

[39] Erlend Torje Berg Lundby et al. 'Sparse Deep Neural Networks for Modeling Aluminum Electrolysis Dynamics'. In: *Applied Soft Computing* 134 (Feb. 2023), p. 109989.

[40] Guy Bunin. 'Ecological Communities with Lotka-Volterra Dynamics'. In: *Physical Review E* 95.4 (2017), pp. 042414–042414.

[41] Behzad Ghanbari and Salih Djilali. 'Mathematical Analysis of a Fractional-Order Predator-Prey Model with Prey Social Behavior and Infection Developed in Predator Population'. In: *Chaos, Solitons & Fractals* 138 (Sept. 2020), p. 109960.

[42] R. M. Goodwin. 'A Growth Cycle'. In: *Essays in Economic Dynamics*. Palgrave Macmillan UK, 1982, pp. 165–170. ISBN: 978-1-349-05504-3.

[43] Roberto Veneziani and Simon Mohun. 'Structural Stability and Goodwin's Growth Cycle'. In: *Structural Change and Economic Dynamics* 17.4 (Dec. 2006), pp. 437–451.

[44] David Harvie, Mark Kelmanson and David Knapp. 'A Dynamical Model of Business-Cycle Asymmetries: Extending Goodwin'. In: *Economic Issues* 12 (Jan. 2007).

[45] Hamel. 'Georg Duffing, Ingenieur: Erzwungene Schwingungen Bei Veränderlicher Eigenfrequenz Und Ihre Technische Bedeutung. Sammlung Vieweg. Heft 41/42, Braunschweig 1918. VI+134 S'. In: *Journal of Applied Mathematics and Mechanics (ZAMM)* 1.1 (1921), pp. 72–73.

[46] Steven H. Strogatz. 'Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering'. Second. CRC Press, 2015. ISBN: 978-0-429-49256-3.

[47] Wojciech Wawrzynski. 'Bistability and Accompanying Phenomena in the 1-DOF Mathematical Model of Rolling'. In: *Ocean Engineering* 147 (2018), pp. 565–579.

[48] B. Van der Pol. 'A Theory of the Amplitude of Free and Forced Triode Vibrations'. In: *Radio Review* 1.Selected Scientific Papers (1920), pp. 701–710.

[49] Kevin Rompala, Richard Rand and Howard Howland. 'Dynamics of Three Coupled van Der Pol Oscillators with Application to Circadian Rhythms'. In: *Communications in Nonlinear Science and Numerical Simulation* 12.5 (2007), pp. 794–803.

[50] Jorge C. Lucero and Jean Schoentgen. 'Modeling Vocal Fold Asymmetries with Coupled van Der Pol Oscillators'. In: *Proceedings of Meetings on Acoustics* 19 (2013), p. 060165.

[51] Gaetan Fautso Kuiate et al. 'Autonomous Van Der Pol-Duffing Snap Oscillator: Analysis, Synchronization and Applications to Real-Time Image Encryption'. In: *International Journal of Dynamics and Control* 6.3 (2018), pp. 1008–1022.

[52] Edward N. Lorenz. 'Deterministic Nonperiodic Flow'. In: *Journal of the Atmospheric Sciences* 20.2 (1963), pp. 130–141.

[53] H. Haken. 'Analogy between Higher Instabilities in Fluids and Lasers'. In: *Physics Letters A* 53.1 (May 1975), pp. 77–78.

[54] David Ruelle. 'The Lorenz Attractor and the Problem of Turbulence'. In: *Quantum Dynamics: Models and Mathematics*. Springer, 1976, pp. 221–239.

[55] Divakar Viswanath. 'The Fractal Property of the Lorenz Attractor'. In: *Physica D: Nonlinear Phenomena* 190.1 (Mar. 2004), pp. 115–128.

[56] Michel Henon and Carl Heiles. 'The Applicability of the Third Integral of Motion: Some Numerical Experiments'. In: *The Astronomical Journal* 69 (Feb. 1964), p. 73.

[57] Euaggelos E. Zotos. 'Comparing the Escape Dynamics in Tidally Limited Star Cluster Models'. In: *Monthly Notices of the Royal Astronomical Society* 452 (2015), pp. 193–209.

[58] Euaggelos E. Zotos. 'Classifying Orbits in the Classical Hénon-Heiles Hamiltonian System'. In: *Nonlinear Dynamics* 79.3 (Oct. 2014), pp. 1665–1677.

[59] Euaggelos E. Zotos. 'An Overview of the Escape Dynamics in the Hénon-Heiles Hamiltonian System'. In: *Meccanica* 52.11-12 (Mar. 2017), pp. 2615–2630.

[60] Ian Goodfellow, Aaron Courville and Yoshua Bengio. 'Deep Learning'. MIT Press, 2016. ISBN: 978-0-262-03561-3.

[61] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105.

[62] Kaiming He et al. 'Deep Residual Learning for Image Recognition'. In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 770–778.

[63] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. Dec. 2019. arXiv: arXiv:1912.01703.

[64] Martin Abadi et al. 'TensorFlow: A System for Large-Scale Machine Learning'. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2016, pp. 265–283.

[65] Yoshua Bengio, Patrice Simard, Paolo Frasconi et al. 'Learning Long-Term Dependencies with Gradient Descent Is Difficult'. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[66] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. 'On the Difficulty of Training Recurrent Neural Networks'. In: *Proceedings of the 30th International Conference on Machine Learning*. PMLR, May 2013, pp. 1310–1318.

[67] Andrew L Maas, Awni Y Hannun, Andrew Y Ng et al. 'Rectifier Nonlinearities Improve Neural Network Acoustic Models'. In: *ICML*. Vol. 30. Atlanta, Georgia, USA. 2013, p. 3.

[68] Prajit Ramachandran, Barret Zoph and Quoc V. Le. 'Searching for Activation Functions'. Oct. 2017. arXiv: arXiv:1710.05941.

[69] Richard S. Sutton and Andrew G. Barto. 'Reinforcement Learning: An Introduction'. Second. MIT Press, 2018. ISBN: 978-0-262-03924-6.

[70] Timothy P. Lillicrap et al. 'Continuous Control with Deep Reinforcement Learning'. July 2019. arXiv: arXiv:1509.02971.

[71] Dimitri P. Bertsekas. 'Dynamic Programming and Optimal Control'. Fourth. Vol. II. Athena Scientific, 2000. ISBN: 978-1-886529-44-1.

[72] Lei Tai et al. 'A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation'. Apr. 2018. arXiv: arXiv:1612.07139.

[73] Richard S Sutton et al. 'Policy Gradient Methods for Reinforcement Learning with Function Approximation'. In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999.

[74] Lex Weaver and Nigel Tao. 'The Optimal Reward Baseline for Gradient-Based Reinforcement Learning'. Jan. 2013. arXiv: arXiv:1301.2315.

[75] John Schulman et al. 'High-Dimensional Continuous Control Using Generalized Advantage Estimation'. Oct. 2018. arXiv: arXiv:1506.02438.

[76] Christopher M. Bishop. 'Pattern Recognition and Machine Learning (Information Science and Statistics)'. Springer-Verlag, 2006. ISBN: 978-0-387-31073-2.

[77] Ronald J. Williams. 'Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning'. In: *Mach. Learn.* 8 (May 1992), pp. 229–256.

[78] Sham Kakade and John Langford. 'Approximately Optimal Approximate Reinforcement Learning'. In: *19th International Conference on Machine Learning (ICML)*. Morgan Kaufmann Publishers Inc., 2002, pp. 267–274.

[79] John Schulman. 'Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs'. PhD thesis. UC Berkeley, 2016.

[80] John Schulman et al. 'Trust Region Policy Optimization'. Apr. 2017. arXiv: arXiv:1502.05477.

[81] John Schulman et al. 'Proximal Policy Optimization Algorithms'. Aug. 2017. arXiv: arXiv:1707.06347.

[82] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. Jan. 2017. arXiv: arXiv:1412.6980.

[83] A. Rupam Mahmood et al. 'Benchmarking Reinforcement Learning Algorithms on Real-World Robots'. In: *Proceedings of The 2nd Conference on Robot Learning*. PMLR, Oct. 2018, pp. 561–591.

[84] Peter Henderson et al. 'Deep Reinforcement Learning That Matters'. In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. Feb. 2018, pp. 3207–3214.

[85] Greg Brockman et al. 'OpenAI Gym'. June 2016. arXiv: arXiv:1606.01540.

[86] Antonin Raffin et al. 'Stable-Baselines3: Reliable Reinforcement Learning Implementations'. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.

[87] Prafulla Dhariwal et al. 'OpenAI Baselines'. In: *GitHub repository* (2017).

[88] Amalie Heiberg et al. 'Risk-Based Implementation of COLREGs for Autonomous Surface Vehicles Using Deep Reinforcement Learning'. In: *Neural Networks* 152 (Aug. 2022), pp. 17–33.

[89] Ben Tearle et al. 'A Predictive Safety Filter for Learning-Based Racing Control'. In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 7635–7642.

[90] Trym Tengesdal et al. 'Ship Collision Avoidance and Anti Grounding Using Parallelized Cost Evaluation in Probabilistic Scenario-Based Model Predictive Control'. In: *IEEE Access* 10 (2022), pp. 111650–111664.

[91] David Silver et al. 'Mastering the Game of Go with Deep Neural Networks and Tree Search'. In: *Nature* 529.7587 (2016), pp. 484–503.

[92] David Silver et al. 'A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play'. In: *Science* 362.6419 (Dec. 2018), pp. 1140–1144.

[93] Oriol Vinyals et al. ‘Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning’. In: *Nature* 575.7782 (Nov. 2019), pp. 350–354.

[94] Øivind Aleksander G. Loe. ‘Collision Avoidance for Unmanned Surface Vehicles’. MA thesis. Norwegian University of Science and Technology, 2008.

[95] Zheping Yan et al. ‘Obstacle Avoidance for Unmanned Undersea Vehicle in Unknown Unstructured Environment’. In: *Mathematical Problems in Engineering* 2013 (Nov. 2013), pp. 1–12.

[96] Y. Koren and J. Borenstein. ‘Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation’. In: *Proceedings of the International Conference on Robotics and Automation*. Vol. 2. IEEE, 1991, pp. 1398–1404.

[97] Yamin Huang et al. ‘Ship Collision Avoidance Methods: State-of-the-art’. In: *Safety Science* 121 (Jan. 2020), pp. 451–473.

[98] Anete Vagale et al. ‘Path Planning and Collision Avoidance for Autonomous Surface Vehicles I: A Review’. In: *Journal of Marine Science and Technology* (2021).

[99] Jacoby Larson et al. ‘Advances in Autonomous Obstacle Avoidance for Unmanned Surface Vehicles’. Tech. rep. ADA475547. Space and Naval Warfare Systems Center, San Diego, CA, Jan. 2007.

[100] Giuseppe Casalino, Alessio Turetta and Enrico Simetti. ‘A Three-Layered Architecture for Real Time Path Planning and Obstacle Avoidance for Surveillance USVs Operating in Harbour Fields’. In: *OCEANS 2009-EUROPE*. 2009, pp. 1–8.

[101] Bjørn-Olav H. Eriksen et al. ‘Hybrid Collision Avoidance for ASVs Compliant With COLREGs Rules 8 and 13-17’. In: *Frontiers in Robotics and AI* 7 (2020).

[102] Randal W. Beard and Timothy W. McLain. ‘Small Unmanned Aircraft: Theory and Practice’. Princeton, 2012. ISBN: 978-0-691-14921-9.

[103] A. Elfes. ‘Sonar-Based Real-World Mapping and Navigation’. In: *IEEE Journal on Robotics and Automation* 3.3 (June 1987), pp. 249–265.

[104] Mauro Candeloro, Anastasios M. Lekkas and Asgeir J. Sørensen. ‘A Voronoi-diagram-based Dynamic Path-Planning System for Underactuated Marine Vessels’. In: *Control Engineering Practice* 61 (Apr. 2017), pp. 41–54.

[105] Mauro Candeloro et al. 'Continuous Curvature Path Planning Using Voronoi Diagrams and Fermat's Spirals'. In: *IFAC Proceedings Volumes* 46.33 (2013), pp. 132–137.

[106] S. Garrido et al. 'Path Planning for Mobile Robot Navigation Using Voronoi Diagram and Fast Marching'. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2006, pp. 2376–2381.

[107] P. E. Hart, N. J. Nilsson and B. Raphael. 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[108] Michael Schuster, Michael Blaich and Johannes Reuter. 'Collision Avoidance for Vessels Using a Low-Cost Radar Sensor'. In: *IFAC Proceedings Volumes*. 19th IFAC World Congress 47.3 (Jan. 2014), pp. 9673–9678.

[109] Steven M. Lavalle. 'Rapidly-Exploring Random Trees: A New Tool for Path Planning'. Technical. Department of Computer Science, Iowa State University, 1998.

[110] Lydia Kavraki et al. 'Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces'. In: *Robotics and Automation, IEEE Transactions on* 12 (Sept. 1996), pp. 566–580.

[111] Y. Chen, H. Peng and J. Grizzle. 'Obstacle Avoidance for Low-Speed Autonomous Vehicles With Barrier Function'. In: *IEEE Transactions on Control Systems Technology* 26.1 (Jan. 2018), pp. 194–206.

[112] I. M. Mitchell, A. M. Bayen and C. J. Tomlin. 'A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games'. In: *IEEE Transactions on Automatic Control* 50.7 (2005), pp. 947–957.

[113] Bjørn-Olav Eriksen et al. 'The Branching-Course MPC Algorithm for Maritime Collision Avoidance'. In: *Journal of Field Robotics* 36 (June 2019), pp. 1222–1249.

[114] I. B. Hagen et al. 'MPC-based Collision Avoidance Strategy for Existing Marine Vessel Guidance Systems'. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 7618–7623.

[115] Glenn Bitar, Morten Breivik and Anastasios M. Lekkas. 'Energy-Optimized Path Planning for Autonomous Ferries'. In: *IFAC-PapersOnLine* 51.29 (2018), pp. 389–394.

[116] Glenn Bitar et al. 'Energy-Optimized Hybrid Collision Avoidance for ASVs'. In: *18th European Control Conference (ECC)*. June 2019, pp. 2522–2529.

[117] Trym Tengesdal, Tor A. Johansen and Edmund F. Brekke. 'Ship Collision Avoidance Utilizing the Cross-Entropy Method for Collision Risk Assessment'. In: *IEEE Transactions on Intelligent Transportation Systems* 23.8 (2022), pp. 11148–11161.

[118] Oussama Khatib. 'Real-Time Obstacle Avoidance for Manipulators and Mobile Robots'. In: *The International Journal of Robotics Research* 5.1 (Mar. 1986), pp. 90–98.

[119] Sang-Min Lee, Kyung-Yub Kwon and Joongseon Joh. 'A Fuzzy Logic for Autonomous Navigation of Marine Vehicles Satisfying COLREG Guidelines'. In: *International Journal of Control, Automation, and Systems* 2.2 (2004), pp. 171–181.

[120] Johann Borenstein and Yoram Koren. 'The Vector Field Histogram - Fast Obstacle Avoidance For Mobile Robots'. In: *Robotics and Automation, IEEE Transactions on* 7 (July 1991), pp. 278–288.

[121] Dimitra Panagou. 'Motion Planning and Collision Avoidance Using Navigation Vector Fields'. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2014, pp. 2513–2518.

[122] D. Fox, W. Burgard and S. Thrun. 'The Dynamic Window Approach to Collision Avoidance'. In: *IEEE Robotics Automation Magazine* 4.1 (1997), pp. 23–33.

[123] O. Brock and O. Khatib. 'High-Speed Navigation Using the Global Dynamic Window Approach'. In: *International Conference on Robotics and Automation*. Vol. 1. IEEE, May 1999, pp. 341–346.

[124] B. H. Eriksen et al. 'A Modified Dynamic Window Algorithm for Horizontal Collision Avoidance for AUVs'. In: *Conference on Control Applications (CCA)*. IEEE, 2016, pp. 499–506.

[125] Paolo Fiorini and Zvi Shiller. 'Motion Planning in Dynamic Environments Using Velocity Obstacles'. In: *The International Journal of Robotics Research* 17.7 (July 1998), pp. 760–772.

[126] Yoshiaki Kuwata et al. 'Safe Maritime Autonomous Navigation With COLREGS, Using Velocity Obstacles'. In: *IEEE Journal of Oceanic Engineering* 39.1 (Jan. 2014), pp. 110–119.

[127] D. Kufoalor, Edmund Brekke and T. Johansen. 'Proactive Collision Avoidance for ASVs Using A Dynamic Reciprocal Velocity Obstacles Method'. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*. Oct. 2018, pp. 2402–2409.

[128] John Canny and John Reif. 'New Lower Bound Techniques for Robot Motion Planning Problems'. In: *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1987, pp. 49–60.

[129] M. S. Wiig. 'Collision Avoidance and Path Following for Underactuated Marine Vehicles'. PhD thesis. Norwegian University of Science and Technology, 2019.

[130] Einvald Serigstad, Bjørn-Olav H. Eriksen and Morten Breivik. 'Hybrid Collision Avoidance for Autonomous Surface Vehicles'. In: *IFAC-PapersOnLine* 51.29 (2018), pp. 1–7.

[131] Society of Naval Architects, Marine Engineers (U. S.). Technical and Research Committee. Hydrodynamics Subcommittee. 'Nomenclature for Treating the Motion of a Submerged Body Through a Fluid: Report of the American Towing Tank Conference'. Society of Naval Architects and Marine Engineers, 1950.

[132] Roger Skjetne, Øyvind Smogeli and Thor I. Fossen. 'Modeling, Identification, and Adaptive Maneuvering of CyberShip II: A Complete Design with Experiments'. In: *IFAC Proceedings Volumes* 37.10 (2004), pp. 203–208.

[133] Pauli Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17.3 (Mar. 2020), pp. 261–272.

[134] Morten Breivik and Thor I. Fossen. 'Guidance Laws for Autonomous Underwater Vehicles'. In: *Underwater Vehicles*. IntechOpen, 2009.

[135] Andreas Bell Martinsen. 'End-to-End Training for Path Following and Control of Marine Vehicles'. MA thesis. NTNU, 2018.

[136] E. Fehlberg. 'Klassische Runge-Kutta-Formeln Vierter Und Niedrigerer Ordnung Mit Schrittweiten-Kontrolle Und Ihre Anwendung Auf Wärmeleitungsprobleme'. In: *Computing* 6.1 (Mar. 1970), pp. 61–71.

[137] Yuhuai Wu et al. 'Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-factored Approximation'. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[138] Volodymyr Mnih et al. 'Asynchronous Methods for Deep Reinforcement Learning'. In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, June 2016, pp. 1928–1937.

[139] Kim Peter Wabersich and Melanie N. Zeilinger. 'A Predictive Safety Filter for Learning-Based Control of Constrained Nonlinear Dynamical Systems'. In: *Automatica* 129 (2021), p. 109597.

[140] Edmund F. Brekke et al. 'milliAmpere: An Autonomous Ferry Prototype'. In: *Journal of Physics: Conference Series* 2311.1 (July 2022), p. 012029.

[141] Glenn Bitar et al. 'Two-Stage Optimized Trajectory Planning for ASVs Under Polygonal Obstacle Constraints: Theory and Experiments'. In: *IEEE Access* 8 (2020), pp. 199953–199969.

[142] Sean Gillies et al. 'Shapely'. Github repository. Nov. 2022.

[143] Joel A. E. Andersson et al. 'CasADi: A Software Framework for Nonlinear Optimization and Optimal Control'. In: *Mathematical Programming Computation* 11.1 (Mar. 2019), pp. 1–36.

[144] Andreas Wächter and Lorenz T. Biegler. 'On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming'. In: *Mathematical Programming* 106.1 (Mar. 2006), pp. 25–57.

[145] Simon Blindheim and Tor Arne Johansen. 'Electronic Navigational Charts for Visualization, Simulation, and Autonomous Ship Control'. In: *IEEE Access* 10 (2022), pp. 3716–3737.

[146] International Maritime Organization. 'Convention on the International Regulations for Preventing Collisions at Sea (COLREGs)'. 1972.

[147] Robin Verschueren et al. 'Acados - a Modular Open-Source Framework for Fast Embedded Optimal Control'. In: *Mathematical Programming Computation* 14 (Oct. 2021), pp. 147–183.

[148] Andrew W. Senior et al. 'Improved Protein Structure Prediction Using Potentials from Deep Learning'. In: *Nature* 577.7792 (Jan. 2020), pp. 706–710.

[149] Priyabrata Saha, Saurabh Dash and Saibal Mukhopadhyay. 'Physics-Incorporated Convolutional Recurrent Neural Networks for Source Identification and Forecasting of Dynamical Systems'. In: *Elsevier Neural Networks* (2021).

[150] Zeyuan Allen-Zhu, Yuanzhi Li and Zhao Song. 'A Convergence Theory for Deep Learning via Over-Parameterization'. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. 2019, pp. 242–252.

[151] Nazatul Aini Abd Majid et al. 'Multivariate Statistical Monitoring of the Aluminium Smelting Process'. In: *Journal of Computers & Chemical Engineering* 35.11 (2011), pp. 2457–2468.

[152] Laura von Rueden et al. 'Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems'. In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (Jan. 2023), pp. 614–633.

[153] Christopher Rackauckas and Qing Nie. 'DifferentialEquations.Jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia'. In: *Journal of Open Research Software* 5.1 (May 2017), p. 15.

[154] Luis Pineda et al. 'Theseus: A Library for Differentiable Nonlinear Optimization'. Jan. 2023. arXiv: arXiv:2207.09442.

[155] Jared Willard et al. 'Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems'. In: *ACM Computing Surveys* 55.4 (Nov. 2022), 66:1–66:37.

[156] Omer San, Adil Rasheed and Trond Kvamsdal. 'Hybrid Analysis and Modeling, Eclecticism, and Multifidelity Computing toward Digital Twin Revolution'. In: *GAMM-Mitteilungen* 44.2 (2021), e202100007.

[157] Brandon Amos and J. Zico Kolter. 'OptNet: Differentiable Optimization as a Layer in Neural Networks'. In: *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[158] Filipe de Avila Belbute-Peres et al. 'End-to-End Differentiable Physics for Learning and Control'. In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018.

[159] Yang Yu, Houpu Yao and Yongming Liu. 'Structural Dynamics Simulation Using a Novel Physics-Guided Machine Learning Method'. In: *Engineering Applications of Artificial Intelligence* 96 (2020), p. 103947.

[160] Alfio Quarteroni and Gianluigi Rozza. 'Reduced Order Methods for Modeling and Computational Reduction'. Vol. 9. Springer, New York, 2014. ISBN: 978-3-319-02090-7.

[161] Shady E. Ahmed et al. 'On Closures for Reduced Order Models—A Spectrum of First-Principle to Machine-Learned Avenues'. In: *Physics of Fluids* 33.9 (Sept. 2021), p. 091301.

[162] Eivind Fonn et al. 'Fast Divergence-Conforming Reduced Basis Methods for Steady Navier-Stokes Flow'. In: *Computer Methods in Applied Mechanics and Engineering* 346 (2019), pp. 486–512.

[163] S. Pawar et al. 'A Deep Learning Enabler for Nonintrusive Reduced Order Modeling of Fluid Flows'. In: *Physics of Fluids* 31.8 (Aug. 2019), p. 085101.

[164] Suraj Pawar et al. 'Data-Driven Recovery of Hidden Physics in Reduced Order Modeling of Fluid Flows'. In: *Physics of Fluids* 32.3 (Mar. 2020), p. 036602.

[165] Maziar Raissi, Paris Perdikaris and George Em Karniadakis. 'Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations'. In: *Journal of Computational Physics* 378.C (Nov. 2018).

[166] Navid Zobeiry and Keith D. Humfeld. 'A Physics-Informed Machine Learning Approach for Solving Heat Transfer Equation in Advanced Manufacturing and Engineering Applications'. In: *Engineering Applications of Artificial Intelligence* 101 (May 2021), p. 104232.

[167] Florian Arnold and Rudibert King. 'State-Space Modeling for Control Based on Physics-Informed Neural Networks'. In: *Engineering Applications of Artificial Intelligence* 101 (2021), p. 104195.

[168] Sheng Shen et al. 'A Physics-Informed Deep Learning Approach for Bearing Fault Detection'. In: *Engineering Applications of Artificial Intelligence* 103 (2021), p. 104295.

[169] Aditi Krishnapriyan et al. 'Characterizing Possible Failure Modes in Physics-Informed Neural Networks'. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 26548–26560.

[170] Harsha Vaddireddy et al. 'Feature Engineering and Symbolic Regression Methods for Detecting Hidden Physics from Sparse Sensors'. In: *Physics of Fluids, Editor's pick* 32 (2020), p. 015113.

[171] Joseph Bakarji and Daniel M. Tartakovsky. 'Data-Driven Discovery of Coarse-Grained Equations'. In: *Journal of Computational Physics* 434 (2021), p. 110219.

[172] Kathleen Champion et al. 'Data-Driven Discovery of Coordinates and Governing Equations'. In: *Proceedings of the National Academy of Sciences* 116.45 (2019), pp. 22445–22451.

[173] Silviu-Marian Udrescu et al. 'AI Feynman 2.0: Pareto-optimal Symbolic Regression Exploiting Graph Modularity'. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 4860–4871.

[174] Steven L. Brunton, Joshua L. Proctor and J. Nathan Kutz. 'Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems'. In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937.

[175] Samuel Kim et al. 'Integration of Neural Network-Based Symbolic Regression in Deep Learning for Scientific Discovery'. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.9 (2021), pp. 4166–4177.

[176] Hao Xu, Dongxiao Zhang and Nanzhe Wang. 'Deep-Learning Based Discovery of Partial Differential Equations in Integral Form from Sparse and Noisy Data'. In: *Journal of Computational Physics* 445 (2021), p. 110592.

[177] Suraj Pawar et al. 'Physics Guided Machine Learning Using Simplified Theories'. In: *Physics of Fluids* 33.1 (Jan. 2021), p. 011701.

[178] Suraj Pawar et al. 'Model Fusion with Physics-Guided Machine Learning: Projection-based Reduced-Order Modeling'. In: *Physics of Fluids* 33.6 (June 2021), p. 067123.

[179] Sindre Stenen Blakseth et al. 'Deep Neural Network Enabled Corrective Source Term Approach to Hybrid Analysis and Modeling'. In: *Neural Netw.* 146.C (Feb. 2022), pp. 181–199.

[180] R. Maulik et al. 'Subgrid Modelling for Two-Dimensional Turbulence Using Neural Networks'. In: *Journal of Fluid Mechanics* 858 (2018), pp. 122–144.

[181] Suraj Pawar et al. 'A Priori Analysis on Deep Learning of Subgrid-Scale Parameterizations for Kraichnan Turbulence'. In: *Theoretical and Computational Fluid Dynamics* 34.4 (Aug. 2020), pp. 429–455.

[182] Rahul Rai and Chandan K. Sahu. 'Driven by Data or Derived Through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques With Cyber-Physical System (CPS) Focus'. In: *IEEE Access* 8 (2020), pp. 71050–71073.

[183] Laura von Rueden et al. 'Combining Machine Learning and Simulation to a Hybrid Modelling Approach: Current and Future Directions'. In: *Advances in Intelligent Data Analysis XVIII*. Lecture Notes in Computer Science. Springer International Publishing, 2020, pp. 548–560. ISBN: 978-3-030-44584-3.

[184] M. Chao et al. 'Fusing Physics-Based and Deep Learning Models for Prognostics'. In: *Reliability Engineering and System Safety* 217 (2022).

[185] William Bradley et al. 'Perspectives on the Integration between First-Principles and Data-Driven Modeling'. In: *Computers & Chemical Engineering* (2022), p. 107898.

[186] Erlend Torje Berg Lundby et al. 'A Novel Hybrid Analysis and Modeling Approach Applied to Aluminum Electrolysis Process'. In: *Journal of Process Control* 105 (2021), pp. 62–77.

[187] Sindre Stenen Blakseth et al. 'Combining Physics-Based and Data-Driven Techniques for Reliable Hybrid Analysis and Modeling Using the Corrective Source Term Approach'. In: *Applied Soft Computing* 128 (2022), p. 109533.

[188] Anders Krogh and John Hertz. 'A Simple Weight Decay Can Improve Generalization'. In: *Advances in Neural Information Processing Systems*. Vol. 4. Morgan Kaufmann Publishers Inc., 1991.

[189] Nitish Srivastava et al. 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[190] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. PMLR, 2015, pp. 448–456.

[191] Torsten Hoefler et al. 'Sparsity in Deep Learning: Pruning and Growth for Efficient Inference and Training in Neural Networks'. In: *The Journal of Machine Learning Research* 22.1 (Jan. 2021), 241:10882–241:11005.

[192] Mark Sandler et al. 'MobileNetV2: Inverted Residuals and Linear Bottlenecks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.

[193] Jonathan Frankle and Michael Carbin. 'The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks'. Mar. 2019. arXiv: arXiv:1803.03635.

[194] E. J. Candes and T. Tao. 'Decoding by Linear Programming'. In: *IEEE Transactions on Information Theory* 51.12 (Dec. 2005), pp. 4203–4215.

[195] B. K. Natarajan. 'Sparse Approximate Solutions to Linear Systems'. In: *SIAM Journal on Computing* 24.2 (Apr. 1995), pp. 227–234.

[196] Serge Gale et al. 'RBF Network Pruning Techniques for Adaptive Learning Controllers'. In: *9th International Workshop on Robot Motion and Control*. July 2013, pp. 246–251.

[197] Suraj Pawar et al. 'Multi-Fidelity Information Fusion with Concatenated Neural Networks'. In: *Scientific Reports* 12.1 (Apr. 2022), p. 5900.

[198] Pantelis R. Vlachas et al. 'Data-Driven Forecasting of High-Dimensional Chaotic Systems with Long Short-Term Memory Networks'. In: *Proceedings of the Royal Society A* 474.2213 (2018), p. 20170844.

[199] Jaideep Pathak et al. 'Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach'. In: *Physical Review Letters* 120.2 (Jan. 2018), p. 024102.

[200] J.R. Dormand and P.J. Prince. ‘A Family of Embedded Runge-Kutta Formulae’. In: *Journal of Computational and Applied Mathematics* 6.1 (1980), pp. 19–26.

[201] Grégoire Montavon et al. ‘Layer-Wise Relevance Propagation: An Overview’. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 193–209. ISBN: 978-3-030-28954-6.

[202] R. M. Isherwood. ‘Wind Resistance of Merchant Ships’. In: *The Royal Institution of Naval Architects* 115 (1972), pp. 327–338.

[203] Werner Blendermann. ‘Parameter Identification of Wind Loads on Ships’. In: *Journal of Wind Engineering and Industrial Aerodynamics* 51.3 (1994), pp. 339–351.

[204] Changliu Liu et al. ‘Algorithms for Verifying Deep Neural Networks’. In: *Foundations and Trends in Optimization*. Vol. 4. 2021, pp. 244–404.

[205] Randall J. LeVeque. ‘Finite Volume Methods for Hyperbolic Problems’. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. ISBN: 978-0-521-00924-9.

[206] Cheuk-Yi Cheung et al. ‘Spatial Temperature Profiles in an Aluminum Reduction Cell under Different Anode Current Distributions’. In: *AIChE Journal* 59.5 (2013), pp. 1544–1556.

[207] Floris Takens. ‘Detecting Strange Attractors in Turbulence’. In: *Dynamical Systems and Turbulence, Warwick 1980*. Lecture Notes in Mathematics. Springer, 1981, pp. 366–381. ISBN: 978-3-540-38945-3.

[208] Oliver Nelles. ‘Nonlinear System Identification: From Classical Approaches to Neural Networks, Fuzzy Models, and Gaussian Processes’. Springer International Publishing, 2020. ISBN: 978-3-030-47438-6.

[209] M. Winter and C. Breitsamter. ‘Nonlinear Identification via Connected Neural Networks for Unsteady Aerodynamic Analysis’. In: *Aerospace Science and Technology* 77 (2018), pp. 802–818.

[210] Hao Li et al. ‘Visualizing the Loss Landscape of Neural Nets’. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018.

[211] Gao Huang et al. ‘Densely Connected Convolutional Networks’. Jan. 2018. arXiv: arXiv:1608.06993.

[212] Jiawei Su, Danilo Vasconcellos Vargas and Kouichi Sakurai. ‘One Pixel Attack for Fooling Deep Neural Networks’. In: *IEEE Transactions on Evolutionary Computation* 23.5 (Oct. 2019), pp. 828–841.

[213] A.L. Cervantes, O.E. Agamennoni and J.L. Figueroa. 'A Nonlinear Model Predictive Control System Based on Wiener Piecewise Linear Models'. In: *Journal of Process Control* 13.7 (2003), pp. 655–666.

[214] D. Mignone, G. Ferrari-Trecate and M. Morari. 'Stability and Stabilization of Piecewise Affine and Hybrid Systems: An LMI Approach'. In: *Proceedings of the 39th IEEE Conference on Decision and Control*. Vol. 1. Dec. 2000, pp. 504–509.

[215] H. Benlaoukli et al. 'On the Construction of Invariant Sets for Piecewise Affine Systems Using the Transition Graph'. In: *2009 IEEE International Conference on Control and Automation*. Dec. 2009, pp. 122–127.

[216] Matteo Fischetti and Jason Jo. 'Deep Neural Networks as 0-1 Mixed Integer Linear Programs: A Feasibility Study'. Dec. 2017. arXiv: arXiv:1712.06174.

[217] Matteo Fischetti and Jason Jo. 'Deep Neural Networks and Mixed Integer Linear Optimization'. In: *Constraints* 23.3 (July 2018), pp. 296–309.

[218] Thiago Serra, Christian Tjandraatmadja and Srikumar Ramalingam. 'Bounding and Counting Linear Regions of Deep Neural Networks'. In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, July 2018, pp. 4558–4566.

[219] Guy Katz et al. 'Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks'. In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer International Publishing, 2017, pp. 97–117. ISBN: 978-3-319-63387-9.

[220] Abhinav Kumar, Thiago Serra and Srikumar Ramalingam. 'Equivalent and Approximate Transformations of Deep Neural Networks'. May 2019. arXiv: arXiv:1905.11428.

[221] Thiago Serra, Abhinav Kumar and Srikumar Ramalingam. 'Lossless Compression of Deep Neural Networks'. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Lecture Notes in Computer Science. Springer International Publishing, 2020, pp. 417–430. ISBN: 978-3-030-58942-4.

[222] Martin Herceg et al. 'Multi-Parametric Toolbox 3.0'. In: *2013 European Control Conference (ECC)*. July 2013, pp. 502–510.

[223] Ronen Eldan and Ohad Shamir. 'The Power of Depth for Feedforward Neural Networks'. In: *Conference on Learning Theory*. PMLR, June 2016, pp. 907–940.

[224] Razvan Pascanu, Guido Montufar and Yoshua Bengio. 'On the Number of Response Regions of Deep Feed Forward Networks with Piece-Wise Linear Activations'. Feb. 2014. arXiv: arXiv:1312.6098.

[225] Guido F Montufar et al. 'On the Number of Linear Regions of Deep Neural Networks'. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014.

[226] Komei Fukuda et al. 'Frequently Asked Questions in Polyhedral Computation'. Technical. Swiss Federal Institute of Technology, 2004.

[227] M. Baotić. 'Polytopic Computations in Constrained Optimal Control'. In: *Automatika* 50.3-4 (2009), pp. 119–134.

[228] Alexandre Maréchal and Michaël Périn. 'Efficient Elimination of Redundancies in Polyhedra by Raytracing'. In: *Verification, Model Checking, and Abstract Interpretation*. Lecture Notes in Computer Science. Springer International Publishing, 2017, pp. 367–385. ISBN: 978-3-319-52234-0.

[229] Richard P. Stanley. 'An Introduction to Hyperplane Arrangements'. In: *Geometric Combinatorics*. Vol. 13. IAS/Park Mathematics Series. American Mathematical Society, 2006, pp. 389–486. ISBN: 978-0-8218-3736-8.

[230] T Zaslavsky. 'Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes'. In: *Memoirs of the American Mathematical Society* 154 (Jan. 1975).

[231] Guido Montufar. 'Notes on the Number of Linear Regions of Deep Neural Networks'. Mar. 2017.

[232] V. Vapnik and A. Chervonenkis. 'On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities'. In: *Theory of Probability & Its Applications* 16.2 (1971), pp. 264–280.

[233] Peter L. Bartlett et al. 'Nearly-Tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks'. In: *Journal of Machine Learning Research* 20.63 (2019), pp. 1–17.

[234] Maithra Raghu et al. 'On the Expressive Power of Deep Neural Networks'. In: *International Conference on Machine Learning*. July 2017, pp. 2847–2854.

[235] Matus Telgarsky. 'Benefits of Depth in Neural Networks'. In: *Conference on Learning Theory*. PMLR, June 2016, pp. 1517–1539.

[236] H. Amin, K. M. Curtis and B. R. Hayes-Gill. 'Piecewise Linear Approximation Applied to Nonlinear Function of a Neural Network'. In: *IEE Proceedings - Circuits, Devices and Systems* 144.6 (Dec. 1997), pp. 313–317.

[237] Shuning Wang, Xiaolin Huang and Khan M. Junaid. 'Configuration of Continuous Piecewise-Linear Neural Networks'. In: *IEEE Transactions on Neural Networks* 19.8 (Aug. 2008), pp. 1431–1445.

[238] Dimitris Bertsimas and Jack William Dunn. 'Machine Learning under a Modern Optimization Lens'. Dynamic Ideas LLC, 2019. ISBN: 978-1-73378-850-2.

[239] Kai Y. Xiao et al. 'Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability'. In: *International Conference on Learning Representations (CoRR)*. 2018.

[240] Francesco Croce, Maksym Andriushchenko and Matthias Hein. 'Provable Robustness of ReLU Networks via Maximization of Linear Regions'. In: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. PMLR, Apr. 2019, pp. 2057–2066.

[241] Tim Salimans and Durk P Kingma. 'Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks'. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016.

[242] Twan van Laarhoven. 'L2 Regularization versus Batch and Weight Normalization'. June 2017. arXiv: arXiv:1706.05350.

[243] Giosuè Cataldo Marinó et al. 'Compression Strategies and Space-Conscious Representations for Deep Neural Networks'. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. Jan. 2021, pp. 9835–9842.

[244] P. Tøndel, T. A. Johansen and A. Bemporad. 'Evaluation of Piecewise Affine Control via Binary Search Tree'. In: *Automatica* 39.5 (May 2003), pp. 945–950.

[245] R. Andrews, J. Diederich and A.B. Tickle. 'Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks'. In: *Knowledge-Based Systems* 8.6 (1995), pp. 373–389.

[246] G. P. J. Schmitz, C. Aldrich and F. S. Gouws. 'ANN-DT: An Algorithm for Extraction of Decision Trees from Artificial Neural Networks'. In: *IEEE Transactions on Neural Networks* 10.6 (1999), pp. 1392–1401.

[247] M. Sato and H. Tsukimoto. 'Rule Extraction from Neural Networks via Decision Tree Induction'. In: *IJCNN'01. International Joint Conference on Neural Networks. Proceedings*. Vol. 3. July 2001, pp. 1870–1875.

[248] J. Ross Quinlan. 'C4.5: Programs for Machine Learning'. First. Morgan Kaufmann Publishers Inc., Oct. 1992. ISBN: 978-1-55860-238-0.

[249] Jan Ruben Zilke, Eneldo Loza Mencía and Frederik Janssen. 'DeepRED - Rule Extraction from Deep Neural Networks'. In: *Lecture Notes in Computer Science*. Vol. 9956 LNAI. Springer Verlag, 2016, pp. 457–473. ISBN: 978-3-319-46306-3.

[250] MOSEK ApS. 'The MOSEK Optimization Toolbox for MATLAB Manual. Version 9.0.' 2019.

[251] Ioannis Z. Emiris and Vissarion Fisikopoulos. 'Practical Polytope Volume Approximation'. In: *ACM Transactions on Mathematical Software* 44.4 (June 2018), 38:1–38:21.

[252] Jean-Philippe Noël and M. Schoukens. 'F-16 Aircraft Benchmark Based on Ground Vibration Test Data'. In: *Workshop on Nonlinear System Identification Benchmarks*. Apr. 2017, pp. 19–23.

[253] David A. Dickey and Wayne A. Fuller. 'Likelihood Ratio Statistics for Autoregressive Time Series with a Unit Root'. In: *Econometrica* 49.4 (1981), pp. 1057–1072. JSTOR: 1912517.

[254] D. Kwiatkowski et al. 'Testing the Null Hypothesis of Stationarity against the Alternative of a Unit Root. How Sure Are We That Economic Time Series Have a Unit Root?' In: *Journal of Econometrics* 54.1-3 (1992), pp. 159–178.

[255] Ramin Hasani et al. 'Liquid Time-constant Networks'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.9 (May 2021), pp. 7657–7666.

[256] Ramin Hasani et al. 'Closed-Form Continuous-Time Neural Networks'. In: *Nature Machine Intelligence* 4.11 (Nov. 2022), pp. 992–1003.

ISBN 978-82-326-7040-6 (trykt utg.)
ISBN 978-82-326-7039-0 (elektr. utg.)
ISSN 1503-8181 (trykt utg.)
ISSN 2703-8084 (online ver.)

NTNU
Kunnskap for en bedre verden