UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXIV CICLO DEL DOTTORATO DI RICERCA IN

SCIENZE DELLA TERRA, FLUIDODINAMICA E MATEMATICA.

INTERAZIONI E METODICHE.

Deep Learning for Abstraction, Control and Monitoring of Complex Cyber-Physical Systems

Settore scientifico-disciplinare: **INF-01**

DOTTORANDA

FRANCESCA CAIROLI

COORDINATORE

PROF. STEFANO MASET

SUPERVISORE DI TESI

PROF. LUCA BORTOLUSSI

ANNO ACCADEMICO 2020/2021

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXIV CICLO DEL DOTTORATO DI RICERCA IN

SCIENZE DELLA TERRA, FLUIDODINAMICA E MATEMATICA.

INTERAZIONI E METODICHE.

Deep Learning for Abstraction, Control and Monitoring of Complex Cyber-Physical Systems

Settore scientifico-disciplinare: **INF-01**

DOTTORANDA

FRANCESCA CAIROLI

COORDINATORE

PROF. STEFANO MASET

SUPERVISORE DI TESI

PROF. LUCA BORTOLUSSI

ANNO ACCADEMICO 2020/2021

First of all, I would like to start this thesis by thanking all the people and the things, material and immaterial, that enriched this journey as it is even more important than the destination itself.

The first person I want to thank is Luca Bortolussi, he earned this privilege by being a guide to me for quite a long time now. He made me consider academia in the first place, he showed me it could be fun and he taught me how to have a playful approach to problem-solving. He encouraged me in being proactive and independent. Most of all, Luca went along with my "rebellious and revolutionary" nature and recognized and valued my own talents. His support may seem unconventional, but, in the end, he is always there when you need it the most.

The second academic guide I would like to thank is Nicola Paoletti. He has been more of a co-supervisor than a co-author to me. Nicola has been extremely supportive and understanding and he introduced me to the magic art of "how to write a good paper". Visiting him in London has been an intense learning opportunity.

My sincere thanks go to the reviewers: Sriram Sankaranarayanan and Jyotirmoy Deshmukh. I thank them for their nice and encouraging words about my work, but, even more, I thank them for their constructive feedback. Their contribution definitely improved the quality and the readability of this thesis.

Finally, I thank all the students I taught or co-supervised, all my collaborators and all my co-authors.

* * *

On the personal side, I feel extremely lucky to have so many people I would like to thank. I promise I will try not to exceed the cheesiness allowed here. Do expect personal notes though.

I start by thanking all the PhD colleagues, the old and the new ones, for making the university a brighter place. We are becoming quite a crowd, so I explicitly mention only Gloria and Ginevra for sharing the office, and thus every emotion, with me. The university gave me an office and two special friends emerged out of it. I also thank the large, and expanding, group of

data scientists for their daily vitality and enthusiasm. In particular, I thank Laura for being a faithful friend, a fellow traveller and a good example. I must also thank all the good and bad music that helped me through the writing of this thesis.

I thank Ester and Vinicio for being the warmest and strongest little family I could desire. You are my home.

I thank Laura Pomicino for teaching me to listen carefully and to respect all of my emotions. Your help is invaluable as it permeates everything. I thank rock climbing and the mountains in general for always being there, for teaching me not to give up and for making my strength, my persistence and my resilience extremely tangible. I thank Laura Pauluzzi and Calice for being two wise and sensitive counsellors. I thank "I Corsari delle Giulie" for the adventures and the good moods and "La scuola di alpinismo Emilio Comici" for making me feel like a real alpinist. My self-esteem is grateful to you all.

I thank the Mellon, for being the best group of friends. Your presence has been, and still is, the lighthouse of my journey. I thank all the people that entered my life during these years. The old ones, the new ones and all those that will travel with me in this transition to a new chapter. Among all, I thank Giacomo who is walking by my side right now and who is filling my days with love. I thank Trieste for being my welcoming home every day. In particular, I thank la Napo e la Valle for being my safe spots.

Last but not least, I thank myself for making it to the end with a vibrant desire to keep going.

There is no room here to mention all the people that made this journey unique, but I am sure that there is enough room in my heart. Cheesiness limit exceeded.

Deep Learning for Abstraction, Control and Monitoring of Complex Cyber-Physical Systems

PhD Student: **Francesca Cairoli**
Supervisor: **Luca Bortolussi**

Eadem mutata resurgo.

Jakob Bernoulli

Contents

1 Introduction **17**

1.1 Motivation 17

1.1.1 Modeling 18

1.1.2 Controller Design 19

1.1.3 Testing and Verification 19

1.2 Approach 21

1.3 Contributions 23

1.4 Structure of the Thesis 25

I Background and Literature **27**

2 Modeling, Control and Verification **29**

2.1 Hybrid Systems Theory 29

2.1.1 Hybrid Automata (HA) 30

2.1.2 Chemical Reaction Networks and Stochastic Simulation 40

2.2 Signal Temporal Logic (STL) 44

2.2.1 STL semantics 44

2.3 Control of a CPS 47
2.3.1 Markov Decision Process 47
2.3.2 Controller synthesis 49
2.4 Verification of a CPS 51
2.4.1 Verification as a reachability problem 51

3 Learning Algorithms **57**
3.1 Supervised Learning 58
3.1.1 Deep Neural Networks 59
3.2 Unsupervised Learning 62
3.2.1 Generative Adversarial Nets 63
3.3 Uncertainty quantification 71
3.3.1 Bayesian Neural Networks for classification 72
3.3.2 Conformal Predictions 75

II Contributions **83**

4 Abstraction **85**
4.1 Problem Statement 87
4.1.1 Preliminaries 87
4.1.2 Learning an abstraction 88
4.1.3 Dataset Generation 90
4.1.4 cWCGAN-GP architecture 90
4.1.5 Model Training 91

4.2 Experimental Results 93
4.2.1 Results 99
4.2.2 Discussion 108
4.3 Conclusions 111

5 Control **115**
5.1 Problem Statement 116
5.2 Methodology 117
5.3 Experiments 122
5.3.1 Cart-Pole balancing 123
5.3.2 Car platooning 126
5.4 Conclusions 131

6 Monitoring **133**
6.1 Problem Statement 134
6.1.1 Uncertainty-based error detection 139
6.2 Uncertainty quantification in Neural Predictive Monitoring . . 142
6.3 Uncertainty-based Rejection Criteria 144
6.3.1 Frequentist error detection via Support Vector Classification (SVC) 145
6.3.2 Bayesian error detection via Gaussian Process Classification (GPC) 147
6.4 Active Learning 150
6.4.1 General active learning algorithm 151
6.4.2 Frequentist active learning algorithm 153

6.5 Experimental Results 155
6.5.1 Case Studies 156
6.5.2 Performance measures 161
6.5.3 Results under Full Observability 163
6.5.4 Results under Partial Observability 176
6.6 Related Work 185
6.7 Conclusion 190

7 Conclusions **193**
7.1 Concluding Remarks 193

Appendices **213**

A Abstraction **215**

B Monitoring **221**

Acknowledgements

All the contributions presented in this thesis have been published. In particular: Chapter 4 is based on [1], coauthored by Luca Bortolussi and Ginevra Carbone, both from the University of Trieste. Chapter 5 is based on [2], a joint work with Luca Bortolussi, Ginevra Carbone, Francesco Franchina and Enrico Regolin, all from the University of Trieste. Chapter 6 is based on [3] and [4], both coauthored by Luca Bortolussi, University of Trieste, and Nicola Paoletti, Royal Holloway University and King's College, London.

All these works have been partially supported by the PRIN project "SEDUCE" n. 2017TWRCNB.

Abstract

Cyber-Physical Systems (CPS) consist of digital devices that interact with some physical components. Their popularity and complexity are growing exponentially, giving birth to new, previously unexplored, safety-critical application domains. As CPS permeate our daily lives, it becomes imperative to reason about their reliability. Formal methods provide rigorous techniques for verification, control and synthesis of safe and reliable CPS. However, these methods do not scale with the complexity of the system, thus their applicability to real-world problems is limited. A promising strategy is to leverage deep learning techniques to tackle the scalability issue of formal methods, transforming unfeasible problems into approximately solvable ones. The approximate models are trained over observations which are solutions of the formal problem. In this thesis, we focus on the following tasks, which are computationally challenging: the modeling and the simulation of a complex stochastic model, the design of a safe and robust control policy for a system acting in a highly uncertain environment and the runtime verification problem under full or partial observability. Our approaches, based on deep learning, are indeed applicable to real-world complex and safety-critical systems acting under strict real-time constraints and in presence of a significant amount of uncertainty.

Tables of Notations

Table 1: Acronyms summary

Notation	Description
CPS	Cyber-Physical System
HS, HA	Hybrid System, Hybrid Automaton
IoT	Internet of Things
MC	Model Checking
(N)PM	(Neural) Predictive Monitoring
GAN	Generative Adversarial Nets
cWCGAN-GP	conditional Wasserstein GAN with Gradient Penalty
MA	Model Abstraction
DNN, CNN, BNN	Deep, Convolutional, Bayesian Neural Network
FN, FP	False Negative, False Positive Error
TPR	True Positive Rate
(I)CP	(Inductive) Conformal Predictions
NCF	CP nonconformity function
STL	Signal Temporal Logic
DTCM, CTMC	Discrete-, Continuous-Time Markov Chain
CRN	Chemical Reaction Network
CME	Chemical Master Equation
SSA	Stochastic Simulation Algorithm
MDP	Markov Decision Process
PO	Partial Observability
(S)GD	(Stochastic) Gradient Descent
HMC, VI	Hamilton Monte Carlo, Variational Inference
KL, JS	Kullback-Leibler, Jensen-Shannon
ELBO	Evidence Lower Bound
(N)SC, (N)SE	(Neural) State Classifier, Estimator
SVC, GPC	Support Vector, Gaussian Process Classifier

Table 2: Symbols summary (part I)

Hybrid Automata	
$\mathcal{M}$	hybrid automaton
$Loc \ni l$, $Var \ni v$,	discrete, continuous space
2^X	power set of a space X, i.e. set of all subspaces of X
$S(\mathcal{M}) \ni s = (l, v)$, $Y \ni y$	state space, partially observable space
$Init$	initial region of the HA
D	unsafe region
H	temporal horizon
$Flow$	continuous dynamics of the HA
$Trans$ ($Jump$, $Reset$)	discrete dynamics of the HA
Inv	invariant set (for nondeterministic HA)
$\mathcal{T} \ni \tau$	set of hybrid time trajectories τ
$\bar{s} : \tau \to S(\mathcal{M})$	hybrid signal
$\bar{v}(\bar{l}) : \tau \to Var(Loc)$	continuous (discrete) signal
$\mathsf{ReachSet}(Init, \mathcal{M})$	set of states of the HA $\mathcal{M}$ reachable from $Init$
R_{over}, R_{under}	over-, under-approximation of the reachable set
$\mathcal{R}each(D, s, H)$	time-bounded reachability of a region D from a state s
Chemical Reaction Network	
$\mathsf{X}_1, \ldots, \mathsf{X}_n$	collection of n species in a population
$s_{i,t}$, $i = 1, \ldots, n$	number of individuals of species X_i in the population at time t
$\mathsf{R}_1, \ldots, \mathsf{R}_m$	set of m possible reactions
$(f_{\mathsf{R}_i}, \nu_i)$	propensity function, update vector of reaction R_i
Signal Temporal Logic	
φ	STL property
$\Diamond_I$, $\Box_I$, $\mathcal{U}_I$	eventually, globally, until operator over a time interval I
$\chi^{\varphi}(s, t)$	Boolean semantics for a formula φ and a signal s
$\rho^{\varphi}(s, t)$	quantitative semantics for a formula φ and a signal s
Markov Decision Process	
S, A	state, action space
P, R	transition, reward function
π, π^*	policy, optimal policy
$J_\pi(s)$	objective function
$\mathbf{w}^H$	disturbance signal of length H

Table 3: Symbols summary (part II)

Learning	
X (x), T (t)	input, target space (variable)
$Z = X \times T$ $(z = (x, t))$, $\mathcal{Z}$	data space (variable), data generating distribution
$Z' \sim \mathcal{Z}$	training set
$K \ni k$	latent space
$f^* : X \to T$	unknown function
$f : X \to T$	approximate function
$f_d : X \to [0, 1]^c$	discriminant function for classification problems (c classes)
$f_{\mathbf{w}} : X \to T$	neural network parametrized by weights $\mathbf{w}$
$\mathcal{L}(\mathbf{w}, Z')$	loss function given a dataset Z' and parameters $\mathbf{w}$
$\nabla_{\mathbf{w}}$	gradient w.r.t. $\mathbf{w}$
λ	learning rate
G_{w_g}, D_{w_d}, C_{w_c}	generator, discriminator, critic network
$\mathbb{P}_r$, $\mathbb{P}_{w_g}$	real and generated data distribution
$W(\mathbb{P}_r, \mathbb{P}_{w_g})$	Wasserstein distance between real and generated distribution
$\mathcal{L}(w_g, w_d)$	GAN loss function
β	penalty coefficient in the WGAN-GP loss
Conformal Prediction	
ε	significance level
$\Gamma^{\epsilon}_*(x) \subseteq T$	ϵ-prediction region for x
$\delta : Z \to \mathbb{R}$	nonconformity function
$\alpha^i_* = \delta(x_*, t^i)$	nonconformity score for input x_* and target t^i
Z_c	calibration set
γ, κ	confidence, credibility
Model Abstraction	
proj	projection function
d	distance among distributions
$\Theta \ni \theta$	parameter space
Control	
ψ	discrete-time evolution of the system
$A_{\mathtt{e}}$, $A_{\mathtt{a}}$	environment, agent action space
$Y_{\mathtt{e}}$, $Y_{\mathtt{a}}$	observable states of the environment, agent
$\mathcal{R}_\phi$	robustness function of requirement ϕ
$\mathcal{A}$, $\mathcal{D}$	attacker, defender network
$\Theta_{\mathcal{A}}$, $\Theta_{\mathcal{D}}$	weights of the attacker (defender) network
Neural Predictive Monitoring	
h, g	state classifier, PO state classifier
μ	measurement function
$f \in \{h, g\}$	generic state classifier
R_f, r_f	rejection criterion, error detection rule
u_f	uncertainty quantification function w.r.t predictor f
U $(U_{\mathcal{B}}, U_{\mathcal{F}})$	uncertainty domain (Bayesian, frequentist)
E	error label space
$W = U \times E$	state space of the error detection criterion

Chapter 1

Introduction

1.1 Motivation

The Internet of Things, or IoT, integrates small computing devices with the surrounding physical world governed by the laws of physics. This physical environment is populated by humans and by animated and inanimate objects. These devices can communicate with one another (via specific wireless communication protocols), can sense part of the outside world (through micro-sensors) and can actuate the computed decisions/actions (via some micro-controllers, known as actuators). The overall system is known as a cyber-physical system (CPS), where the cyber part, i.e. the computing devices, interacts with the physical part, i.e. the outside world. CPS is typically modeled as a hybrid system as it undergoes hybrid discrete-continuous dynamics.

Recent technological advances made embedded hardware available to a larger public, causing an important paradigm shift in the technological panorama, known as the IoT revolution. Today, CPSs populate our lives. We find them inside our homes, at work, in the cities. Their popularity is expected to keep growing. In fact, as the cost of components (sensors, actuators and communication mechanisms) decreases, the number of CPSs, together with their complexity, is about to increase exponentially.

That said, it remains imperative for developers to ensure the safety of the

complex CPSs that permeate our daily life. You should trust your Roomba enough to leave it home alone without having to fear for the life of your beloved cat (the vice-versa would also be nice). The IoT revolution caused the appearance of novel, previously unexplored, safety-critical domains. Safety-critical means that a failure may result in catastrophic consequences, such as endangering or hurting someone, environmental damage or a huge monetary loss. Traditional safety-critical domains, such as aerospace, medical, chemical and nuclear industries, have a long and strong research history focused on safety [5, 6, 7, 8]. Experts in each field focused, with a lot of ingenuity, on the safety of a very specific system. They also kept a very conservative approach towards any extension that could compromise the safety of the system. Despite there being emerging efforts towards the deployment of safe artificial intelligence [9, 10] and safe reinforcement learning [11, 12, 13] applications, it remains clear how novel areas do not share such a deeply-rooted culture of safety and most of the time developers do not have the resources, the expertise and the time to create one.

1.1.1 Modeling

Having an abstract representation, or a digital twin, of the developed system is an essential requirement to reason about its correctness and reliability. In general, a model can either follow some well-grounded theory, such as physics, biology, etc., or it can be data-driven, meaning it is extrapolated from observations of the system. Most complex hybrid system models integrate data-driven and theoretical ingredients.

In modeling a complex CPS we face a wide range of difficulties:

- *Lack of a unified modeling language:* CPSs often integrate elements coming from diverse disciplines, which lack a common jargon to specify the behaviour of the system as a whole.

- *Black-box components:* CPSs may contain components that are not transparent, meaning that they behave as black-box components. This happens when the devices come from multiple diverse vendors, when there is a human involved in the decision system or when the system interacts with highly unpredictable and suddenly changing environments.

- *Noisy measurements:* the model can be built out of observable realizations of the system. In practice, the measurement process provides only partial and noisy information about the state of the system. This results in an under-modeling that may lead to a poor representation.

- *Loss of interpretability:* the number of interconnected devices and their inner complexity is constantly increasing, making a full description of the system highly unfeasible. This growth in dimension and complexity makes the system very hard to interpret and thus to model.

All the problems above contribute in making the modeling process extremely difficult. However, the model provides an understanding and a descriptive representation of the knowledge we have about the developed system. In other words, it delineates the boundary of knowledge we have about the system. Without such a representation, it is impossible to analyse the system.

1.1.2 Controller Design

Controlling a CPS is a well-established problem in classic control theory [14]. State of the art solutions apply to all those models in which a complete knowledge of the system is available, i.e., scenarios in which the environment is supposed to follow deterministic rules. For such models, a high level of predictability, along with good robustness, is achieved. However, as soon as these unpredictable scenarios come into play, traditional controllers are challenged and could fail. The so-called *open world scenarios* are difficult to model and to control, due to the significant amount of stochastic variables that are needed in their modeling and to the variety of uncertain scenarios that they present. Therefore, while trying to ensure safety and robustness, we need to be cautious about not trading them with model effectiveness.

1.1.3 Testing and Verification

Two main approaches are typically used to gain insights about the correctness (safety and reliability) of a CPS: the first approach is based on *testing and simulation*, whereas the second one is based on *formal verification*.

- *Testing and simulation* empirically checks the behaviour for a certain set of conditions. This approach scales well with respect to the dimensionality of the system at the cost of providing limited coverage over the state space of the system. Reaching exhaustive coverage is highly unfeasible as the tested area decreases as the cost of simulation increases. Besides, for very complex systems, e.g. systems with multi-scale dynamics or with hierarchical timelines, the test area is potentially infinite and the cost of simulation is extremely high, making testing an unfeasible solution.

- *Formal verification and model-checking* provide rigorous and more thorough analysis of the system. Model-checking techniques start from a mathematical specification of the system to automatically prove its correctness. Bounded model-checking is usually expressed in terms of a time-bounded reachability problem, i.e. given a set of initial states and a set of unsafe states, it determines whether there exists a trajectory of the system starting in an initial state and ending up in an unsafe state within a given temporal bound. Time-bounded reachability is decidable for a very limited class of hybrid automata, e.g. initialied rectangular hybrid automata, whereas, for a generic system, the solution to time-bounded reachability has been shown to be undecidable [15]. Approximate solutions [16] allow to cope with such undecidability, yet they are still very expensive. The cost is strongly related to the dimension of the initial region, to the length of the temporal horizon, and, of course, to the complexity of the model. If the analysis is performed offline, there are no strict temporal constraints on the time needed to evaluate the safety of a certain region. Thus, one can consider long temporal horizons and large initial regions. On the contrary, in a runtime application, we want to monitor the safety of the system as it evolves. This means that we want to preemptively know if a failure is about to happen in the nearby future. The answer about the safety of the current state has to be promptly communicated so that we can try to prevent eventual failures from happening. This problem is known as *predictive monitoring*. Therefore, safety evaluation has to be quick and it must cover the entire state space. Current approaches, even approximate ones, are not efficient enough to satisfy the aforementioned requirements.

In general, if a model attempts to describe all aspects of a system, then typically the analysis about its safety, either via testing or via verification, is impossible. In order to preserve provability, one has to limit the complexity, meaning that the computational model should hamstring the operations allowed. On one hand, developers may prefer a simpler simulation model, ignore physical effects that are irrelevant to the property of interest, to ensure faster simulations. On the other hand, these limitations in expressiveness can be to restrictive. Finding a middle ground between complexity and verification (or testing) is of paramount importance to make the analysis of the reliability of real-world applications possible.

1.2 Approach

The main issue in this thesis is the following: can we create a mathematical model expressive enough to capture all the complex dynamics we are interested in in a CPS and still be able to use it to design efficient, safe and robust controllers, governing its controllable parts? Once such a controller is integrated into the complex mathematical representation of the system, can we use the whole model to reason about the correctness and the reliability of the system?

The above issue can be better specified by the following questions:

1. Can we define a modeling framework able mathematically specify the dynamics of a complex CPS undergoing all the difficulties presented before?

2. If the system, or its model, is very complex, can we reduce the computational load of simulation so that testing becomes feasible, allowing for a larger, even if not complete, coverage of the state space?

3. Can we design the controllable parts of the system to behave safely and robustly even if the system is acting in an adverse, or highly uncertain, environment?

4. Can we solve the predictive monitoring problem to monitoring runtime safety for arbitrarily complex systems?

In the thesis, we provide answers to the questions above. The common ingredient is to leverage the powerful tools of machine learning to reach highly accurate solutions that reduce considerably the computational burden. The rationale is to consider observed realizations of the original complex system as samples of an unknown data generating process. The goal is to exploit the information present in these realizations to infer a function that behaves as similarly as possible to the original process. The inferred function will generalize the behaviour of the process over the entire state space so that from a limited sample we can evaluate the behaviour of the original process under any condition. In supervised learning scenarios, where each example has an associated ground truth target value, the learned function is used to map new examples to the most likely target value. When there is no ground truth information about the desired output, as for unsupervised learning scenarios, machine learning infers a probabilistic model that mimics the unknown data generating distribution, capturing the hidden structure of the observed data. Machine learning provides black-box approximations, meaning that we trade interpretability with high accuracy and high computational efficiency. In addition, to reinforce the reliability of the reconstruction, we provide, whenever possible, statistical guarantees about the quality of the approximation.

Questions one and two are solved by developing an abstraction based on generative adversarial nets (GAN), called *GAN-based model abstraction*. Given a complex and highly stochastic model with a simulation process prohibitively expensive to run, we train a GAN that generates trajectories similar to the original ones, but with a highly reduced computational cost. This speed-up could be beneficial in improving the efficiency of all the simulation-based analysis, e.g. empirical testing.

Question three is solved by designing two neural networks, one that models the controllable parts of the system and one that models the environmental settings. These two networks are trained against each other so that the controller network chooses the *best* action for the safety of the system, whereas the environmental network decides which is the *worst* environmental setting with respect to the safety of the system. The result is a *safe and robust controller*, though without formal guarantees.

Finally, question four is solved with a technique, called *Neural Predictive Monitoring* (NPM), that provides an approximate solution to the predictive monitoring problem. NPM builds on a neural network that infer the safety

of a system over the entire state space from a limited set of observations of safe and unsafe scenarios. This neural network will be extremely efficient in providing information about the safety of the system.

A more in-depth presentation of each of these solutions is outlined in the following section.

1.3 Contributions

More in detail, our contribution can be divided into solving the following inference problems.

1. **GAN-based Model Abstraction.** The aim is to reduce the computational cost of simulation, i.e. the realization of a trajectory of a fixed finite length, for a stochastic system with complex dynamics. This issue is framed as an unsupervised learning problem, where a pool of trajectories, generated by the costly simulation process, is considered as a sample of a very complex unknown distribution over the space of fixed-length trajectories. This pool is then used as a dataset to train a deep generative model that approximates such distribution as precisely as possible. Because of their strength and flexibility, we choose generative adversarial networks as deep generative models. The abstract model automatically captures the features that are more relevant to characterise the evolution of the system. This abstraction reduces both the computational time needed to simulate a batch of trajectories and the ingenuity needed to perform an efficient abstraction. The same technique can potentially address a data-driven approach to modeling. If we only have access to a set of realizations of an unknown system, we can learn a deep generative abstraction that infers the behaviour of the system from these observations and that generalizes over the entire state space. In other words, it can learn a data-driven abstraction of a stochastic system.

2. **Safe and Robust Controller.** We design a controller, that acts safely and robustly under uncertainty, by training a neural network that competes with another black-box model, representing the potentially adverse environment. The first network, the controller, takes as input the

state of the system and outputs the *best* control action, whereas the second network, the environment, takes as input the state of the system and outputs the *worst* environmental setting. The terms *best* and *worst* are interpreted with respect to a requirement describing a property that the system should satisfy. Given a formal requirement, expressed as a signal temporal logic (STL) formula, we can quantify how much a realization of the system satisfies such requirement by leveraging the STL quantitative semantics. This quantification depends both on the output of the first network, the controller's actions, and on the output of the second network, the environmental setting. The training of the controller is nothing but an optimization process that aims at maximizing such quantification with respect to its actions, whereas the training of the environmental network aims at minimizing such quantification with respect to the proposed environmental setting. In other words, the environment repeatedly attacks the controller, so that when the training phase is over, we have learned a controller strong enough to robustly defend itself even from an adversarial environment.

3. **Neural Predictive Monitoring (NPM).** The aim of NPM is to improve the computational efficiency of a predictive monitor so that it can work at runtime even for complex systems. At runtime means that we need information about the safety of the system as it evolves. The general idea is to approximate the predictive monitor from a pool of costly realizations of the latter, typically used for offline analysis. The rationale is to frame this approximation as a supervised learning problem, more precisely a state classification problem, where states that have a trajectory that reaches the unsafe region are labeled as unsafe. We generate a dataset, by randomly selecting a pool of initial states and by labeling them as safe or unsafe using a costly model checker. Then we simply train a binary classifier that generalizes the information about safety over the entire state space and that is very efficient to evaluate. Building on two parallel statistical techniques to quantify uncertainty, one frequentist and one Bayesian, we quantify the trustworthiness of each approximate prediction and use this information to decide whether to accept it or not. NPM works well even on partially observable state spaces.

1.4 Structure of the Thesis

After this introduction (Chapter 1), we divide the thesis into two parts.

Part I presents the background material together with a literature review.

In Chapter 2, we first summarize the traditional modeling approach of complex cyber-physical systems, borrowing from hybrid systems theory (Section 2.1). In particular, we illustrate hybrid systems undergoing deterministic, nondeterministic and stochastic dynamics. We introduce the concepts of verification, predictive monitoring (Sec. 2.4) and predictive control (Sec. 2.3). In Section 2.2 we present a logic-based approach to formally specify the behaviour of a system. In particular, we focus on Signal Temporal Logic (STL).

Chapter 3 describes the learning algorithms used to approximate costly processes and the techniques used to estimate uncertainty. We present both discriminative (Sec. 3.1) and generative (Sec. 3.2) models. Section 3.3 presents two approaches to quantify the uncertainty of deep learning predictions. In particular, we present Bayesian Neural Networks (Sec. 3.3.1) and Conformal Predictions (Sec. 3.3.2).

Part II discusses our contributions and it is organized in 3 chapters.

Chapter 4 presents the GAN-based model abstraction technique that speeds up the cost of simulation for stochastic systems and that could also be used as a data-driven modeling approach for stochastic systems.

Chapter 5 presents a technique to train a safe and robust controller for a complex system acting in a critical and possibly adversarial environment.

Chapter 6 presents Neural Predictive Monitoring, a technique to perform efficient predictive monitoring to deploy it at runtime in safety-critical applications.

Finally, in the last chapter, Chapter 7, we report the concluding remarks and discuss future directions.

Part I

Background and Literature

Chapter 2

Modeling, Control and Verification

2.1 Hybrid Systems Theory

A cyber-physical system (CPS) is a system combining physical and digital or cyber components. The physical components represent entities of the physical world that have to be monitored or controlled. The cyber components represent the computer algorithms that monitor and control such entities. Control actions, meaning the states of the digital components, typically assume discrete values and can cause discrete jumps or shifts over the dynamics of the physical components. This is why hybrid models are the best solution to capture the mixed continuous and discrete behaviour of a CPS.

A hybrid system (HS) is a system whose dynamics exhibit both continuous and discrete dynamic behaviours. In other words, it has both flows, described by differential equations, and jumps, described by a state machine or an automaton. The continuous behaviour of the system depends on the discrete state as well as the changes of the discrete state are determined by the continuous state. The structure of HS allows for a lot of flexibility in modeling dynamic phenomena. In general, the state of an HS is defined by the values of the continuous variables and by a discrete mode. The state changes either continuously, according to a flow condition, or discretely ac-

cording to a control graph. Continuous flow is permitted as long as so-called invariants hold, while discrete transitions can occur as soon as given jump conditions are satisfied. Discrete transitions may be associated with events.

A hybrid automaton is a formal model that mathematically describes the evolution in time of a hybrid system.

2.1.1 Hybrid Automata (HA)

Let us start by defining a general *differential equation*. Let $Flow : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ be a vector field, with state space $\mathbb{R}^n$ and input space $\mathbb{R}^m$, such that

$$\begin{cases} \dot{v} = Flow(l, v), \\ v(0) = v_0. \end{cases} \tag{2.1}$$

Given an input signal $l : [0, \infty) \to \mathbb{R}^m$, a signal $v : [0, \infty) \to \mathbb{R}^n$ is a solution of (2.1) if:

1. v is piecewise differentiable w.r.t. time;
2. $v(t) = v_0 + \int_0^t Flow(l(t'), v(t'))dt' \qquad \forall t \geq 0.$

If v is a solution, then $\dfrac{dv}{dt}(t) = Flow(l(t), v(t))$ at any time t for which the derivative exists.

Deterministic Hybrid Automaton

Let $Loc = \{l_1, \ldots, l_m\}$ be a set of discrete states, referred to as *modes*, and let $Var \subset \mathbb{R}^n$ be a continuous state space. The function $Flow : Loc \times Var \to Var$ is a vector field, i.e., a vector-valued function, describing the *flow* of the continuous dynamics in each mode. The *discrete transition* between hybrid states is modeled by a *jump* function $Jump : Loc \times Var \to Loc$ and by a *reset* map $Reset : Loc \times Var \to Var$. The jump and the reset transitions happen simultaneously, so they can be modeled as a function $Trans : Loc \times Var \to Loc \times Var$, such that $(l', v') = Trans(l, v) = (Jump(l, v), Reset(l, v))$.

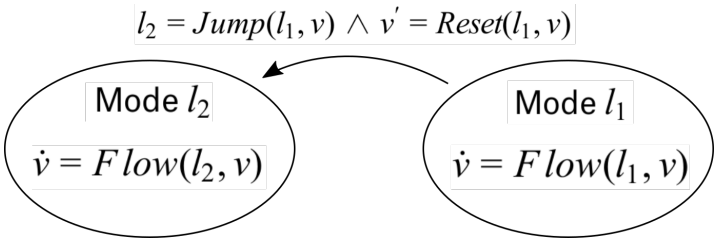


Figure 2.1: Deterministic Hybrid Automaton.

Figure 2.1 shows an example of a diagram representing the dynamics of a deterministic HA.

In a compact way, a hybrid automaton can be represented as a tuple

$$\mathcal{M} = (Loc, Var, Flow, Trans), \tag{2.2}$$

such that $\dot{v} = Flow(l, v)$ and $(l', v') = Trans(l, v)$ for $l \in Loc$ and $v \in Var$. $S(\mathcal{M}) = Loc \times Var$ is the *state space* of $\mathcal{M}$ and $s = (l, v) \in S(\mathcal{M})$ is a state of the $\mathcal{M}$.

A *solution* of the HA defined above is a pair of continuous signals $v : [0, \infty) \to Var$ and $l : [0, \infty) \to Loc$ such that

1. v is piecewise differentiable and l is piecewise constant w.r.t. time;
2. on an interval (t_1, t_2) on which l is constant and v continuous

$$v(t) = v(t_1) + \int_{t_1}^{t} Flow(l(t'), v(t_1))dt' \qquad \forall t \in [t_1, t_2);$$

3. $(l(t), v(t)) = Trans(l(t^-), v(t^-)) \qquad \forall t \geq 0$, where $t^- := \lim_{\delta t \to 0}(t - \delta t)$.

Nondeterministic Hybrid Automaton

Suppose that the jump condition does not force the transition but it simply allows it, meaning that we can stay in a mode until a certain invariance condition holds. Then we have a *nondeterministic* hybrid system modeled as a tuple

$$\mathcal{M} = (Loc, Var, Flow, Trans, Inv), \tag{2.3}$$

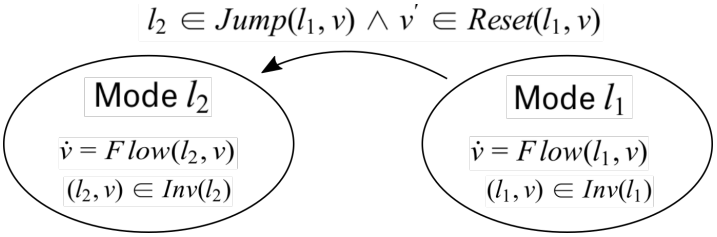


Figure 2.2: Nondeterministic Hybrid Automaton.

where $Inv : Loc \to 2^{Var}$ denotes the invariant set, or the domain of the system, at each mode. The discrete transition is modeled by a set-valued jump function $Jump : Loc \times Var \to 2^{Loc}$ and by a set-valued reset map $Reset : Loc \times Var \to 2^{Var}$, resulting in a set-value discrete transition function $Trans : Loc \times Var \to 2^{Loc \times Var}$, such that

$$Trans(l, v) = \Big(Jump(l, v) \times Reset(l, v)\Big) \cap \Big(\{l\} \times Inv(l)\Big).$$

The symbol 2^A denotes the power set of a general space A, in other words, 2^A is the set of all subspaces of A. Figure 2.2 shows an example of a diagram representing the dynamics of a nondeterministic HA.

A *solution* to the nondeterministic hybrid system is a pair of continuous signals $v : [0, \infty) \to Var$ and $l : [0, \infty) \to Loc$ such that

1. v is piecewise differentiable and l is piecewise constant w.r.t. time;
2. on an interval (t_1, t_2) on which l is constant and v continuous

$$v(t) = v(t_1) + \int_{t_1}^{t} Flow(l(t'), v(t_1))dt' \qquad \forall t \in [t_1, t_2);$$

3. $(l(t), v(t)) \in Trans(l(t^-), v(t^-)) \qquad \forall t \geq 0.$

Stochastic Hybrid Automaton

When discrete transitions happen according to a certain probability distribution we have a *stochastic* hybrid automaton. It is defined as a tuple

$$\mathcal{M} = (Loc, Var, Flow, Trans), \tag{2.4}$$

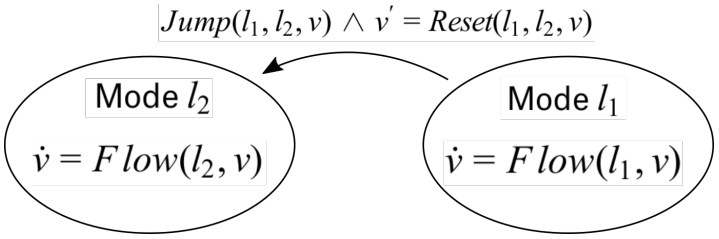


Figure 2.3: Stochastic Hybrid Automaton. The value on the edge denotes the probability for that transition to occur.

where the flow and the reset functions are deterministic ($Reset : Loc \times Loc \times Var \rightarrow Var$), whereas the discrete transition is represented as the combination of a discrete jump probability $Jump : Loc \times Loc \times Var \rightarrow [0, 1]$. In particular, $Jump(l_1, l_2, v)$ denotes the probability of jumping to a mode l_2 when in state (l_1, v). Mathematically,

$$Jump(l_1, l_2, v) = \lim_{dt \rightarrow 0} \frac{\mathbb{P}(l(t + dt) = l_2 | s(t) = (l_1, v))}{dt},$$

where $s(t) := (l(t), v(t)$. For a system to be well-defined, the following property must hold: $\forall l_1 \in Loc$ and $\forall v \in Var$, $\sum_{l_2 \in Loc} Jump(l_1, l_2, v) = 1$. The discrete transition function $Trans : Loc \times Loc \times Var \times Var \rightarrow [0, 1]$ is defined as

$$Trans(l_1, l_2, v_1, v_2) = \lim_{dt \rightarrow 0} \frac{\mathbb{P}(s(t + dt) = (l_2, v_2) | s(t) = (l_1, v_1))}{dt}.$$

Figure 2.3 shows an example of a diagram representing the dynamics of a stochastic HA. Here, we assume that some of the variables undergo continuous deterministic dynamics, whereas the discrete variables undergo stochastic transitions. However, it is possible to encapsulate eventual deterministic dynamics in a fully stochastic setting, as presented in Section 2.1.2. In simple words, a deterministic event is nothing but a probabilistic event happening with a probability of 1.

Trajectories of an HA

An hybrid trajectory has to satisfy the properties of an HA solution, specified above, so it can be defined as follows. First, let us define a *hybrid time*

trajectory as a (finite or infinite) sequence of closed intervals

$$\tau = \{[t_i, t_{i+1}] | t_i \leq t_{i+1}, i = 1, 2, \dots\}.$$

If τ is infinite the last interval may be open on the right. $\mathcal{T}$ denotes the set of hybrid time trajectories. Then, for a given $\tau \in \mathcal{T}$ a *hybrid signal* defined on τ with values in a generic hybrid space $S(\mathcal{M})$ is a sequence of functions

$$\bar{s} = \{\bar{s}_i : [t_i, t_{i+1}] \to S(\mathcal{M}), [t_i, t_{i+1}] \in \tau\}.$$

The map $\bar{s} : \tau \to S(\mathcal{M})$ defines the hybrid signal defined on τ with values in $S(\mathcal{M})$.

A *trajectory* of the hybrid automaton $\mathcal{M}$ is thus a pair of hybrid signals $\bar{v} : \tau \to Var$ and $\bar{l} : \tau \to Loc$ with $\tau \in \mathcal{T}$, such that

1. on any $[t_i, t_{i+1}] \in \tau$, $\bar{l}_i$ is constant and

$$\bar{v}_i(t) = \bar{v}_i(t_i) + \int_{t_i}^{t} Flow\left(\bar{l}_i(t_i), \bar{v}_i(t')\right) dt' \qquad \forall t \in [t_i, t_{i+1}];$$

2. $\left(\bar{l}(t_{i+1}), \bar{v}(t_{i+1})\right) = Trans\left(\bar{l}(t_i), \bar{v}(t_i)\right)$.

Properties of an HA

Let $\overline{Var}$ be the set of all piecewise continuous signals $\bar{v} : [0, T) \to Var$ and $\overline{Loc}$ be the set of all piecewise constant signals $\bar{l} : [0, T) \to Loc$ for $T \in (0, \infty]$. A *sequence property* is a map $\varphi : \overline{Loc} \times \overline{Var} \to \{false, true\}$, such that a pair of signals $(\bar{l}, \bar{v}) \in \overline{Loc} \times \overline{Var}$ *satisfies* φ if $\varphi(\bar{l}, \bar{v}) = true$. An HA $\mathcal{M}$ *satisfies* φ, formally $\mathcal{M} \models \varphi$, if $\varphi(\bar{l}, \bar{v}) = true$ for every solution $(\bar{l}, \bar{v})$ of $\mathcal{M}$.

The goal of *sequence analysis* is that of showing, given an HA $\mathcal{M}$, a set of initial states $Init \subset S(\mathcal{M})$ and a sequence property φ, that $(\mathcal{M}, Init) \models \varphi$, i.e. showing that, for every solution $(\bar{l}, \bar{v})$ of $\mathcal{M}$ with $(\bar{l}(0), \bar{v}(0)) \in Init$, $\varphi(\bar{l}, \bar{v}) = true$. When this is not the case, it must find a *counter-example*, i.e. a solution $(\bar{l}, \bar{v}) \in \overline{Loc} \times \overline{Var}$ with $(\bar{l}(0), \bar{v}(0)) \in Init$, such that $\varphi(\bar{l}, \bar{v}) = false$.

Given a signal $\bar{s} : [0, T) \to S(\mathcal{M})$, $T \in (0, \infty]$, a sub-signal $\bar{s}^* : [0, T^*) \to S(\mathcal{M})$ is called a *prefix* to s if $T^* \leq T$ and $\bar{s}^*(t) = \bar{s}(t)$ for every $t \in [0, T^*)$.

Safety properties. A sequence property φ is a *safety property* if it is:

1. *non-empty*, meaning that $\exists(\bar{l}, \bar{v})$ such that $\varphi(\bar{l}, \bar{v}) = true$;
2. *prefix-closed*, meaning that, given signals $(\bar{l}, \bar{v})$, $\varphi(\bar{l}, \bar{v}) \implies \varphi(\bar{l}^*, \bar{v}^*)$ for every prefix $(\bar{l}^*, \bar{v}^*)$ to $(\bar{l}, \bar{v})$;
3. *limit-closed*, meaning that, given an infinite sequence of signals $(\bar{l}_1, \bar{v}_1), (\bar{l}_2, \bar{v}_2), \ldots$ such that each element satisfies φ and such that, for every k, $(\bar{l}_k, \bar{v}_k)$ is a prefix to $(\bar{l}_{k+1}, \bar{v}_{k+1})$, then $(\bar{l}, \bar{v}) := \lim_{k\to\infty} (\bar{l}_k, \bar{v}_k)$ also satisfies φ.

In simpler words, the characterization above can be interpreted as follow. A safety property should check that something bad never happens, and this means that the property is non-trivial (point 1, non-empty), that a prefix to a good signal is always a good signal (point 2, prefix-closed) and, finally, that if something bad happens, it will happen in finite time (point 3, limit-closed).

Liveness properties. A sequence property φ is a *liveness property* if for every finite $(\bar{l}^*, \bar{v}^*) \in \overline{Loc} \times \overline{Var}$ there is some $(\bar{l}, \bar{v}) \in \overline{Loc} \times \overline{Var}$ such that:

1. $(\bar{l}^*, \bar{v}^*)$ is a prefix to $(\bar{l}, \bar{v})$;
2. $(\bar{l}, \bar{v})$ satisfies φ.

In simpler words, the characterization above can be interpreted as follows. A liveness property should check that something good will eventually happen. Thus, for any sequence, there is a good continuation.

The following two theorems imply that if we are able to verify safety and liveness properties we are indeed able to verify any sequence property. For their proof see [17].

Theorem 1. *If φ is both a liveness and a safety property, then every $(\bar{l}, \bar{v}) \in \overline{Loc} \times \overline{Var}$ satisfies φ, i.e., φ is always true.*

Theorem 2. *For every nonempty, i.e., not always false, sequence property φ there is a safety property φ_s and a liveness property φ_l such that: $(\bar{l}, \bar{v})$ satisfies φ if and only if $(\bar{l}, \bar{v})$ satisfies both φ_s and φ_l.*

Sequence properties are typically expressed in terms of Signal Temporal Logic (STL) formulas, a formalism presented in Section 2.2.

Reachability

Given an HA $\mathcal{M} = (Loc, Var, Flow, Trans)$ with set of initial states $Init \subset S(\mathcal{M})$, let $\mathsf{ReachSet}(Init, \mathcal{M})$ define the set of pairs $(l_f, v_f) \in S(\mathcal{M})$ for which there is a solution $(\bar{l}, \bar{v})$ to $\mathcal{M}$ for which:

1. $(\bar{l}(t_0), \bar{v}(t_0)) \in Init$;
2. $\exists t \geq t_0$ such that $(\bar{l}(t), \bar{v}(t)) = (l_f, v_f)$.

In simpler words, $\mathsf{ReachSet}(Init, \mathcal{M})$ is the set of states for which there exists a trajectory of $\mathcal{M}$, starting in $Init$, that passes through these states (see Fig. 2.4). A set $I \subset S(\mathcal{M})$ for which $\mathsf{ReachSet}(I, \mathcal{M}) = I$ is called an *invariant* set.

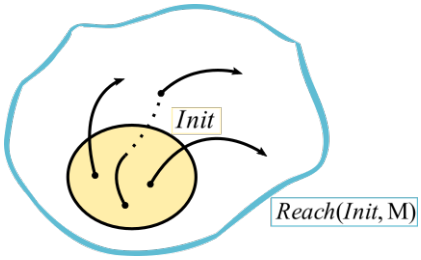


Figure 2.4: Reachable set from an initial region $Init$.

Reachability and safety. An HA $\mathcal{M}$ satisfies a safety property φ of the form $\varphi(\bar{l}, \bar{v}) = \Box\left((\bar{l}(t), \bar{v}(t)) \in F\right)$, where $F \subset S(\mathcal{M})$ is a nonempty set and $\Box$ denotes the "globally over time"operator, if and only if $\mathsf{ReachSet}(Init, \mathcal{M}) \subset F$. The above statement means that, every point in every trajectory starting from $Init$ satisfies φ (see Fig. 2.5).

Reachability algorithm. Consider an HA $\mathcal{M}$ with state space $S(\mathcal{M})$, the alphabet of events E can be defined as $\mathsf{E} = \{\tau\} \cup \{(l_i, l_j) | l_i, l_j \in Loc\}$, where

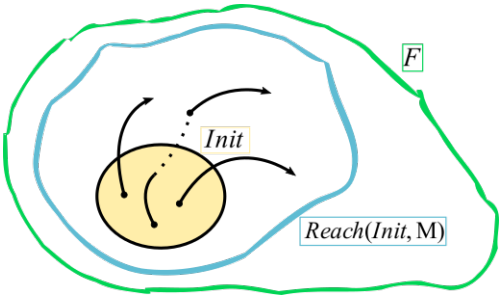


Figure 2.5: Safety property over a set F and reachable set from an initial region $Init$.

τ denotes the continuous evolution event and (l_i, l_j) denotes the jump event. The transition relation $\mathsf{T} \subset S(\mathcal{M}) \times \mathsf{E} \times S(\mathcal{M})$ can be defined as

$$\begin{aligned}\mathsf{T} = \{ &((l_0, v_0), ([t_0, t_f], (l_0, l_f)), (l_f, v_f)) \mid \\ &(l_f, v_f) = Trans(l_0, v(t_f)), \dot{v} = Flow(l_0, v), v(t_0) = v_0, \\ &(l_0, v(t)) = Trans(l_0, v(t)) \ \forall t \in (t_0, t_f)\}.\end{aligned}$$

An *execution*, or trajectory, of a transition system is a sequence of states $\{s_0, s_1, s_2, \dots\}$ such that there exists a sequence of events $\{e_0, e_1, e_2, \dots\}$ for which, for every i, $(s_i, e_i, s_{i+1}) \in \mathsf{T}$. Given a set of initial states $Init \subset S(\mathcal{M})$, $\mathsf{ReachSet}(Init, S(\mathcal{M}))$ is the set of states $s \in S(\mathcal{M})$ for which there is a finite trajectory that starts in $Init$ and ends at s.

Algorithm 1 Reachability algorithm

procedure REACHABLESET($Init$)
  $\mathsf{ReachSet}_{-1} = \emptyset$ ▷ Initialization
  $\mathsf{ReachSet}_0 = Init$
  $i = 0$
  while $\mathsf{ReachSet}_i \neq \mathsf{ReachSet}_{i-1}$ **do** ▷ Iterative algorithm
    $S' = \{s' \in S(\mathcal{M}) | \exists s \in \mathsf{ReachSet}_i, e \in \mathsf{E} \ s.t. \ (s, e, s') \in \mathsf{T}\}$
    $\mathsf{ReachSet}_{i+1} = \mathsf{ReachSet}_i \cup S'$
    i = i+1

Theorem 3. *If $S(\mathcal{M})$ is finite then*

(i) the reachability algorithm (Algorithm 1) finishes in a finite number of steps and

(ii) upon exiting the while-loop $\mathsf{ReachSet}_i = \mathsf{ReachSet}(Init, \mathcal{M})$.

The biggest problem with HA is that typically $S(\mathcal{M})$ is not finite and thus the algorithm may not terminate. Moreover, in the while loop, when computing S' so that $\mathsf{ReachSet}_{i+1} = \mathsf{ReachSet}_i \cup S'$, we have to find events $e = (\tau, (l_i, l_j)) \in \mathsf{E}$ for which a transition relation exists. Finding pairs (l_i, l_j) is simple, whereas finding τ is not in general that simple, as it requires to find if a certain continuous evolution inside a mode does exist.

Back to safety, let φ denote a generic safety property and let F be the set of safe states, then the safety algorithm can be framed as shown in Algorithm 2. The main difference is that the algorithm can terminate immediately if one of the $\mathsf{ReachSet}_i$ is outside of F.

Algorithm 2 Reachability and Safety algorithm

procedure SAFETY(φ, F, $Init$)
  $\mathsf{ReachSet}_{-1} = \emptyset$ ▷ Initialization
  $\mathsf{ReachSet}_0 = Init$
  $i = 0$
  while $\mathsf{ReachSet}_i \neq \mathsf{ReachSet}_{i-1}$ or $\mathsf{ReachSet}_i \not\subset F$ **do** ▷ Iterative algorithm
    $S' = \{s' \in S(\mathcal{M}) | \exists s \in \mathsf{ReachSet}_i, e \in \mathsf{E}\ s.t.\ (s, e, s') \in \mathsf{T}\}$
    $\mathsf{ReachSet}_{i+1} = \mathsf{ReachSet}_i \cup S'$
    $i = i + 1$
  if $\mathsf{ReachSet}_i = \mathsf{ReachSet}_{i-1}$ **then**
    $\mathcal{M}$ satisfy the safety property φ
  else
    $\mathcal{M}$ does not satisfy the safety property φ

Backward Reachability. Given a HA $\mathcal{M}$ and a set of final states $Final \subset S(\mathcal{M})$, BackReach(Final, $\mathcal{M}$) denotes the set of pairs $(l_0, v_0) \in S(\mathcal{M})$ for which there is a solution $(\bar{l}, \bar{v})$ to $\mathcal{M}$ for which: $(\bar{l}(t_0), \bar{v}(t_0)) = (l_0, v_0)$ and $\exists t \geq t_0 : (\bar{l}(t), \bar{v}(t)) \in Final$ (see Fig. 2.6).

In general,

$$Final \subset \mathsf{ReachSet}(\mathsf{BackReachSet}(Final))$$

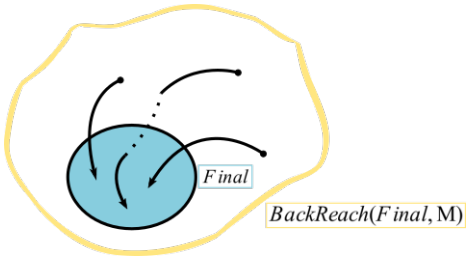


Figure 2.6: Backward reachability for a target set $Final$.

and

$$Init \subset \mathsf{BackReachSet}(\mathsf{ReachSet}(Init)).$$

Thus, the HA $\mathcal{M}$ satisfies a safety property φ over F if and only if

$$\mathsf{BackReachSet}(\neg F) \cap Init = \emptyset.$$

Algorithm 3 Backward Reachability algorithm

procedure BACKREACHSET($Final$)
  $\mathsf{BackReachSet}_{-1} = \emptyset$ ▷ Initialization
  $\mathsf{BackReachSet}_0 = Final$
  $i = 0$
  while $\mathsf{BackReachSet}_i \neq \mathsf{BackReachSet}_{i-1}$ **do** ▷ Iterative algorithm
    $S'' = \{s \in S(\mathcal{M}) | \exists s' \in \mathsf{BackReachSet}_i, e \in \mathsf{E}\ s.t.\ (s, e, s') \in \mathsf{T}\}$
    $\mathsf{BackReachSet}_{i+1} = \mathsf{BackReachSet}_i \cup S''$
    $i = i + 1$

Theorem 4. *If $S(\mathcal{M})$ is finite then*

(i) the backwards reachability algorithm (Algorithm 3) finishes in a finite number of steps and

(ii) upon exiting the while-loop $\mathsf{BackReachSet}_i = \mathsf{BackReachSet}(Final, \mathcal{M})$.

2.1.2 Chemical Reaction Networks and Stochastic Simulation

In the definition of a stochastic HA, we presented a system with deterministic *Flow*, i.e. continuous dynamics, and stochastic *Jump* transitions. However, a wide variety of complex stochastic hybrid systems have dynamics that can be modeled as Markov population models, where, in each state, the process changes according to an exponential random variable and then move to a different state as specified by the probabilities of a stochastic matrix. Eventual deterministic dynamics are easily captured by events of probability 1.

Chemical Reaction Networks (CRNs) use the formalism of chemical equations to capture the dynamics of population models. Let $\mathsf{X}_1, \ldots, \mathsf{X}_n$ be a collection of n species and $s_{i,t}$, $i = 1, \ldots, n$, denote the number of individuals of species X_i present in the population at time t. The dynamics of a CRN is described by a set of reactions $\mathsf{R}_1, \ldots, \mathsf{R}_m$. The firing of reaction R_i results in a transition of the system from state $s_t = (s_{1,t}, \ldots, s_{n,t}) \in S = \mathbb{N}^n$ to state $s_t + \nu_i$, with ν_i being the update vector. Under the well-stirred assumption, it is possible to ignore spatial inhomogeneities, leading to a discrete state-space in which transitions happen at random time instants as a result of accidental collisions between different chemical compounds. A general reaction R_i is identified by the tuple $(f_{\mathsf{R}_i}, \nu_i)$, where $f_{\mathsf{R}_i} : S \to \mathbb{R}_{\geq 0}$, known as *propensity function* of reaction R_i, depends on the state of the system.

Individual chemical reactions are denoted as:

$$\mathsf{R}_i : \sum_{j \in \{1,\ldots,n\}} \alpha_{ij} \mathsf{X}_j \to \sum_{j \in \{1,\ldots,n\}} \beta_{ij} \mathsf{X}_j, \tag{2.5}$$

where α_{ij} and β_{ij} are nonnegative integers called *stoichiometry coefficients*. The compounds on the left-hand and right-hand sides are usually referred to as the reactants and, respectively, the products of reaction R_i. The arrow indicates that the reaction happens only in that direction. If the converse transformation can happen as well, then the reaction is called reversible. The stoichiometry coefficients can be arranged so that they form the respective update vector $\nu_i = \beta_i - \alpha_i$. Equation (2.5) shows the algebraic interpretation of CRN. However, they can be represented as weighted directed graphs, where the weights are given by the stoichiometry coefficients and the number of

nodes is given by the number of species, i.e. n.

Example 1. *Consider a population with N individuals that can be in one of the three states ($n = 3$): susceptible (species X_S), infected (species X_I) and recovered (species X_R). The system is subject to three possible reactions ($m = 3$):*

- R_{inf}*:* $f_{inf}(s) = k_I \frac{s_I}{N} s_S$ *and* $\nu_{inf} = (-1, 1, 0)$*;*
- R_{rec}*:* $f_{rec}(s) = k_R s_I$ *and* $\nu_{rec} = (0, -1, 1)$*;*
- R_{susc}*:* $f_{susc}(s) = k_S s_R$ *and* $\nu_{suc} = (1, 0, -1)$.

The graph of this system is shown in Fig. 2.7, whereas the algebraic expression of the reactions (as in Eq.(2.5)) is the following:

- $\mathsf{R}_{inf} : X_S + X_I \underset{k_I s_I s_S/N}{\rightarrow} X_I + X_I$
- $\mathsf{R}_{rec} : X_I \underset{k_R s_I}{\rightarrow} X_R$
- $\mathsf{R}_{susc} : X_R \underset{k_S s_R}{\rightarrow} X_S$

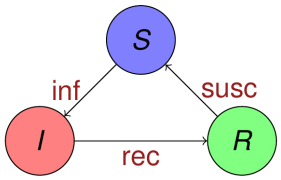


Figure 2.7: Diagram of the SIR epidemiological model.

The time evolution of a CRN can be modelled as a Continuous Time Markov Chain (CTMC) [18] on the discrete space S. Motivated by the well-known memoryless property of CTMC, let $\mathbb{P}_{s_0}(s_t = s)$ denote the probability of finding the system in state s at time t given that it was in state s_0 at time

t_0. This probability satisfies a system of ODEs known as Chemical Master Equation (CME):

$$\partial_t \mathbb{P}_{s_0}(s_t = s) = \sum_{j=1}^{m} \left[f_{\mathsf{R}_j}(s - \nu_j)\mathbb{P}_{s_0}(s_t = s - \nu_j) - f_{\mathsf{R}_j}(s)\mathbb{P}_{s_0}(s_t = s) \right]. \quad (2.6)$$

The equation above is nothing but the Kolmogorov equation for a population process. The Kolmogorov equation model the probability mass, by considering the inflow and outflow probability at time t for a state s. Since the CME is a system in general with countably many differential equations, its analytic or numeric solution is almost always unfeasible. An alternative computational approach is to generate trajectories using stochastic algorithms for simulation, like the well-known Gillespie's SSA [19]. The Stochastic Simulation Algorithm (SSA) is an exact procedure for simulating the time evolution of a CRN. An exact simulation is a trajectory (possible solution) of the CME (Eq. (2.6)). A large collection of samples (trajectories) induced by the CME can be used to extract information about the process via statistical methods.

We will now present some of these simulation algorithms.

Direct method. In each state s, the m reactions compete in a *race condition*: the fastest wins and is executed (see Algorithm 4).

Algorithm 4 Direct simulation method.

1: **procedure** DIRECT(s, t)
2: sample m uniform r.v. u_i;
3: compute $T_i = -\frac{1}{f_{\mathsf{R}_i}(s)} \log(u_i)$;
4: find $i^* = \underset{i \in \{1,\dots,m\}}{\operatorname{argmin}} T_i$;
5: execute reaction R_{i^*} updating the current state from s to $s + \nu_{i^*}$ and the current time to $t + T_{i^*}$.

SSA. We can improve the previous simulation by using the characterization with jump chain at holding times, typical of population CTMC that use exponential propensity functions. Let $f_{\mathsf{R}}(s) = \sum_{i=1}^{m} f_{\mathsf{R}_i}(s)$ be rate of the holding time and let $\mathbb{P}(\mathsf{R_j}|s) = \frac{f_{\mathsf{R}_j}}{f_{\mathsf{R}}}$ be the probabilities associated with the

jump chain of the system in state s, then the Stochastic Simulation Algorithm can be expressed as follows:

Algorithm 5 Stochastic Simulation Algorithm. This method in biochemistry and system biology is also known as Gillespie Algorithm.

1: **procedure** SSA(s, t)
2: sample the next transition from the jump chain;
3: sample the holding time from an $Exp(f_{\mathsf{R}}(x))$;
4: update current state and current time.

τ-leaping method. It is an approximate method of simulation. It aims at improving the computational efficiency with respect to SSA by considering the Poisson representation Ψ of the population CTMC at time τ:

$$s(\tau) = s(0) + \sum_{i=1}^{m} \nu_i \Psi_i \left(\int_0^{\tau} f_{\mathsf{R}_i}(s(t))dt \right). \tag{2.7}$$

The key idea behind the Poisson representation is to perform all reactions for a temporal interval of length τ before updating the propensity functions, so that the rates are updated less often allowing for more efficient simulation. If τ is sufficiently small, we may assume that the rates $f_{\mathsf{R}_i}(s(t))$ are approximately constant in $[0, \tau]$ and equal to a_i. Then $\int_0^{\tau} f_{\mathsf{R}_i}(s(t))dt \approx a_i\tau$, hence

$$s(\tau) = s(0) + \sum_{i=1}^{m} \nu_i \Psi_i(a_i\tau).$$

Algorithm 6 τ-leaping algorithm.

1: **procedure** τ-LEAPING(s, t)
2: choose τ;
3: for each R_i, sample ψ_i from the Poisson r.v. $\Psi_i(a_i\tau)$;
4: update s to $s + \sum_i \nu_i\psi_i$.

Choosing the *leaping condition*, meaning choosing τ, is a complex task. It has to be small enough so that the rates are constant in the interval $[t, t + \tau]$ but it should also be large enough to make $\Psi_i(a_i\tau)$ large to gain in computational efficiency. It is important to find a fine balance between the accuracy of the approximation and the actual speed-up of the simulation algorithm. The τ-leaping algorithm is shown in Algorithm 6.

2.2 Signal Temporal Logic (STL)

Signal temporal logic (STL) [20] was originally developed in order to specify and monitor the expected behaviour of physical systems, including temporal constraints between events. STL allows the specification of properties of dense-time, real-valued signals, and the automatic generation of monitors for testing these properties on individual simulation traces.

A signal is a function $\bar{s} : \mathbb{T} \to \mathbb{V}$, where $\mathbb{T} \subset \mathbb{R}^+$ is the time domain, whereas $\mathbb{V}$ determines the nature of the signal. If $\mathbb{V} = \mathbb{B} := \{true, false\}$, we have a *Boolean signal*. If $\mathbb{V} = \mathbb{R}$, we have a *real-valued signal*. In general, $\mathbb{V}$ can be an arbitrary Cartesian product of sets such as Boolean, reals, etc. As we are dealing with CPS, the signals would be the solutions of the relative HA used to model the CPS. As for executions or traces, we follow the definition of an HA trajectory introduced in Section 2.1. The rationale of STL is to transform real-valued signals into Boolean ones, using predicates built on the following *STL syntax*:

$$\varphi := true \mid g(\bar{s}) \geq 0 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathcal{U}_{\mathsf{I}} \varphi, \tag{2.8}$$

where $\mathsf{I} \subseteq \mathbb{T}$ is a temporal interval, either bounded, $\mathsf{I} = [a, b]$, or unbounded, $\mathsf{I} = [a, +\infty)$, for any $0 \leq a < b$. From this essential syntax it is easy to define other operators, used to abbreviate the syntax in a STL formula: $false := \neg true$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $\Diamond_{\mathsf{I}} := true \, \mathcal{U}_{\mathsf{I}} \varphi$ and $\Box_{\mathsf{I}} := \neg\Diamond_{\mathsf{I}}\neg\varphi$.

2.2.1 STL semantics

Consider a signal $\bar{s}[t] = (\bar{s}_1[t], \ldots, \bar{s}_n[t])$, then atomic predicates are of the form $\mu = (g(\bar{s}_1[t], \ldots, \bar{s}_n[t]) > 0)$, where $g : \mathbb{R}^n \to \mathbb{R}$. The signal $\bar{s}$ is called *primary signal*, whereas $g(\bar{s}[t])$ is called *secondary signal*.

Boolean semantics. The satisfaction of a formula φ by a signal $\bar{s} = (\bar{s}_1, \ldots, \bar{s}_n)$ at time t is defined as:

- $(\bar{s}, t) \models \mu \iff g(\bar{s}_1[t], \ldots, \bar{s}_n[t]) > 0$;
- $(\bar{s}, t) \models \varphi_1 \wedge \varphi_2 \iff (\bar{s}, t) \models \varphi_1 \wedge (\bar{s}, t) \models \varphi_2$;

- $(\bar{s}, t) \models \neg\varphi \iff \neg((\bar{s}, t) \models \varphi)$;
- $(\bar{s}, t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2 \iff \exists t' \in [t+a, t+b]$ s.t. $(\bar{s}, t') \models \varphi_2 \wedge \forall t'' \in [t, t'), (\bar{s}, t'') \models \varphi_1$.

- Eventually: $(\bar{s}, t) \models \Diamond_{[a,b]} \varphi \iff \exists t' \in [t+a, t+b]$ s.t. $(\bar{s}, t') \models \varphi$;
- Globally: $(\bar{s}, t) \models \Box_{[a,b]} \varphi \iff \forall t' \in [t+a, t+b] \quad (\bar{s}, t') \models \varphi$.

Given formula φ and a trace ξ over a time interval T, we can define the *satisfaction*, or *Boolean*, *signal* $\chi^{\varphi}(\xi, \cdot)$ as:

$$\chi^{\varphi}(\xi, t) := \begin{cases} true & \text{if } (\xi, t) \models \varphi \\ false & \text{otherwise,} \end{cases} \qquad \forall t \in \mathsf{T}.$$

Monitoring the satisfaction of a formula is done recursively, by computing $\chi^{\varphi_i}(\xi, \cdot)$ for each sub-formula φ_i of φ. The recursion is performed by leveraging the tree structure of the STL formula, where each node represents a sub-formula, in an incremental fashion, so that the leaves are the atomic propositions and the root represents the whole formula. Thus the procedure goes bottom-up from atomic predicated to the top formula.

Quantitative semantics. The quantitative semantics, meaning the robustness, of a formula φ is defined as a function ρ^{φ}:

- $\rho^{\mu}(\bar{s}, t) = g(\bar{s}_1[t], \ldots, \bar{s}_n[t])$;
- $\rho^{\neg\varphi}(\bar{s}, t) = -\rho^{\varphi}(\bar{s}, t)$;
- $\rho^{\varphi_1 \wedge \varphi_2}(\bar{s}, t) = \min(\rho^{\varphi_1}(\bar{s}, t), \rho^{\varphi_2}(\bar{s}, t))$;
- $\rho^{\varphi_1 \mathcal{U}_{[a,b]} \varphi_2}(\bar{s}, t) =$
$$\sup_{t' \in [t+a, t+b]} \left(\min \left(\rho^{\varphi_2}(\bar{s}, t'), \inf_{t'' \in [t, t']} \rho^{\varphi_1}(\bar{s}, t'') \right) \right).$$

The sign of ρ^{φ} indicates the satisfaction status:

- $\rho^{\varphi}(\bar{s}, t) > 0 \Rightarrow (\bar{s}, t) \models \varphi$;
- $\rho^{\varphi}(\bar{s}, t) < 0 \Rightarrow (\bar{s}, t) \not\models \varphi$.

In this thesis, we use the original definition of quantitative semantics proposed in [21], although there are multiple competing definitions of such robustness value.

As already shown for the Boolean semantics, it is possible to automatically generate monitors for the quantitative semantics as well. The algorithm follows a similar bottom-up approach over the syntax tree of the formula. The main difference is that atomic predicates map a trace on a real value instead of a Boolean value. So to iteratively move up along the tree, we need to have an algorithm to compute the robustness value related to each of the operators in the STL syntax, see (2.8). These `Compute()` algorithms exist and are straightforward to derive. We refer to [22] for the details. Thus for a generic operator $*$, we call `Compute(*)` its algorithm to compute the robustness value. Thus, given a trace ξ, the algorithm starts by computing the quantitative signals of all the atomic propositions, then it goes up on the tree computing the quantitative signals of a node using the signals of its child (see Algorithm 7).

Algorithm 7 Recursive bottom-up algorithm to compute the quantitative semantics of an STL formula.

```
procedure ROBUSTNESS(φ, ξ)
    switch φ do
        case φ = true
            return true                          ▷ a constant true signal
        case f(ξ_i) ≥ 0
            return ξ_i
        case *φ_1
            y:= ROBUSTNESS(φ, ξ)
            return Compute(*,y)
        case φ_1 * φ_2
            y:= ROBUSTNESS(φ_1, ξ)
            y':= ROBUSTNESS(φ_2, ξ)
            return Compute(*,y, y')
```

2.3 Control of a CPS

The aim of this section is to properly formalize the decision making process for HA $\mathcal{M}$, where the continuous part describes the behaviour of the system and the discrete part identifies the possible actions decided by the controller. The state space S coincides with the HA continuous domain, $S := Var$, and the action space A coincides with the HA discrete domain, $A := Loc$. The dynamics of the system are assumed to be Markovian, meaning that the current state of the system is always a sufficient statistics for the future evolution of the system. Mathematically speaking, a system is Markovian if $\mathbb{P}(s_{t+1}|s_1, \ldots, s_t) = \mathbb{P}(s_{t+1}|s_t)$. This property is known as the *memory-less property*.

2.3.1 Markov Decision Process

Consider a scenario in which an agent interacts with a system, as depicted in Fig. 2.8, so that the latter reaches a certain goal. At time t, the agent reads the state of the system s_t and leverages this information to decide the best action a_t. When this action is actuated, the system evolves to a new state s_{t+1}. The system provides the agent an additional numerical feedback: the *reward* associated to the tuple (s_t, a_t, s_{t+1}). The agent will use such rewards to learn how to act optimally in order to reach the goal as soon as possible. Mathematically, such systems are modeled as Markov Decision Processes.

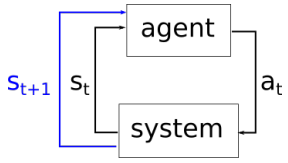


Figure 2.8: Diagram of interaction between an agent and a system.

A Markov Decision Process (MDP) is defined as a tuple (S, A, P, R), where:

- S is the state space;
- A is the action space, i.e. the state space of the agent;

- $P : S \times A \times S \to [0, 1]$, such that $\mathbb{P}(s'|s, a)$ describes the effects over the system of taking action $a \in A$ when the system is in state $s \in S$. This function can be either deterministic or stochastic.
- $R : S \times A \times S \to \mathbb{R}$, such that $R(s, a, s')$ describes the immediate reward related to taking action $a \in A$ that moved the system from s to s'. The reward is the numerical value that the agent receives on performing a certain action at a certain state of the system. Since the system evolution, governed by P, can be either deterministic or stochastic, one can define the immediate reward as the expected reward over $\mathbb{P}(s'|s, a)$, i.e., a function $\mathcal{R} : S \times A \to \mathbb{R}$ such that

  $$\mathcal{R}(s, a) := \mathbb{E}_{s' \sim \mathbb{P}(\cdot|s,a)}[R(s, a, s')|s, a].$$

  The agent can leverage this value to search for the best action given that the system is in state s. The desire is to define a reward strongly related to the overall goal of the system. Typically, an action that brings the system closer to accomplishing the goal should receive a positive reward. Vice-versa, an action that leads to an undesirable state should be penalized by a negative reward. Therefore, the reward is typically defined as a sort of "distance" from the desired configuration of the system.

Policy. A *policy* $\pi : S \to A$ maps a state s to an action a, i.e. $\pi(s) = a$. Given a fixed policy π, the system can evolve following an iterative procedure. Let s_0 be the initial state, the next state is sampled from P, meaning $s_1 \sim \mathbb{P}(\cdot|s_0, \pi(s_0)) := \mathbb{P}_\pi(\cdot|s_0)$. The current state is now s_1 and we simply repeat the sampling procedure, $s_2 \sim \mathbb{P}_\pi(\cdot|s_1)$, and so on.

Objective function. The overall goal of an MDP is to find a good decision-making policy. As always, good is not a universal characterization. Thus we have to define an objective function to measure the goodness of a policy. A typical choice in MDPs is to define a *return* function, i.e., a cumulative function of the rewards R, and maximize this quantity over π. For instance, one can consider the expected discounted sum over a potentially infinite horizon:

$$J_\pi(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right], \tag{2.9}$$

where the expectation is taken over $s_{t+1} \sim P_\pi(\cdot|s_t)$ and $\gamma \in [0, 1]$ is a *discount factor* which values the importance of immediate rewards against that of delayed rewards. When $\gamma \approx 0$, we have a myopic evaluation that prefers shorter solutions, meaning actions taken early. On the other hand, when $\gamma \approx 1$ we have a more far-sighted evaluation. The discount factor is fundamental to avoid infinite returns in case of cyclic MDPs.

A policy that maximizes this objective function is referred to as the *optimal policy* π^*:

$$\pi^* = \arg\max_\pi J_\pi(s_0). \tag{2.10}$$

Partial observability. The solution above assumes that the information about a state s is fully accessible when action is to be taken, otherwise $\pi(s)$ can not be calculated. In case of a *partially observable* state space, $Y \subset S$, we define the policy function as a map from the observed state $y \in Y$ to the action $a \in A$, $\pi : Y \to A$. The MDP system is called a *partially observale* MDP (POMDP). The system, however, is evolving according to the model defined by P.

Defining the objective function as the cumulative reward is the typical solution deployed in reinforcement applications. However, another solution, that is becoming increasingly popular, is to leverage the expressiveness of STL specification to define the behaviour of the system that we want to positively reward and use the quantitative STL semantics as the objective function J_π. More details about this approach will be discussed in Chapter 5.

2.3.2 Controller synthesis

Consider an MDP evolving according to a model P, that can be mathematically formulated as follows. Given a time interval Δt, regulating the sampling time of the controller, we can model the dynamics P of the system as a discrete-time evolution of the form

$$s_{i+1} = F(s_i, a_i, w_i), \tag{2.11}$$

where $s_i \in S$, $a_i \in A$ and $w_i \in W$ are respectively the state, the controller's action and the external disturbance of the system at time $t_i = t_0 + i \cdot \Delta t$.

Eventual stochasticities present in the dynamic P are captured by the external disturbances. The objective function J would thus depend not only on the controller's actions but also on the behaviour of the external disturbances, i.e. the environmental variables. Thus, the objective function $J_\pi(s, \mathbf{w})$ depends on the current state of the system s, on the policy π and on the sequence of disturbances $\mathbf{w}$.

The problem of synthesizing a closed-loop optimal controller, i.e. finding an optimal policy $\pi : S \to A$, can be formulated as follows.

Problem 1 (Deterministic environment)**.** *Consider an MDP, evolving as per Eq. (2.11) from an initial state $s_0 \in S$ for N time steps, a reference disturbance signal $\mathbf{w} = w_1 \cdots w_N$ (known in advance) and a horizon $0 < H < N$, then at each time step $0 \leq k \leq N - H$, find the optimal policy π_k^*, i.e. the policy that satisfies*

$$\pi_k^* = \arg\max_{\pi} \quad J_\pi(s_k, \mathbf{w}_{[\mathbf{k},\mathbf{k+H}]}).$$

The above formulation corresponds to a model predictive control scheme. This means that, in order to be robust to eventual modeling uncertainties, we iteratively apply policy π_k^* for a single step, meaning $s_{k+1} = F(s_k, \pi_k^*, w_k)$, and from s_{k+1} we repeat the optimization procedure.

On the other hand, if the external disturbances are a priori uncertain and potentially adversarial, we formulate the problem of *reactive control synthesis* as follows.

Problem 2 (Uncertain environment)**.** *Consider an MDP, evolving as per Eq. (2.11) from an initial state $s_0 \in S$ for N time steps, and a horizon $0 < H < N$, then at each time step $0 \leq k \leq N - H$, find the optimal policy π_k^*, i.e. the policy that satisfies*

$$\pi_k^* = \arg\max_{\pi} \quad \min_{\mathbf{w}^H \sim \mathcal{W}^H} \quad J_\pi(s_k, \mathbf{w}^H),$$

where $\mathcal{W}^H$ is the distribution governing the evolution of disturbances over an horizon H.

2.4 Verification of a CPS

CPSs are often dealing with safety-critical scenarios. As a consequence, the reliability of their models is a central issue. However, the behaviours of a hybrid automaton are often complex, and it may thus be difficult to reason about them. This is why, since the early works on HA, the emphasis has been on the use of formal methods for computer-aided analysis. The applicability of formal methods, such as verification, control and synthesis, is indeed extremely important. These methods involve a search over the state space of the model. The latter can be, and typically is, infinite, making exact computations extremely challenging [23].

2.4.1 Verification as a reachability problem

Verification of a safety property typically amounts to solving a hybrid automata (HA) reachability checking problem [15]. The aim of reachability analysis is to convert safety properties into sets and compare them with the reachable set to prove or disprove the properties. The reachable set $\mathsf{ReachSet}(\mathit{Init}, \mathcal{M})$ (introduced in Section 2.1) is the region of the hybrid state space that can be reached under the dynamical evolution of the HA starting from the initial region *Init*. A state belongs to the reachable space if there exists a trajectory starting in one point of the initial region that eventually passes through this state. To verify the safety of a CPS, modeled as an HA $\mathcal{M}$, given a set $I \subset \mathit{Init}$ of initial states of $\mathcal{M}$, and a set D of unsafe/dangerous states, we should check whether D is reached along any time-bounded path of $\mathcal{M}$ starting from a state in I. Formally,

$$\mathcal{M} \models \mathcal{R}\mathit{each}(D, I, H) \iff \mathsf{ReachSet}_H(I, \mathcal{M}) \cap D = \emptyset,$$

where H denotes the finite temporal horizon and $\mathsf{ReachSet}_H$ denotes the reachable set after a time H. However, *Init* may be very large or infinite and the HA is not necessarily deterministic, meaning that it is typically unfeasible to compute all the trajectories of the automaton. Therefore, reachability checking is extremely costly from a computational perspective and thus it is usually limited to design-time (offline) analysis [23].

The computation of the reachable set is based on an iterative algorithm (see Algorithm 1). The crucial point in this algorithm is the computation

of the continuous successors, which typically requires numerical integration. Such an algorithm may not converge since the HA reachability problem has been proved to be undecidable [23]. Hence, no algorithm is guaranteed to converge in every case. There exists a variety of approximate techniques that help the algorithm to converge and provide answers about the safety of the system. In particular, an *over-approximation* of the reachable set (see Fig. 2.9) may provide positive answers, meaning proofs that the system is safe, whereas, an *under-approximation* (see Fig. 2.10) may provide negative answers, meaning proofs that the system is unsafe. Nonetheless, computing

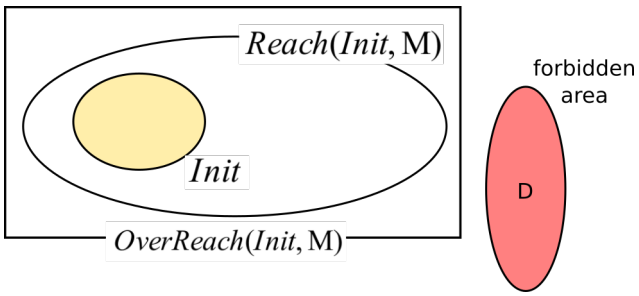


Figure 2.9: Reachable space and an over-approximation of it. If $OverReach(Init, M) \cap D = \emptyset$, then the system is proved to be safe.

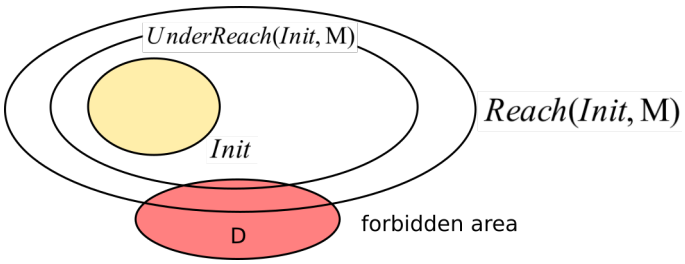


Figure 2.10: Reachable space and an under-approximation of it. If $UnderReach(Init, M) \cap D \neq \emptyset$, then the system is proved to be unsafe.

a sequence of over or under-approximations that converges to the reachable set may be extremely difficult, because of nonlinearities and constraints. The hardest challenge is to identify the best approximation technique. Some tools have no control over the error, meaning that safety information could be unreliable. Some others, more sophisticated, work with fixed precision,

meaning that sometimes they are unable to verify the safety of systems that are indeed safe.

The rationale of these approximate techniques is that of considering a partition of the state space, e.g. a grid of fixed size or a partition tree, and define *denotable sets* as those sets that are representable as the exact union of cells of the partition. A *non-denotable set* can be over or under-approximated by the closest denotable set (see Fig. 2.11). More precisely, when states are hybrid, we need hybrid grids or hybrid partition trees. The chosen partition controls the precision of the approximation. The finer the partition the more accurate the approximation.

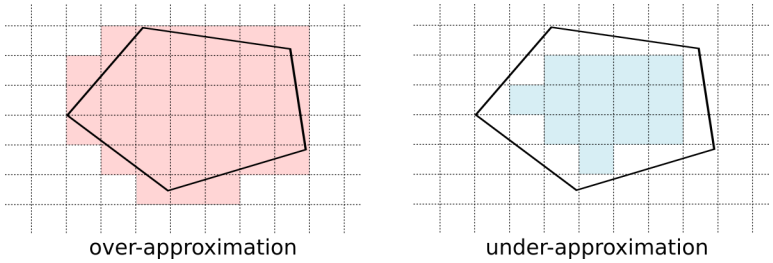


Figure 2.11: Over (left) and under (right) approximation of non-denotable sets defined by the union of cells of a fixed-size grid partition.

Over-approximation. To build an over-approximation, we follow the chain structure of Algorithm 1 and add a bounding set, composed of a finite number of cells, to guarantee termination. At each step, we evolve the current over-approximation by marking the new visited cells. Because of the modularity, multiple integrators (e.g. affine or Euler integrators) can be used for the continuous evolution. If a cell becomes too large during its continuous evolution, it can be subdivided to avoid large errors. In the discrete evolution, all the enabled transitions are evolved so that it is guaranteed to be an over-approximation. Suppose a grid with size ϵ_1 provides an over-approximation $R^{\epsilon_1}_{over}$. If $R^{\epsilon_1}_{over} \cap D = \emptyset$ the system is safe and we can terminate. Otherwise, we can choose $\epsilon_2 < \epsilon_1$, i.e. a finer grid, and compute $R^{\epsilon_2}_{over}$ and repeat until we can prove the safety of the system. It is important to stress that until we find a denotable set R_{over} disjoint from D, we cannot conclude anything about the safety of the system.

Under-approximation. Computing an under-approximation R_{under} could determine whether a system is unsafe. R_{under}, defined as the union of cells $R_{under} = \bigcup_i C_i$, is an under-approximation of ReachSet if each of its cells contains at least a point of ReachSet. If there exists a cell C_i such that $S_i \subset D$, then there is at least one point of the reachable set that reaches the unsafe region D. An under-approximation is harder to compute. For instance, if during the continuous evolution a cell becomes too large, it cannot be sub-divided, since we could not guarantee that all the sub-cells contain at least one point of ReachSet. Moreover, in the discrete evolution, it may be impossible to determine whether a transition is enabled. These two problems typically cause a great loss of accuracy

Falsification. An alternative approach to reason about the safety of a CPS is to search for counterexamples, i.e., for traces that start from $Init$ and reach D in finite-time. The state space is explored by simulating a bunch of trajectories. If the algorithm finds a counterexample, the CPS is proved unsafe, otherwise, no conclusion about the safety of the system can be drawn. Typically, the search for counterexamples is expressed as an optimization problem. For instance, one can use the STL robustness (see Section 2.2) as the objective function to be minimized [24, 25, 26]. The main advantage is that falsification does not require an exhaustive reconstruction of the reachable set. This makes falsification applicable to CPS that are too large or too complex for exhaustive formal verification. Moreover, falsification needs no prior knowledge about the system, except for the ability to run simulations, which makes it applicable to black-box CPS.

Verification tools. Several tools have been developed for automatic verification of CPS properties. Some of them, such as UPPAAL [27] and HyTech [28], perform exact verification but apply only to a restricted class of HS, e.g. rectangular systems or systems with simple dynamics. Other tools, such as PHAVer [29], SpaceEx [30], Flow*, HyPro/HyDra [31], Ariadne [32] and JuliaReach [33] compute an approximation of the reachable set. The most used tools for falsification are Breach [34], S-Taliro [35], C2E2 [36], HyLAA [37]. However, this list is far from providing an exhaustive panoramics of all the available tools for verification. We remind the interested reader

to the ARCH-COMP [1] friendly competitions, where state-of-the-art verification tools are compared on a set of well-known benchmarks. The choice of the best tool depends strongly on the problem at hand and its application domain.

In general, reachability analysis is not an objective on its own but a step in a more general analysis about safety analysis or safe controller synthesis. Our focus is on the online analysis of hybrid systems and, in particular, on the *predictive monitoring* (PM) problem [38]; i.e., the problem of predicting, *at runtime*, whether or not an unsafe state can be reached from the current system state within a given time-bound. PM is at the core of architectures for runtime safety assurance such as Simplex [39], where the system switches to a certified-safe baseline controller whenever PM indicates the potential for an imminent safety violation. In such approaches, PM is invoked periodically and frequently. Thus, reachability needs to be determined rapidly, from a single state (the current system state), and typically for short time horizons. This is in contrast with offline reachability checking, where long or unbounded time horizons and sizable regions of initial states are typically considered. PM also differs from traditional runtime verification [40] in that PM is *preemptive*: it detects potential safety violations before they occur, not when or after they occur. Any solution to the PM problem involves a tradeoff between two main requirements: *accuracy* of the reachability prediction and computational *efficiency*, as the analysis must execute within strict real-time constraints and typically with limited hardware resources.

[1] https://cps-vo.org/group/ARCH/FriendlyCompetition

Chapter 3

Learning Algorithms

This chapter presents the main machine learning techniques used throughout the thesis.

The term machine learning, in general, denotes a set of techniques that allow a computer to acquire knowledge about a system from experience. In other words, it is capable of extracting patterns from observed data. This learning approach avoids the need to formally specify all of the knowledge needed to the computer to understand a certain problem since its knowledge is automatically gathered from experience. For instance, if we want a robot to be able to walk, we could either program the robot to learn to walk or we could attempt to directly write a program that specifies how to walk manually.

Learning algorithms undergo a training process, that improves the performance by learning, at every step, a better representation for the data. In this sense, we can say that machine learning algorithms improve with experience.

Machine learning algorithms can be broadly categorized as *unsupervised* or *supervised* by what kind of experience they are allowed to have during the learning process. Unsupervised learning algorithms experience a dataset and learn useful properties of the structure of this dataset, e.g. it may learn the probability distribution that generated a dataset or it may divide the dataset into clusters of similar examples. Supervised learning algorithms experience a dataset but each example is also associated with a label or target value and

they learn to predict the target to associate to a given example.

3.1 Supervised Learning

Let X be the input space, T be the target space, and define $Z = X \times T$. Let $\mathcal{Z}$ be the data-generating distribution, i.e., the distribution of the points $(x, t) \in Z$. We assume that the target t of a point $(x, t) \in Z$ is the result of the application of a function $f^* : X \to T$, typically unknown or very expensive to evaluate. The goal of a supervised learning algorithm is to find a function $f : X \to T$ that, from a finite set of observations, learns to behave as similarly as possible to f^* over the entire input space. For a generic input $x \in X$, we denote with t the true target value of x and with $\hat{t}$ the prediction by f, i.e. $\hat{t} = f(x)$. Test inputs, whose unknown true target values we aim to predict, are denoted by x_*.

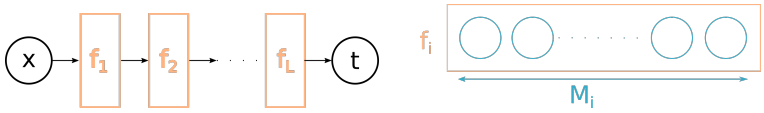
Mathematically, the training problem can be expressed as an optimization problem

$$f^* = \underset{f}{\operatorname{argmin}} \Big(\mathbb{E}_{(x,t)\sim\mathcal{Z}} \left[\|t - f(x)\|^2 \right] \Big). \tag{3.1}$$

If trained on infinitely many samples from the true data generating distribution, $\mathcal{Z}$, minimizing the *mean squared error* loss function gives a function that predicts the *mean* of t for each value of x, $f^*(x) = \mathbb{E}_{t\sim p_{t|x}}[t]$ [41]. Different loss functions give different statistics, for instance the L_1-norm predicts the *median* value of t for each x. Such loss is commonly called *mean absolute error*. As for any optimization procedure, the chosen loss function and the chosen family of models play extremely important roles.

Classification problems. Machine learning classifiers often admit an underlying *discriminant function*, which is used to determine the final classifier output. In neural networks, the discriminant is the function mapping inputs into the class likelihoods (i.e., the softmax probabilities of the last network layer), such that the predicted class is the one with the highest likelihood.

Definition 1 (Discriminant function)**.** *Let $T = \{t^1, \ldots, t^c\}$ be a set of classes. A function $f_d : X \to [0, 1]^c$ is a discriminant for a classifier*



(a) Network structure. (b) Layer structure.

Figure 3.1: Deep Neural Network: composition of successive layers.

$f : X \to T$ *iff for any input* $x \in X$, $f(x) \in \arg\max_{t^i \in T} f_d^i(x)$, *where* $f_d^i(x)$ *is the* i*-th component of* $f_d(x)$.

Remark 1. *If* $\arg\max_{t^i \in T} f_d^i(x)$ *contains more than one class, then the discriminant implies multiple possible predictions for* x*. To avoid this ambiguity, we will assume that* f_d *is always well-defined, meaning that* $\arg\max_{t^i \in T} f_d^i(x)$ *is a singleton, and thus, only one prediction is possible for the classifier* f*. Any discriminant can be made well-defined by, for instance, imposing a total ordering on the classes to select in such ambiguous cases and adequately adjusting the discriminant output. As a matter of fact, any discriminant* f_d *can be turned into a well-formed one by* $f'_d(x) = \{f_d(x)$ *if* $|T_{\max}| = 1$*;* $f_d(x) \cdot (1 + \mathbf{0}_{k \mapsto \epsilon})/(1 + \epsilon)$ *otherwise*$\}$*, where* $T_{\max} = \arg\max_{t^i \in T} f_d^i(x)$, $\epsilon \in \mathbb{R}^+$, $k = \max_{t^i \in T_{\max}} i$ *(based on the ordering on* T*) and* $\mathbf{0}_{k \mapsto \epsilon} \in \mathbb{R}^c$ *is a vector whose components are all zeros but the* k*-th component equals to* ϵ.

3.1.1 Deep Neural Networks

Neural networks choose a family of parametric functions $f_{\mathbf{w}} : X \to T$ as candidate approximations of f^*. These functions have a common network structure, the training is then translated in searching for the parameters (weights) $\mathbf{w}$ that best approximate f^*.

Network. A *feedforward neural network* can be described as a directed acyclic graph describing functions that are composed together. The information flows from the input x, through the intermediate computations, to the output t. For instance, we might have L parametric functions, $f_{w_i}^{(i)}$ for $i = 1, \ldots, L$, connected in a chain (see Figure 3.1(a)). Then $f_{\mathbf{w}}(x) = f_{w_L}^{(L)}(f_{w_{L-1}}^{(L-1)} \cdots (f_{w_2}^{(2)}(f_{w_1}^{(1)}(x))))$ is the function resulting from the composition of these functions, i.e. $f_{\mathbf{w}} = f_{w_1}^{(1)} \circ f_{w_2}^{(2)} \cdots \circ f_{w_L}^{(L)}$. In such scenario, $f_{w_1}^{(1)}$ is

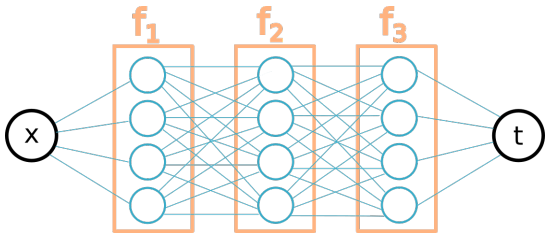


Figure 3.2: Structure of a deep feedforward neural network with 3 hidden layers ($L = 3$) and width 4 ($M = 4$).

called the *first layer*, $f^{(2)}_{w_2}$ is called the *second layer*, and so on. L represent the *depth* of the network. The final layer is called *output layer*. As a group they are called *hidden layers*.

Neural. Each hidden layer $f^{(i)}_{w_i}$ contains a certain number M_i of neurons, meaning it is represented by a M_i-dimensional vector (see Figure 3.1(b)). The dimensionality of these hidden layers determines the *width* of the model. All the neurons can be seen as many units, acting in parallel, each representing a vector-to-scalar function.

Nonlinearity. In order to make the network capable of representing nonlinear functions of x, we can choose the hidden layers $f^{(i)}_{w_i}$ to contain nonlinear transformations $f^{(i)}_{w_i} = \phi\left(f^{(i-1)}_{w_{i-1}}\right)$, where ϕ is a nonlinear *activation function*.

Training of a neural network

The aim of the training phase is to drive $f_{\mathbf{w}}$ as close as possible to f^*. The training data provides us with noisy observations of f^* evaluated at different training inputs. Given a dataset $Z' = \{(x_i, t_i) \in Z\}_{i=1}^N$ and a loss function, $\mathcal{L}(\mathbf{w}, Z')$, that somehow measures the error introduced by $f_{\mathbf{w}}$ over the dataset Z' we can express the training process as an optimization problem

$$\bar{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\Big(\mathcal{L}(\mathbf{w}, Z')\Big) = \underset{\mathbf{w}}{\operatorname{argmin}}\Big(\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\mathbf{w}, (x_i, t_i))\Big).$$

Unfortunately, most of the commonly used loss functions become non-convex because of the nonlinearity of the neural network. Global convergence is thus not guaranteed.

Gradient-based optimization. Gradient descent (GD) [41] is an iterative optimization algorithm that aims at finding a local minimum of a scalar-valued differentiable function. In training a neural network, the function to be minimized, w.r.t. the network weights, is the loss function $\mathcal{L}$ introduced before. Let p be the dimension of the weights space, the gradient $\nabla_w \mathcal{L} : \mathbb{R}^p \to \mathbb{R}^p$ is the vector field whose evaluation at a point $\hat{w}$ is the p-dimensional vector whose components are the partial derivatives of $\mathcal{L}$ at $\hat{w}$: $\nabla_w \mathcal{L}(\hat{w}) = \left[\frac{\partial \mathcal{L}}{\partial w_1}(\hat{w}), \ldots, \frac{\partial \mathcal{L}}{\partial w_p}(\hat{w})\right]$. The direction of the gradient vector in $\hat{w}$ can be interpreted as the direction in which function $\mathcal{L}$ increases most quickly from $\hat{w}$, whereas the magnitude of the gradient is the rate of increase in that direction. The negative of the gradient indicates thus the steepest descent. Gradient descent is an optimization algorithm used to minimize a certain objective function by iteratively moving in the direction of steepest descent. Starting from a random initial point $\hat{w} = w_0$, we iteratively take a step as

$$\hat{w} = \hat{w} - \lambda \nabla_w \mathcal{L}(\hat{w}),$$

where the parameter λ is the *learning rate*, governing the size of each step. The choice of the learning rate is extremely important because if the steps are too big, the local minimum may not be reached because the algorithm bounces back and forth between the convex function (Fig. 3.3-left), if the learning rate is too small, the algorithm may be very slow in reaching the local minimum (Fig. 3.3-right). The best solution is to adapt the learning rate to the dimension of the parameter space and to different phases of the optimization process. Ideally, we should perform a larger step in the early phases, when the minimum is still far, and then reduce the learning rate as we evolve (Fig. 3.3-center). One of the most popular gradient-based optimization algorithms is Adam [42] (Adaptive Moment Estimation).

Mini-batch Gradient Descent. The biggest downside of a GD algorithm is the number of computations needed to perform each step. As a matter

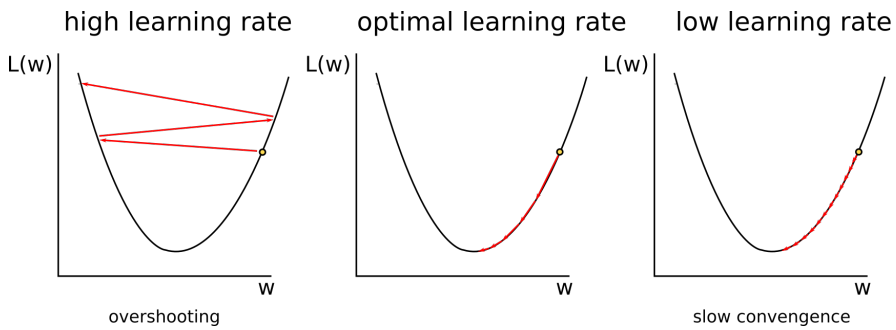


Figure 3.3: Importance of learning rate in gradient descent algorithms.

of fact, every time we update the weights $\hat{w}$ we must evaluate the loss function $\mathcal{L}(\hat{w}, Z')$ over the entire training set Z', which is typically extremely large. Mini-batch GD comes to alleviate such computational costs. At each iteration, mini-batch GD randomly selects a batch of data points out of the training set Z' and, by doing so, it enormously reduces the computational burden of GD. Mini-batch GD is nothing but a stochastic approximation of the GD optimization since it replaces the actual gradient (calculated from the entire data set) with an estimate thereof (calculated from a randomly selected subset of the dataset). If the batch consists of single data points, we call the approach *stochastic gradient descent* (SGD).

Backpropagation. The last missing ingredient is how to compute the gradient of the loss function with respect to the weights. Backpropagation [43] provides the solution. The computation is tackled with an algorithm that involves repeated use of the chain-rule for partial derivatives. In simple terms, after each forward pass through the network, backpropagation performs a backward pass while adjusting the weights. We refer the interested reader to [41, 44] for the mathematical details of the forward and backward passes in a neural network.

3.2 Unsupervised Learning

The goal of unsupervised learning algorithms is that of capturing properties about the structure of a dataset of observations. The dataset can be

considered as a set of samples drawn from an unknown distribution $\mathbb{P}_r$, the data generating distribution, which explains which data are more likely to be found in the data manifold. A *generative model* aims at learning a probabilistic model that mimics this unknown distribution as closely as possible, i.e., learning a distribution $\mathbb{P}_{w_g}$ as similar as possible to $\mathbb{P}_r$. In other words, $\mathbb{P}_{w_g}$ aims at describing how the training dataset is generated. By sampling from this probabilistic model, we are able to generate new data, meaning data outside the training dataset.

Generative modeling could work either on an *unlabeled dataset*, where it learns to estimate the distribution $p(x)$, i.e. the probability of observing x, or on a labeled dataset, where it learns to estimate the conditional distribution $p(x|t)$, i.e., the probability of observing x given a class/target t. The first scenario is known as a *density estimation* problem. The second is referred to as *conditional density estimation* or *density regression* problem.

Representation learning. Instead of modeling a high dimensional space, we typically resort to a low-dimensional *latent space* and then we learn a mapping from points in the latent space to the original domain. By doing so, we learn which features are the most important to describe a set of observations. Mathematically speaking, this method tries to find the highly non-linear *manifold* on which the data lies. It is of paramount importance to this end to establish the dimension that is capable of fully describing this space.

Deep learning methods can be used as generative models, typically referred to as deep generative models. Two states of the art techniques are *Variational Auto Encoders* (VAE) [45] and *Generative Adversarial Networks* (GAN) [46]. In this thesis, we focus on the latter.

3.2.1 Generative Adversarial Nets

Generative Adversarial Nets (GANs) are deep learning-based generative models. More precisely, GANs propose an adversarial model architecture to learn a generative model. It is then common practice to use deep learning models in such architecture. The adversarial architecture proposes a clever way of addressing the problem of training a generative model. The rationale is to

frame the problem as a supervised learning one and divide it into two sub-models. One sub-model, the *generator*, is trained to generate new examples, the other sub-model, the *discriminator*, is trained to classify examples as either real (part of the dataset) or fake (generated by the generator). The two sub-models are trained together in a zero-sum game, until the discriminator is fooled about half the time, meaning the generator is generating plausible examples that are difficult to distinguish from real ones.

Generator. The generator model takes as input a fixed-length random vector k, sampled from a Gaussian distribution, and transforms it into a sample in the data domain. The vector space K is the latent space introduced before. The generative process is reframed as a function transforming a Gaussian distribution into a more complex one. Since we can sample from the latent distribution, we can indeed sample from the more complex generated distribution. The latent variables compress complex observed data. The generator applies meaningful decoding to latent variables. Once the training phase is over, the generator is used to generate new samples.

Discriminator. The discriminator takes as input samples from the data domain, either from the dataset (real) or from the generator's outputs (fake), and labels them as real (class 1) or fake (class 0). Training the discriminator is thus a classical binary classification problem. Once the training phase is over, the discriminator is discarded as it is not useful for the generating process.

Two-player zero-sum game. Although GAN solve an unsupervised learning problem, their training is crafted so that it becomes a supervised learning problem. The clever idea behind GANs is to label samples coming from the training dataset as *real* and samples generated from the generator are labeled as *fake*. The generator and discriminator are then trained together. The generator generates a batch of fake samples, and these, along with real samples are fed into the discriminator that learns a model able to distinguish between real and fake. During the training phase, the goal of the discriminator is to get better at discriminating real and fake samples, whereas the goal of the generator is to try to fool the discriminator, based on how well, or not, the

generated samples fooled the discriminator in previous training epochs. In simple words, the two models are competing against each other, thus they are *adversarial* in the sense of game theory. In particular, they play a *zero-sum game*, meaning that when one model performs well, the other is strongly penalized.

Let $\mathbb{P}_K(k)$ be the prior placed over the latent space K, $\mathsf{G}_{w_g} : K \to X$ be a differentiable function (a neural network) with parameters (weights) w_g and $\mathsf{D}_{w_d} : X \to [0, 1]$ be a neural network with weights w_d. $\mathsf{D}_{w_d}(x)$ represents the probability that x came from the $\mathbb{P}_r$ rather than from $\mathbb{P}_{w_g}$. We train D_{w_d} to maximize the probability of assigning the correct label to real and generated samples and simultaneously we train G_{w_g} to minimize $\log\big(1 - \mathsf{D}_{w_d}(\mathsf{G}_{w_g}(k))\big)$. Mathematically speaking, D_{w_d} and G_{w_g} play the following two-player min-max game with value function $V(\mathsf{G}_{w_g}, \mathsf{D}_{w_d})$:

$$\begin{aligned} V(\mathsf{G}_{w_g}, \mathsf{D}_{w_d}) := \mathbb{E}_{x\sim\mathbb{P}_r(x)}\Big[\log\big(\mathsf{D}_{w_d}(x)\big)\Big] + \\ + \mathbb{E}_{k\sim\mathbb{P}_K(k)}\Big[\log\Big(1 - \mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k)\big)\Big)\Big]. \end{aligned} \tag{3.2}$$

The min-max game is then expressed as

$$\min_{w_g} \max_{w_d} V(\mathsf{G}_{w_g}, \mathsf{D}_{w_d}). \tag{3.3}$$

In practice, we implement the training phase following an iterative process so that: when the discriminator's weights are updated, the generator's weights should be held constant; and when the generator's weights are updated, the discriminator's weights should be held constant.

Vanishing gradients. Each side of the GAN can overpower the other. If the discriminator is too good, it will return values so close to 0 or 1 that the generator will struggle to read the gradient. If the generator is too good, it will persistently exploit weaknesses in the discriminator that lead to poor reconstructions. This may be mitigated by the nets' respective learning rates. Moreover, Eq. (3.3) may not provide sufficient gradient to let G_{w_g} learn properly. Early in learning, when G_{w_g} is still poor, D_{w_d} can easily

discriminate samples with high confidence because they are clearly different from the training data. In this case, $\log\Big(1-\mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k)\big)\Big)$ saturates. A good solution to such problem, known in literature as *vanishing gradients* problem, is to train G to maximize $\log\Big(\mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k)\big)\Big)$ rather than minimizing $\log\Big(1-\mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k)\big)\Big)$. This modified value function results in the same fixed point of the dynamics of G_{w_g} and D_{w_d} but provides much stronger gradients early in learning. The min-max problem is thus split into two separate problems. The discriminator is trained in order to maximize the value function of Eq. (3.2) with respect to w_d:

$$\max_{w_d} V(\mathsf{G}_{w_g}, \mathsf{D}_{w_d}).$$

On the other hand, the generator is trained by solving the following problem:

$$\max_{w_g} \log\Big(\mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k)\big)\Big).$$

Algorithm 8 summarizes the step on the training procedure and Figure 3.4 shows the GAN architecture.

We refer the reader to [46] for a more detailed discussion of the theoretical properties of this approach. For example how, under an optimal discriminator, the value function 3.2 is proportional to the Jensen-Shannon divergence[1].

Mode collapse. The goal of a well-performing GAN is to produce a wide variety of outputs to obtain a reconstruction of the entire target distribution and not only of its mode. However, the generator tries to find the one output that seems most plausible to the discriminator. If it finds an especially plausible output, it may decide to produce that output only. At each iteration, the generator may over-optimize for a particular discriminator. As a result, the generators rotate through a small set of output types. This form of GAN failure is called *mode collapse*. The reason for such behaviour is mathematically grounded and related to instabilities of the gradients of

[1]The Jensen-Shannon divergence between two distributions P and Q is defines as $JSD(P||Q) = (KL(P||M) + KL(Q||M))/2$, where $M = (P+Q)/2$ and $KL(A||B) := -\int_X \log \frac{dB}{dA} dA$ denotes the Kullback-Leibler divergence between two probability measures A and B over a space X, where $\frac{dB}{dA}$ is the Radon–Nikodym derivative of B w.r.t. A.

Algorithm 8 Minibatch stochastic gradient descent training of generative adversarial nets. Hyperparameters: number of epochs n_{epochs}, number of iterations for the discriminator $n_{discrim}$, number of iterations for the generator n_{gen}, batch size b, learning rate λ.

procedure TRAIN GAN
  for $e = 1, \ldots, n\_epochs$ **do**
    for $j = 1, \ldots, n\_discr$ **do**
      for $i = 1, \ldots b$ **do**
        Sample a latent variable $k^{(i)} \sim \mathbb{P}_K(k)$;
        Sample a real data $x^{(i)} \sim \mathbb{P}_r(x)$;
        $\mathcal{L}^{(i)} \leftarrow \log \mathsf{D}_{w_d}\big(x^{(i)}\big) + \log\Big(1 - \mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k^{(i)})\big)\Big)$
      Update the discriminator:
      $w_d \leftarrow \text{Adam}\left(\nabla_{w_d} \frac{1}{b} \sum_{i=1}^{b} \mathcal{L}^{(i)}, \lambda\right)$;
    for $k = 1, \ldots, n_{gen}$ **do**
      Sample batch of latent variables $\{k^{(i)}\}_{i=1}^{b} \sim \mathbb{P}_K(k)$;

      Update the generator:
      $w_g \leftarrow \text{Adam}\Big(\nabla_{w_g} \frac{1}{b} \sum_{i=1}^{b} \log\Big(1 - \mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k^{(i)})\big)\Big), \lambda\Big)$.

the Jensen Shannon divergence, used to measure the difference between the distributions in a GAN. For this reason, using a different distance would alleviate the mode collapse problem (see Wasserstein GAN below).

Networks architecture. The type of networks used for the two models depends strongly on the type of samples present in the dataset, i.e. it depends on the structure of the data domain. If images or fixed-length sequences are used, it is common practice to use a Convolutional Neural Network (CNN) [41] for both the discriminator and the generator. On the contrary, feedforward and recurrent neural networks can be used if they better suit the data domain.

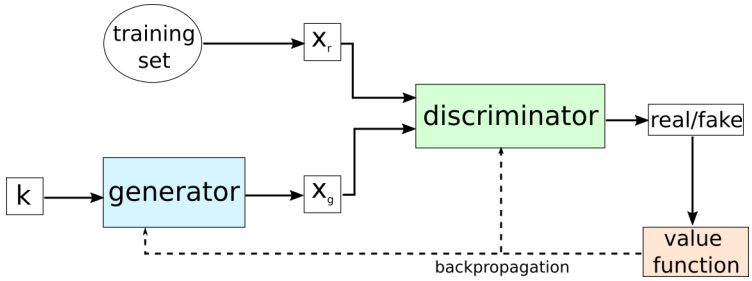


Figure 3.4: Diagram of a Generative Adversarial Network. x_r denotes real samples coming from the dataset, whereas x_g denotes generated samples, i.e. $x_g = \mathsf{G}_{w_g}(k)$.

Conditional GAN

When data in the domain belong to different classes, we may decide to generate a new sample, ignoring the class it will belong to, or we could aim at generating new samples that are likely to belong to a specific class. In the first scenario, a traditional GAN (introduced before) is enough, whereas in the second scenario a conditional GAN is needed. A conditional GAN (cGAN) [47] is an extension, allowing the generative process to condition on certain information. The conditioning input could either be a categorical discrete information, such as a class label, or a continuous value.

In a cGAN both the generator and discriminator are conditioned on some extra information t. The discriminator will take as input both a sample x, either real or fake, and the condition t. In turn, the generator will take as input a sample k from the latent space and the condition t, so that it will learn to generate examples of that class in order to fool the discriminator. The value function of Eq. (3.2) is then reformulated as:

$$\begin{aligned} V(\mathsf{G}_{w_g}, \mathsf{D}_{w_d}) := \mathbb{E}_{x\sim\mathbb{P}_r(x)}\Big[\log\big(\mathsf{D}_{w_d}(x|t)\big)\Big] + \\ + \mathbb{E}_{k\sim\mathbb{P}_K(k)}\Big[\log\Big(1-\mathsf{D}_{w_d}\big(\mathsf{G}_{w_g}(k|t)\big)\Big)\Big]. \end{aligned} \tag{3.4}$$

Figure 3.5 shows the cGAN architecture.

GANs are an extremely powerful yet very unstable tool. They are highly sensitive to chosen hyper-parameters and are subject to vanishing gradients

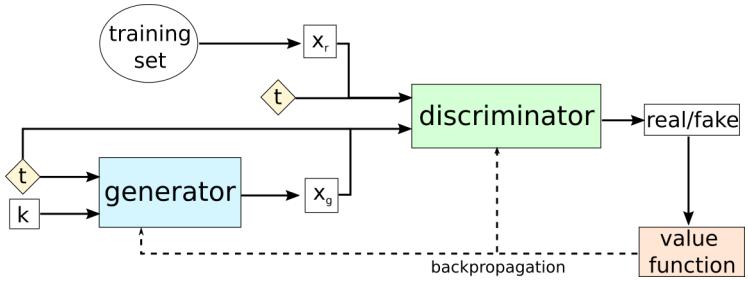


Figure 3.5: Diagram of a conditional Generative Adversarial Network. x_r denotes real samples coming from the dataset, whereas x_g denotes generated samples, i.e. $x_g = \mathsf{G}_{w_g}(k)$.

and the mode collapse problem. In this perspective, Wasserstein GAN proposes a new formulation that overcomes most of the aforementioned problems.

Wasserstein GAN

Wasserstein GAN (WGAN) [48, 49] are a version of GAN known to be more stable and less sensitive to the choice of model architecture and hyperparameters compared to a traditional GAN. WGANs use the Wasserstein distance (also known as Earth-Mover's distance), rather than the Jensen Shannon divergence, to measure the difference between the model distribution $\mathbb{P}_{w_g}$ and the target distribution $\mathbb{P}_r$.

The *Wasserstein distance* (*Earth-Mover (EM) distance*) between two distributions is defined as

$$W(\mathbb{P}_r, \mathbb{P}_{w_g}) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_{w_g})} \mathbb{E}_{(x,y)\sim\gamma} \left[||x - y|| \right], \tag{3.5}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_{w_g})$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_{w_g}$. Intuitively, $\gamma(x, y)$ indicates how much "mass" must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_{w_g}$. The EM distance then is the "cost" of the optimal transport plan. The EM is a much more sensible cost function for optimization problems than the Jensen-Shannon divergence [48]. Even if computing the infimum in (3.5) is intractable, the Kantorovich-Rubinstein

duality [50] allow us to rewrite such distance as the supremum over all the 1-Lipschitz functions $f : X \to \mathbb{R}$:

$$W(\mathbb{P}_r, \mathbb{P}_{w_g}) = \sup_{||f||_L \leq 1} \left(\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_{w_g}}[f(x)] \right). \tag{3.6}$$

If we have a parametrized family of functions $\{f_\theta\}_{\theta \in \Theta}$ that are all 1-Lipschitz, we could solve the problem:

$$\max_{\theta \in \Theta} \mathbb{E}_{x \sim \mathbb{P}_r}[f_\theta(x)] - \mathbb{E}_{x \sim \mathbb{P}_{w_g}}[f_\theta(x)], \tag{3.7}$$

whose computation would yield the distance $W(\mathbb{P}_r, \mathbb{P}_{w_g})$. In order to select such a parametric family of functions, we define a neural net C_{w_c} parametrized by weights w_c. To enforce the Lipschitz constraint we follow [49] and introduce a penalty over the norm of the gradients. It is known that a differentiable function is 1-Lipchitz if and only if it has gradients with norm at most 1 everywhere. The objective function, to be maximized w.r.t. w_c, becomes:

$$\mathcal{L}(w_c, w_g) := \mathbb{E}_{x \sim \mathbb{P}_r}[\mathsf{C}_{w_c}(x)] - \mathbb{E}_{x \sim \mathbb{P}_{w_g}}[\mathsf{C}_{w_c}(x)] - \beta \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}(\|\nabla_{\hat{x}} \mathsf{C}_{w_c}(\hat{x})\|_2 - 1)^2], \tag{3.8}$$

where β is the penalty coefficient and $\mathbb{P}_{\hat{x}}$ is defined by sampling uniformly along straight lines between pairs of points sampled from $\mathbb{P}_r$ and $\mathbb{P}_{w_g}$. The C_{w_c} network is referred to as *critic* and it outputs different scores for real and fake samples, its objective function (Eq. (3.8)) provides an estimate of the Wasserstein distance among the two distributions. On the other hand, the distribution $\mathbb{P}_{w_g}$ is parametrized by w_g; we seek the parameters that make it as close as possible to $\mathbb{P}_r$. To achieve this, we consider a random variable K with a fixed simple distribution $\mathbb{P}_K$ and pass it through a parametric function, the *generator*, $\mathsf{G}_{w_g} : K \to X$ that generates samples following the distribution $\mathbb{P}_{w_g}$. Therefore, the WGAN architecture consists of two deep neural nets, a generator that proposes a distribution and a critic that estimate the distance between the proposed and the real (unknown) distribution. Using WGAN brings several important advantages compared to traditional GAN: it avoids the mode collapse problem, which makes WGAN more suitable for capturing stochastic dynamics, it drastically reduces the problem of vanishing gradients and it also has an objective function that correlates with the quality of generated samples, making the results easier to interpret. Another possible approach to enforce the Lipschitz constraint over the critic neural net is to clip the weights to lie in a compact space $\overline{W}$. If $\overline{W}$ is compact, then f_w is K-Lipschitz and thus we get an estimate of $K \cdot W(\mathbb{P}_r, \mathbb{P}_{w_g})$ [48]. Although

Algorithm 9 Conditional WGAN with gradient penalty. Default values used for hyper-parameters: $\beta = 10$, $n_{critic} = 5$, $\alpha = 0.0001$, $\beta_1 = 0.5$, $\beta_2 = 0.9$. Variable x denotes the input, variable t denotes the condition.

procedure TRAIN CWGAN-GP(number of epochs n_{epochs}, number of critic iterations per generator iteration n_{critic}, batch size b, learning rate β)
  for $e = 1 \ldots, n_{epochs}$ **do**
    for $k = 1 \ldots, n_{critic}$ **do**
      for $i = 1 \ldots, b$ **do** ▷ Mini-Batching
        Sample a real data $(t, x) \sim \mathbb{P}_r$;
        Sample a latent variable $k \sim \mathbb{P}_K(k)$;
        Sample a random number $\epsilon \sim U[0, 1]$
        $\tilde{x} \leftarrow \mathsf{G}_{w_g}(k, t)$
        $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
        $\mathcal{L}^{(i)} \leftarrow \mathsf{C}_{w_c}(\tilde{x}, t) - \mathsf{C}_{w_c}(x, t) + \beta(\| \nabla_{\hat{x}} \mathsf{C}_{w_c}(\hat{x}, t) \|_2 - 1)^2$
      $w_c \leftarrow \text{Adam}\left(\nabla_{w_c} \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}^{(i)}, w_c, \alpha, \beta_1, \beta_2\right)$
    Sample a batch of latent variables $\{k^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_K(k)$
    Sample a batch of random conditions $\{t^{(i)}\}_{i=1}^{m} \sim \mathbb{P}(t)$
    $w_g \leftarrow \text{Adam}\left(\nabla_{w_g} \frac{1}{m} \sum_{i=1}^{m} -\mathsf{C}_{w_c}(\mathsf{G}_{w_g}(k^{(i)}, t^{(i)}), t^{(i)}), w_g, \beta\right)$

easier to implement, this approach has been shown to lead to optimization difficulties [49], and even when optimization succeeds the resulting critic can have a pathological value surface. Therefore the WGAN-GP approach (Algorithm 9) is preferable.

To summarize, GAN and WGAN are in general very strong and flexible as they can handle high-dimensional data and they can easily learn to generate multi-modal outputs. Interesting application areas are data augmentation and data-editing.

3.3 Uncertainty quantification

The predictions made by the deep-learning algorithms presented in Section 3.1, although accurate, are doomed to encounter some errors. It is the very nature of an approximation to allow for some error. However, it could

be extremely beneficial to have an understanding of how much a prediction is trustworthy. Quantification of the predictive uncertainty allows us to define alternative strategies in the most uncertain scenarios. We present to alternative approaches: a *Bayesian* (Section 3.3.1) and a *frequentist* (Section 3.3.2) one.

3.3.1 Bayesian Neural Networks for classification

A neural network is a function $f_{\mathbf{w}} : X \to [0,1]^c$, which maps an input $x \in X$ into a vector of class likelihoods, $f_{\mathbf{w}}(x) = [f^1_{\mathbf{w}}(x), \ldots, f^c_{\mathbf{w}}(x)]$, depending on some parameters $\mathbf{w}$, namely weights and biases. A trained neural network is typically a complex but deterministic model. The core idea of Bayesian neural networks (BNNs) is to place a probability distribution over its parameters $\mathbf{w}$, thereby transforming the neural network into a stochastic model. The advantage of Bayesian methods, and BNNs in particular, is that they provide a distribution of predictions, called *predictive distribution*, rather than a single prediction like deterministic NNs. Such distribution captures both the aleatoric uncertainty, i.e., the noise inherent in the observations, and the epistemic uncertainty, i.e. model uncertainty about its prediction [51]. We will therefore leverage this predictive distribution, its mean and variance to be precise, to compute our Bayesian uncertainty measures.

The Bayesian learning process starts by defining a prior distribution for $\mathbf{w}$ that expresses our initial belief about the parameter values. As we observe data $Z' \sim \mathcal{Z}$, we update this prior to a posterior distribution using Bayes' rule:

$$p(\mathbf{w}|Z') = \frac{p(Z'|\mathbf{w})p(\mathbf{w})}{p(Z')}. \tag{3.9}$$

Note that placing a prior distribution over $\mathbf{w}$ is analogous to the random weight initialization required to train a traditional (deterministic) neural network. A common choice is to choose a zero-mean Gaussian prior.

Similarly, we assume that the conditional distribution $p(t|x)$ is a softmax likelihood

$$p(t = t^j|x, \mathbf{w}) = \frac{\exp(f^j_{\mathbf{w}}(x))}{\sum_{i=1}^{c} \exp(f^i_{\mathbf{w}}(x))}. \tag{3.10}$$

It follows that, given a set of i.i.d. observations Z', the likelihood function

can be expressed as

$$p(Z' \mid \mathbf{w}) = \prod_{(x_i,t_i)\in Z'} p(t_i|x_i, \mathbf{w}). \tag{3.11}$$

Note that, because of the non-linearity introduced by the neural network function $f_{\mathbf{w}}(x)$ and by the softmax likelihood, the posterior $p(\mathbf{w}|Z')$ is non-Gaussian. Finally, in order to predict the value of the target for an unobserved input x_*, we marginalize the predictions with respect to the posterior distribution of the parameters, obtaining

$$p(\hat{t}_*|x_*, Z') = \int p(\hat{t}_*|x_*, \mathbf{w})p(\mathbf{w}|Z')d\mathbf{w}. \tag{3.12}$$

The latter is called *posterior predictive* distribution and it can be used to retrieve information about the uncertainty of a specific prediction $\hat{t}_*$. Unfortunately, the integration is analytically intractable due to the non-linearity of the neural network function [44, 52].

Empirical approximation of the predictive distribution . Assume, for the moment, that we are able to sample from the posterior distribution (3.9) and let $[w_1, \ldots, w_N]$ denote a vector of N realizations of the random variable $\mathbf{w} \sim p(\mathbf{w}|Z')$. Each realization w_i induces a deterministic function f_{w_i} that can be evaluated at x_*, the unobserved input. The likelihood for a target $\hat{t}_*$ can be computed using (3.10). The empirical approximation of the predictive distribution (3.12) can be expressed as

$$\begin{aligned} p(\hat{t}_*|x_*, Z') &\approx \frac{1}{N}\sum_{i=1}^{N} p(\hat{t}_*|x_*, w_i) \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\exp\big(f^*_{w_i}(x_*)\big)}{\sum_{j=1}^{c}\exp\big(f^j_{w_i}(x_*)\big)}\right], \end{aligned} \tag{3.13}$$

where $f^*_{w_i}$ denotes the component of f_{w_i} corresponding to class $\hat{t}_*$. By the strong law of large numbers, the empirical approximation converges to the true distribution as $N \to \infty$ [53]. The sample size N can be chosen, for instance, to ensure a given width of the confidence interval for a statistic

of interest [54] or to bound the probability that the empirical distribution differs from the true one by at most some given constant [55].

Bayesian inference techniques. Since precise inference is infeasible, various approximate methods have been proposed to infer a BNN. We consider two approximate solution methods: Hamiltonian Monte Carlo and Variational Inference.

Let w be a weight vector sampled from the posterior distribution $p(\mathbf{w}|Z')$, i.e., a realization of the random variable $\mathbf{w}$. We denote with $f_w(x)$ the corresponding deterministic neural network having weights fixed to w.

Hamiltonian Monte Carlo (HMC) [56] defines a Markov chain whose invariant distribution is exactly the posterior $p(\mathbf{w}|Z')$.

The Hamiltonian dynamics is used to speed up space exploration. HMC does not make any assumption on the form of the posterior distribution and is asymptotically correct. After convergence, HMC returns a trace of explored network weights $w_0, w_1, \ldots, w_N$ that, all together, can be interpreted as an empirical approximation of the posterior $p(\mathbf{w}|Z')$. Controlling in a precise way the convergence rate and how well the chain explores the parameter space is, however, far from trivial.

Variational Inference (VI) [57] directly approximates the posterior distribution with a known parametric distribution $q(\mathbf{w};\psi)$, typically a distribution easy to sample from. Its parameters ψ, called variational parameters, are learned by minimizing the Kullback-Leibler (KL) divergence between the proposed distribution and the posterior. The KL divergence between $q(\mathbf{w};\psi)$ and $p(\mathbf{w}|Z')$ is defined as

$$KL(q(\mathbf{w};\psi)||p(\mathbf{w}|Z')) = \int q(\mathbf{w};\psi)\log\frac{q(\mathbf{w};\psi)}{p(\mathbf{w}|Z')}d\mathbf{w}. \tag{3.14}$$

Since the posterior distribution is not known, a different objective function, called Evidence Lower Bound (ELBO), is introduced. It is defined as

$$ELBO_\psi = \mathbb{E}_{q(w;\psi)}\{\log p(Z'|\mathbf{w}) - KL(q(\mathbf{w};\psi)||p(\mathbf{w}))\}. \tag{3.15}$$

ELBO [58]. In VI, the variational objective, i.e., the negative ELBO, becomes the loss function used to train a Bayesian neural network [58]. A

common choice for $q(\mathbf{w};\psi)$ is the Gaussian distribution (where ψ are its mean and variance).

The predictive distribution is a non-linear combination of Gaussian distributions, and thus it is not Gaussian. However, samples can be easily extracted from $q(\mathbf{w};\psi)$, which allows us to obtain an empirical approximation of the predictive distribution.

Prediction uncertainty. Having shown how to derive empirical approximations of the predictive distribution (either with HMC or with VI) we can now extract statistics to characterize the latter. We stress that the predictive distribution, and hence its statistics, effectively captures predictive uncertainty.

3.3.2 Conformal Predictions

In the following, we provide background on conformal prediction considering a generic prediction model.

Conformal Prediction associates measures of reliability to any traditional supervised learning problem. It is a very general approach that can be applied across all existing classification and regression methods [59, 60]. CP produces *prediction regions with guaranteed validity.*

Definition 2 (Prediction region)**.** *For significance level $\epsilon \in (0,1)$ and test input x_*, the ϵ-prediction region for x_*, $\Gamma^\epsilon_* \subseteq T$, is a set of target values s.t.*

$$\Pr_{(x_*,t_*)\sim\mathcal{Z}}(t_* \in \Gamma^\epsilon_*) = 1-\epsilon. \tag{3.16}$$

The idea of CP is to construct the prediction region by "inverting" a suitable hypothesis test: given a test point x_* and a tentative target value t', we *exclude* t' from the prediction region only if it is unlikely that t' is the true value for x_*. The test statistic is given by a so-called *nonconformity function (NCF)* $\delta : Z \to \mathbb{R}$, which, given a predictor f and a point $z = (x,t)$, measures the deviation between the true value t and the corresponding prediction $f(x)$. In this sense, δ can be viewed as a generalized residual function. In other words, CP builds the prediction region Γ^ϵ_* for a test point x_* by excluding

all targets t' whose NCF values are unlikely to follow the NCF distribution of the true targets:

$$\Gamma^{\epsilon}_{*} = \left\{ t' \in T \mid Pr_{(x,t)\sim\mathcal{Z}} \left(\delta(x_*, t') \geq \delta(x, t) \right) > \epsilon \right\}. \tag{3.17}$$

The probability term in Eq. 3.17 is often called the p-value. From a practical viewpoint, the NCF distribution $Pr_{(x,t)\sim\mathcal{Z}}(\delta(x,t))$ cannot be derived in an analytical form, and thus we use an empirical approximation derived using a sample Z_c of $\mathcal{Z}$. This approach is called *inductive CP* [61] and Z_c is referred to as *calibration set*.

Remark 2 (Assumptions and guarantees of inductive CP)**.** *Importantly, CP prediction regions have finite-sample validity [59], i.e., they satisfy* (3.16) *for any sample of $\mathcal{Z}$ (of reasonable size), and not just asymptotically. On the other hand, CP's theoretical guarantees hold under the exchangeability assumption (a "relaxed" version of iid) by which the joint probability of any sample of $\mathcal{Z}$ is invariant to permutations of the sampled points. Independent observations are exchangeable but sequential ones are not (due to the temporal dependency). Even though sequential data violate CP's theoretical validity, we will find that the prediction regions still attain empirical coverage, i.e. the probability that the prediction region contains the correct target value, is consistent with the nominal coverage (see results section), that is, the probabilistic guarantees still hold in practice (as also found in previous work on CP and time-series data [59]).*

Validity and Efficiency. CP performance is measured via two quantities: 1) *validity* (or *coverage*), i.e. the empirical error rate observed on a test sample, which should be as close as possible to the significance level ϵ, and 2) *efficiency*, i.e. the size of the prediction regions, which should be small. CP-based prediction regions are automatically valid (under the assumptions of Remark 1), whereas the efficiency depends on the chosen nonconformity function and thus on the underlying model.

CP for classification

In classification, the target space is a discrete set of possible labels (or classes) $T = \{t^1, \ldots, t^c\}$. We represent the classification model as a function f_d :

$X \to [0,1]^c$ mapping inputs into a vector of class likelihoods, such that the predicted class is the one with the highest likelihood[2].

The inductive CP algorithm for classification is divided into an offline phase, executed only once, and an online phase, executed for every test point x_*. In the offline phase (steps 1–3 below), we train the classifier f and construct the calibration distribution, i.e., the empirical approximation of the NCF distribution. In the online phase (steps 4–5), we derive the prediction region for x_* using the computed classifier and distribution.

1. Draw sample Z' of $\mathcal{Z}$. Split Z' into training set Z_t and calibration set Z_c.
2. Train classifier f using Z_t. Use f_d to define an NCF δ.
3. Construct the calibration distribution by computing, for each $z_i \in Z_c$, the NCF score $\alpha_i = \delta(z_i)$.
4. For each label $t^j \in T$, compute $\alpha_*^j = \delta(x_*, t^j)$, i.e., the NCF score for x_* and t^j, and the associated p-value p_*^j:

$$p_*^j = \frac{|\{z_i \in Z_c \mid \alpha_i > \alpha_*^j\}|}{|Z_c|+1} + \theta \frac{|\{z_i \in Z_c \mid \alpha_i = \alpha_*^j\}|+1}{|Z_c|+1}, \tag{3.18}$$

   where $\theta \in \mathcal{U}[0,1]$ is a tie-breaking random variable.
5. Return the prediction region

$$\Gamma_*^\epsilon = \{t^j \in T \mid p_*^j > \epsilon\}. \tag{3.19}$$

In defining the NCF δ, we should aim to obtain high δ values for wrong predictions and low δ values for correct ones. Thus, a natural choice in classification is to define

$$\delta(x, t^j) = 1 - f_d^j(x), \tag{3.20}$$

where $f_d^j(x)$ is the likelihood predicted by f_d for class t_j. Indeed, if t^j is the true target for x and f correctly predicts t^j, then $f_d^j(x)$ is high (the highest among all classes) and $\delta(x, t^j)$ is low; the opposite holds if f does not predict t^j.

[2]Ties can be resolved by imposing an ordering over the classes.

Prediction uncertainty. A CP-based prediction region provides a set of plausible predictions with statistical guarantees, and as such, also captures the uncertainty about the prediction. Indeed, if CP produces a region Γ^ε_* with more than one class, then the prediction for x_* is *ambiguous* (i.e., multiple predictions are plausible), and thus, potentially erroneous. Similarly, if Γ^ε_* is empty, then there are no plausible predictions at all, and thus, none can be trusted. The only reliable prediction is the one where Γ^ε_* contains only one class. In this case, $\Gamma^\varepsilon_* = \{\hat{t}_*\}$, i.e., the region only contains the predicted class. This is always true for our NCF function (Eq. (3.20)), as shown in the following proposition.

Proposition 1. *For the NCF function* (3.20)*, if* $\Gamma^\varepsilon_* = \{t^{j_1}\}$*, then* $t^{j_1} = f(x_*)$*.*

Proof. Suppose by contradiction that $\Gamma^\varepsilon_* = \{t^{j_1}\}$ and $t^{j_1} \neq f(x_*) = t^{j_2}$. Then, by Equation 3.19, this implies that $p^{j_1}_* > \varepsilon$ and that $p^{j_2}_* \leq \varepsilon$, i.e., $p^{j_1}_* > p^{j_2}_*$. In turn, this implies that the corresponding NCF scores are such that $\alpha^{j_1}_* \leq \alpha^{j_2}_*$ (the inequality is not strict due to the tie-braking factor θ in Equation 3.18). But according to the definition of our NCF function (3.20), this means that $f_d^{t^{j_1}}(x_*) \geq f_d^{y^{j_2}}(x_*)$, i.e., that the likelihood of the non-predicted class t^{j_1} is not below than that of the predicted class t^{j_2}, which, by Definition 1 and the assumption of well-formed discriminant, is a contradiction. □

The size of the prediction region is determined by the chosen significance level ε and by the p-values derived via CP. Specifically, from Equation (3.19) we can see that, for levels $\varepsilon_1 \geq \varepsilon_2$, the corresponding prediction regions are such that $\Gamma^{\varepsilon_1} \subseteq \Gamma^{\varepsilon_2}$. It follows that, given a test input x_*, if ε is lower than all its p-values, i.e. if $\varepsilon < \min_{j=1,\dots,c}\ p^j_*$, then the region Γ^ε_* contains all the classes, and Γ^ε_* shrinks as ε increases. In particular, Γ^ε_* is empty when $\varepsilon \geq \max_{j=1,\dots,c}\ p^j_*$.

In the classification scenario, CP introduces two additional point-wise measures of uncertainty, called confidence and credibility, defined in terms of two p-values, independently of the significance level ε. The intuition is that these two p-values identify the range of ϵ values for which the prediction is reliable, i.e., $|\Gamma^\varepsilon_*| = 1$.

Definition 3 (Confidence and credibility)**.** *Given a predictor* F*, the confidence of a point* $x_* \in X$*, denoted by* $1 - \gamma_*$*, is defined as:*

$$1 - \gamma_* = \sup\{1 - \varepsilon : |\Gamma^\varepsilon_*| = 1\}, \tag{3.21}$$

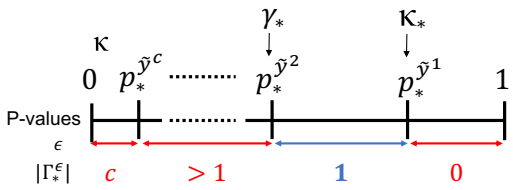


Figure 3.6: CP p-values over the $[0, 1]$ interval and corresponding sizes of prediction interval. $\tilde{y}^i$ is the class with the i-th largest p-value, so $p_*^{\tilde{\ell}^1} = \kappa_*$ and $p_*^{\tilde{\ell}^2} = \gamma_*$.

and the credibility of x_*, denoted by κ_*, is defined as:

$$\kappa_* = \inf\{\varepsilon : |\Gamma_*^\varepsilon| = 0\}. \tag{3.22}$$

Therefore, the so-called confidence-credibility interval $[\gamma_*, \kappa_*)$ contains all the values of ε such that $|\Gamma_*^\varepsilon| = 1$.

The confidence $1-\gamma_*$ is the highest probability value for which the corresponding prediction region contains only $\hat{t}_*$, and thus it measures how likely (according to the calibration set) our prediction for x_* is.

In particular, γ_* corresponds to the second largest p-value. The credibility κ_* is the smallest level for which the prediction region is empty, i.e., no plausible prediction is found by CP. It corresponds to the highest p-value, i.e., the p-value of the predicted class. Figure 3.6 illustrates CP p-values and corresponding prediction region sizes. In binary classification problems, each point x_* has only two p-values: κ_* (p-value of the predicted class) and γ_* (p-value of the other class).

It follows that the higher $1-\gamma_*$ and κ_* are, the more reliable the prediction $\hat{t}_*$ is, because we have an expanded range $[\gamma_*, \kappa_*)$ of ε values by which $|\Gamma_*^\varepsilon| = 1$. Indeed, in the degenerate case where $\kappa_* = 1$ and $\gamma_* = 0$, then $|\Gamma_*^\varepsilon| = 1$ for any value of $\varepsilon < 1$. This is why, as we will explain in the next section, our uncertainty-based rejection criterion relies on excluding points with low values of $1-\gamma_*$ and κ_*. Hence, our frequentist uncertainty measure associates with each input its confidence and credibility values.

Mondrian approach. The validity property, as stated above, guarantees an error rate over all possible labels, not on per-label basis. The latter can be achieved with a CP variant, called *label-conditional CP*, which is in turn a variant of the Mondrian CP approach. The only change is in the calculation of the p-values. The p-value associated to class t^j on a test point x_* is defined as:

$$p_*^j = \frac{|\{z_i \in Z_c : t_i = t^j, \alpha_i > \alpha_*^j\}|}{|\{z_i \in Z_c : t_i = t^j\}| + 1} + \tag{3.23}$$

$$+ \theta \frac{|\{z_i \in Z_c : t_i = t^j, \alpha_i = \alpha_*^j\}| + 1}{|\{z_i \in Z_c : t_i = t^j\}| + 1}. \tag{3.24}$$

In words, we consider only the α_i corresponding to examples with the same label t^j as the hypothetical label that we are assigning at the test point.

Label-conditional validity [62] is extremely important when the CP is applied to an unbalanced dataset. It has been shown empirically, that, with the plain validity property, the overall error rates tend to the chosen significance level, but the minority class are disproportionally affected by errors. The Mondrian approach ensures that, even for the minority class, the expected error rate will tend to the chosen significance level ε. We refer the reader to the existing literature [63, 64] for further details.

NCF for Support Vector Classifiers. CP relies on the definition of a nonconformity measure, every classification algorithm has a specific nonconformity measure. SVC is a kernel-based method that transforms the original data by mapping them into a new space, called feature space, via a feature map $\phi(x)$. By doing so, patterns that are not linearly separable can be converted to be linearly separable in the feature space [44]. The linear decision boundary for a binary SVC is defined as $d(x) = a \cdot \phi(x) + b = 0$, for x in the original space. Support vectors are the data points that lie closest to the decision hyper-plane. SVC maximizes the margin around the separating hyperplane and the decision function, which depends only on the support vectors. For ease of notation, we are assuming $Y = \{-1, 1\}$, rather than $\{0, 1\}$. The distance of a point x_* from the separating hyperplane of the SVC is given by:

$$d_*^h = \frac{|d(x_*)|}{||a||},$$

where $d(x_*)$ is SVC decision function $d(\cdot)$ evaluated in x_* and $||a||$ is the weighted sum of the support vectors. Then, the distance to the margin boundary of the class under consideration is given by:

$$d_*^m = \frac{|d(x_*)| - 1}{||a||}.$$

The non-conformity measure is thus defined as

$$\alpha_*^j = \exp(-d_*^m).$$

Such definition is presented in [65]. In the case of transductive CP (TCP), the NCM can be derived directly from the value of Lagrange multipliers associated with the support vectors, as proposed in [64].

CP for Regression

In regression, we have a continuous target space $T \subseteq \mathbb{R}^n$. The CP algorithm for regression is similar to the classification one. In particular, the offline phase of steps 1–3, i.e., training of regression model f and definition of NCF δ, is the same (with obviously a different kind of f and δ).

The online phase changes though, because T is a continuous space and thus, it is not possible to enumerate the target values and compute for each a p-value. Instead, we proceed in an equivalent manner, that is, identify the critical value $\alpha_{(\epsilon)}$ of the calibration distribution, i.e., the NCF score corresponding to a p-value of ϵ. The resulting ϵ-prediction region is given by $\Gamma_*^\epsilon = f(x_*) \pm \alpha_{(\epsilon)}$, where $\alpha_{(\epsilon)}$ is the $(1-\epsilon)$-quantile of the calibration distribution, i.e., the $\lfloor \epsilon \cdot (|Z_c| + 1) \rfloor$-th largest calibration score[3].

A natural NCF in regression, and the one used in our experiments, is the norm of the difference between the real and the predicted target value, i.e., $\delta(x) = ||t - f(x)||$.

[3]Such prediction intervals have the same width ($\alpha_{(\epsilon)}$) for all inputs. There are techniques like [66] that allow constructing intervals with input-dependent widths, which can be equivalently applied to our problem.

Part II

Contributions

Chapter 4

Model Abstraction

A wide range of complex systems can be modeled as a network of chemical reactions (see Section 2.1.2). Stochastic simulation is typically the only feasible analysis approach that scales in a computationally tractable manner with the increase in system size, as it avoids the explicit construction of the state space. The well known Gillespie Stochastic Simulation Algorithm [67] is widely used for simulating models, as it samples from the exact distribution over trajectories. This algorithm is effective to simulate systems of moderate complexity, but it does not scale well to systems with many species and reactions, large populations, or internal stiffness. In these scenarios, a more effective choice is to rely on approximate simulation algorithms such as tau-leaping [68] and hybrid simulation [69]. Nonetheless, when the number of simulations required is extremely large and possibly costly, e.g. when one needs to simulate a large population of heterogeneous cells in a multi-scale model of a tissue or to simulate many heterogeneous individuals in a population ecology scenario, all these methods become extremely computationally demanding, even for HPC facilities.

A viable approach to address such a problem is model abstraction, which aims at reducing the underlying complexity of the model and thus reducing its simulation cost. However, building effective model abstractions is difficult, requiring a lot of ingenuity and manpower. Here we advocate the strategy of learning an abstraction from simulation data. Our strategy is to frame model abstraction as an unsupervised learning problem and learn an abstract

probabilistic model using state of the art deep learning. The probabilistic model should then be able to generate approximate trajectories efficiently and in constant time, i.e., independent of the complexity of the original system, thus sensibly reducing the simulation cost.

Related work. The idea of using machine learning as a model abstraction tool to approximate and simplify the dynamics of a Markov Population Process has received some attention in recent years. In [70] the authors use a Mixture Density Network (MDN) [44] to approximate the transition kernel of the stochastic process. In [71] the authors extend the previous approach by introducing an automated search of the MDN architecture that better fit the data. In [72] the authors present a Bayesian model abstraction technique, based on Dirichlet Processes, that allows the quantification of the reconstruction uncertainty. In all cases, what is learned is an approximate transition kernel, i.e., the probabilistic distribution of a single simulation step. In this work, we address a more general and more complex problem. Instead of learning an approximate transition kernel, we learn the distribution of an entire trajectory of fixed length. This latter problem is not solvable with any of the previously adopted approaches, and its major goal is to keep abstraction error under control. In fact, training the abstract model on a full trajectory, rather than on pairs of subsequent states, allows the abstract model to retain and capture more information about the dynamics of the Markov process.

Contributions. Our approach leverages Generative Adversarial Nets, which are one of the most strong and flexible techniques to learn probabilistic models. In fact, the GAN-based model abstraction technique is capable of learning a conditional distribution over the trajectory space, keeping into account the correlation, both spatial and temporal, among all the different species and conditioning both on initial states and model parameters. All the previous approaches focus on learning the distribution of the state of the system after a time Δt, the so-called *transition kernel*. However, such approaches perform poorly when the time interval is small and the dynamics are transient, showing a clear propagation of the error as the approximate kernel is applied iteratively to form a trajectory. Furthermore, producing a full trajectory reduces, even more, the computational cost of simulating a large pool of trajectories for different initial settings.

4.1 Problem Statement

Let us briefly recall the formalism of a chemical reaction network, introduced in Section 2.1.2, and the concept of a condition Wasserstein Generative Adversarial Network (c-WGAN), introduced in Section 3.2.1.

4.1.1 Preliminaries

Chemical Reaction Networks. Consider a system with n species evolving according to a stochastic model defined as a Chemical Reaction Network. Under the well-stirred assumption, the time evolution can be modelled as a Continuous Time Markov Chain (CTMC) on a discrete state space. The vector $s_t = (s_{1,t}, \ldots, s_{n,t}) \in S \subseteq \mathbb{N}^n$ denotes the state vector at time t, where $s_{i,t}$ is the number of individuals in species i at time t. The dynamics is encoded by a set of m reactions with parametric propensity functions that depends on the state of the system. Due to the memoryless property of CTMC, the probability of finding the system in state s at time t given that it was in state s_0 at time t_0 can be expressed as a system of ODEs known as Chemical Master Equation (CME).

Conditional WGAN-GP. Wasserstein GAN (WGAN) are a deep generative model based on the use of the Wasserstein distance to measure the difference between the model distribution $\mathbb{P}_{w_g}$ and the target distribution $\mathbb{P}_r$, i.e., $W(\mathbb{P}_r, \mathbb{P}_{w_g})$. Such metric can be expressed in a functional form as:

$$W(\mathbb{P}_r, \mathbb{P}_{w_g}) = \sup_{||f||_L \leq 1} \Big(\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_{w_g}}[f(x)] \Big) . \tag{4.1}$$

The WGAN rationale is to define a neural net C_{w_c}, parametrized by weights w_c and representing a parametric family of functions, and to enforce the Lipschitz constraint, we introduce a penalty over the norm of the gradients. On the other hand, the neural net *generator*, G_{w_g}, parametrized by weights w_g, generates samples following the distribution $\mathbb{P}_{w_g}$. We seek the parameters that make it as close as possible to $\mathbb{P}_r$. The WGAN architecture consists of two deep neural nets, a generator that proposes a distribution and a critic that estimate the distance between the proposed and the real (unknown)

distribution (see Section 3.2.1 for more details). Furthermore, we are going to use a conditional formulation of WGAN.

4.1.2 Learning an abstraction

The underlying idea is the following: given a stochastic process $\{s_t\}_{t\geq 0}$ with transition probabilities $\mathbb{P}_{s_0}(s_t = s) = \mathbb{P}(s_t = s \mid s_{t_0} = s_0)$, we aim at finding another stochastic process whose trajectories are faster to simulate but similar to the original ones. Time has to be discretized, meaning we fix an initial time t_0 and a time step Δt that suits our problem. We define $\tilde{s}_i := s_{t_0+i\cdot\Delta t}$, $\forall i \in \mathbb{N}$. In addition, given a fixed time horizon H, we define time-bounded trajectories as

$$\tilde{\xi}_{[1,H]} = \tilde{s}_1\tilde{s}_2\cdots\tilde{s}_H \in S^H \subseteq \mathbb{N}^{H\times n}.$$

Given a state s_0 and a set of parameters θ, we can represent a trajectory of length H as a realization of a random variable over the state space S^H. The probability distribution for such random variable is given by the product of the transition probabilities at each time step:

$$\mathbb{P}_{s_0,\theta}(\tilde{\xi}_{[1,H]} = \tilde{s}_1\tilde{s}_2\cdots\tilde{s}_H) = \prod_{i=1}^{H}\mathbb{P}_{\tilde{s}_{i-1},\theta}(\tilde{\xi}_i = \tilde{s}_i).$$

The CTMC, $\{s_t\}_{t\geq 0}$, is now expressed as a time-homogeneous Discrete Time Markov Chain $\{\tilde{s}_i\}_i$. An additional approximation has to be made: the abstract model takes values in $S' \subseteq \mathbb{R}^n_{\geq 0}$, a continuous space in which the state space $S \subseteq \mathbb{N}^n$ is embedded. In constructing the approximate probability distribution for trajectories we can decide to restrict our attention to arbitrary aspects of the process, rather than trying to preserve the full behaviour. A *projection* proj from S^H to an arbitrary space Y^H can be used to reach this purpose, for instance, to monitor the number of molecules belonging to a certain subset of chemical species, i.e., $Y \subseteq S$. Note that $\mathsf{proj}(\tilde{\xi}_{[0,H]})$ is a random variable over Y^H. Such flexibility could be extremely helpful in capturing the dynamics of systems in which some species are not observable.

Abstraction accuracy. A meaningful quantification of the error introduced by the abstraction procedure, i.e., the reconstruction accuracy, is another important ingredient. Such quantification must be based on a distance,

d, among distributions. We choose the Wasserstein distance, together with the absolute and relative difference among means and variances of the histograms. Given a distribution over initial states s_0 and a distribution over parameters θ, we would like to measure the expected error at every time instant $t_i = t_0 + i \cdot \Delta t$ with $i \in \{1, \ldots, H\}$. Formally, we want to measure

$$\mathbb{E}_{s_0,\theta}\left[d\big(\mathsf{proj}(\xi_{[1,H]})\big|_i, \mathsf{proj}'(\xi'_{[1,H]})\big|_i\big)\right],$$

where $\mathsf{proj}(\xi_{[1,H]})\big|_i$ denotes the i-th time components of the projected trajectory $\mathsf{proj}(\xi_{[1,H]}) \in Y^H$. To estimate such quantity we use a well-known unbiased estimator, which is the average over the distances computed over a large sample set of initial settings. Computing the distance among SSA and abstract distributions at each time step quantify how small the expected error is and, more importantly, how it evolves in time. As a matter of fact, it shows whether the error tends to propagate or not and how much each species contributes to the abstraction error. In practice, we compute $H \cdot n$ distances among distributions over $\mathbb{N}$ as we want to know how each species contributes to the reconstruction error.

Definition 4 (Abstract process)**.** *Let (s_0, θ) be the initial settings and let $\xi_{s_0,\theta} = \{\tilde{s}_i\}_{i=1}^H$ be a discrete time stochastic process, conditioned on the initial settings, over an arbitrary space S^H, the space of trajectories of length H, and let $\mathsf{proj} : S^H \to Y^H$ be a projection function. An* abstraction *of $(\xi_{s_0,\theta}, \mathsf{proj})$ is a discrete time stochastic process $(\xi'_{s_0,\theta}, \mathsf{proj}')$ where:*

- *$S' \subseteq \mathbb{R}^n_{\geq 0}$ is the abstract state space,*
- *$\mathsf{proj}' : S'^H \to Y^H$ is the abstract projection function,*
- *$\xi'_{s_0,\theta} = \{s'_i\}_{i=1}^H$ is an abstract discrete stochastic process over S'^H.*

Given the distribution over initial states and the distribution over parameters, the abstract stochastic process should minimize the expected error *at every time instant $t_i = t_0 + i \cdot \Delta t$, with $i \in \{1, \ldots, H\}$, i.e., it should minimize*

$$\mathbb{E}_{s_0,\theta}\left[d\big(\mathsf{proj}(\xi_{s_0,\theta})\big|_i, \mathsf{proj}'(\xi'_{s_0,\theta})\big|_i\big)\right],$$

where $\mathsf{proj}(\xi'_{s_0,\theta})\big|_i$ denotes the i-th time components of the projected trajectory $\mathsf{proj}(\xi_{s_0,\theta}) \in Y^H$ and d denotes an arbitrary distance among distributions.

4.1.3 Dataset Generation

Training set. Choose a set of N_{train} initial settings and for each setting simulate k_{train} SSA trajectory of length H. The training set is composed of $N_{train} \cdot k_{train}$ pairs initial setting-trajectory, i.e. pairs $(\theta^i, s_0^i, \xi_{[1,H]}^{ij})$ for $i = 1, \ldots, N_{train}$ and $j = 1, \ldots, k_{train}$.

Test set. Choose a set of N_{test} initial settings and for each setting simulate a large number, $k_{test} \gg k_{train}$, of SSA trajectory of length H. The test set is composed of $N_{test} \cdot k_{test}$ pairs initial setting-trajectory, i.e. pairs $(\theta^i, s_0^i, \xi_{[1,H]}^{ij})$ for $i = 1, \ldots, N_{test}$ and $j = 1, \ldots, k_{test}$.

Partial observability. In case of partial observability, $Y \subseteq S$, we fix an initial condition for species in Y, and simulate a pool of trajectories each time sampling the initial value of species in $S \setminus Y$. As a result, we are learning an abstract distribution that marginalizes over unobserved variables.

4.1.4 cWCGAN-GP architecture

Dealing with inputs that are trajectories, i.e. sequences of fixed length, requires the use of convolutional neural networks (CNNs) [73] for both the generator and the critic. The condition can be either the initial state or some parameters or both. The architecture used in this work is thus a conditional Wasserstein Convolutional GAN with gradient penalty, it is going to be referred to as cWCGAN-GP.

The critic C_{w_c} takes as input a batch of initial states, $s_0^1, \ldots, s_0^b$, a batch of parameters, $\theta_1, \ldots, \theta_b$, and a batch of subsequent trajectories, $\xi_{[1,H]}^1, \ldots, \xi_{[1,H]}^b$. For each $i \in \{1, \ldots, b\}$ the inputs, $\xi_{[1,H]}^i$, s_0^i and θ_i, are concatenated to form an input with dimension $b \times (H+1) \times (n+m)$ (see Fig. 4.1).

Formally, $C_{w_c} : S^{H+1} \times \Theta \to \mathbb{R}$. To enforce the Lipschitz property over C_{w_c} we add a gradient penalty term over $\mathbb{P}_{\hat{x}}$. Samples of $\mathbb{P}_{\hat{x}}$ are generated by sampling uniformly along straight lines connecting points coming from a batch of real trajectories and points coming from a batch of generated

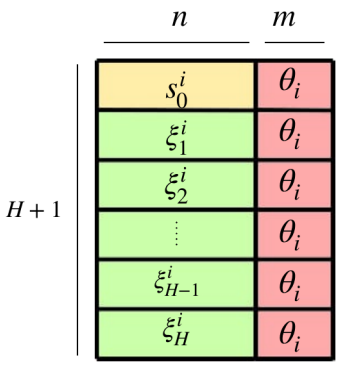


Figure 4.1: Concatenated input of the critic net C_{w_c}.

trajectories.

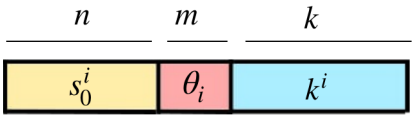


Figure 4.2: Concatenated input of the generator net G_{w_g}.

On the other hand, the generator G_{w_g} takes as input a batch of initial states, $s_0^1, \ldots, s_0^b$, a batch of parameters, $\theta_1, \ldots, \theta_b$, and a batch of random noise, $k^1, \ldots, k^b$, with dimension k, a user-defined hyper-parameter. For each $i \in \{1, \ldots, b\}$ the two inputs are, once again, concatenated to form an input with dimension $b \times (n+m+k)$ (see Fig. 4.2). The generator outputs a batch of generated trajectories $\hat{\xi}_{[1,H]}^1, \ldots \hat{\xi}_{[1,H]}^b$. Formally, $G_{w_g} : S \times \Theta \times K \to S^H$, such that $G_{w_g}(s_0, \theta, k) = \hat{\xi}_{[1,H]} = s_1 \cdots s_H$. See the diagram in Fig. 4.3 and the pseudocode for the algorithm in Section 3.2.1.

4.1.5 Model Training

The cWCGAN-GP-based model abstraction framework consists in training two different CNNs. The loss function, introduced in Eq. (3.8), is a parametric function depending both on the generator weights w_g and the critic weights w_c. When training the critic, we keep the generator weights constant

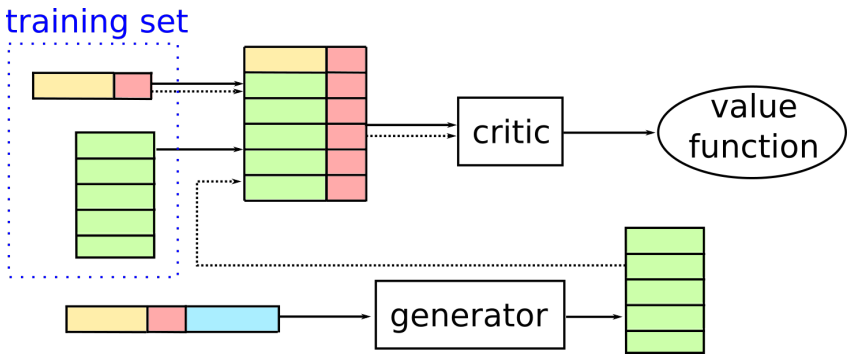


Figure 4.3: Diagram of the cWCGAN architecture.

$\overline{w}_g$, and we maximize $\mathcal{L}(w_c, \overline{w}_g)$ w.r.t. w_c. Formally, we solve the problem

$$w_c^* = \underset{w_c}{\text{argmax}}\Big\{\mathcal{L}(w_c, \overline{w}_g)\Big\}.$$

On the other hand, in training the generator, we keep the critic weights constant $\overline{w}_c$, and we minimize $\mathcal{L}(\overline{w}_c, w_g)$ w.r.t. w_g. Formally, we solve the problem

$$w_g^* = \underset{w_g}{\text{argmin}}\Big\{\mathcal{L}(\overline{w}_c, w_g)\Big\} = \underset{w_g}{\text{argmin}}\Big\{ - \mathbb{E}_{k,(s_0,\theta)}\Big[C_{\overline{w}_c}(G_{w_g}(z, s_0, \theta), s_0, \theta)\Big]\Big\}.$$

As mentioned in Section 3, the loss function derives from the Wasserstein distance between the real and generated distributions, see [48, 49] for the mathematical details.

Intuitively, the generator generates a batch of samples, and these, along with real examples from the dataset, are provided to the critic, which is then updated to get better at estimating the distance between the real and the abstract distribution. The generator is then updated based on scores obtained by the generated samples from the critic. An important collateral advantage is that WGANs have a loss function that correlates with the quality of generated examples.

Training the cWCGAN-GP has a cost. Nonetheless, once it has been trained, its evaluation is extremely fast. Details about training and evaluation costs are discussed in Section 6.5.

Abstract Model Simulation. Once the training is over, we can discard the critic and focus only on the trained generator G. In order to generate an abstract trajectory starting from a state s_0^* with parameters θ^*, we just have to sample a value k from the random noise variable K and evaluate the generator on the pair (s_0^*, θ^*, k). The output is a stochastic trajectory of length H: $G(s_0^*, \theta^*, k) = \hat{\xi}_{[1,H]}$. The stochasticity is provided by the random noise variable, de facto the generator acts as a distribution transformer that maps a simple random variable into a complex distribution. In order to generate a pool of p trajectories, we simply sample p different values from the random noise variable: $k_1, \ldots, k_p$. Therefore, the generation of a trajectory has a fixed computational cost.

4.2 Experimental Results

In this section, we validate our GAN-based model abstraction procedure on the following case studies.

- **SIR Model (Absorbing state).** The SIR epidemiological model describes a population divided in three mutually exclusive groups: susceptible (S), infected (I) and recovered (R). The system state at time t is $\eta_t = (S_t, I_t, R_t)$. The possible reactions, given by the interaction of individuals (representing the molecules of a CRN), are the following:

  - $R_1 : S + I \xrightarrow{\theta_1 \cdot I_t S_t / (S_t + I_t + R_t)} 2I$ (infection),
  - $R_2 : I \xrightarrow{\theta_2 \cdot I_t} R$ (recovery).

  The model describes the spread, in a population, of an infectious disease that grants immunity to those who recover from it. As the SIR model is well-known and stable, we use it as a testing ground for our GAN-based abstraction procedure. The ranges for the initial state are $S_0, I_0, R_0 \in [30, 200]$. An important aspect of the SIR model is the presence of absorbing states. In fact, when $I = 0$ or when $R = N$ no more reaction can take place.

- **Ergodic SIRS Model.** Small perturbations of the SIR model force the system to be ergodic. We called this revised version ergodic SIRS

(eSIRS). This model has no absorbing state. In particular, we assume that the population is not perfectly isolated, meaning there is always a chance of getting infected by some external individuals. In addition, we also assume that immunity is only temporary. The possible reactions are now the following:

- $R_1 : S + I \xrightarrow{\theta_1 \cdot I_t S_t/(S_t+I_t+R_t)+\theta_2 \cdot S_t} 2I$ (infection),
- $R_2 : I \xrightarrow{\theta_3 \cdot I_t} R$ (recovery),
- $R_3 : R \xrightarrow{\theta_4 \cdot R_t} S$ (immunity loss),

Both epidemiological models are essentially unimodal. The ranges for the initial state are $S_0, I_0, R_0 \in [0, N]$ such that $S_0 + I_0 + R_0 = N$. In our experiments $N = 100$. The range for parameter θ_1 is $[0.5, 5]$.

- **Genetic Toggle Switch Model (Bistability).** The toggle switch is a well-known bistable biological circuit. Briefly, this system consists of two genes, G_1 and G_2, that mutually repress each other. The system displays two stable equilibrium states in which either of the two gene products represses the expression of the other gene. The possible reactions are:
  - $prod_i : G_i^{on} \xrightarrow{kp_i \cdot G_i^{on}} G_i^{on} + P_i$, for $i = 1, 2;$,
  - $bind_i : 2P_j + G_i^{on} \xrightarrow{kb_i \cdot G_i^{on} \cdot P_j \cdot (P_j - 1)} G_i^{off}$, for $i = 1, 2$ and $j = 2, 1$ resp.;
  - $unbind_i : G_i^{off} \xrightarrow{ku_i \cdot G_i^{off}} G_i^{on} + 2P_j$, for $i = 1, 2$ and $j = 2, 1$ resp.;
  - $deg_i : P_i \xrightarrow{kd_i \cdot Pi} \emptyset$, for $i = 1, 2$.

  The ranges for the initial state are $G_{1,0}, G_{2,0}^{on} \in \{0, 1\}$ and $P_{1,0}, P_{2,0} \in [5, 20]$.

- **Oscillator Model.** The oscillator circuit consists of three species A, B and C and three reactions, in which A converts B to itself, B converts C to itself, and C converts A to itself. The three species regulate each other in a cyclic manner. This circuit was found to exhibit oscillations in the concentrations of the three species.

- $R_1 : A + B \xrightarrow{\theta \cdot \frac{A \cdot B}{A+B+C}} 2A$ (B transformation),
- $R_2 : B + C \xrightarrow{\theta \cdot \frac{B \cdot C}{A+B+C}} 2B$ (C transformation),
- $R_3 : C + A \xrightarrow{\theta \cdot \frac{C \cdot A}{A+B+C}} 2C$ (A transformation).

The ranges for the initial state are $A_0, B_0, C_0 \in [20, 100]$.

- **MAPK Model.** Mitogen-activated protein kinase cascade is a particular type of signal transduction into protein phosphorylation (PP) whose function is the amplification of a signal. The sensitivity increases with the number of cascade levels, such that a small change in a stimulus results in a large change in the response. Negative feedback from MAPK-PP to the MAKKK activating reaction with ultra-sensitivity to an input stimulus, governed by parameter V_1.

  - $R_1 : MKKK \xrightarrow{f_{R_1}} MKKK\_P$,
  - $R_2 : MKKK\_P \xrightarrow{V_2 \cdot MKKK\_P/(K_2+MKKK\_P)} MKKK$,
  - $R_3 : MKK \xrightarrow{k_3 \cdot MKKK\_P \cdot MKK/(K_3+MKK)} MKK\_P$,
  - $R_4 : MKK\_P \xrightarrow{k_4 \cdot MKKK\_P \cdot MKK\_P/(K_4+MKK\_P)} MKK\_PP$,
  - $R_5 : MKK\_PP \xrightarrow{V_5 \cdot MKK\_PP/(K_5+MKK\_PP)} MKK\_P$,
  - $R_6 : MKK\_P \xrightarrow{V_6 \cdot MKK\_P/(K_6+MKK\_P)} MKK$,
  - $R_7 : MAPK \xrightarrow{k_7 \cdot MKK\_PP \cdot MAPK/(K_7+MAPK)} MAPK - P$,
  - $R_8 : MAPK\_P \xrightarrow{f_{R_8}} MAPK\_PP$,
  - $R_9 : MAPK\_PP \xrightarrow{V_9 \cdot MAPK\_PP/(K_9+MAPK\_PP)} MAPK\_P$,
  - $R_{10} : MAPK\_P \xrightarrow{V_{10} \cdot MAPK\_P/(K_{10}+MAPK\_P)} MAPK$,

  where

  $$f_{R_1} = V_1 \cdot MKKK/((1 + (MAPK\_PP/K_l)^n) \cdot (K_1 + MKKK))$$
  $$f_{R_8} = k_8 \cdot MKK - PP \cdot MAPK\_P/(K_8 + MAPK\_P).$$

  The ranges for the initial state are:

$MKKK_0, MKKK\_P_0 \in [0, 100]$ such that

$$MKKK_0 + MKKK\_P_0 = 100;$$

$MKK_0, MKK\_P_0, MKK\_PP_0 \in [0, 300]$ such that

$$MKK_0 + MKK\_P_0 + MKK\_PP_0 = 300;$$

$MAPK_0, MAPK\_P_0, MAPK\_PP_0 \in [0, 300]$ such that

$$MAPK_0 + MAPK\_P_0 + MAPK\_PP_0 = 300.$$

To evaluate the performance of our abstraction procedure, we consider two important measures: the accuracy of the abstract model, evaluated for each species at each time step of the time grid, and the computational gain compared to SSA simulation time.

Experimental Settings. The workflow can be divided into steps: (1) define a CRN model, (2) generate the synthetic datasets via SSA simulation, (3) learn the abstract model by training the cWCGAN-GP and, finally, (4) evaluate such abstraction. All the steps have been implemented in Python. In particular, CRN models are defined in the `.psc` format, CRN trajectories are simulated using Stochpy [74] (stochastic modeling in Python) and PyTorch [75] is used to craft the desired architecture for the cWCGAN-GP and to evaluate the latter on the test data. All the experiments were performed on a Intel Xeon Gold 6140 with 24 cores and 128GB RAM and a Tesla V100 GPU. The source code for all the experiments can be found at the following link: https://github.com/francescacairoli/WGAN_ModelAbstraction.

Datasets. For each case study with fixed parameters, the training set consists of 20K different SSA trajectories. In particular, $N_{train} = 2\text{K}$ and $k_{train} = 10$. The test set, instead, consists of 25 new initial settings and from each of these we simulate 2K trajectories, so to obtain an empirical approximation of the distribution targeted by model abstraction. When a parameter is allowed to vary, the training set consists of 50K SSA trajectories ($N_{train} = 1\text{K}$ and $k_{train} = 50$). The dimension of the training set is

required to increase exponentially with the dimension of the conditioning space $S \times \Theta$ [76]. However, in order to alleviate the curse of dimensionality, one can condition on a subset Y of $S \times \Theta$, marginalizing over the remaining variables $(S \times \Theta) \setminus Y$. Alternatively, when the intrinsic dimension of a data manifold is considerably lower than its full dimension [77], one can leverage eventual correlations among the conditioning variables and thus reduce the amount of training observations needed to reach accurate reconstructions. We manually choose H and Δt so that the system is close to steady state at time $H \cdot \Delta t$, without spending there too many steps. The time interval should be small enough to capture the full transient behavior of the system. If it is to large, the abstraction looses in granularity. For systems with no steady state, such as the oscillating models, we choose H and Δt so to observe a full period of oscillation. The chosen values are the following: SIR: $\Delta t = 0.5$, $H = 16$; e-SIRS: $\Delta t = 0.1$, $H = 32$; Toggle Switch: $\Delta t = 0.1$, $H = 32$; Oscillator: $\Delta t = 1$, $H = 32$; MAPK: $\Delta t = 60$, $H = 32$. It is important to remember that if $H \cdot \Delta t$ is very large, trajectories could be dominated by the steady-state behaviour. This could results in a poor reconstruction of the transient behaviours. On the other hand, a shorter horizon $H \cdot \Delta t$ should not affect the performances of the abstract model. Furthermore, we note that higher values of H result in a longer dataset generation time, but they also translate in a higher computational gain once the abstract model is learnt.

Data Preparation. Data have been scaled to the interval $[-1, 1]$ to enhance the performance of the two CNNs and to avoid sensitivity to different scales in species counts. During the evaluation phase, the trajectories have been scaled back. Hence, results and errors are shown in the original scale.

Architecture details. The same architecture and the same set of hyperparameters work well for all the analyzed case studies, showing great stability and usability of the proposed solution. The Wasserstein formulation of GANs, with gradient penalty, strongly contributes to such stability. Traditional GANs have been tested as well, but they do not have such strength. The details of the architecture follow the best practice suggestions provided in [49]. The critic network has two hidden one-dimensional convolutional layers, with $n + m$ channels, each containing 64 filters of size 4 and stride 2. We use a leaky-ReLU activation function with slope 0.2, we do layer

normalization and at each layer, we introduce a dropout with probability 0.2. An additional dense layer, with a linear activation function, is used to connect the single output node, that contains the critic value (see Fig. 4.4). To enforce the Lipschitz constraint on the critic's model, we add a gradient penalty term, as described in Section 3.2.1.

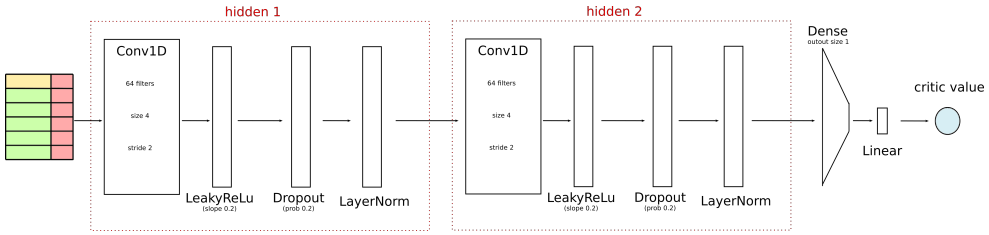


Figure 4.4: Architectural details of the critic network.

On the other hand, the generator network takes as input the noise and the initial settings and it embeds the inputs in a larger space with N_{ch} channels (512 in our experiments) through a dense layer. Four one-dimensional convolutional transpose layers are then inserted, containing respectively 128, 256, 256 and 128 filters of size 4 with stride 2. Here we do batch normalization and use a leaky-ReLU activation function with slope 0.2. Finally, a traditional convolutional layer is introduced to reduce the number of output channels to n (see Fig. 4.5). The Adam algorithm [78] is used to optimize the loss function of both the critic and the generator. The learning rate is set to 0.0001 and $\beta = \{0.5, 0.9\}$. The above settings are shared by all the case studies, the only exception is the more complex MAPK model for which a deeper cWCGAN-GP architecture is selected: a critic with five layers, each containing 256 filters of size 4 and stride 2, and a generator with five layers, containing respectively 128, 256, 512, 256 and 128 filters of size 4 with stride 2.

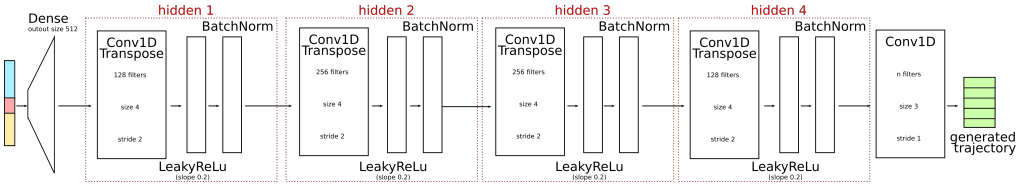


Figure 4.5: Architectural details of the generator network.

Model	SIR	e-SIRS	Switch	Osc.	MAPK
SSA (direct)	0.043 s	0.047 s	0.041 s	0.042 s	0.224 s
- CPU:	0.024 s	0.024 s	0.020 s	0.021 s	0.211 s
SSA (τ-leaping)	0.054 s	0.052 s	0.044 s	0.042 s	0.26 s
- CPU:	0.018 s	0.028 s	0.024 s	0.021 s	0.24 s
cWCGAN-GP	0.00175 s	0.00175 s	0.00175 s	0.00175 s	0.00175 s
- CPU:	0.0008 s	0.0008 s	0.0008 s	0.0008 s	0.0008 s
- GPU: $10^{-5}\times$	1.9 s	1.9 s	1.9 s	1.9 s	1.9 s

Table 4.1: Comparison of the average computational time required to simulate a single trajectory either via SSA (both direct or approximate methods) or via the cWCGAN-GP abstraction. 200 trajectories are needed to reduce the CPU (single processor) overhead, whereas, 2000 trajectories are required for the on GPU overhead.

Training details. Training times depend on the dimension of the dataset, on the size of mini-batches, on the number of species, and on the architecture of the cWCGAN-GP. The latter has been kept constant for all the case studies. Batches of 256 samples have been used and the number of epochs varies from 200 to 500 depending on the complexity of the model. Moreover, each training iteration of the generator corresponds to 5 iterations of the critic, to balance the power of the two players. The average time required for each training epoch is around one minute. Therefore, training the cWCGAN-GP model for 500 epochs takes around 8 hours leveraging the GPU.

4.2.1 Results

Computational gain. The time needed to generate abstract trajectories does not depend on the complexity of the original system. Moreover, as the cWCGAN-GP architecture is shared by all the case studies, the computational time required to generate abstract trajectories is the same for all the case studies. In particular, considering a noise variable of size 480, it takes around 1.75 milliseconds (ms) to simulate a single trajectory. However, when generating batches of at least 200 trajectories the overhead reduces and the time to generate a single trajectory stabilizes around 0.8 ms. The same does not hold for the SSA trajectories, whose computational costs depend on the complexity of the model and on the chosen reaction rates. In the case studies

considered the time required to simulate a single trajectory varies from 0.04 to 0.22 seconds, but it easily increases for more complex models or for smaller reaction rates, whereas the cost of abstract simulation stays constant. Details about the computational gain for each model are presented in Table 4.1. Computations are performed exclusively on a single CPU processor, to perform a fair comparison. However, the evaluation of cWCGAN-GP can be further sped up using GPUs, especially for large batches of trajectories, but this would have introduced a bias in their favour. It is important to stress how GPU parallelization is extremely straightforward in PyTorch and how the time to generate a single trajectory decrease to 1.9×10^{-5} seconds when generating a batch of at least $2K$ trajectories (see the last line of Table 4.1).

The training phase introduces a fixed overhead that affects the overall computational gain. For instance, the training phase of the MAPK model takes around 8 hours, which is equivalent to the time needed to generate 140K SSA trajectories. It follows that, together with the trajectories needed to generate the training set, the cost of the training procedure is paid off when we simulate at least 200K trajectories. In a typical biological multi-scale scenario in which we seek to simulate the evolution in time of a tissue containing hundreds of thousands or millions of cells, simulating also some of their internal pathways, the number of trajectories needed for the training phase becomes negligible making our approach extremely useful.

Measures of performance. Results are presented as follows. For each model, we present a small batch of trajectories, both real and abstract. From the plots of such trajectories, we can appreciate if the abstract trajectories are similar to real ones and if they capture the most important macroscopic behaviours. For visualization purposes, we also show the histograms of empirical distributions at time t_H for each species to quantify the behaviour over all the 2K trajectories present in the test set (see Fig. 4.6- 4.11). We choose time t_H, among all, to see if the abstract model is able to capture the steady-state behaviours of the original model.

Measuring error propagation. The reconstruction accuracy of the proposed abstraction procedure is performed on test sets consisting of 25 different initial settings. For each of these points, 2K SSA trajectories represent the empirical approximation of the true distribution over S^H. From each

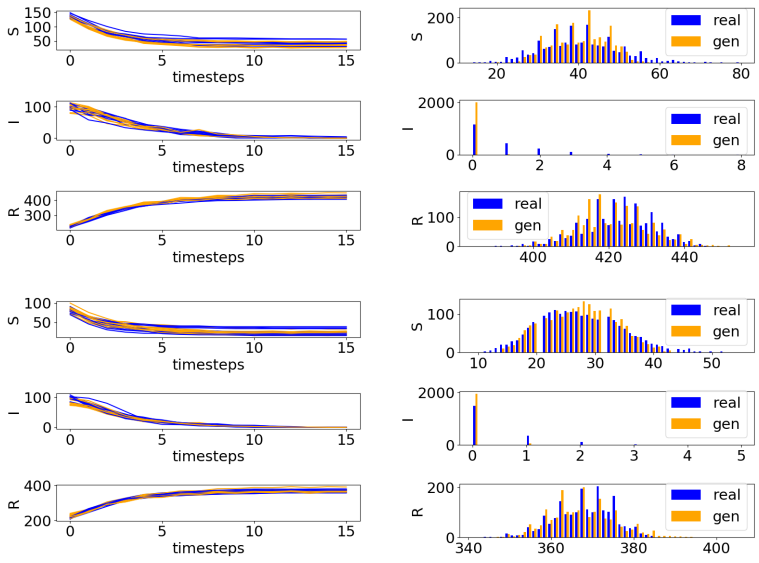


Figure 4.6: SIR model: **(left)** comparison of trajectories generated with a cWCGAN-GP (orange) and the trajectories generated with the SSA algorithm (blue); **(right)** comparison of the real and generated histogram at the last timestep. Performance on two randomly chosen test points represented by three trajectories: the top one (species S), the central one (species I) and the bottom one (species R).

of these initial settings we also simulate 2K abstract trajectories. Given a species $i \in \{1, \dots n\}$ and a time step $j \in \{1, \dots H\}$, we have the real one-dimensional distribution $\xi_{i,j}$ and the generated abstract distribution $\hat{\xi}_{i,j}$, where $\xi_{i,j}$ denotes the counts of species i at time t_j in a trajectory $\xi_{[1,H]}$. To quantify the reconstruction error, we compute five quantities: the Wasserstein distance among the two one-dimensional distributions, the absolute and relative difference among the two means and the absolute and relative difference among the two variances. By doing so, we are capable of seeing whether the error propagates in time and whether some species are harder to reconstruct than others. The error plots for the Wasserstein distance are shown in Figure 4.12. Plots of means and variances distances are provided in (Fig. 4.13-4.16). In addition, for two-dimensional models, i.e. eSIRS, Toggle Switch and MAPK, we show the landscapes of these five measures of the reconstruction error at three different time steps: step t_1, step $t_{H/2}$ and step

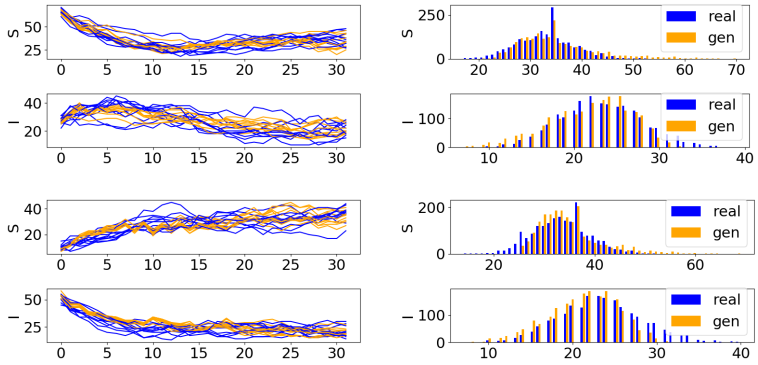


Figure 4.7: e-SIRS model: **(left)** comparison of trajectories generated with a cWCGAN-GP (orange) and the trajectories generated with the SSA algorithm (blue); **(right)** comparison of the real and generated histogram at the last timestep. Performance for two, randomly chosen, test points. Each point is represented by a pair of trajectories: the top one (species S) and the bottom one is for (species I).

t_H (Fig. A.1-A.7 in Appendix A). We observe that, in all the models, each species seems to contribute equally to the global error and, in general, the error stays constant w.r.t. time, i.e., it does not propagate. This was a major concern in previous methods, based on the abstraction of transition kernels. In fact, to simulate a trajectory of length H, the abstract kernel has to be applied iteratively H times. As a consequence, this results in propagation of the error introduced in the approximation of the transition kernel.

SIR. The results for the SIR model are presented in Fig. 4.6, which shows the performance on two, randomly chosen, test points. Each point is represented by three trajectories, the top one is for species S, the central one is for species I and the bottom one is for species R. The population size, given by $S + I + R$, is variable. The abstraction was trained on a dataset with fixed parameters, $\theta = \{3, 1\}$. Likewise, in the test set, only the initial states are allowed to vary. We observe that our abstraction method is able to capture the absorbing nature of SIR trajectories. It is indeed very important that once state $I = 0$ or state $R = N$ are reached, the system should not escape from it. Abstract trajectories satisfy such property without requiring the imposition of any additional constraint. The empirical distributions, real and generated, at time t_H are almost indistinguishable.

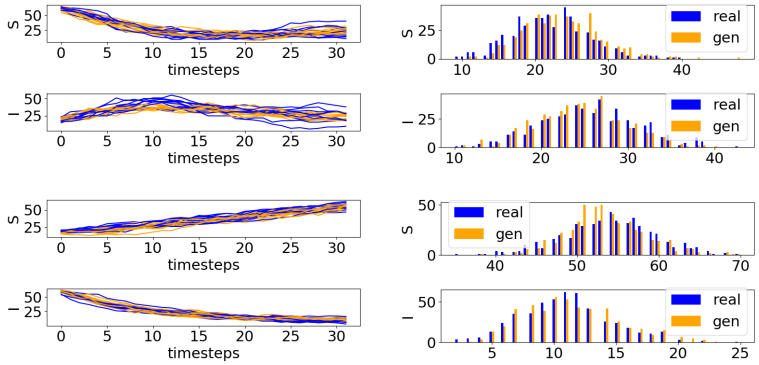


Figure 4.8: e-SIRS model with one varying parameter: **(left)** comparison of trajectories generated with a cWCGAN-GP (orange) and the trajectories generated with the SSA algorithm (blue); **(right)** comparison of the real and generated histogram at the last timestep.

e-SIRS. The e-SIRS model represents our baseline. We train two abstractions: in the first case, the model is trained on a dataset with fixed parameters, $\theta = \{2.36, 1.67, 0.9, 0.64\}$, and in the second case we let parameter θ_1 vary as well. Results are very accurate in both scenarios. In the fixed-parameters case, Fig. 4.7, the results are shown for two, randomly chosen, initial states. In the second case, Fig. 4.8, the results are shown on two, randomly chosen, pairs (s_0, θ_1). Each point is represented by a pair of trajectories, the top one is for species S and the bottom one is for species I. We performed further analysis on the generalization capabilities of the abstraction learned on the dataset with one varying parameter, using larger test sets and computing mean and standard deviation of the distribution of Wasserstein distances over such sets. The mean stays around 0.04 with a tight standard deviation ranging from 0.01 to 0.05, showing the little impact of the chosen conditional setting (see Fig. 4.17).

Toggle Switch. The results for the Toggle Switch model, on two, randomly chosen, test points, are shown in Fig. 4.9. The abstraction was trained on a dataset with fixed symmetric parameters ($kp_i = 1, kb_i = 1, ku_i = 1, kd_i = 0.01$ for $i = 1, 2$). Likewise, in the test set, only the initial states are allowed to vary. In this model, we tried to abstract only trajectories of the proteins $P1$ and $P2$, which are typically the observable species, ignoring the state of the genes. By doing so, we reduce the dimensionality of the

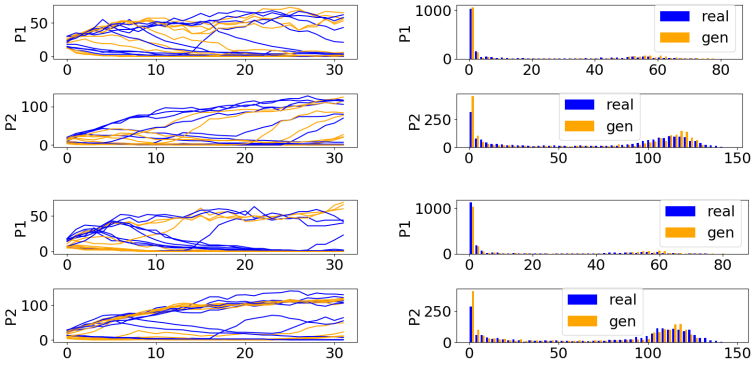


Figure 4.9: Toggle Switch model: **(left)** comparison of trajectories generated with a cWCGAN-GP (orange) and the trajectories generated with the SSA algorithm (blue); **(right)** comparison of the real and generated histogram at the last timestep. Performance for two randomly chosen test points represented by a pair of trajectories: the top one (species P1) and the bottom one (species P2).

problem but we also lose some information about the full state of the system. Nonetheless, the cWCGAN-GP abstraction is capable of capturing the bistable behaviour of such trajectories. In Fig. 4.9, each point is represented by two trajectories, the top one is for species $P1$, whereas the bottom one is for species $P2$.

Oscillator. The results for the Oscillator model, on two, randomly chosen, test points, are shown in Fig. 4.10. The abstraction was trained on a dataset with a fixed parameter ($\theta = 1$). Likewise, in the test set, only the initial states are allowed to vary. Each point is represented by three trajectories, the top one is for species A, the central one is for species B and the bottom one is for species C. The abstract trajectories well capture the oscillating behaviour of the system.

MAPK. The results for the MAPK model, on three, randomly chosen, test points, are shown in Fig. 4.11. The abstraction was trained on a dataset considering only a varying V_1 parameter and the dynamics of species $MAPK\_PP$. This case study represents a complex scenario in which the abstract distribution should capture the marginalization over the other seven unobserved variables. Moreover, the emergent behaviour of the only observed

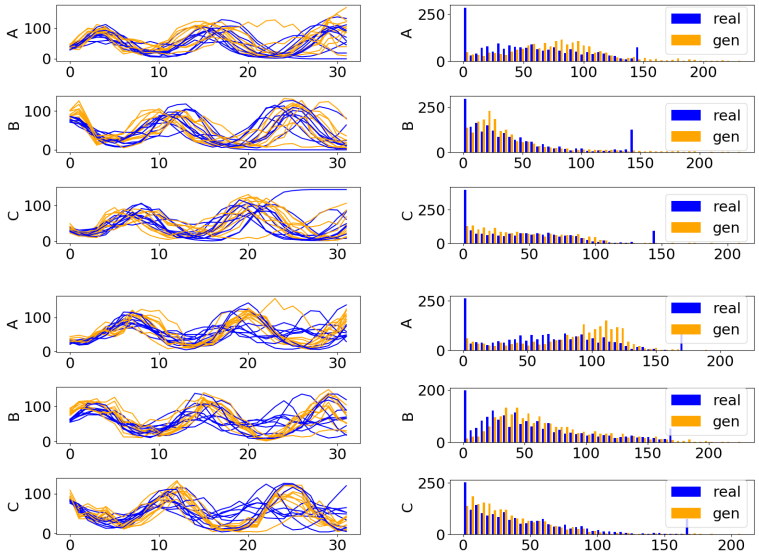


Figure 4.10: Oscillator model: **(left)** comparison of trajectories generated with a cWCGAN-GP (orange) and the trajectories generated with the SSA algorithm (blue);**(right)** comparison of the real and generated histogram at the last timestep. Performance on two randomly chosen test points is represented by three trajectories: the top one (species A), the central one (species B) and the bottom one (species C).

variable, $MAPK\_PP$, is strongly influenced by the input parameter V_1 and further amplified by the multi-scale nature of the cascade: for some values of V_1 the system oscillates, whereas for others, it stabilizes around an equilibrium. Results show that our abstraction technique is flexible enough to capture such sensitivity.

Satisfaction probability. We seek a formal way to quantify whether the abstract model captures and preserves the emergent macroscopic behaviours of the original system. In order to do so, we can resort to formal languages, such as Signal Temporal Logic (STL) (see Section 2.2). The first step is to express formally the property that we would like abstract trajectories to preserve. Then we can measure the satisfaction probability of such property for both real and abstract trajectories and check if it is similar on a large pool of initial settings. Examples are shown in Fig. 4.18. For the e-SIRS

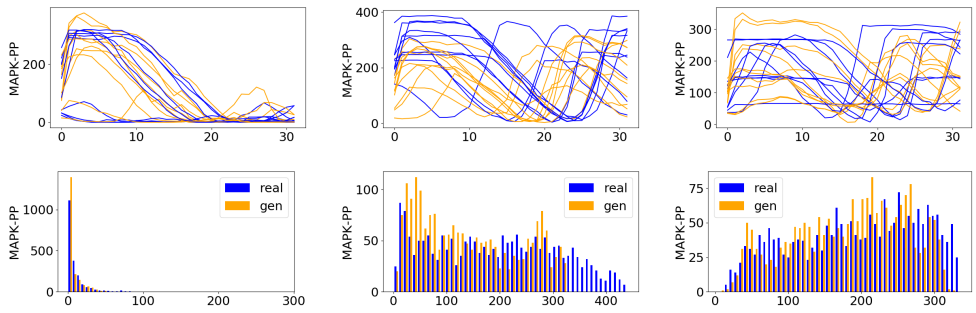


Figure 4.11: MAPK model: **(top)** comparison of trajectories generated with a cWCGAN-GP (orange) and the trajectories generated with the SSA algorithm (blue);**(bottom)** comparison of the real and generated histogram at the last timestep. Performance on three, randomly chosen, test points. Each point is represented by the output species MAPK_PP.

model we consider the property "eventually the number of infected remains below a threshold of 25 individual", i.e. $\Diamond_{[t_0,t_H]}\Box(I < 25)$. For abstract trajectories of the SIR model we check, for each test point, the percentage of valid trajectories, i.e. such that the state $I = 0$ is absorbing, i.e. $G_{[t_0,t_H]}((I = 0) \Rightarrow G(I = 0))$. Finally, for the Toggle Switch model, we check the property "eventually the level of protein P_2 stays above a threshold of 50", i.e. $\Diamond_{[t_0,t_H]}\Box(P_2 > 50)$. For each case study, we compare, for each test point, the percentage of SSA and abstract trajectories that satisfy the property. These comparisons produce a measurable qualitative estimate of how good the reconstruction is. In future work, we intend to use such qualitative measures as a query strategy for an active learning approach, so that the obtained abstract model is driven in the desired direction. In general, STL properties can be defined over a generic time interval $[0, T]$ so that we compare trajectories of arbitrary length. In such scenarios, the generator is iteratively applied to generate a joint trajectory of the desired length. More precisely, we start by generating a trajectory of length H over the temporal interval $[t_0^{(0)}, t_H^{(0)}]$. The state of the system at time $t_H^{(0)}$ is provided as new initial state to the generator that generates a new trajectory of length H over $[t_0^{(1)}, t_H^{(1)}]$, where $t_0^{(1)} := t_H^{(0)}$ and $t_H^{(1)} := t_0^{(1)} + H\Delta t$. This trajectory is concatenated to the previous one to form a trajectory over $[t_0^{(0)}, t_H^{(1)}]$. The procedure is iterated as long as $t_H^{(i)}$ is lower than T.

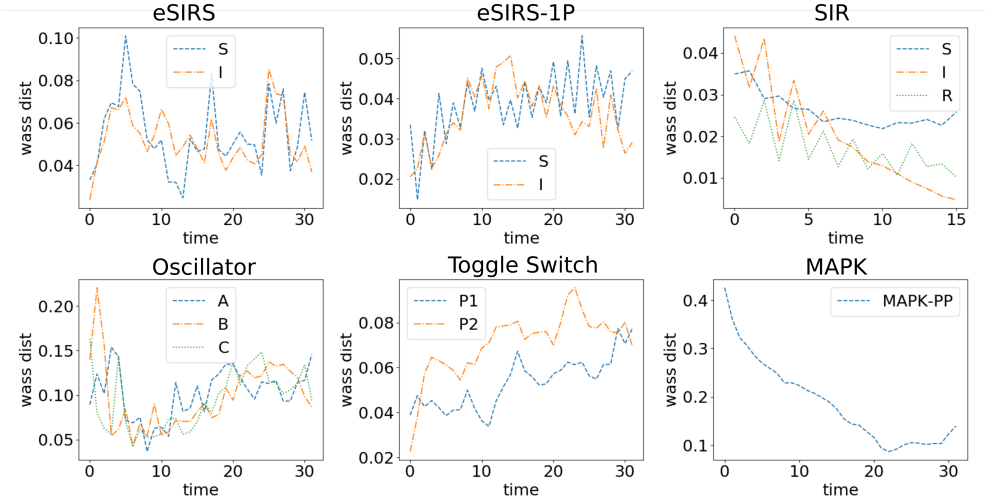


Figure 4.12: Plots of the error over time for each model and each species. Errors are computed using the Wasserstein distance over the entire test set. Generated trajectories have been kept scaled to the interval $[-1, 1]$ so that the scale of the system does not affect the scale of the error measure.

Statistical tests. For each case study, we use a statistical test to compare real and abstract distributions. In particular, we use a two-sample statistical test based on the energy distance among distributions [79]. For each initial setting and for each species we compute the distance statistics and the p-value among the empirical approximation of the SSA distribution and the empirical approximation of the abstract distribution over the trajectory space, i.e. a H-dimensional space. In Fig. 4.19 we report the mean and the standard deviation of p-values over the initial settings present in the test set. Clearly, the p-value decreases as the number of samples used to approximate the distributions increases. Fig. 4.19 shows how the p-values for each species varies according to the number of samples used. These results come with no surprise as the abstract model was trained having only 10 observations for each initial setting. It is interesting to observe how the Energy test is passed by a large percentage of points when the number of samples is around 10. To enhance the resilience of the abstract model to such statistical tests, we should increase the number of samples per point in the training set. This comes at the cost of reducing the number of initial settings, so that the resulting training set is not too large. In this regard, the active learning

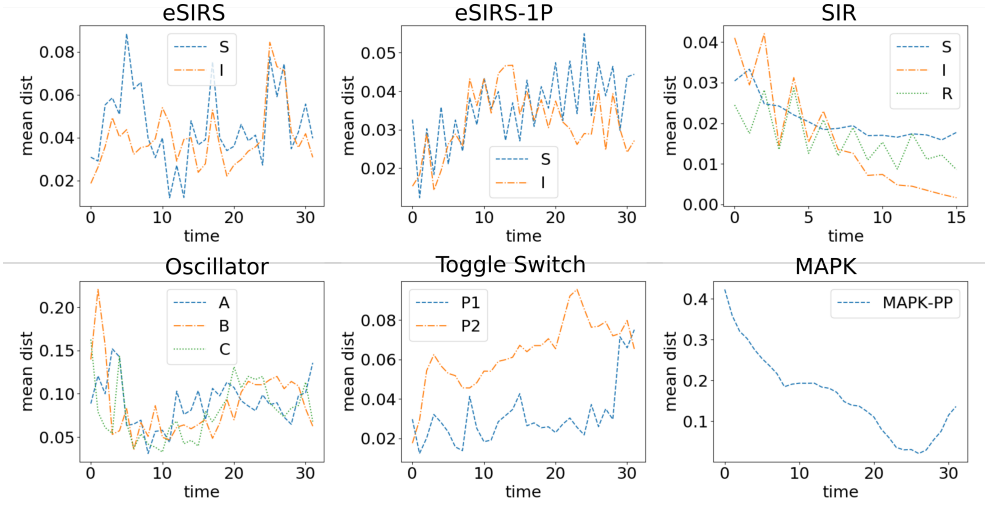


Figure 4.13: Plots of the average difference in the means over time for each model and each species. Errors are computed over the entire test set. Generated trajectories have been keep scaled to the interval $[-1, 1]$ so that the scale of the system does not affect the scale of the error measure.

technique, proposed in the paragraph above, could be extremely beneficial.

4.2.2 Discussion

Previous approaches for learning model abstraction, see Related work at the beginning of this chapter, focus on approximating the transition kernel, meaning the distribution of possible next states after a time Δt, rather than learning the distribution of full trajectories of length H. The main reason for such a choice is the limited scalability of the tool used for learning the abstraction. In fact, learning a distribution over $S^H \subseteq \mathbb{N}^{H \times n}$ with a Mixture Density Network is infeasible even for small H. Moreover, in learning to approximate the transition kernel one must split the SSA trajectories of the dataset in pairs of subsequent states. By doing so, a lot of information about the temporal correlation among states is lost. Having a tool strong and stable enough to learn distributions over S^H allows us to preserve this information and make abstraction possible even for systems with complex dynamics, which the abstraction of the transition kernel was failing to capture. For

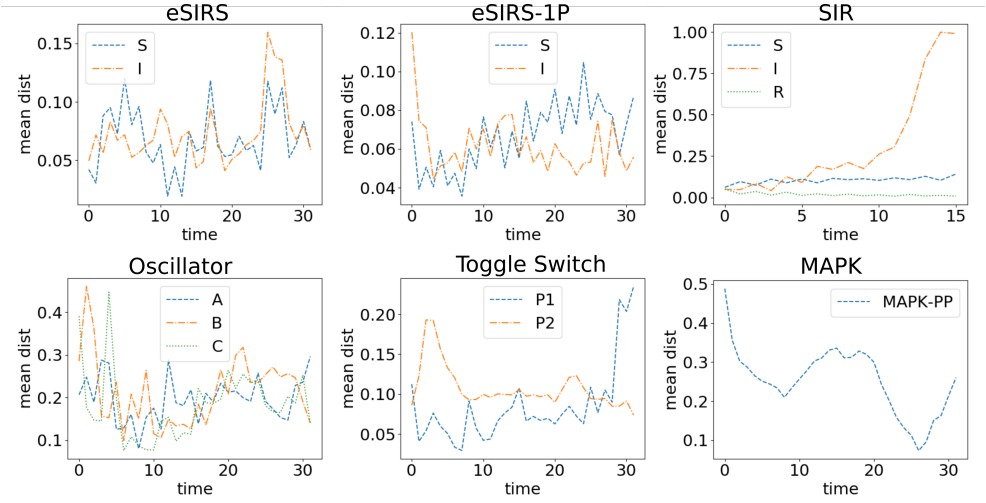


Figure 4.14: Plots of the average relative difference in the means over time for each model and each species. Errors are computed over the entire test set. Generated trajectories have been keep scaled back to $\mathbb{N}$.

instance, we are now able to abstract the transient behaviour of multi-stable or oscillating systems. When attempting to abstract the transition kernel, either via MDN or via c-GAN, for such complex systems, we did not succeed in learning meaningful solutions. A collateral advantage in generating full trajectories, rather than single subsequent states, is that it introduces an additional computational speed-up in the time required to generate a large pool of trajectories of length H. For instance, if a cWGAN is used to approximate the transition kernel, it takes around 31 seconds to simulate the 50K trajectories of length 32 present in the test set. Our trajectory-based method takes only 3.4 seconds to generate the same number of trajectories. Furthermore, our cWCGAN-GP was trained with relatively small datasets, which leaves room for further improvements where needed. An additional strength of our method is that one can train the abstract model only on species that are observable, reducing the complexity of the CRN model while preserving an accurate reconstruction for the species of interest. Once again, this was not possible with transition kernels and it may be extremely useful in real-world applications.

In general, the cWCGAN-GP approximation does not provide any sta-

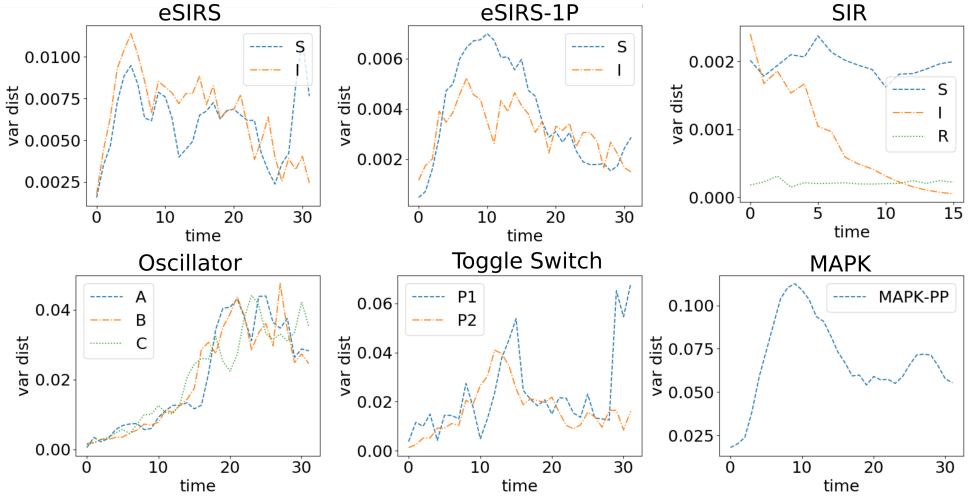


Figure 4.15: Plots of the average difference in the variances over time for each model and each species. Errors are computed over the entire test set. Generated trajectories have been keep scaled to the interval $[-1, 1]$ so that the scale of the system does not affect the scale of the error measure.

tistical guarantee about the reconstruction error. In addition, the set of observations used to learn the abstraction is rather small, typically 10 samples for each initial setting. Therefore, it is not surprising that the real and the abstract distributions are not indistinguishable from a statistical point of view. However, the abstract model is indeed capable of capturing, from the little amount of information provided, the emergent features of the behaviour of the original system, such as multimodality or oscillations. In this regard, formal languages can be used to formalize and check such qualitative properties. In particular, we can check whether the satisfaction probability (of non-rare events) is similar in real and abstract trajectories. Furthermore, such quantification of qualitative properties can be used to measure how good the reconstruction is. As future work, we intend to use it as a query strategy for an active learning approach, so that the obtained abstract model is driven in the desired direction. Moreover, the abstract model is only able to generate trajectories that are similar to those observed in the dataset. Therefore, it may be difficult to uncover edge cases of the model, meaning small regions of the parameter space where trajectories behave very differently from the rest of the parameter space, e.g. regions of bifurcation. In this regard, an

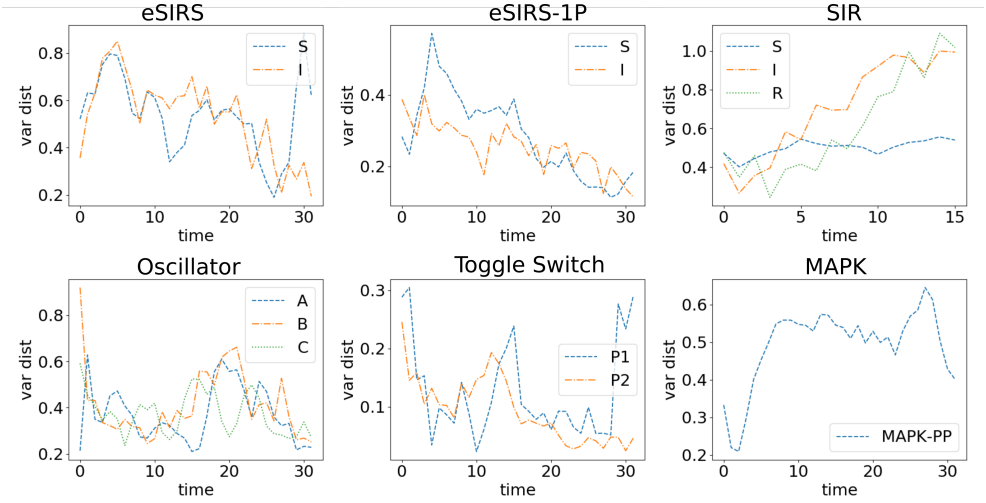


Figure 4.16: Plots of the average relative difference in the variances over time for each model and each species. Errors are computed over the entire test set. Generated trajectories have been keep scaled back to $\mathbb{N}$.

active learning approach, based on a qualitative error quantification, could drive the abstract model to detect edge cases and improve its accuracy by enriching the dataset with more observations from that specific region of the parameter space.

4.3 Conclusions

We presented a technique to abstract the simulation process of stochastic trajectories for various CRNs. The WGAN-based abstraction improves considerably the computational efficiency, which is no more related to the complexity of the underlying CRN. This would be extremely helpful in all those applications in which a large number of simulations is required, i.e., applications whose solution is unfeasible via SSA simulation. It would enable the simulation of multi-scale models for very large populations, it would speed up statistical model checking [80] and it can be used in particular cases of parameter estimation, for example when only a few parameters have to be estimated multiple times. In conclusion, the c-WCGAN-based solution to

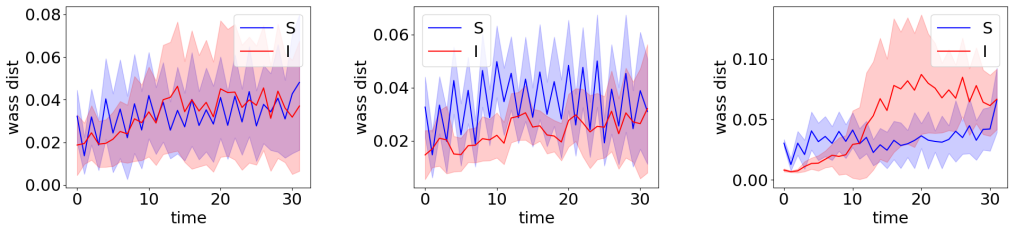


Figure 4.17: Analysis of the generalization capabilities of the abstract model on various test sets: 100 different pairs (s_0, θ) **(left)**, fixed-parameter and 100 different initial states **(middle)** and a fixed initial state with 100 different parameters **(right)**. For each test set we compute the mean and the standard deviation of the distribution of Wasserstein distances over such sets.

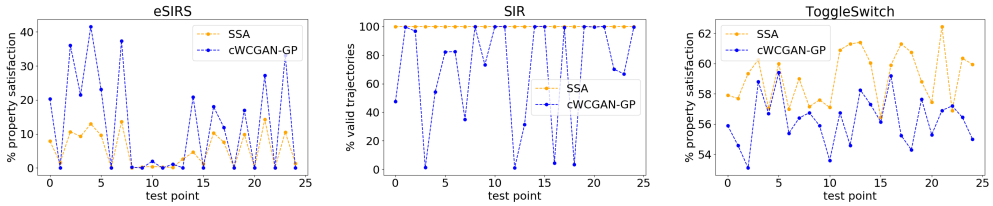


Figure 4.18: **(eSIRS)** Given the property "eventually the number of infected stays below a threshold of 25 individual", we check for each test point (x-axis) the percentage of SSA (orange) and abstract (blue) trajectories that satisfy such property. **(SIR)** For abstract trajectories of the SIR model we check, for each test point, the percentage of valid trajectories, i.e. such that the state $I = 0$ is absorbing. **(Toggle Switch)** Given the property "eventually the level of protein P2 stays above a threshold of 50", we check for each test point (x-axis) the percentage of SSA (orange) and abstract (blue) trajectories that satisfy such property.

model abstraction performs well in scenarios that are very complex and challenging, requiring relatively little data and very little fine-tuning.

As future work, we plan to study how our abstraction technique works on real data. In this regard, we do not aim at capturing the underlying dynamical system, but we would rather be able to reproduce the trajectories observed in real applications. A great strength of our method, compared to the state of the art solutions, is that it is able to generate trajectories only for a subset of the species present in the system domain, ignoring the information that is not observable, even during the training phase. Another interesting extension is to adapt our technique to sample bridging trajecto-

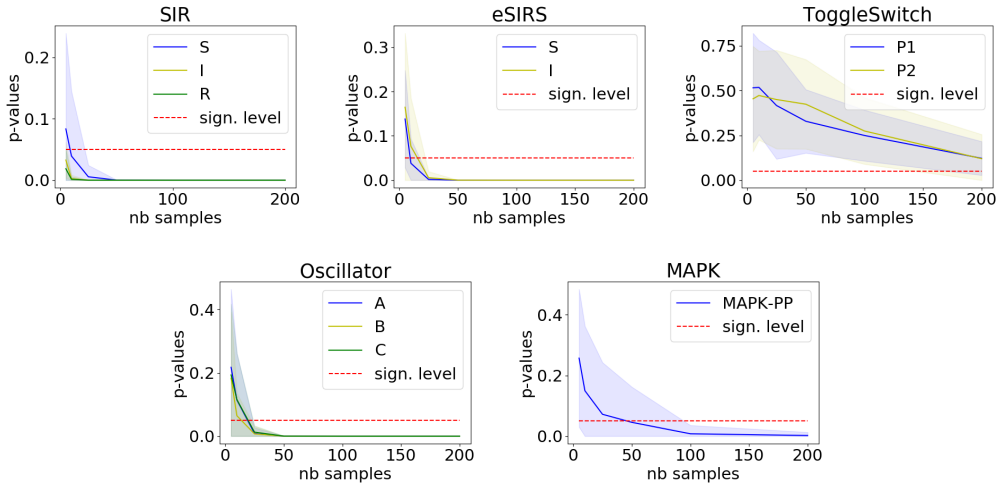


Figure 4.19: Average over the initial setting of the p-values (with confidence interval) for each species computing by the two-sample statistical test w.r.t. the number of samples present in the empirical distributions.

ries, where both the initial and the terminal states are fixed. Typically, the simulation of such trajectories requires expensive Monte Carlo simulations, which makes clear the benefits of resorting to model abstraction. To conclude, an interesting extension to this work would be to incorporate STL formulas as a constraint during the GAN training phase, so that the GAN can only generate trajectories that also satisfy the imposed STL formulas.

Chapter 5

Adversarial-Learning of Safe and Robust Controllers

Controlling Cyber-Physical Systems (CPS) is a well-established problem in classic control theory [14]. State of the art solutions apply to all those models in which complete knowledge of the system is available, i.e., scenarios in which the environment is supposed to follow deterministic rules. For such models, a high level of predictability, along with good robustness, is achieved. However, as soon as unpredictable scenarios come into play, traditional controllers are challenged and could fail. Ongoing research is trying to guarantee more flexibility and resilience in this context by using Deep Learning [81] and, in particular, Reinforcement Learning for robust control. State of the art solutions perform reasonably well, but they still present evident limits in case of unexpected situations. The so-called *open-world scenarios* are difficult to model and to control, due to the significant amount of stochastic variables that are needed in their modelling and the variety of uncertain scenarios that they present. Therefore, while trying to ensure safety and robustness, we need to be cautious about not trading them with model effectiveness.

In this work [2], we investigate the autonomous learning of safe and robust controllers in open-world scenarios. Our approach consists of training two neural networks, inspired by Generative Adversarial Networks (GAN) [46], that have opposite goals: the *attacker* network tries to generate troubling scenarios for the *defender*, which in turn tries to learn how to face them

without violating some safety constraints. The outcome of this training procedure is twofold: on the one hand, we get a robust controller, whereas, on the other hand, we get a generator of adverse tests.[1]

The learned controller is a black-box device capable of dealing with adverse or unobserved scenarios, though without any worst-case guarantee. In this regard, one could complement our method with a shield-based approach, as proposed e.g. by [82].

5.1 Problem Statement

The safety of a system can be formalised as the satisfaction of a set of safety requirements. In this application, we express safety requirements only by means of time-bounded STL formulas (see Section 2.2) over fixed-length trajectories. In particular, we rely on STL *quantitative semantics*, which is capable of capturing, for each trajectory, the level of satisfaction of the desired property. Such measure is often referred to as *robustness* and is exploited in this work as the objective function of an optimization problem.

We model the interaction of an agent with an adversarial environment as a *zero-sum game*, similarly to the strategy behind GANs. The concept of zero-sum game is borrowed from game theory and denotes those situations in which one player's gain is equivalent to another's loss. In such situations, the best strategy for each player is to minimize its loss, while assuming that the opponent is playing at its best. This concept is known in the literature as *min-max strategy*. In practice, we use GAN architectural and theoretical design to reach two main objectives: a controller, that safely acts under adverse conditions, and an attacker, which gains insights about troubling scenarios for the opponent.

Related work. The proposed concept is closely related to that of *Robust Adversarial Reinforcement Learning (RARL)* [83], a Reinforcement Learning framework, involving an agent and a destabilizing opponent, that is robust to adverse environmental disturbances. In this case, the term "robustness" does

[1] Code is available at: https://github.com/ginevracoal/adversarialGAN/

not refer to Signal Temporal Logic, but instead, to the cumulative reward computed on the learned policy with respect to the varying test conditions. Other recent RL techniques involving STL constraints include [84, 85, 86]

5.2 Methodology

Agent-Environment Model. CPS is modeled as a hybrid model $\mathcal{M}$ (see Section 2.1) where the continuous part is represented by differential equations that describe the behaviour of the plant and the discrete part, instead, identifying the possible states of the controller. We decompose our model in two interacting parts: the *agent* $\mathtt{a}$ and the *environment* $\mathtt{e}$. Therefore, the state space S of the system coincides with the HA continuous domain Var, whereas the set of possible controller's actions A coincides with the HA discrete domain Loc. Mathematically, $S := Var$ and $A := Loc = A_{\mathtt{a}} \times A_{\mathtt{e}}$. Both the agent and the environment are able to observe at least part of the whole state space S, i.e. they are aware of some observable states $Y \subset S$. By distinguishing between the observable states of the agent $Y_{\mathtt{a}} \subseteq Y$ and of the environment $Y_{\mathtt{e}} \subseteq Y$, we are able to force uneven levels of knowledge between them.

Notice that the observable states, for both the agent and the environment, could also include environmental variables involved in the evolution of the system.

Let $A_{\mathtt{a}}$ and $A_{\mathtt{e}}$ be the spaces of all possible actions for the two components. We discretize the evolution of the system as a discrete-time system with step Δt, which evolves according to a function $\psi : S \times A_{\mathtt{a}} \times A_{\mathtt{e}} \times \mathbb{R} \longrightarrow S$. By taking control actions at fixed time intervals of length Δt, we obtain a discrete evolution of the form $s_{i+1} = s_i + \psi(s_i, a^i_{\mathtt{a}}, a^i_{\mathtt{e}}, t_i)$, where $t_i := t_0 + i \cdot \Delta t$, $a^i := a(t_i)$ and $s_i := s(t_i)$. Therefore, we are able to simulate the entire evolution of the system over a time horizon H via ψ and to obtain a discrete trajectory $\xi = s_0 \dots s_{H-1}$.

A safety requirement is expressed as an STL formula ϕ; we call h its temporal depth.[2]

[2]The temporal depth of a formula is defined recursively as the sum of maximum bounds of nested temporal operators.

The robustness of a trajectory quantifies the level of satisfaction w.r.t. ϕ and it determines how safe the system is in that configuration. Robustness is denoted as a function $\mathcal{R}_\phi : S^h \to \mathbb{R}$, measuring the maximum perturbation that can be applied to a given trajectory of length h without changing its truth value w.r.t. ϕ. It is straightforward to use this measure as the objective function in our min-max game.

Multi-objective formulation. In the case of multiple safety requirements, we define a different STL formula for each of these requirements and likewise, we compute the respective robustness values. As a matter of fact, the order of magnitude of each robustness value depends on the order of magnitude of the CPS variables involved. Therefore, a single STL formula that combines the safety requirements all together may result in an unbalanced objective function, skewed towards some components that are not necessarily the most safety-critical. To overcome this problem we normalize the variables involved and we define the objective function as a weighted sum of the robustness values $\mathcal{R}_\phi$ resulting from each requirement ϕ_i. Let $\Phi = \{\phi_1, \ldots, \phi_m\}$ denote a set of m safety requirements, the combined robustness score $\mathcal{R}_\Phi$ is defined as:

$$\mathcal{R}_\Phi(\cdot) := \frac{1}{\alpha} \sum_{i=1}^{m} \alpha_i \cdot \mathcal{R}_{\phi_i}(\cdot), \tag{5.1}$$

where $\alpha = \sum_{i=1}^{m} \alpha_i$. By tuning the weights $\alpha_1, \ldots, \alpha_m$ we are able to explicitly influence the importance of each factor in the training objective. The choice of these hyper-parameters will be case-specific. This formulation is typically used in multi-objective optimization scenarios [87]. In (5.1), we assume w.l.o.g. for notational simplicity that each formula ϕ_i has the same time depth h. However, our framework can be straightforwardly extended to more general STL properties with different time depths.

Attacker-Defender architecture. The proposed framework builds on GAN architectural design, in which two NNs compete in a min-max game to reach opposite goals. One network, denoted by $\mathcal{A}$, represents the *attacker*, while the other, denoted by $\mathcal{D}$, represents the *defender*. The aim of the former is to generate environment configurations in which the defender is not able to act safely, whereas, the latter tries to keep the CPS as safe as possible. In practice, the defender $\mathcal{D}$ can be interpreted as a controller for the agent.

Optimization strategy. Given a time horizon H, an initial state s_0, meaning the state at time t_0, and two sequences of actions $a_{\mathtt{a}} = (a_{\mathtt{a}}^0, \ldots, a_{\mathtt{a}}^{H-1})$ and $a_{\mathtt{e}} = (a_{\mathtt{e}}^0, \ldots, a_{\mathtt{e}}^{H-1})$, one for the agent and one for the environment, it follows that the evolution of a trajectory ξ is obtained by evaluating ψ at each time steps $t_i \in \{t_0, \ldots, t_{H-1}\}$.

The min-max problem can be expressed as finding the sequences $a_{\mathtt{e}}$ and $a_{\mathtt{a}}$ that solve

$$\min_{a_{\mathtt{e}}} \max_{a_{\mathtt{a}}} \left[\mathcal{J}(s_0, a_{\mathtt{a}}, a_{\mathtt{e}})\right]. \tag{5.2}$$

The objective function $\mathcal{J}$ is the cumulative sum of the robustness scores computed at each timestep during the generation of the whole trajectory ξ on the sub-trajectory $\xi[t, t+h-1]$ available at timestep t, i.e.

$$\mathcal{J}(s_0, a_{\mathtt{a}}, a_{\mathtt{e}}) := \sum_{t=0}^{H-h} \mathcal{R}_\Phi(\xi[t, t+h-1]).$$

In our setting, the sequences of actions are iteratively determined by the two adversarial networks. In particular, let $\theta_{\mathcal{A}}$ be the weights of the attacker's network $\mathcal{A}$ and $\theta_{\mathcal{D}}$ the weights of the defender's network $\mathcal{D}$. At each timestep, the attacking network,

$$\begin{aligned} \mathcal{A} : \Theta_{\mathcal{A}} \times Y_{\mathtt{e}} \times K &\longrightarrow A_{\mathtt{e}} \\ (\theta_{\mathcal{A}}, y_{\mathtt{e}}, k) &\longmapsto a_{\mathtt{e}}, \end{aligned}$$

receives the current observable state of the environment $y_{\mathtt{e}}$, the noise coefficient k and outputs the coefficients $a_{\mathtt{e}}$, defining the adversarial environmental components.

$$\begin{aligned} \mathcal{D} : \Theta_{\mathcal{D}} \times Y_{\mathtt{a}} &\longrightarrow A_{\mathtt{a}} \\ (\theta_{\mathcal{D}}, y_{\mathtt{a}}) &\longmapsto a_{\mathtt{a}}, \end{aligned}$$

reads the current observable state of the agent $y_{\mathtt{a}}$ and produces the control action $a_{\mathtt{a}}$.

To ease the notation, we introduce a function

$$\begin{aligned} \psi_t : S \times \Theta_{\mathcal{D}} \times \Theta_{\mathcal{A}} &\longrightarrow S^t \\ (s_0, \theta_{\mathcal{D}}, \theta_{\mathcal{A}}) &\longmapsto s_0 \ldots s_{t-1}, \end{aligned}$$

which iteratively applies ψ for each pair of actions

$$a_{\mathtt{a}}^j = \mathcal{D}(\theta_{\mathcal{D}}, y_{\mathtt{a}}^j)$$
$$a_{\mathtt{e}}^j = \mathcal{A}(\theta_{\mathcal{A}}, y_{\mathtt{e}}^j, k),$$

where $j \in \{0, \dots, t-1\}$ indexes the simulation interval.

The formalism introduced by the two policy networks transforms the problem of finding the best sequences of actions, $a_{\mathtt{a}}$ and $a_{\mathtt{e}}$, to that of finding the best networks' parameters, $\theta_{\mathcal{D}}$ and $\theta_{\mathcal{A}}$. This leads to the objective

$$J(s_0, \theta_{\mathcal{A}}, \theta_{\mathcal{D}}) = \sum_{t=0}^{H-h} \mathcal{R}_{\Phi}[\psi_h(s_t, \theta_{\mathcal{D}}, \theta_{\mathcal{A}})] \tag{5.3}$$

and the minmax game $\min_{\theta_{\mathcal{A}}} \max_{\theta_{\mathcal{D}}} J(s_0, \theta_{\mathcal{A}}, \theta_{\mathcal{D}})$ is now directly expressed in terms of the training parameters.

In such a setting, the defender aims at generating safe actions by tuning its weights in favour of a maximization of the objective function, i.e., a maximization of the cumulative robustness score. The attacker, on the other hand, aims at generating troubling scenarios for the opponent by minimizing the objective function, i.e., minimizing the cumulative robustness score.

The horizon H represents the number of simulation steps performed while keeping the parameters $\theta_{\mathcal{A}}$ and $\theta_{\mathcal{D}}$ fixed. In principle, we could choose $H = h$, without the need of having a summation in (5.3), possibly taking a larger time-bound h in the formulae of Φ. However, this would make the objective excessively rigid. In fact, if a controller would work well everywhere but on a small sub-region of the trajectory, such an objective would return a penalization also for the regions where the controller performs well. Similarly, in (5.3), one could choose alternative operators, other than the sum, for instance a max or a softmax operator. This would result in an objective function that ignores the global behaviour of the controller in favor of the local sub-interval where it performs best. This effects are avoided by considering an objective as (5.3), where $h \ll H$ and the robustness values over each sub-trajectory are summed together.

Algorithm 10 shows the pseudocode for the training phase.

Algorithm 10 Training phase.

procedure TRAIN($\mathcal{M}$, H_{train}, iters$_{\mathcal{A}}$, iters$_{\mathcal{D}}$)
  $s_0 \leftarrow$ SampleRandomState()

  for iters$_{\mathcal{A}}$ **do** ▷ Train $\mathcal{A}$
    $a_{\mathtt{e}}, a_{\mathtt{a}}$=[ ]
    for $i \leftarrow 0 \dots H_{\text{train}} - 1$ **do**
      $k \leftarrow \mathcal{N}(0, 1)$
      $a^i_{\mathtt{e}} \leftarrow \mathcal{A}(\theta_{\mathcal{A}}, y^i_{\mathtt{e}}, k)$
      $a^i_{\mathtt{a}} \leftarrow \mathcal{D}(\theta_{\mathcal{D}}, y^i_{\mathtt{a}})$
      $a_{\mathtt{e}}[i] \leftarrow a^i_{\mathtt{e}}$
      $a_{\mathtt{a}}[i] \leftarrow a^i_{\mathtt{a}}$

    $\theta_{\mathcal{A}} \leftarrow$ Adam $(\mathcal{J}(s_0, a_{\mathtt{a}}, a_{\mathtt{e}}), \lambda_{\mathcal{A}})$

  for iters$_{\mathcal{D}}$ **do** ▷ Train $\mathcal{D}$
    $a_{\mathtt{e}}, action_{\mathtt{a}}$=[ ]
    for $i \leftarrow 0 \dots H_{\text{train}} - 1$ **do**
      $k \leftarrow \mathcal{N}(0, 1)$
      $a^i_{\mathtt{e}} \leftarrow \mathcal{A}(\theta_{\mathcal{A}}, y^i_{\mathtt{e}}, k)$
      $a^i_{\mathtt{a}} \leftarrow \mathcal{D}(\theta_{\mathcal{D}}, y^i_{\mathtt{a}})$
      $a_{\mathtt{e}}[i] \leftarrow a^i_{\mathtt{e}}$
      $a_{\mathtt{a}}[i] \leftarrow a^i_{\mathtt{a}}$

    $\theta_{\mathcal{D}} \leftarrow \text{Adam}(\mathcal{J}(s_0, a_{\mathtt{a}}, a_{\mathtt{e}}), \lambda_{\mathcal{D}})$

Testing phase. In the testing phase, we generate a trajectory of length H and we check separately each safety requirement on such trajectory, in particular, we check that the requirement ϕ_i is globally satisfied, i.e., the condition $\square_{[0,H]}\phi_i$. We stress that, at test time, the temporal depth h of the STL formula is set to H. Therefore, for each property, a positive value of robustness at test time means that the requirement is met during the whole evolution of the system w.r.t. the time horizon H. Algorithm 11 shows the pseudocode for the testing phase.

Algorithm 11 Test phase.

procedure TEST($\mathcal{M}$, H_{test})
  $s_0 \leftarrow \text{GetState}(\mathcal{M})$
  $\xi := [s_0]$
  for $i \leftarrow 0 \dots H_{\text{test}} - 1$ **do**
    $\mathbf{z} \leftarrow \mathcal{N}(0, 1)$
    $a^i_{\mathtt{e}} \leftarrow \mathcal{A}(\theta_{\mathcal{A}}, y^i_{\mathtt{e}}, k)$
    $a^i_{\mathtt{a}} \leftarrow \mathcal{D}(\theta_{\mathcal{D}}, y^i_{\mathtt{a}})$

    $\xi[i+1] \leftarrow \psi(s_i, a^i_{\mathtt{a}}, a^i_{\mathtt{e}}, t_i)$
  $\rho = \mathcal{R}_\Phi(\xi)$

5.3 Experiments

We test the proposed architecture on two different case studies: a cart-pole balancing problem and a platooning problem. Both systems are embedded into environments with a stochastic evolution.

The Attacker-Defender networks are trained against each other for a given number of epochs, as discussed below. Once the training is over, the performances of the trained Defender are tested in two different ways. On one hand, we generate a test set containing 1k different initial configurations, uniformly sampled from pre-defined compact sets. From each of these points we generate a trajectory evolving according to the trained Attacker and Defender networks, then check each requirement separately on each trajectory as specified in the previous section.

The second approach to evaluate the performances of the trained Attacker-Defender network is to consider an environment that evolves unaware of the state of the system. In both cases, we compare the performance of Defender with that of a classical controller.

Hyperparameter tuning is necessary for the GAN architecture to achieve the desired performance in terms of safety. The choice of the architecture (number and size of the layers), the training hyperparameters, the time horizon H and weights for the cumulative robustness have a strong impact on the final results. In particular, the number of training iterations performed by

the Attacker network and by the Defender network has a strong impact on the performances of the trained networks. By tuning this number we are able to ensure that the Attacker is strong enough to generate challenging configurations of the environment, without preventing the Defender network from learning a secure controller. We performed manual tuning on a combination of hyperparameters and architectures, however, one could also automate this process by maximizing the percentage of safe trajectories produced by the learned controller.

5.3.1 Cart-Pole balancing

The *Cart-Pole* system [88] (also known as Inverted Pendulum) consists of a cart and a vertical pole attached to the cart by an un-actuated joint. The cart is allowed to move along the horizontal axis, while the pole moves in the vertical plane parallel to the track. The goal is to keep the pole balanced by learning an optimal policy for the cart, which influences the swinging movement of the pole. This problem is a well-known benchmark in both classical control [89, 90] and reinforcement learning [91, 92] applications.

Moving target and track-cart friction. In order to test the full potential of our framework, we consider a complex stochastic environment made of a moving target for the cart to follow and a friction coefficient between the cart and its track. These two components, governed by the Attacker network, represent the two potentially adversarial components of the system.

Model. The observable states $y_{\mathtt{a}}$ for the Defender and $y_{\mathtt{e}}$ for the Attacker are: cart position x, cart velocity $\dot{x}$, pole angle θ, pole angular velocity $\dot{\theta}$ and target position $\hat{x}$. Given $y_{\mathtt{e}}$, the Attacker's policy network $\mathcal{A}$ generates the adverse coefficients, i.e., friction μ and target position $\hat{x}$, both constrained to assume realistic values w.r.t. the physical settings of our application. The Defender reads the current state $y_{\mathtt{a}}$ and generates the desired control action f for the cart, which is meant to keep the pole balanced during the whole trajectory. The dynamic of the system is described by the following

equations [93]:

$$\begin{cases} \ddot{x} &= \frac{f-\mu\dot{x}+m_p\, l\, \dot{\theta}^2\, \sin\theta - m_p\, g\, \cos\theta \sin\theta}{m_c+m_p\, \sin\theta^2} \\ \ddot{\theta} &= \frac{g\, \sin\theta - \cos\theta \ddot{x}}{l} \end{cases}, \tag{5.4}$$

where m_p is the mass of the pole, m_c is the mass of the cart, l is half the pole length and g is the gravitational constant.

We impose the STL requirement $\phi_d = \square_{[0,h]}(d \leq d_{\max} \wedge d \geq d_{\min})$ on the distance $d = \|x - \hat{x}\|$ between the cart and its target, where $d_{\min}$ and $d_{\max}$ are the minimum and maximum distances allowed. Similarly, we set the requirement $\phi_\theta = \square_{[0,h]}(\theta \leq \theta_{\max} \wedge \theta \geq \theta_{\min})$ on the angle.

The objective function is the combination of two cumulative robustness components, one on the distance, $\mathcal{R}_{\phi_d}$, and one on the angle, $\mathcal{R}_{\phi_\theta}$, whose contributions are weighted by a coefficient $\alpha \in [0, 1]$:

$$J(s_0, \theta_{\mathcal{A}}, \theta_{\mathcal{D}}) = \alpha \sum_{t=0}^{H-h} \mathcal{R}_{\phi_d}[\xi_t] + (1-\alpha) \sum_{t=0}^{H-h} \mathcal{R}_{\phi_\theta}[\xi_t],$$

where $\xi_t = \psi_h(s_t, \theta_{\mathcal{D}}, \theta_{\mathcal{A}})$.

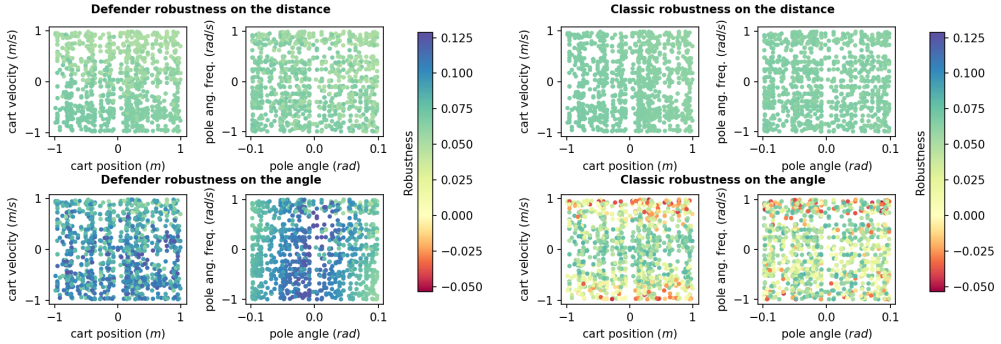
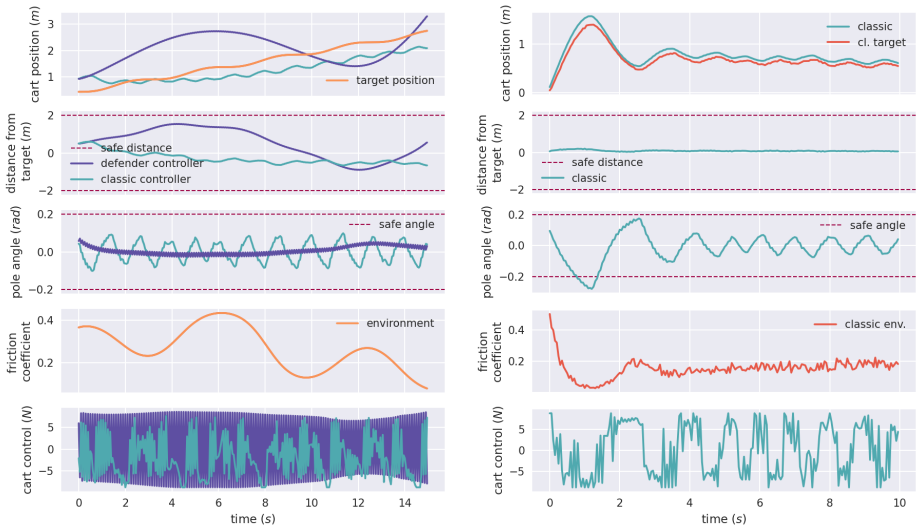


Figure 5.1: Cartpole balancing problem in which the friction and the moving target are generated by the Attacker network. This plot shows the robustness values achieved by the Defender and the classic controller. Robustness differences are computed separately for the two requirements imposed on the distance and on the angle. Trajectories start from 1k random initial states and evolve on a time horizon $H = 200$ with a step of $\Delta t = 0.05\, s$, resulting in $\Delta t \cdot H = 10\ s$ long simulations.



(a) (b) moving target and cart friction

Figure 5.2: **(a)** Sample of the system evolution for the cartpole balancing problem, using a classical controller and the Defender network against the same environment, represented by fixed trajectories of target positions and the cart-track friction coefficients. The *defender* cart is controlled by the Defender network, while the *classic* one is managed a classical controller. The common initial state is: cart position $x = 0.9055\ m$, cart velocity $\dot{x} = 0.0348\ m/s$, pole angle $\theta = 0.0808\ rad$, pole angular velocity $\dot{\theta} = 0.0399\ rad/s$. The trajectory evolves on a time horizon $H = 300$ with time step size $\Delta t = 0.05\ s$ ($\Delta t \cdot H = 15\ s$ long simulation).
(b) Evolution of the system for cart-pole balancing problem with moving target, using a classical controller for the cart against the Attacker network, which generates track-cart friction coefficient and target. Initial configuration: cart position $x = 0.7835\ m$, cart velocity $\dot{x} = 0.9550\ m/s$, pole angle $\theta = 0.0715\ rad$, pole angular velocity $\dot{\theta} = 0.8155\ rad/s$.

Results. The experimental settings are presented in Table 5.1. Fig. 5.1 shows that the trajectories evolving according to the Defender network all achieved positive robustness, despite the adversarial reactive environment governed by the Attacker. Moreover, Fig. 5.2-(a) shows the evolution of the system in a fixed environmental setting, for two different controllers: the Defender network and a classical robust controller based on a Sliding Mode Control (SMC) architecture [94]. The Defender is able to maintain safety during the whole trajectory on both θ and d, adequately counteracting the cart-track friction, while the classical controller has worse overall performance and in a few cases failed to guarantee safety within the specified initialization grid. In this setting, the Defender exhibits chattering in its control signal. In fast-evolving systems, such as cart-pole, this phenomenon could be avoided by including a regularization term on the control signal during the training phase. It should be noticed that the relatively low sampling frequency of $20Hz$ could also affect the SMC based controller.

5.3.2 Car platooning

A *platoon* [95] is a group of vehicles travelling together very closely and safely. This problem is usually faced with techniques that coordinate the actions of the entire pool of vehicles as a single entity [96]. This approach, though, requires specific hardware and a distributed system of coordination that might be difficult to realise in complex scenarios. Our method, instead, builds a robust controller for individual decision-making, hence it fits into the autonomous driving field. In this setting, we assume that all vehicles are equipped with a hardware component called *LIDAR scanner*, which is able to measure the distance between two cars by using a laser beam. In the basic scenario, only involving two cars, the car in front is called the *leader* and acts according to the Attacker network, while the second one is the *follower*, whose behaviour is determined by the Defender network. This setting trivially extends to the case of n cars, where the first car is the leader and the other ones all act as followers, controlled by the same Defender.

Platooning with Power Consumption. An additional problem that can be addressed in the platooning problem is the optimization of the energy consumption of the follower car, similarly to what has been proposed by [97],

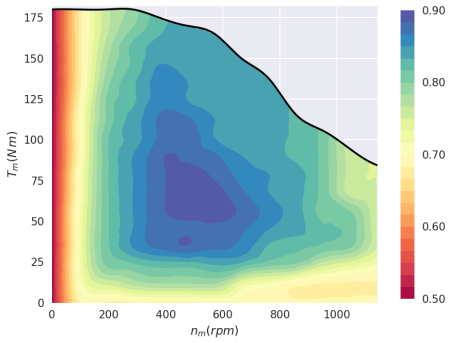


Figure 5.3: Powertrain efficiency map, speed n_m expressed in rpm. Efficiency values are assumed symmetrical in the negative torque case. For positive values, electric torque is bounded by the maximum torque value $T_m^{MAX}(\omega_m) = 180\,Nm$, speed is bounded by the maximum value $n_m^{MAX} = 1140\,rpm$.

who exploited a non-cooperative distributed MPC framework.

We are given a vehicle with mass m and effective wheel radius R_e, which moves on a flat straight line covering the distance $x(t)$, with the driver inputs producing torque at wheel level T_w. We factor the loss in two terms, which include the effects of rolling resistance (deformation of the rolling wheel on the ground) and aerodynamic resistance. The dynamics follow the following equation

$$m\ddot{x}(t) = \frac{T_w(t)}{R_e} - C_r m g \dot{x}(t) - \frac{1}{2}\rho C_a S_v \dot{x}^2(t), \tag{5.5}$$

where C_r is the rolling resistance coefficient, g the gravitational acceleration, ρ the air density, C_a the aerodynamic coefficient and S_v the equivalent vehicle surface.

The torque at wheel level $T_w(t)$ is a function of time given by the combined effect of electrical torque at motor level, T_m, and the one due to the conventional brake action, T_b, that acts directly on the brake callipers. By factoring in the gear ratio between motor and wheels, one has:

$$T_w(t) = T_m(t) \cdot r_g + T_b(t). \tag{5.6}$$

From an energy efficiency standpoint, it is reasonable to expect an optimal platooning policy to minimize the overall consumed electrical energy. This

can be achieved by operating as much as possible the electric powertrain at its most efficient working point, and by avoiding situations in which the conventional brake has to be operated for safety reasons, i.e. when the safety requirement on the distance is violated.

With regards to powertrain efficiency, we define an efficiency map $\eta(T_m(t), \omega_m(t))$ which combines the effects of battery and e-motor, where ω_m is the motor speed (see Fig. 5.3. At time t the consumed electric power is

$$P_m(t) = T_m(t)\omega_m(t)\eta(T_m(t), \omega_m(t))^{-sign(T_m(t))}. \tag{5.7}$$

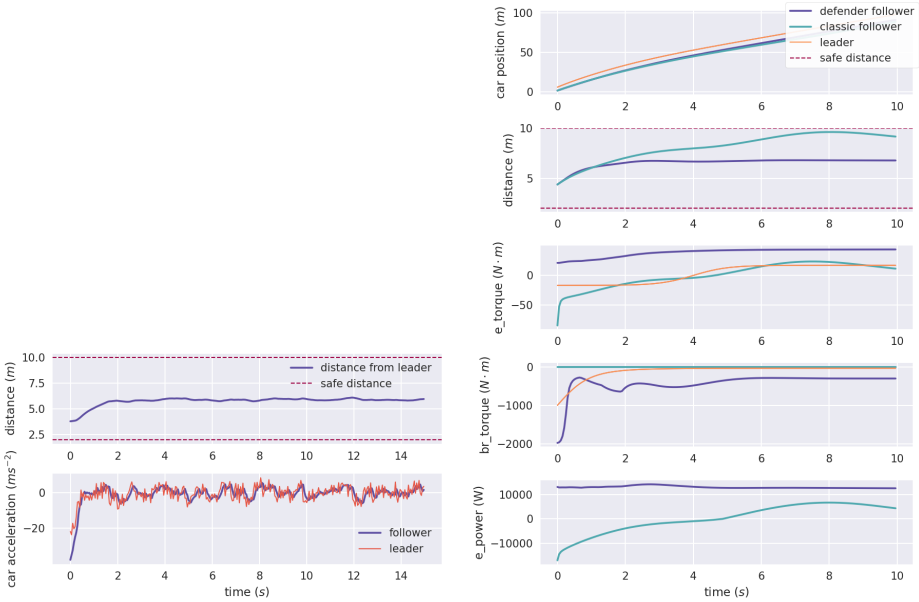
In a single-gear setting, the motor speed is related to vehicle speed through the relation $\omega_m(t) = \frac{r_g}{R_e}\dot{x}(t)$.

Model. We consider the simple case of two cars, one *leader* `l` and one *follower* `f`, whose internal states are position x, velocity v and acceleration a. The follower `f` acts as the agent of this system, while the leader `l` is considered to be part of the adversarial environment, to simulate a cyberattack scenario. They have the same observable states $y_{\mathtt{a}} = y_{\mathtt{e}} = (\dot{x}_{\mathtt{l}}, \dot{x}_{\mathtt{f}}, d)$, given by their velocities and by their relative distance d. In the basic platooning setting the policy networks $\mathcal{A}$ and $\mathcal{D}$ output the accelerations $a_{\mathtt{a}} = \ddot{x}_{\mathtt{f}}$ and $a_e = \ddot{x}_{\mathtt{l}}$, which are used to update the internal states of both cars. When the energy consumption evaluation is involved, the policy networks output electric and conventional torque values for the two cars, $a_{\mathtt{a}} = (T^{\mathtt{a}}_{el}, T^{\mathtt{a}}_{b}), a_{\mathtt{e}} = (T^{\mathtt{e}}_{el}, T^{\mathtt{e}}_{b})$, that are used to compute the corresponding accelerations. The dynamic of a car with mass m and velocity v is described as $m\frac{\mathrm{d}v}{\mathrm{d}t} = ma_{\text{in}} - \nu mg$, where a_{in} is the input acceleration produced by one of the two policies, ν is the friction coefficient and g is the gravity constant. We impose the following STL requirements: $\phi_d = \square_{[0,h]}(d \leq d_{\max} \wedge d \geq d_{\min})$ on the distance $d = x_{\mathtt{l}} - x_{\mathtt{f}}$ between the two vehicles and $\phi_e = \square_{[0,h]}(e \leq e_{\max})$ on the power energy consumption e (note that the higher the robustness of ϕ_e, the lower the energy consumption). The objective of our optimization problem is

$$J(s_0, \theta_{\mathcal{A}}, \theta_{\mathcal{D}}) = \alpha \sum_{t=0}^{H-h} \mathcal{R}_{\phi_d}[\xi_t] + (1-\alpha) \sum_{t=0}^{H-h} \mathcal{R}_{\phi_e}[\xi_t],$$

where $\xi_t = F_h(s_t, \theta_{\mathcal{D}}, \theta_{\mathcal{A}})$, R_{ϕ_d} is the robustness on the distance d and R_{ϕ_e} is the robustness of the energy consumption.

The extension to a platoon of n cars is straightforward: the first car is the leader and each of the other cars follows the one in front. The first pair of subsequent cars act as described in the two-cars model, while the other followers are controlled by copies of the same Defender network.



(a) (b)

Figure 5.4: **(a)** Defender network (follower) against Attacker network (leader) for car platooning problem. Initial state: distance between cars $d = 4.2439\ m$, leader velocity $v_{\mathtt{l}} = 18.1229\ m/s$, follower velocity $v_{\mathtt{f}} = 15.7211\ m/s$. The time horizon is $H = 300$ and the step width is $\Delta t = 0.05\,s$. **(b)** Two different followers against the same leader: *defender* car acts according to the Defender network, *classic* car is managed by a classical controller. In both cases the evolution begins with the same initial configuration: distance between cars $d = 2.9356\ m$, leader velocity $v_{\mathtt{l}} = 17.3107\ m/s$, follower velocity $v_{\mathtt{f}} = 16.3543\ m/s$. The time horizon is $H = 200$ and the step width is $\Delta t = 0.05\,s$.

Results. In the basic platooning scenario, i.e. ignoring energy consumption, the leader acts according to the Attacker's policy, with sudden accelerations and brakes. In such a case, the follower learns to manage the un-

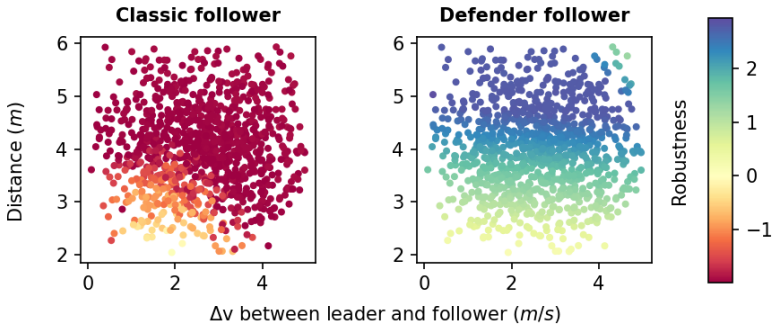


Figure 5.5: Global robustness values on the distance requirement for a classic follower and the Defender follower against the Attacker leader, starting from $1k$ different random configurations of the system (leader and follower cars positions and velocities). Random initializations of the states are described in Tab. 5.1. The time horizon is $H = 200$ and the step width is $\Delta t = 0.05\,s$.

predictable behaviour of the attacker by maintaining their relative distances within the safety range, as shown in Fig. 5.4-(a).

Introducing requirements on energy consumption makes the problem of platooning more challenging. Nonetheless, our architecture is still able to provide a safe controller. Fig. 5.4-(b) shows an evolution of the system in which the leader car is unaware of the state of the follower and the two followers are managed by the Defender network and by a PID classical controller, which implements a basic "energy-aware" distance-tracking strategy. In this scenario, both controllers are able to ensure safety during the whole trajectory, although at the moment the defender appears to be focusing mostly on keeping the ideal distance from the leader, even applying an energy-inefficient strategy.

However, when the leader actions are adverse reactions to the state of the follower, meaning actions chosen by the Attacker, the global robustness achieved by the Defender is in general much higher than the one achieved by the classic controller, as shown in Fig. 5.5.

5.4 Conclusions

Classical control theory struggles in giving adequate safety guarantees in many complex real-world scenarios. We proposed a new learning technique, whose architecture is inspired by Generative Adversarial Networks, and tested its full potential against the vehicle platooning and the cart-pole problem with additional stochastic components. We tested the learned controllers against black-box adversarial policies in the case of completely observable systems, but our framework could also be straightforwardly extended to partially observable systems. Our approach has been able to enforce the safety of the model, while also gaining insights about adverse configurations of the environment.

As future work, we plan to improve the performances in the multi-objective case, to test more complex scenarios, to investigate the scalability of this approach, and to introduce a regularization in the objective function to make the Defender policy less stiff, removing chattering behaviour. We also intend to test the proposed method on properties with temporal behaviours more complex than simple global formulas.

Table 5.1: Training parameters and constraints for the cart-pole problem with adversarial cart-track friction and moving target and for the platooning problem with energy consumption requirements.

Cart-pole with moving target

Training iterations	500 overall steps 1 attacker step 2 defender steps
Time step size	$\Delta t = 0.05\ s$
Time horizon	$H = 40$
Temporal depth	$h = 10$
Initial cart position	$x_0 \sim \mathcal{U}(-1, 1)\ m$
Initial cart velocity	$\dot{x}_0 \sim \mathcal{U}(-1, 1)\ m/s$
Initial pole angle	$\theta_0 \sim \mathcal{U}(-0.1, 0.1)\ rad$
Initial angular velocity	$\dot{\theta}_0 \sim \mathcal{U}(-1, 1)\ m/s$
Position constraint	$x \in [-30\ m, 30\ m]$
Velocity constraint	$\dot{x} \in [-10\ m/s, 10\ m/s]$
Angle constraint	$\theta \in [-1.5\ rad, 1.5\ rad]$
Friction constraint	$\mu \in [0, 1]$
Target position offset constraint	$\dot{\epsilon} \in [-5\ m/s, 5\ m/s]$
Robustness weight	$\alpha = 0.4$
Attacker's architecture	1 layer with 10 neurons each and Leaky ReLU activations
Defender's architecture	2 layers with 10 neurons each and Leaky ReLU activations
Noise space	$K = \mathbb{R}^3$

Car platooning with energy consumption

Training iterations	1000 overall steps 1 attacker step 2 defender steps
Time step size	$\Delta t = 0.05\ s$
Time horizon	$H = 40$
Temporal depth	$h = 10$
Initial distance	$d_0 \sim \mathcal{U}(2, 6)\ m$
Initial velocity	$\dot{x}_{l0}, \dot{x}_{f0} \sim \mathcal{U}(15, 20)\ m/s$
Acceleration constraints	$\ddot{x}_l, \ddot{x}_f \in [-5\ m/s^2, 5\ m/s^2]$
Velocity constraints	$\dot{x}_l, \dot{x}_f \in [0\ m/s^2, 37\ m/s^2]$
Torque constraints	$T_m \in [-T_{m,MAX}(\omega_m), T_{m,MAX}(\omega_m)]$
Robustness weight	$\alpha = 0.98$
Attacker's architecture	1 layer with 10 neurons each and Leaky ReLU activations
Defender's architecture	2 layers with 10 neurons each and Leaky ReLU activations
Noise space	$K = \mathbb{R}^2$

Chapter 6

Neural Predictive Monitoring

Verification of a CPS typically amounts to solving a hybrid automata (HA) reachability checking problem (see Section 2.4.1). Due to its high computational cost, reachability checking is usually limited to design-time (offline) analysis. Our focus is on the online analysis of hybrid systems and, in particular, on the *predictive monitoring* (PM) problem [38]; i.e., the problem of predicting, *at runtime*, whether or not an unsafe state can be reached from the current system state within a given time-bound. PM is at the core of architectures for runtime safety assurance such as Simplex [39], where the system switches to a certified-safe baseline controller whenever PM indicates the potential for an imminent safety violation. In such approaches, PM is invoked periodically and frequently. Thus, reachability needs are determined rapidly, from a single state (the current system state), and typically for short time horizons. This is in contrast with offline reachability checking, where long or unbounded time horizons and sizable regions of initial states are typically considered. PM also differs from traditional runtime verification [40] in that PM is *preemptive*: it detects potential safety violations before they occur, not when or after they occur. Any solution to the PM problem involves a tradeoff between two main requirements: *accuracy* of the reachability prediction, and computational *efficiency*, as the analysis, must execute within strict real-time constraints and typically with limited hardware resources.

In this chapter, we present *Neural Predictive Monitoring* (NPM), a machine-learning-based approach to PM that provides highly accurate predictions in

a highly efficient manner. Moreover, NPM offers principled methods for detecting potential *prediction errors*, which significantly enhances the reliability of PM estimates.

Chapter overview. The chapter is structured as follows. Section 6.1 states the problem of approximating a predictive monitor for a fully or a partially observable HA. Section 6.1.1 introduces the uncertainty-based error detection problem that aims at recognizing errors made by the approximate predictive monitor. Section 6.2 provides the technical details on how the predictive uncertainty is quantified, both in a frequentist and in an Bayesian setting. Section 6.3 specifies how the detection criteria is trained in both the frequentist (Section 6.3.1) and the Bayesian (Section 6.3.2) framework. Section 6.4 presents the possible active learning strategies built on top of the uncertainty measures introduces before. Finally, Section 6.5 presents the experimental results both under full (Section 6.5.3) and partial (Section 6.5.4) observability. Sections 6.5.1 and 6.5.2 introduce respectively the case studies and the measures of performance. Related works are discussed in Section 6.6, whereas conclusions are drawn in Section 6.7.

Let us start by describing the predictive monitoring problem for hybrid automata reachability and the related problem of finding an optimal criterion for rejecting erroneous reachability predictions.

6.1 Problem Statement

Reachability checking of HA is concerned with establishing whether given an initial HA state s and a set of target states D – typically a set of unsafe states to avoid – the HA admits a trajectory starting from s that reaches D.

Definition 5 (Time-bounded reachability)**.** *Given an HA $\mathcal{M}$, either deterministic or nondeterministic, with state space $S(\mathcal{M})$, a set of states $D \subseteq S(\mathcal{M})$, state $s \in S(\mathcal{M})$, and time bound H_f, decide whether there exists a trajectory ξ of $\mathcal{M}$ starting from s and $t \in [0, H_f]$ such that $\rho(t) \in D$, denoted as $\mathcal{M} \models \mathcal{R}each(D, s, H_f)$.*

We aim to derive a predictive monitor for HA reachability, i.e., a function

that can predict whether or not a state in D (an unsafe state) can be reached from the current system state within time H_f. In solving this problem, we assume a distribution $\mathcal{S}$ of HA states and seek the monitor that predicts HA reachability with minimal error probability w.r.t. $\mathcal{S}$. The choice of $\mathcal{S}$ depends on the application at hand and can include a uniform distribution on a bounded state space or a distribution reflecting the density of visited states in some HA executions [98]. We stress that, in general, the HA $\mathcal{M}$ can be characterized either by a deterministic or by a nondeterministic dynamics. In the latter, reachability is a more complex problem to solve. More details would be provided later in the chapter.

Problem 3 (Predictive monitoring for HA reachability)**.** *Given an HA $\mathcal{M}$ with state space $S(\mathcal{M})$, a distribution $\mathcal{S}$ over $S(\mathcal{M})$, a time bound H_f and set of unsafe states $D \subseteq S(\mathcal{M})$, find a function $h^* : S(\mathcal{M}) \to \{0, 1\}$ that minimizes the probability*

$$Pr_{s\sim\mathcal{S}}\left(h^*(s) \neq \mathbf{1}(\mathcal{M} \models \mathcal{R}each(D, s, H_f))\right),$$

where $\mathbf{1}$ is the indicator function. A state $s \in S(\mathcal{M})$ is called positive *w.r.t. a predictor $h : S(\mathcal{M}) \to \{0, 1\}$ if $h(s) = 1$. Otherwise, s is called* negative *(w.r.t. h).*

Any practical solution to the above PM problem must also assume a space of functions within which to restrict the search for the optimal predictive monitor h^*. Following the neural state classification method of [98], in this work we consider functions described by deep neural networks (DNNs). Finding h^*, i.e., finding a function approximation with minimal error probability, is indeed a classical machine learning problem, a *supervised classification* problem in particular, with h^* being the *classifier*, i.e., the function mapping HA state inputs s into one of two classes: 1 (x is positive, can reach D) and 0 (s is negative, cannot reach D).

Training set. In supervised learning, one minimizes a measure of the empirical prediction error w.r.t. a *training set* (see Section 3.1). In our case, the training set Z' is obtained from a finite sample S' of $\mathcal{S}$ by labelling the training inputs $s \in S'$ using some reachability oracle, that is, a hybrid automata reachability checker like [16, 99, 100, 33] (see Section 2.4.1). If the system is deterministic it is sufficient to simulate the system with an ODE solver

and use a event-detection method to check guard conditions and whether the trajectory reaches D. On the other hand, if the system is nondeterministic, solvers that support bounded model-checking are needed. Hence, given a sample S' of $\mathcal{S}$, the training set is defined by

$$Z' = \{(s, \mathbf{1}(\mathcal{M} \models \mathcal{R}each(D, s, H_f)) \mid s \in S'\}.$$

Prediction errors. It is well known that neural networks are universal approximators, i.e., they are expressive enough to approximate arbitrarily well the output of any measurable mathematical function [101]. Even though arbitrarily high precision might not be achievable in practice, state-of-the-art optimization methods based on gradient descent via back-propagation [102] can effectively learn highly accurate neural network approximators (as explained in Section 3). However, such methods cannot completely avoid prediction errors (no supervised learning method can). Therefore, we have to deal with predictive monitors h that are prone to prediction errors, which are of two kinds:

- *false positives* (FPs), when, for a state $s \in S(\mathcal{M})$, $h(s) = 1$ (s is positive w.r.t. h) but $\mathcal{M} \not\models \mathcal{R}each(D, s, H_f)$, and
- *false negatives* (FNs), when $h(s) = 0$ (s is negative w.r.t. h) but $\mathcal{M} \models \mathcal{R}each(D, s, H_f)$.

These errors are respectively denoted by predicates $fn(s)$ and $fp(s)$. In what follows we consider general kinds of prediction error, described by predicate $pe(s)$, and defined by any arbitrary combination of $fn(s)$ and $fp(s)$.

We will interchangeably use the term "(reachability) predictor" for the classifier h and for its discriminant h_d (see Definition 1). Likewise, we will also call h a state classifier, and in particular, a neural state classifier (NSC) when its discriminant h is described by a deep neural network.

Partial Observability

Problem 3 relies on the full observability (FO) assumption, i.e. the assumption that full knowledge about the HS state is available. However,

in most practical applications, state information is partial and noisy. Consider a discrete-time deterministic HS[1] modeled as a HA $\mathcal{M}$ (defined as in Section 2.1.1). The discrete-time deterministic dynamics of the system can be expressed by $v_{i+1} = Flow(l_i)(v_i)$, where $s_i = (l_i, v_i) = (l(t_i), v(t_i))$ and $t_i = t_0 + i \cdot \Delta t$. The measurement process can be modeled as

$$y_i = \mu(s_i) + w_i, \tag{6.1}$$

assuming that partial and noisy observations $y_i \in Y$ are produced by the *observation function* $\mu : S(\mathcal{M}) \to Y$ and the additive measurement noise $w_i \sim \mathcal{W}$.

Under *partial observability* (PO), we only have access to a sequence of past observations $\mathbf{y}_t = (y_{t-H_p}, \ldots, y_t)$ modeled as per (6.1), i.e. we can generate it by applying the observation function μ and measurement noise to the *unknown* state sequence $\mathbf{s}_t = (s_{t-H_p}, \ldots, s_t)$. Let $\mathcal{Y}$ denote the distribution over Y^{H_p}, the space of sequences of observations $\mathbf{y}_t$ induced by the sequence of states $\mathbf{s}_t \sim \mathcal{S}^{H_p}$ and a sequence of i.i.d. noise $\mathbf{w}_t = (w_{t-H_p}, \ldots, w_t) \sim \mathcal{W}^{H_p}$.

Problem 4 (PM for HS under noise and partial observability)**.** *Given the HS and reachability specification of Problem 3, find a function* $g^* : Y^{H_p} \to \{0, 1\}$ *that minimizes*

$$Pr_{\mathbf{y}_t \sim \mathcal{Y}}\Big(g^*(\mathbf{y}_t) \neq \mathbf{1}\big(\mathcal{M} \models \mathcal{R}each(D, s_t, H_f)\big)\Big).$$

In other words, g^* should predict reachability values given in input only a sequence of past observations, instead of the true HS state. In particular, we require a sequence of observations for the sake of identifiability. Indeed, for general non-linear systems, a single observation does not contain enough information to infer the HS state[2]. Problem 4 considers only deterministic systems. Dealing with partial observability and noise in nondeterministic systems remains an open problem as state identifiability is a non-trivial issue.

There are two natural learning-based approaches to tackle Problem 4 (see Fig. 6.1):

[1] In case of partial observability we restrict our analysis to deterministic systems.

[2] Feasibility of state reconstruction is affected by the time lag and the sequence length. Our focus is to derive the best predictions for fixed lag and sequence length, not to fine-tune these to improve identifiability.

1. an **end-to-end** solution that learns a direct mapping from the sequence of past measurements $\mathbf{y}_t$ to the reachability label $\{0, 1\}$.

2. a **two-step** solution that combines steps (a) and (b) below:

   (a) learns a *state estimator* able to reconstruct the history of full states $\mathbf{s}_t = (s_{t-H_p}, \ldots, s_t)$ from the sequence of measurements $\mathbf{y}_t = (y_{t-H_p}, \ldots, y_t)$;

   (b) learns a *state classifier* mapping the sequence of states $\mathbf{s}_t$ to the reachability label $\{0, 1\}$;

Training set. We need reachability oracles to label states s as safe (negative), if $\neg\mathcal{R}each(U, s, H_f)$, or unsafe (positive) otherwise. Given that we consider deterministic HS dynamics, we use simulation, rather than reachability checkers, to label the states. The reachability of the system at time t depends only on the state of the system at time t, however, one can decide to exploit more information and make a prediction based on the previous H_p states. Formally, the generated dataset under FO can be expressed as $Z' = \{(\mathbf{s}_t^i, l^i)\}_{i=1}^N$, where $\mathbf{s}_t^i = (s_{t-H_p}^i, s_{t-H_p+1}^i, \ldots, s_t^i)$ and $l^i = \mathbf{1}(\mathcal{M} \models \mathcal{R}each(U, s_t^i, H_f))$. Under PO, we use the (known) observation function $\mu : S \to Y$ to build a dataset Z'' made of tuples $(\mathbf{y}_t, \mathbf{s}_t, l_t)$, where $\mathbf{y}_t$ is a sequence of noisy observations for $\mathbf{s}_t$, i.e., such that $\forall j \in \{t - H_p, \ldots, t\}$ $y_j = \mu(s_j) + w_j$ and $w_j \sim \mathcal{W}$. The distribution of $\mathbf{s}_t$ and $\mathbf{y}_t$ is determined by the distribution $\mathcal{S}$ of the initial state of the sequences, s_{t-H_p}. We consider two different distributions: *independent*, where the initial states s_{t-H_p} are sampled independently, thus resulting in independent state/observation sequences; and *sequential*, where states come from temporally correlated trajectories in a sliding-window fashion. The latter is more suitable for real-world runtime applications, where observations are received in a sequential manner. On the other hand, temporal dependency violates the exchangeability property, which affects the theoretical validity guarantees of CP, as we will soon discuss.

The predictors, either h or g, are approximate solutions and, as such, they can commit safety-critical prediction errors. We endow the predictive monitor with an error detection criterion R. This criterion should be able to *preemptively* identify – and hence, reject – inputs, either states or sequences of observations, where the prediction is likely to be erroneous (in which case

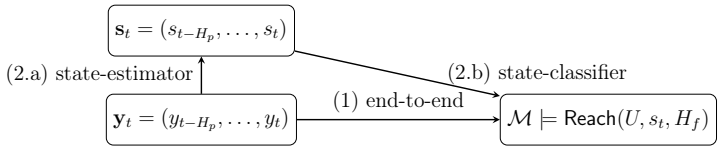


Figure 6.1: Diagram of NSC under PO.

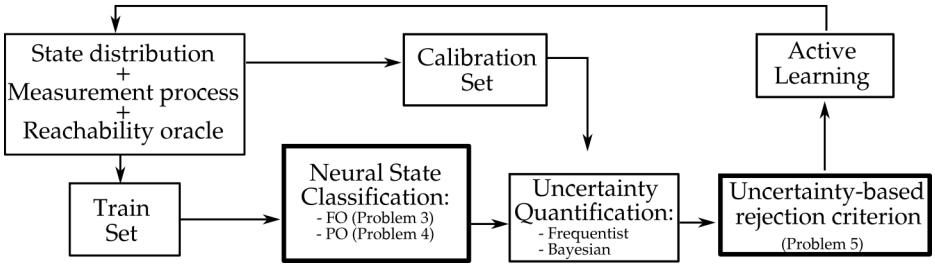


Figure 6.2: Overview of the NPM framework.

R evaluates to 1, 0 otherwise). R should also be optimal in that it has a minimal probability of detection errors. The rationale behind R is that those uncertain predictions are more likely to lead to prediction errors. Hence, rather than operating directly over states or observations, the detector R receives in input a measure of predictive uncertainty of h (or g) about s (or $\mathbf{y}$).

6.1.1 Uncertainty-based error detection

We first introduce a little bit of notation to refer interchangeably to both the full and the partial observability scenario. Let f be the predictor, either h of Problem 3 or g of Problem 4, and let $x \in X$ be the input of such predictor f, either a state s or a sequence of measurements $\mathbf{y}$. The distribution over the generic input space X is denoted by $\mathcal{X}$.

A central objective of this work is to derive, given a predictor f, a rejection criterion R_f able to identify inputs x that are wrongly classified by f, i.e., FNs and FPs or any combination of the two, without knowing the true reachability value of x. Further, R_f should be optimal, that is, it should ensure minimal probability of rejection errors w.r.t. the input distribution

$\mathcal{X}$. For this purpose, we propose to utilize information about the reliability of reachability predictions, to detect and reject potentially erroneous (i.e., unreliable) predictions.

Our solution relies on enriching each prediction with a measure of predictive uncertainty: given f, we define a function $u_f : X \to U$ mapping an HA input $x \in X$ into some measure $u_f(x)$ of the uncertainty of f about x. The set U is called the *uncertainty domain*.

Defining u_f and U is a non-trivial task, details are provided in Section 6.2. The only requirements are that u_f should be point-specific and should not use any knowledge about the true reachability value. Once defined, u_f can be used to build an optimal error detection criterion, as explained below.

Problem 5 (Uncertainty-based error detection)**.** *Given a reachability predictor f, a distribution $\mathcal{X}$ over HA states X, a predictive uncertainty measure $u_f : X \to U$ over some uncertainty domain U, and a kind of error* pe *find an optimal error detection rule $r^*_{f,pe} : U \to \{0, 1\}$, i.e., a function that minimizes the probability*

$$Pr_{x\sim\mathcal{X}}(pe(x) \neq r^*_{f,pe}(u_f(x))).$$

Note that Problem 5 requires specifying the kind of prediction errors to reject. Indeed, depending on the application at hand, one might desire to reject only a specific kind of error. For instance, in safety-critical applications, FNs are the most critical errors while FPs are less important.

As for Problem 3 and 4, we can obtain a sub-optimal solution $r_{f,pe}$ to Problem 5 by expressing the latter as a supervised learning problem, where the inputs are, once again, sampled according to $\mathcal{X}$ and labelled using a reachability oracle. We call *validation set* the set of labelled observations used to learn $r_{f,pe}$. These observations need to be independent of the above-introduced training set Z', i.e., those used to learn the reachability predictor f. In the simplest scenarios, learning $r_{f,pe}$ reduces to identifying an optimal threshold. However, the proposed supervised learning solution is capable of identifying complex and multi-dimensional decision boundaries in an automatic fashion, making it suitable also for complex scenarios.

For an error pe, the final rejection rule $R_{f,pe}$ for detecting HA states where the reachability prediction should not be trusted, and thus rejected,

is readily obtained by the composition of the uncertainty measure and the error detection rule

$$R_{f,pe} = r_{f,pe} \circ u_f : X \to \{0, 1\},$$

where $R_{f,pe}(x) = 1$ if the prediction on state x is rejected; $R_{f,pe}(x) = 0$, otherwise. We remark that rejection rules for different kinds of errors could be combined together to express more sophisticated criteria.

The general goal of Problems 3, 4 and 5 is to minimize the risk of making mistakes in predicting reachability and predicting prediction errors, respectively. We are also interested in establishing probabilistic guarantees on the expected error rate, in the form of predictions regions guaranteed to include the true reachability value with arbitrary probability.

Problem 6 (Probabilistic guarantees)**.** *Given the HS and reachability specification of Problem 3 and 4, find a function* $\Gamma^\epsilon : X \to 2^{\{0,1\}}$*, mapping an input* x *into a prediction region for the corresponding reachability value, i.e., a region that satisfies, for any error probability level* $\epsilon \in (0, 1)$*, the validityom a pool of observed realizations. The accurate abstrac- tion could then be used to efficiently simulate new, previously unobserved, 187 188 CHAPTER 7. CONCLUSIONS trajectories. Once the dynamics of the system is known, we can synthesize a controller that manipulates some controllable variables of the system, so that the system can meet a desired behavioural requirement. The controller would be robust to uncertainties, meaning that it takes into account some of the inevitable modeling uncertainty coming from the abstraction procedure. Finally, the hybrid automata coming from the combination of the abstract model and of the controller can be analysed efficiently using the neural pre- dictive monitoring approach. By doing so, we can check that the running system is actually moving in a safe region. However, every tool provides an approximate solution that introduces in- evitable errors in each step. Moreover, these errors will propagate along the combined system. For these reasons, it is extremely important to leverage the uncertainty quantification techniques presented in Section 3.3. By doing so, we can reduce the number of errors, by actively refining the synthetic train- ing sets, we can detect untrustworthy solutions and can provide statistical guarantees over the performances of the combined approximate solution. [TODO: estendere la discussione secondo i commenti di Jyo] property below*

$$Pr_{x\sim\mathcal{X}}\Big(\mathbf{1}\big(\mathcal{M} \models \mathcal{R}each(D, s, H_f)\big) \in \Gamma^\epsilon\big(x\big)\Big) \geq 1 - \epsilon.$$

Among the maps that satisfy validity, we seek the most *efficient* one, meaning the one with the smallest, i.e. less conservative, prediction regions.

6.2 Uncertainty quantification in Neural Predictive Monitoring

We explore two approaches to quantify the uncertainty produced by a neural network-based reachability predictor f, i.e., to derive the uncertainty measures u_f introduced in Section 6.1.1.

The first approach, referred to as the *frequentist approach*, is based on Conformal Prediction (Section 3.3.2). The second one, referred to as the *Bayesian approach*, relies on Bayesian learning and employs probability distributions to express and measure uncertainty. In particular, we leverage the theory of Bayesian Neural Networks (Section 3.3.1), which combines neural networks and probabilistic modeling.

We present the two uncertainty quantification approaches for a generic classification problem, where we consider an input space X, a set of c classes T, and a classifier $f : X \to T$ with discriminant $f_d : X \to [0,1]^c$. For an input x, we will use the notation $\hat{t}$ as a shorthand for $f(x)$, the classifier prediction on x. We define the data domain by $Z = X \times T$, and we denote with $\mathcal{Z}$ the distribution of the data over Z.

In the context of PM of HA reachability, X is the HA state space (in case of FO) or the space of sequences of measurements Y^{H_p} (in case of PO), $T = \{0,1\}$ ($c = 2$) is the set of possible reachability values, $\mathcal{Z} = Pr_{x\sim\mathcal{X}}(x, \mathbf{1}(\mathcal{M} \models \mathcal{R}each(D, x, T)))$, where $\mathcal{X}$ is the distribution of HA inputs and $\mathcal{R}each(D, s, H_f)$ is the reachability specification, and f is the reachability predictor, i.e., a (sub-)optimal solution of Problem 3 or 4.

Frequentist predictive uncertainty

In Section 3.3.2, we introduce the CP-based quantification of predictive uncertainty for a classification problem. In particular, in Definition 3, we introduce the concepts of confidence and credibility of a prediction, which can

easily be interpreted as two point-wise measures of classification uncertainty.

Definition 6 (Frequentist uncertainty measure)**.** *Given a predictor f with discriminant f_d, we define the frequentist uncertainty measure $u_f : X \to U_{\mathcal{F}} = [0,1]^2$ as the function mapping inputs x into their corresponding confidence and credibility values, obtained as per Definition 3, i.e., $\forall x \in X$, $u_f(x) = (1-\gamma, \kappa)$.*

Bayesian predictive uncertainty

In Section 3.3.1, we show how to derive empirical approximations of the predictive distribution of a BNN (either with HMC or with VI). From such empirical approximation, we can extract statistics to characterize the predictive distribution. In this work, our Bayesian uncertainty measure is based on the empirical mean and variance of the predictive distribution.

Definition 7 (Bayesian uncertainty measure)**.** *Given observations $Z' \sim \mathcal{Z}$ and a Bayesian neural network $f_{\mathbf{w}}$ with $\mathbf{w} \sim p(\mathbf{w}|Z')$, we define the Bayesian uncertainty measure $u_f : X \to U_{\mathcal{B}} \subseteq \mathbb{R}^2$ as the function mapping inputs x into the the empirical mean and variance of the predictive distribution $p(\hat{t} \mid x, Z')$ (3.12). Formally, $\forall x \in X$, $u_f(x) = (\mu_x, \sigma_x^2)$.*

Remark 3 (Softmax probabilities as uncertainty measures)**.** . *A popular method to assess the quality of a prediction is to use the probabilities outputted by the softmax layer of the DNN (i.e., the output of the discriminant function). However, previous works have shown that such probabilities are not well calibrated, meaning that the probability associated with the predicted class label does not reflect its ground-truth correctness likelihood [103, 104]. For example, in a binary classification problem, one can consider the difference between the probability of the two classes as a measure of uncertainty, where small differences indicate uncertain predictions. Later in Fig. 6.6 we show that such a measure yields poor error detection in NPM because the measure is overconfident in predictions that turn out to be erroneous. Such observation supports our claim that more principled and calibrated methods to measure uncertainty are needed. On one hand, our uncertainty measure based on Conformal Prediction overcomes this limitation, as it makes predictions with statistical evidence, rather than probabilistic evidence. On the other hand, our Bayesian measure of uncertainty remains consistent also in*

regions where no data has been observed and, in particular, where the deterministic DNN will behave almost randomly.

6.3 Uncertainty-based Rejection Criteria

In this section we show how to leverage the measures of uncertainty introduced in Section 6.2 to learn an optimal, uncertainty-based error detection rule for reachability predictions, thereby solving Problem 5.

The rationale is that an unseen input x must have sufficiently low uncertainty values for the prediction to be accepted. However, manually determining such decision boundaries on the uncertainty domain U is a non-trivial task. As discussed in Section 6.1, optimal error detection thresholds can be automatically identified by solving an additional supervised learning problem.

For a type of error $e \in \{pe, fp, fn\}$, given a set of validation inputs X_v, sampled from $\mathcal{X}$, and an uncertainty measure u_f, we build a *validation set* W_v^e, defined as

$$W_v^e = \{(u_f(x), \mathbf{1}(e(x))) \mid x \in X_v\}. \tag{6.2}$$

The inputs of W_v^e, referred to as U_v, are the uncertainty measures evaluated on X_v:

$$U_v = \{u_f(x) \mid x \in X_v\}. \tag{6.3}$$

Similarly, each input $u_f(x) \in U_v$ is then labelled respectively with 1 or 0 depending respectively on whether or not the classifier f makes a error of type e on x:

$$E_v^e = \{\mathbf{1}(e(x)) \mid x \in X_v\}. \tag{6.4}$$

For ease of notation, in the following, we omit the type of error considered. However, it is important to keep it in mind when addressing a specific application.

We seek to find those uncertainty values that optimally separate the points in U_v in relation to their classes, that is, separate points yielding errors from those that do not. Finding such a separation corresponds to finding a (sub-)optimal solution to Problem 5.

As for the uncertainty measures of the previous section, we present two alternative solutions to this problem. The first one leverages Support Vector Classification (SVC) and applies to the frequentist case, based on CP, where the uncertainty values are given by confidence and credibility. The second solution leverages Gaussian Process classification (GPC) and applies to the Bayesian case, based on BNNs, where uncertainty values are given by mean and standard deviation of the predictive distribution.

Remark 4 (Higly unbancanced dataset)**.** *In predictive monitoring, the neural network-based reachability predictors that we use have typically very high accuracy (see results in Section 6.5). Therefore, the dataset W_v is typically highly unbalanced, as it contains more examples of correct classifications (class 0) than of classification errors (class 1). In binary classification problems, in particular, in both SVC and GPC, accuracy can be a misleading measure when dealing with imbalanced datasets. Indeed, for instance, a constant function mapping any input into the most frequent class will have high accuracy. Therefore, a model trained on accuracy maximization tends to misinterpret the behaviour of the observations belonging to the minority class, causing misclassification. However, in our method, the less frequent class (i.e., the prediction errors) is indeed the most interesting one, which we want to classify correctly.*

6.3.1 Frequentist error detection via Support Vector Classification (SVC)

For data parsimony, since calibration points were not used to train the reachability predictor, we build the validation set W_v from the calibration set Z_c. A cross-validation strategy is used to compute values of confidence and credibility for points in Z_c. The cross-validation strategy consists of removing the j-th score, α_j, in order to compute γ_j and κ_j, i.e. the p-values at $x_j \in X_c$, where $X_c = \{x \mid \exists t \in T : (x,t) \in Z_c\}$. In this way, we can compute our frequentist uncertainty measure given by confidence $1-\gamma$ and credibility c for every point in the calibration set. The Support Vector Classifier (SVC) is then trained on pairs $((1-\gamma, \kappa), e)$, where e indicates whether or not a prediction error is observed.

In a nutshell, SVC is a kernel-based method that maps the original data into a new space, called feature space, via a feature map ϕ. By doing so, patterns that are not linearly separable in the original data can be converted to be linearly separable in the feature space [44]. The linear decision boundary for a binary SVC is defined as

$$d(x) = a \cdot \phi(x) + b = 0, \tag{6.5}$$

for x in the original space. Training a SVC reduces to find the values of a and b that maximize the margin around the separating hyperplane and the decision function $d(x)$. In the dual formulation of the SVC problem, whose details are out of the scope of this paper, the optimization is performed using kernel functions rather than feature maps. Recall that a kernel can be defined as $k(u, u') = \phi(u)^T \cdot \phi(u')$. In practice, given a test point x_* with predicted label $\hat{t}_*$ and uncertainty measure $u_f(x_*) = (1 - \gamma_*, \kappa_*)$, error detection at x_* boils down to evaluating the learned SVC, i.e., its decision function d, at $u_f(x_*)$. If $d(u_f(x_*)) > 0$ the point x_* is classified as potentially erroneous, class 1, and 0 otherwise.

Tuning of SVC hyperparameters. A simple method to handle imbalanced classes in SVC is via cost-sensitive learning [105]. The aim is to find the classifier that minimizes the mean predictive error on the training set. Each misclassified example by a hypothetical classifier contributes differently to the error function. One way to incorporate such costs is the use of a penalty matrix, which specifies the misclassification costs in a class dependent manner [106]. We design an empirical penalty matrix $\mathcal{P}$, as follows: the (i, j)-th entry of $\mathcal{P}$ gives the penalty for classifying an instance of class i as class j. Of course, when $i = j$, the penalty is null. The penalty matrix for dataset W_v is defined as

$$\mathcal{P} = \begin{bmatrix} 0 & \frac{q}{2r_e(q-n_e)} \\ \frac{r_e q}{2n_e} & 0 \end{bmatrix}, \tag{6.6}$$

where n_e is the number of points belonging to class 1 in W_v, r_e is a parameter influencing how many errors we are willing to accept and $q = |W_v|$. The term $\frac{r_e q}{2n_e}$, which represents the penalty for wrongly classifying an error as correct, increases as n_e decreases. Note that, when $r_e = 1$ and the dataset is perfectly balanced ($q = 2n_e$), the penalties are equal: $\frac{r_e q}{2n_e} = \frac{q}{2r_e(q-n_e)} = 1$. Further, if $r_e > 1$, the penalty term increases, leading to more strict rejection thresholds

and higher overall rejection rates. On the contrary, if $r_e < 1$, the penalty decreases, leading to possibly missing some errors.

Definition 8 (Frequentist error detection criterion)**.** *Given a state $x_* \in X$, a reachability predictor f and an uncertainty measure $u_f(x_*) = (1 - \gamma_*, \kappa_*)$ as per Definition 6, the frequentist error detection function $r_f : U_{\mathcal{F}} \to \{0, 1\}$ rejects a reachability estimate $f(x_*)$, i.e., $r_f\,(1 - \gamma_*, \kappa_*) = 1$, if and only if*

$$d\,(1 - \gamma_*, \kappa_*) > 0,$$

where d is the binary SVC of (6.5) trained on W_v (6.2).

6.3.2 Bayesian error detection via Gaussian Process Classification (GPC)

Recall that we define our Bayesian uncertainty measure like the mean and variance of the empirical approximation of the BNN predictive distribution. Therefore, the prediction $\hat{t}_*$ made on an unseen input x_* is associated with a vector of uncertainty $(\mu_*, \sigma_*^2) \in U_{\mathcal{B}} \subset \mathbb{R}^2$.

To keep the approach fully Bayesian, we propose a probabilistic solution to the error detection problem based on Gaussian Processes [107].

Formally, a *Gaussian Process (GP)* is a stochastic process, i.e., a collection of random variables indexed by some input variable, in our case $u \in U$, such that every finite linear combination of them is normally distributed. In practice, a GP defines a distribution over real-valued functions of the form $\ell : U_{\mathcal{B}} \to \mathbb{R}$ and such distribution is uniquely identified by its mean and covariance functions, respectively denoted by $m(u) = \mathbb{E}[\ell(u)]$ and $k(u, u')$. The GP can thus be denoted as $\mathcal{GP}(m(u), k(u, u'))$. This means that the function value at any point u, $\ell(u)$, is a Gaussian random variable with mean $m(u)$ and variance $k(u, u)$. Typically, the covariance function $k(\cdot, \cdot)$ depends on some hyper-parameter γ.

As mentioned before, GPs can be used to perform probabilistic binary classification, i.e., learning a Bayesian classifier $G_f : U_{\mathcal{B}} \to \{0, 1\}$ from a set of observations. GPs model the posterior probabilities by defining latent functions $\ell : U_{\mathcal{B}} \to \mathbb{R}$, whose output values are then mapped into the $[0, 1]$

interval by means of a so-called link function Φ. Typically, in a binary classification problem, the logit or the probit function are used as Φ.

Given an input u_i, let $\ell_i = \ell(u_i)$ denote its latent variable, i.e., the latent function ℓ evaluated at u_i. Also denote $l_v = [\ell(u_i) \mid u_i \in U_v]$, where U_v is a set of input points defined as per 6.3. From U_v it is possible to compute the mean vector m_v of the GP, by evaluating the mean function $m(\cdot)$ at every point in the set, and the covariance matrix K_v^γ, by evaluating the covariance function on every pair of points in the set: $m_v = [m(u_i)|u_i \in U_v]$ and $K_v^\gamma = [k_\gamma(u_i, u_j)|u_i, u_j \in U_v]$.

The first step of a GPC algorithm is to place a GP prior over the latent function ℓ, defined by

$$p(\ell|U_v) = \mathcal{N}(\ell|m_v, K_v^\gamma).$$

Let now consider a test input u_* with latent variable ℓ_*. In order to do inference, that is, predict its label e_*, we have to compute

$$p(e_* = 1|u_*, U_v, E_v) = \int \Phi(\ell_*)p(\ell_*|u_*, U_v, E_v)d\ell_*, \tag{6.7}$$

where E_v, see Equation 6.4, denotes the set of labels corresponding to points in U_v, see Equation 6.3. The discriminant function $r_{f,d} : U_{\mathcal{B}} \to [0, 1]^2$ of the error detection classifier r_f is defined as

$$r_{f,d}(u_*) = [1 - p(e_* = 1|u_*, U_v, E_v), p(e_* = 1|u_*, U_v, E_v)]. \tag{6.8}$$

To compute equation (6.7), we have to marginalize the posterior over the latent Gaussian variables:

$$p(\ell_*|u_*, U_v, E_v) = \int p(\ell_*|u_*, U_v, \ell_v)p(\ell_v|U_v, E_v)d\ell_v, \tag{6.9}$$

where the posterior $p(\ell_v|U_v, E_v)$ can be obtained using the standard Bayes rule

$$p(\ell_v|U_v, E_v) = \frac{p(E_v|\ell_v, U_v)p(\ell_v|U_v)}{p(E_v|U_v)}.$$

Therefore, performing inference reduces to solving two integrals, Eq. 6.7 and 6.9. In classification, the first integral is not available in closed form since it is the convolution of a Gaussian distribution, $p(\ell_v)$, and a non-Gaussian

one, $p(E_v|\ell, U_v)$. Hence, we have to rely on approximations in order to compute and integrate over the posterior $p(\ell_v|E_v)$. In our experiments, we use the Laplace method, which provides a Gaussian approximation $q(\ell_v|E_v)$ of the posterior $p(\ell_v|E_v)$, which can then be easily computed and integrated over.

Tuning of GPC hyperparameters. In the prior distribution, the covariance for the latent variables depends on some hyperparameters γ. A classical strategy to select the optimal values for such parameters is to find the values of γ that maximize the marginal likelihood, which, intuitively, measures how likely the data are, given a certain value of γ. However, as mentioned in Remark 4, the marginal likelihood may be a poor choice because class 1 is very little represented. An alternative solution is to compute, for different values of γ, the confusion matrix of the GPC on the training set W_v. The entries of such a matrix can be used to define more clever measures of performance that apply to binary classification. In our experiments, we use the true positive rate (TPR), as it is well-suited for datasets presenting a strong disproportion. TPR measures the fraction of points in class 1 that have been correctly classified:

$$TPR_\gamma := \frac{TP}{TP+FN}, \tag{6.10}$$

where TP indicates the number of true positives and FN indicates the number of false negatives. Alternative measures, such as the Matthews correlation coefficient (MCC) [108], may apply. Note that, during the training phase, the GPC assigns to each point the class with the highest likelihood.

Another key step is the following. The discriminant function, $r_{f,d}$, returns a vector containing the probability of belonging to each of the two classes. However, such probabilities might not separate well in cases of highly unbalanced datasets. Therefore, choosing the class with the highest probability, as per Definition 1, may lead to bad performance. Therefore, after the GPC has been trained with an optimal value for γ, it may be useful to find the decision threshold that maximizes the GPC accuracy on the training set W_v. This can be done, for instance, using the ROC curve. In other words, we classify as correct (class 0) only those points that have an extremely high probability of belonging to that class. We do so by searching for the threshold τ that maximizes the quantity $TPR - FPR$, i.e., the proportion of recognized errors minus the proportion of points wrongly rejected. Below we provide

the formal definition of the Bayesian error detection function for a generic threshold τ.

Definition 9 (Bayesian error detection criterion)**.** *Given a state $x_* \in X$, a reachability predictor f, a Bayesian uncertainty measure $u_f(x_*) = (\mu_*, \sigma_*^2) = u_*$ as per Definition 7, and a decision threshold $\tau \in [0, 1)$, the error detection function $r_f : U_{\mathcal{B}} \to \{0, 1\}$ rejects a reachability estimates $f(x_*)$ if and only if*

$$p(e_* = 1 \mid u_*, U_v, E_v) > \tau,$$

where U_v, E_v are the inputs and outputs of the validation set, see (6.3) and (6.4).

6.4 Active Learning

Recall that we are dealing with two related learning problems: learning a prediction rule (i.e., a reachability predictor) using the training set Z_t, and learning a rejection rule using the validation set W_v (via learning an adequate uncertainty measure first).

As the accuracy of a classifier increases with the quality and the quantity of observed data, adding samples to Z_t will generate a more accurate predictor, and similarly, adding samples to W_v will lead to more precise error detection. Ideally, one wants to maximize accuracy while using the least possible amount of additional samples, because obtaining labeled data is expensive (in NPM, labelling each sample entails solving a reachability checking problem), and the size of the datasets affects the complexity and the dimension of the problem. Therefore, to improve the accuracy of our learning models efficiently, we need a strategy to identify the most "informative" additional samples.

For this purpose, we propose an *uncertainty-aware active learning* solution, where the re-training points are derived by first sampling a large pool of unlabeled data, and then considering only those points where the current predictor f is still uncertain. The criterion used to decide whether a point is uncertain enough to be considered informative is indeed our rejection rule R_f. In particular, recall that the uncertainty function $u_f : X \to U$

maps input states to their level of uncertainty, and the error detection function $r_f : U \to \{0, 1\}$ maps uncertainty values to a binary class interpreted as accepting/rejecting a prediction. The rejection rule $R_f : X \to \{0, 1\}$, introduced in Section 6.1, is defined as the combination of these two functions, $R_f = r_f \circ u_f$. Therefore, such a rejection rule provides an effective uncertainty-based query strategy. Points rejected by R_f, i.e., points whose predictions are expected to be erroneous, are indeed the most uncertain ones.

The proposed active learning method should reduce the overall number of erroneous predictions, because it improves the predictor on the inputs where it is most uncertain, and, as a consequence, also reduces the overall rejection rate. However, it cannot be excluded in general that the retraining process introduces new prediction errors. We stress that, with our method, these potential new errors can be effectively detected.

6.4.1 General active learning algorithm

Our active learning algorithm works as follows. The rejection rule R_f is used as a query strategy to identify, from a batch of randomly selected unlabeled points, those with a high degree of uncertainty. We then query the oracle, i.e., the HA reachability checker, to label such uncertain points, and finally, we divide them into two groups: one group is added to the training set $Z_t \subseteq X \times Y$, producing the augmented dataset Z_t^a; the other is added to the validation set Z_v, producing Z_v^a. The set of uncertain points must be divided according to the splitting probability used to originally divide Z' into Z_t and Z_v. The first step consists in retraining the reachability predictor on Z_t^a. Let f_a denote the new predictor. The second step requires extracting the augmented validation dataset W_v^a from Z_v^a. To do so, we must first train a new uncertainty measure u_{f_a} to reflect the new predictor f_a. Then, the new error detection rule r_{f_a} is trained on

$$W_v^a = \{(u_{f_a}(x), \mathbf{1}(e_{f_a}(x)) \mid x \in X_v^a\},$$

where e_{f_a} is the error predicate introduced in Section 6.1 (the f_a index is added to stress dependency on the updated predictor). In conclusion, this process leads to an updated rejection rule $R_{f_a} = r_{f_a} \circ u_{f_a}$, which is expected to have a reduced rate of incorrect rejections.

Algorithm 12 Active Learning algorithm

Inputs: training set Z_t, validation set Z_v, predictor f, uncertainty function u_f, rejection rule R_f, maximum iterations n_{it}.

Outputs: enhanced predictor f_a, enhanced rejection rule R_f^a.

procedure ACTIVE LEARNING
  for $i = 1, \ldots, n_i t$ **do**
    # Select re-training inputs
    Randomly sample a set of input points.
    Identify the subset A of points rejected based on R_f.
    # Derive augmented datasets
    Invoke the reachability oracle to label the points in A.
    Divide the data into two groups and add them respectively to Z_t and Z_v, obtaining an augmented training set, Z_t^a, and an augmented validation set, Z_v^a.
    Train a new predictor f_a from Z_t^a.
    Build the training set $W_v^{f_a}$ using Z_v^a and f_a.
    Train a new error detection rule r_{f_a}, using u_f and the method of Section 6.3, and obtain the enhanced rejection rule R_{f_a}.
  $Z_t \leftarrow Z_t^a$, $Z_v \leftarrow Z_v^a$, $f \leftarrow f_a$, $R_f \leftarrow R_f^a$.

We now describe in detail our uncertainty-aware active learning algorithm, which given an initial training set Z_t, a predictor with discriminant f trained on Z_t, an initial validation set Z_v, and a rejection rule R_f trained on Z_v computes an enhanced predictor f_a and enhanced rejection rule R_{f_a} as follows.

The above algorithm is used as-is in the Bayesian framework. For the frequentist case, we present a refined version of the algorithm that overcomes issues with the sensitivity of CP-based measures. The aforementioned sensitivity issues are explained below.

Sensitivity of CP-based uncertainty measures. The distribution of calibration scores depends both on the case study at hand and on the trained classifier. If such a classifier f has high accuracy, then most of the calibration scores $\alpha_1, \ldots, \alpha_q$ will be close to zero. Each p-value p_*^j of an unseen test point x_* counts the number of calibration scores greater than α_*^j, the non-

conformity score for label j at x_*. Credibility, which is the p-value associated with the class predicted by f_d, is expected to have a small score and therefore a high p-value. On the contrary, γ, which is the p-value associated with the other (non-predicted) class, is expected to have a larger score. However, given the high accuracy of f, the number of calibration scores significantly greater than zero is very small. Therefore, the fraction of calibration scores determining γ is not very sensitive to changes in the value of α_*, which is determined by $f_d(x_*)$. On the contrary, credibility is extremely sensitive to small changes in α_*. In general, the sensitivity of confidence with respect to α_* increases as the accuracy of f decreases, and vice versa for credibility. Figure 6.3 shows the credibility landscapes for two different training instances of model f_d on the same training set for a concrete case study. We observe that even if regions, where misclassifications take place, are always assigned low credibility values, outside those regions credibility values are subject to high variance.

This sensitivity results in an over-conservative rejection criterion, leading to a high rejection rate and in turn, to an inefficient query strategy. However, if we enrich the calibration set using additional samples with non-zero α-scores, we can reduce such sensitivity, thereby making credibility more robust with respect to retraining. This process is illustrated in Figure 6.3, where the additional non-zero α-scores (right) leads to a more robust credibility landscape, where low-credibility regions are now more tightly centred around areas of misclassification.

Observing that samples with uncertain predictions will have non-zero α-scores[3], we will use the original rejection rule to enrich the calibration set, thereby deriving a refined rejection rule and in turn, a refined and more effective query strategy for active learning.

6.4.2 Frequentist active learning algorithm

Provided that the credibility measure is extremely sensitive in our application, we found that dividing the frequentist active learning algorithm into two phases dramatically improves performances. In the first phase, we refine the query strategy: we use the current rejection rule R_f to identify a

[3]The α-score of a sample (x_i, y_i) is zero only if $f_d^{t_i}(x_i) = 1$.

Algorithm 13 Frequentist AL algorithm

Inputs: training set Z_t, calibration set Z_c, predictor f, uncertainty function u_f, rejection rule R_f, maximum iterations n_{it}.

Outputs: enhanced predictor f_a, enhanced rejection rule R_f^a.

procedure FREQUENTIST ACTIVE LEARNING
  for $i = 1, \ldots, n_i t$ **do**
    #Refine the query strategy
    Randomly sample a set of input points.
    Identify the subset Q of points rejected by R_f.
    Identify the subset A of points rejected based on R_f.
    Invoke the reachability oracle to label the points in Q.
    Define a query set Z_Q by adding these points to Z_c.
    Obtain an updated rejection rule R_f^Q from Z_Q using the method of Section 6.3.
    # Active phase
    Randomly sample a set of input points.
    Identify the subset A of points rejected by R_f^Q.
    Invoke the reachability oracle to label the points in A.
    Divide the labeled data into two groups and add them respectively to Z_t and Z_c, obtaining an augmented training set, Z_t^a, and an augmented calibration set, Z_c^a.
    Train a new predictor f_a from Z_t^a.
    $Z_t \leftarrow Z_t^a$, $Z_c \leftarrow Z_c^a$, $f \leftarrow f_a$, $R_f \leftarrow R_f^a$.

batch of uncertain points, temporarily add these points to the calibration set, thereby obtaining an updated, more robust, rejection rule that we use as a query strategy. In the second phase, we simply perform an active learning iteration i.e., steps 2–5 above) but using the refined query strategy to identify the re-training inputs.

We now describe the details of this variant of the active learning algorithm designed for the frequentist framework. Given an initial training set Z_t, a prediction rule f trained on Z_t, an initial calibration set Z_c, a rejection rule R_f trained on Z_c and a rejection ratio r_e, we proceed as follows.

It is important to observe that, for the active learning algorithm to preserve the statistical soundness of conformal prediction, the augmented train-

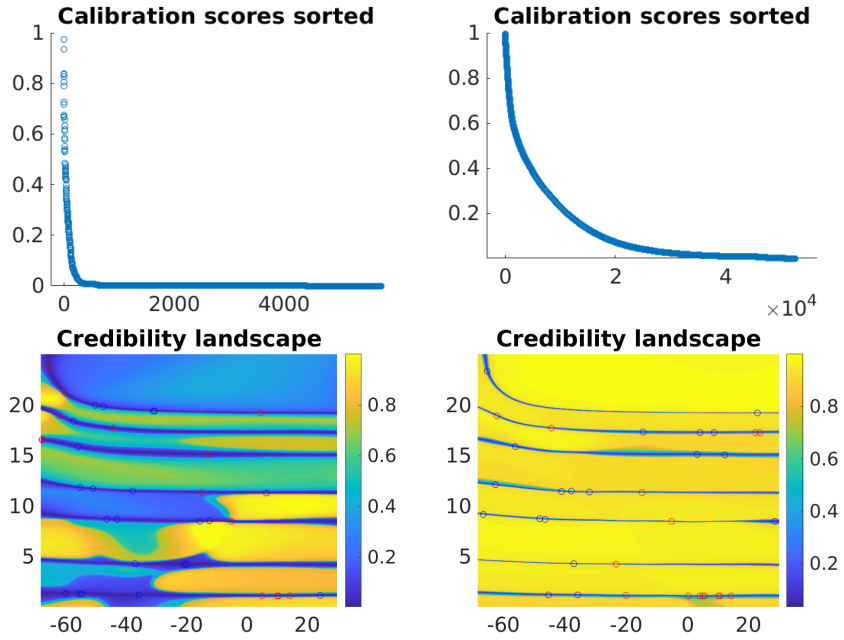


Figure 6.3: Credibility values in the spiking neuron case study. Calibration scores (first row) and credibility landscapes using the initial calibration set Z_c (left column) versus the query set Z_Q (right column). The landscapes are obtained for different instances of the predictor f, trained on the same dataset Z_t.

ing and calibration sets Z_t^a and Z_c^a must be sampled from the same distribution. This is guaranteed by the fact that, in the active learning phase, we add new points to both the training and the calibration dataset, and these points are sampled from the same distribution (in particular, we apply the same random sampling method and same rejection criterion). The only caveat is ensuring that the ratio between the number of samples in Z_c and Z_t is preserved on the augmented datasets.

6.5 Experimental Results

We experimentally evaluate the proposed method both in the FO and the PO scenario. In the FO scenario, we compare the frequentist and Bayesian

approaches on a benchmark of six hybrid system models with varying degrees of complexity. In the PO scenario, we use the frequentist aprroach[4] to evaluate both end-to-end and two-step approaches on six benchmarks of cyber-physical systems with dynamics presenting a varying degree of complexity and with a variety of observation functions.

6.5.1 Case Studies

Full Observability

We consider four deterministic case studies:

- $\mathbf{SN_{fo}}$: this model describes the evolution of a neuron's action potential (as on the Flow* website[5]). It is a deterministic HA with two continuous variables, one mode, one jump and nonlinear polynomial dynamics, defined by the ODE

$$\begin{cases} \dot{v_2} & = 0.04v_2^2 + 5v_2 + 140 - v_1 + I \\ \dot{v_1} & = a \cdot (b \cdot v_2 - v_1) \end{cases} \tag{6.11}$$

  The jump condition is $v_2 \geq 30$, and the associated reset is $v_2' := c \wedge v_1' := v_1 + d$, where, for any variable v, v' denotes the value of v after the reset. The parameters are $a = 0.02$, $b = 0.2$, $c = -65$, $d = 8$, and $I = 40$ as reported on the Flow* website. The state space is $68.5 < v_2 \leq 30 \wedge 0 \leq v_1 \leq 25$. We consider the unsafe set D defined by $v_2 \leq 68.5$, expressing that the neuron should not undershoot its resting potential. The time bound for the reachability property is $H_f = 20$.

- $\mathbf{IP_{fo}}$: this model describes the classic inverted pendulum on a cart, which is a two-dimensional model with non-linear dynamics, defined by the ODE

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = \sin(\theta) - \cos(\theta) \cdot u \end{cases} \tag{6.12}$$

[4]We choose to focus on the frequentist approach since it has been shown to perform better and since it scales better.

[5]https://flowstar.org/examples/

The control input $u := F/g$ is a force, divided by the gravitational acceleration g, applied to the cart to keep the pendulum in the upright position, i.e., $\theta = 0$. The mass of the pendulum is $1/g$, and the length of the rod is 1. We consider the control law of Eq. 6.13.

$$u = \begin{cases} \dfrac{2 \cdot \omega + \theta + \sin(\theta)}{\cos(\theta)}, & E \in [-1, 1], \mid \omega \mid + \mid \theta \mid \leq 1.85 \\ 0, & E \in [-1, 1], \mid \omega \mid + \mid \theta \mid > 1.85 \\ \dfrac{\omega}{1 + \mid \omega \mid} \cos(\theta), & E < -1 \\ \dfrac{-\omega}{1 + \mid \omega \mid} \cos(\theta), & E > 1 \end{cases} \tag{6.13}$$

where $E = 0.5 \cdot \omega + (\cos(\theta) - 1)$ is the pendulum energy. We consider the unsafe set D defined by $|\theta| > \pi/4$, corresponding to the safety property that keeps the pendulum within 45° of the vertical axis. The time bound is $H_f = 5$.

- $\mathbf{AP_{fo}}$: this model describes the dynamics of an artificial pancreas [109], which is a six-dimensional non-linear model. The unsafe set D corresponds to hypoglycemia states, i.e., $D = BG \leq 3.9$ mmol/L, where BG is the blood glucose variable. The state distribution considers uniformly distributed values of plasma glucose and insulin. The insulin control input is fixed to the basal value. The time bound is $H_f = 240$;

- $\mathbf{HC_{fo}}$: the helicopter model [98], which is a linear model with 29 state variablesWe augment the 28-variable helicopter controller available on SpaceEx website[6] with a variable z denoting the helicopter's altitude. The dynamics of z is given by $\dot{z} = v_z$, where v_z is the vertical velocity and represented by variable x_8. The unsafe set D is defined by $z \leq 0$. The time bound is $H_f = 5$. Since this model is large and publicly available on SpaceEx website, we do not provide the details here.

In addition, we analyze two non-deterministic models with non-linear dynamics:

[6] http://spaceex.imag.fr/

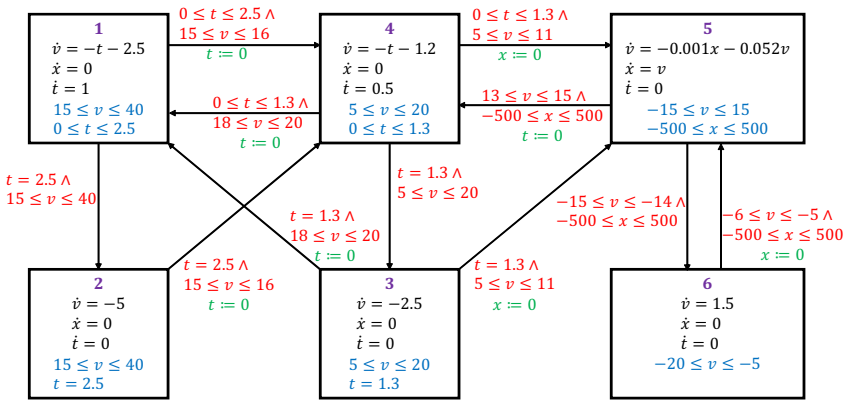


Figure 6.4: Hybrid automaton for the cruise control system. Invariants are in blue, guards are in red, and reset mappings are in green.

- **$\mathbf{CC_{fo}}$**: this models describes a Cruise Controller [98], whose input space has four dimensions and a nonlinear polynomial dynamics. The unsafe set D is defined by $v \leq -1$, which expresses that the vehicle's speed should not be below a reference speed by $1m/s$ or more. The reachability time-bound is $H_f = 10$. The full model has three continuous variables, six modes, eleven jumps as shown in Fig. 6.4. The continuous variable v denotes the difference between the vehicle's speed and the cruise speed in m/s, x is the integral term for the proportional-integral (PI) controller in mode 5, and t is a clock. In mode 5, the PI controller tries to stabilize v to zero, i.e., to match the vehicle's speed with the cruise speed. Mode 3 and 4 represent the first level of brakes where deceleration increases smoothly from 1.2 to 2.5 m/s^2 in mode 4 and stays constant at 2.5 m/s^2 in mode 3. Mode 1 and 2 represent the second level of brakes and work in the same way but with higher starting and peak deceleration. Mode 6 constantly accelerates the vehicle. The guards are designed to prevent chattering or Zeno behaviour.

- **$\mathbf{TWT_{fo}}$**: this model describes the Triple Water Tank system[7], which is a three-dimensional model. The unsafe D is given by states where the water level of any of the tanks falls outside a given safe interval I, i.e., $D = \vee_{i=1}^{3} x_i \notin I$, where x_i is the water level of tank i. The state distribution considers water levels uniformly distributed within the safe

[7] http://dreal.github.io/benchmarks/networks/water/

interval. The time-bound is $H_f = 1$.

Partial Observability

In the PO scenario, we consider deterministic models only.

- **IP$_{\mathbf{po}}$**: classic two-dimensional non-linear model of an Inverted Pendulum on a cart. Given a state $s = (s_1, s_2)$, we observe a noisy measure of the energy of the system $y = s_2/2 + \cos(s_1) - 1 + w$, where $w \sim \mathcal{N}(0, 0.005)$. Unsafe region $D = \{s : |s_1| \geq \pi/6\}$. $H_p = 1$, $H_f = 5$.

- **SN$_{\mathbf{po}}$**: a two-dimensional non-linear model of the Spiking Neuron action potential. Given a state $s = (s_1, s_2)$ we observe a noisy measure of s_2, $y = s_2 + w$, with $w \sim \mathcal{N}(0, 0.1)$. Unsafe region $D = \{s : s_1 \leq -68.5\}$. $H_p = 4$, $H_f = 16$.

- **CVDP$_{\mathbf{po}}$**: a four-dimensional non-linear model of the Coupled Van Der Pol oscillator [110], modeling two coupled oscillators. The dynamics is defined by the following ODE:

$$\begin{cases} \dot{s}_1 &= s_2 \\ \dot{s}_2 &= (1 - s_1^2)s_2 - 2s_1 + s_3 \\ \dot{s}_3 &= s_4 \\ \dot{s}_4 &= (1 - s_3^2)s_4 - 2s_3 + s_1 \end{cases} \tag{6.14}$$

  Given a state $s = (s_1, s_2, s_3, s_4)$ we observe $y = (s_1, s_3) + \mathbf{w}$, with $\mathbf{w} \sim \mathcal{N}(\underline{0}, 0.01 \cdot I_2)$. Unsafe region $D = \{s : s_2 \geq 2.75 \wedge s_2 \geq 2.75\}$. $H_p = 8$, $H_f = 7$.

- **LALO$_{\mathbf{po}}$**: the seven-dimensional non-linear Laub Loomis model [110] of a class of enzymatic activities. The dynamics is defined by the

following ODE:

$$\begin{cases} \dot{s}_1 &= 1.4s_3 - 0.9s_1 \\ \dot{s}_2 &= 2.5s_5 - 1.5s_2 \\ \dot{s}_3 &= 0.6s_7 - 0.8s_2s_3 \\ \dot{s}_4 &= 2 - 1.3s_3s_4 \\ \dot{s}_5 &= 0.7s_1 - s_4s_5 \\ \dot{s}_7 &= 0.3s_1 - 3.1s_6 \\ \dot{s}_8 &= 1.8s_6 - 1.5s_2s_7. \end{cases} \tag{6.15}$$

The system is asymptotically stable with equilibrium at the origin. The unsafe region is defined as $D = \{s : s_4 \geq 4.5\}$. Given a state $s = (s_1, \ldots, s_7)$ we observe $y = (s_1, s_2, s_3, s_5, s_6, s_7) + \mathbf{w}$, with $\mathbf{w} \sim \mathcal{N}(0, 0.01)$, $H_p = 5$ and $H_f = 20$.

- $\mathbf{TWT_{po}}$: a three-dimensional non-linear model of a Triple Water Tank. Given a state $s = (s_1, s_2, s_3)$ we observe $y = s + w$, with $w \sim \mathcal{N}(\underline{0}, 0.01 \cdot I_3)$. Unsafe region $D = \{s : \vee_{i=1}^{3} s_i \notin [4.5, 5.5]\}$. $H_p = 1$, $H_f = 1$.
- $\mathbf{HC_{po}}$: the 28-dimensional linear model of an Helicopter controller. We observe only the altitude, i.e. $y = s_8 + w$, with $w \sim \mathcal{N}(0, 1)$. Unsafe region $D = \{s : s_8 < 0\}$. $H_p = 5$, $H_f = 5$.

Implementation. The workflow can be divided into steps: (1) define the CPS models, (2) generate the synthetic datasets, (3) train the NPM, (4) train the CP-based error detection rules, (5) perform active learning and (6) evaluate both the initial and the active NPM on a test set. From here on, we call *initial setting* the one with no active learning involved.

Experimental settings. The experiments were performed on a computer with a CPU Intel x86, 24 cores and 128GB RAM and a GPU Tesla V100. The entire pipeline is implemented in Python, and the neural networks are trained with TensorFlow [111] and PyTorch [75]. More precisely, PyTorch, a Python deep learning library, is used to train the deterministic DNN, Edward [112], a Python library for probabilistic modeling built on TensorFlow, is used to train the BNN with HMC inference, and Pyro [113], a probabilistic programming library built on PyTorch, is used to train the BNN with VI.

The source code for all the experiments under FO can be found at:
https://github.com/francescacairoli/NPM.
The source code for the experiments under PO can be found at:
https://github.com/francescacairoli/PO-NPM.

6.5.2 Performance measures

We want our method to be capable of working at runtime, which means it must be extremely fast in making predictions and deciding whether to trust them. We emphasize that the time required to train the reachability predictor and the error detection rule does not affect its runtime efficiency, as it is performed in advance (offline) only once. Also, we do not want an over-conservative rejection rule, as unnecessary rejections would reduce the effectiveness of our predictive monitor[8]. Keeping that in mind, the relevant performance metrics for NPM are the *accuracy of the reachability predictor* f, the *error detection rate* (or *recognition rate*) and the overall *rejection rate* of the rejection rule R_f. The error detection rate measures the proportion of errors made on the test set by f that is actually recognized by R_f, whereas, the rejection rate measures the overall proportion of test points rejected by R_f. Clearly, we want our method to be reliable and thus, detect the majority of prediction errors (high detection rate) without being overly conservative, i.e., keeping a low rejection rate. Another important remark is about the interaction of the two classifiers, f and r_f: as the accuracy of f increases, it commits fewer errors, which makes it harder for the detection rule r_f to learn how to capture them because the validation set W_v for training r_f will contain few examples of prediction errors. The opposite holds as well: if f performs poorly, it produces a less unbalanced validation set W_v, which may result in a more accurate rejection rule R_f. These two behaviours are balanced against one another, as discussed above, by tuning the training of the rejection rules.

[8]Defining countermeasures to a rejected prediction is out of the scope of our method, but these may include switching to a fail-safe mode of the system or querying the HA model checker for the true reachability value. Both cases consistently affect the runtime efficiency of our monitor.

Computational Performances

NPM is designed to work at runtime in safety-critical applications, which translates into the need for high computational efficiency together with high reliability. The time needed to generate the dataset and to train both methods does not affect the runtime efficiency of the NPM, as it is performed only once (offline).

Offline costs. Training an NPM requires the following steps: (i) training the state classifier, (ii) generating the datasets W_v, which requires computing the uncertainty values for each point in Z_v, and (iii) training the error rejection rule. All these steps are performed offline.

Online costs. The online cost consists of the time needed to evaluate the NPM, i.e., the time needed to make a prediction and choose whether to accept it or not. Importantly, this time does not depend on the dimension or dynamics of the hybrid system. However, in the Bayesian approach, it depends on the number of observations used to empirically approximate the predictive distribution, which is a fixed cost, whereas, in the frequentist approach the evaluation time is affected by the size of the calibration set Z_c, which may increase as we add observations[9]. On this aspect, the query strategy refinement we propose for active learning ensures that the augmented calibration set is as small as possible, which translates into the runtime efficiency of our method.

Active learning overhead (offline). Active learning carries two additional training costs: the time needed to compute uncertainty values for a large pool of data, and the time the oracle needs to compute labels for the most uncertain points. The latter dominates, especially for non-deterministic systems, since they require fully-fledged reachability checking, which is more expensive than a simulation of a deterministic system. Therefore, if the

[9]The size of Z_c affects only the computation of the uncertainty measures, which reduces to computing two p-values (confidence and credibility). Each p-value is derived by computing a nonconformity score, which has the same cost as evaluating the state classifier, and one search over the array of calibration scores.

	$\mathbf{IP_{fo}}$	$\mathbf{AP_{fo}}$	$\mathbf{CC_{fo}}$	$\mathbf{TWT_{fo}}$	$\mathbf{HC_{fo}}$	$\mathbf{SN_{fo}}$
DNN-S	**99.84**	**99.56**	**99.92**	**99.91**	98.33	**99.78**
SNN	99.74	99.49	99.91	99.81	**98.96**	99.51
DNN-R	99.61	99.41	99.91	99.82	98.75	97.59
SVM	98.85	99.17	99.50	99.32	96.54	67.94
RF	99.66	96.61	99.19	99.24	91.67	99.51
NBOR	99.66	96.61	99.19	99.24	91.67	98.43
BNN-VI	99.47	99.29	99.49	99.56	**99.32**	**99.45**
BNN-HMC	99.12	**99.63**	**99.88**	**99.77**	97.79	98.87
GP	**99.80**	99.61	99.86	99.76	96.16	98.43
BLR	57.85	97.68	97.96	87.16	90.14	56.72

Table 6.1: Empirical accuracy of the state classifiers for each case study. Values are in percentage. For each model, the best result for deterministic classifiers, and the best result for Bayesian classifiers, are highlighted in bold.

rejection rate is relatively high and we consider a large pool of randomly selected points, the procedure may take a long time. The pool of new inputs has indeed to be large in order to have good exploration and find significant instances. As we will show experimentally, our uncertainty-aware active learning approach results in a more precise rejection rule with a lower rejection rate. Therefore, the time spent on offline retraining pays off in improving the online performance of the NPM.

6.5.3 Results under Full Observability

Table 6.1 compares the performances of DNN and BNN against different types, respectively deterministic and Bayesian, of classifiers. In particular, in the deterministic case we compare: a sigmoid-DNN (**DNN-S**) with 3 hidden layers of 10 neurons each, tanh activations for the hidden layers and Sigmoid function for the output layer; a shallow NN (**SNN**) of 20 neurons; a ReLu DNN (**DNN-R**), with 3 hidden layers of 10 neurons each and rectified linear unit (ReLU) activations for all layers; a support vector machine with radial kernel (**SVM**); a random forest classifier (**RF**); and a k-nearest neighbors classifier (**NBOR**). For the Bayesian case, we compare: a Bayesian NN (**BNN**) with 3 hidden layers of 10 neurons each, standard Gaussian priors,

trained with both variational inference (**BNN-VI**) and Hamiltonian Monte Carlo (**BNN-HMC**) methods; a Gaussian Process (**GP**); and a Bayesian Logistic Regression model (**BLR**). In the deterministic framework, differences in accuracy values are relatively small, even though the DNNs outperform the other classifiers in all FO case studies but HC_{fo}. In the Bayesian scenario, we observe that GP and BNN have comparable performances on the simplest models. However, BNN works better as soon as the dimension of the system increases. Furthermore, BNNs offer better scalability than GPs. Indeed, the scalability of GP inference depends heavily on the size of the dataset n, with time complexity of $O(n^3)$, whereas for BNNs with VI this is $O(n \cdot m)$, where m is the number of epochs, and for BNN with HMC the complexity is $O(n \cdot k)$, where k is the number of steps of the Markov chain. In our experiments, $m \ll k$, and k and n have the same order of magnitude. On the other hand, BLR shows limited performances for systems whose dynamics are intrinsically nonlinear. In general, despite the overall difference in performance may seem small, we would like to stress that we target safety-critical applications, for which we seek accuracies as close as possible to 100% and even small improvements become important.

Motivated by the results presented in Table 6.1, we choose the sigmoid DNN architecture described above for our reachability predictions. In particular, the output of the DNN with parameters w in a state $x \in X$, denoted by $f_w(x)$, is the likelihood of class 1, i.e., the likelihood that the hybrid automaton state is positive. Therefore, the discriminant function f_d evaluated at x returns a vector of probabilities $f_d(x) = [1 - f_w(x), f_w(x)]$. To avoid overfitting, we did not tune the architecture (i.e., number of neurons and hidden layers) to optimize the performance for our data and, for the sake of simplicity, we choose the same architecture for all the case studies, as we found no specific DNN architecture with consistently better performances. See Fig. 6.5 for detailed performance analysis for different choices of the DNN architecture. In particular, we use the same architecture for deterministic DNNs and their Bayesian counterpart.

Datasets. For every model, we generate an initial dataset Z' of 20K samples and a test set Z_{test} of 10K samples. The helicopter model is the only exception, where, due to the higher dimensionality, a set Z' of 100K samples is used. Both Z' and Z_{test} are drawn from the same distribution $\mathcal{Z}$; see [98]

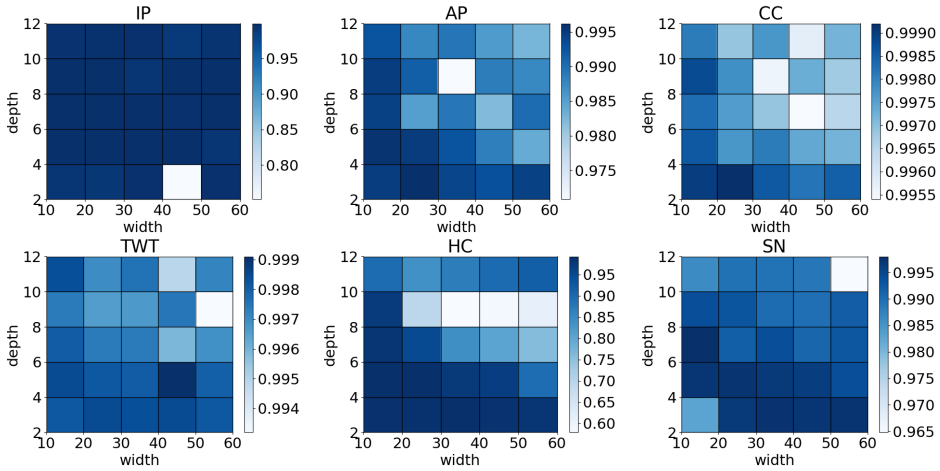


Figure 6.5: Sensitivity analysis: for each model the width and depth of the DNN have been varied. The colormap indicates the accuracy of the predictive monitor.

and the next paragraph for more details on how data are labeled and on the distributions for each case study. The training and validation sets are two subsets of Z' extracted as follows: a sample $z \in Z'$ has probability s of falling into Z_t and probability $1 - s$ of falling into Z_v, where $s = 0.7$ is the splitting ratio. Recall that the calibration set Z_c of the frequentist approach coincides with the validation set Z_v and that we use the same splitting rate s when augmenting the datasets during active learning. The dReal solver [16] is used as a reachability oracle to label the datasets for the non-deterministic case studies. For the deterministic case studies, we used an HA simulator implemented in Matlab.

Labeling data. Labeling a state s of an HA $\mathcal{M}$ means deciding whether $\mathcal{M} \models \mathcal{R}each(D, s, H_f)$, i.e., solving a reachability checking problem. For nondeterministic HAs, we use an SMT solver that supports bounded model checking of hybrid systems. In particular, we choose dReal [16], which provides sound unsatisfiability proofs and approximates satisfiability up to a user-defined precision (δ-satisfiability). So, we label s as negative (positive) if $\mathcal{M} \models \mathcal{R}each(D, s, H_f)$ ($\mathcal{M} \models \neg\mathcal{R}each(D, s, H_f)$) is unsatisfiable. If both $\mathcal{R}each(D, s, H_f)$ and $\neg\mathcal{R}each(D, s, H_f)$ are δ-sat, then the model checker cannot make a decision about s, and in this case, we choose to be conserva-

tive and mark the state as positive. However, choosing a small δ makes this situation less likely to happen. In the case of deterministic systems, it is sufficient to simulate the system with an ODE solver and use an event-detection method to check guard conditions and whether the trajectory reaches D.

During dataset generation, we sample the HA states to label using either a uniform sampling or a balanced sampling strategy. The former ensures that all states in $S \setminus D$ are equiprobable. The latter produces a balanced amount of positive and negative samples and it is used in cases when the unsafe region D is a small subset of the state space, where a uniform sampling strategy would result in imbalanced datasets with insufficient positive samples. See [98] for more details.

Kernel choice. Both error detection rules, based on SVC for the frequentist approach and GPC for the Bayesian approach, are kernel-based methods. The radial basis function (RBF) kernel has been chosen in both cases, as it outperforms the polynomial and linear kernels. The RBF is defined as $k_\gamma(u, u') = \exp(\gamma \|u - u'\|^2)$, where $\|u - u'\|^2$ is the squared Euclidean distance between the two input vectors.

Error type selection. Below we focus on detecting all kinds of prediction errors, including false positives and false negatives. However, it is possible – and this is a very useful feature of our approach – to focus on a specific type of error. For example, in safety-critical applications, one could focus on detecting false negatives, which are the most critical kind of errors. An alternative solution is to learn two distinct rejection rules, one for false positives and one for false negatives, and combine them into a global rejection rule that suits the case study at best.

In addition, in the frequentist case, one can tune the SVC penalty matrix $\mathcal{P}$ (see equation (6.6)) to penalize specific kinds of errors. For instance, setting $r_{fn} > 1$ will result in a detection rule that is stricter on recognizing false negatives. In the Bayesian case, the GPC decision threshold can be tuned by maximizing scores other than $TPR - FPR$.

Tuning of the Bayesian approach. Training the deterministic DNNs was straightforward in our experiments. All models share the same initialization settings and all reach an extremely high accuracy (always higher than 99%, see Table 6.3). On the contrary, training a Bayesian Neural Network requires careful tuning of the inference hyperparameters, e.g. the choice of prior distributions and the sample size used to empirically approximate the predictive distribution. Furthermore, in the HMC framework, the parameters governing the Hamiltonian dynamics affect the capabilities of the Monte Carlo algorithm to explore and eventually converge. In the VI framework, the hyper-parameters by which we maximize the ELBO (3.15) may change from one model to the other. The main drawback is that this may limit the effectiveness of the active learning framework, as explained later in this section.

Offline computational costs. Executing the entire pipeline, i.e., learning a working NPM, when $|Z'| = 20$K, takes around 3 minutes in the frequentist case and around 11 minutes in the Bayesian case. When $|Z'| = 100$K, it takes around 6.5 $(120 - 190)$ minutes in the frequentist (Bayesian) case. The time required to execute 20K VI epochs is comparable with the time required to perform 2K HMC steps (see Table 6.2, bottom-left frame)).

Online computational costs. Given a test input x_*, it takes from 1.4 up to 31 milliseconds to evaluate the NPM, i.e., to make a prediction and choose whether to accept it or not (see Table 6.2, top frame).

In our study, the pool used to refine the query strategy (required only with CP) contains 50K samples, whereas the pool used for the active learning phase contains 100K samples. In particular, one iteration of the active learning procedure takes, for the simplest deterministic models, around 10 minutes in the frequentist scenario, and around 20 minutes in the Bayesian scenario (both VI and HMC approaches). The helicopter model needs a longer time, as it is trained for a higher number of epochs: it takes around 1 hour in the frequentist case against the 10 hours of the Bayesian case. For the non-deterministic models (triple water tank and cruise controller), an active learning iteration takes approximately the same time as a simple deterministic model, except for the overhead introduced in labeling new points (see Table 6.2: bottom-right frame). dReal, the non-deterministic reachability

	Online costs (ms)			**Offline costs** (min)			**AL overhead** (min)		
Model	**CP**	**VI**	**HMC**	**CP**	**VI**	**HMC**	**CP**	**VI**	**HMC**
IP_{fo}	1.4	13.0	8.1	2.0	8.9	14.5	9.6	32.4	27.8
AP_{fo}	2.1	15.0	7.9	2.5	11.6	9.3	10.0	30.4	16.4
CC_{fo}	2.0	14.0	8.2	2.9	12.2	14.1	57.9	95.0	135.5
TWT_{fo}	2.1	14.0	8.1	3.1	12.8	11.0	21.2	74.4	207.6
HC_{fo}	4.0	31.0	14.5	6.5	194.0	121.0	26.0	593.0	917.1
SN_{fo}	1.6	15.0	8.3	4.5	9.9	15.2	11.4	29.9	57.5

Table 6.2: **(Left)** Online computational costs: time to evaluate the NPM, i.e., time to obtain a reachability prediction and decide whether to trust it, on a single state. Time is measured in milliseconds (ms). **(Right)** Offline computational costs: time required to initially train the NPM. AL overhead: time to complete an active learning iteration (on average). Time is measured in minutes.

checker, takes around 1.5 minutes to label 100 observations of the CC model and around 4 minutes to label the same amount of points of the TWT model. In general, the time required for a single active learning iteration is expected to decrease for subsequent iterations, as the rejection rate will be lower (leading to fewer retraining samples). Note that retraining is performed offline and does not affect the runtime performance of our approach.

Benefit of Conformal Predictions. The key advantage of the CP approach is that predictions are rejected on rigorous statistical grounds. We experimentally compare it with a naive approach based on the DNN output. We define the naive uncertainty metric as the difference between the likelihoods of the two classes, that is, $|f_d^0(x) - f_d^1(x)|$, where x is a generic input and $f_d^i(x)$ is the output of the discriminator for class i. Intuitively, small differences should indicate uncertain predictions. Although this simple approach does not provide any statistical guarantee, we may still look for a rejection threshold that allows us to reject the misclassified examples and keep the overall rejection rate low. However, Figure 6.6 (right) shows that this naive metric is not sufficiently discriminative, especially for the spiking neuron model. This supports our claim that a more principled method to measure uncertainty and define rejection criteria is needed. On the contrary, Figure 6.6 (left) shows that the values of confidence-credibility pairs for mis-

classified points are easily separated from the majority of properly classified points. Furthermore, the distribution of points in the confidence-credibility plane helps us choose the proper value for ϵ, which leads to a statistically significant measure of uncertainty.

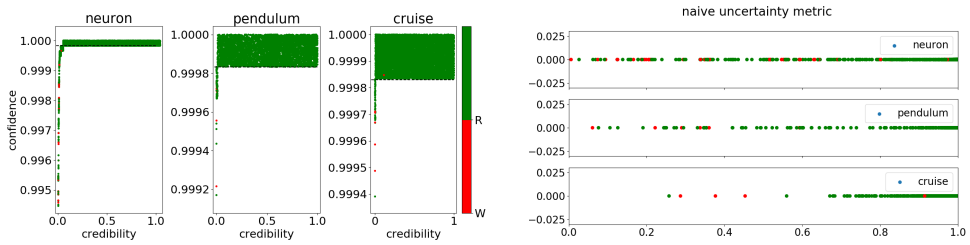


Figure 6.6: Experimental superiority of conformal predictions over naive discrimination based on the DNN class likelihood. **Left**: Confidence-credibility pairs for the test datasets. The horizontal dashed line indicates the empirical and qualitative choice of ϵ. **Right**: Values of the naive uncertainty metric for the test datasets. In both cases (top and bottom) the true test labels were used to check the performances of the uncertainty metrics a posteriori. Green dots indicate properly classified points, red dots misclassified points.

Experiments

We evaluate our approach on three configurations: the *initial* configuration, where the predictor f and error detection rule r_f are derived via supervised learning; the *active* configuration, where the initial models are retrained via our uncertainty-aware active learning; and the *passive* configuration, where the initial models are retrained using a uniform sampling strategy to augment the dataset and with the same number of observations of the active configuration. In this way, we can evaluate the benefits of employing an uncertainty-based criterion to retrain our models.

Table 6.3 presents the experimental performances (on the test set Z_{test}) in the initial configuration. The results are averaged over five runs, where, in each run, we resample Z_t and Z_c from Z' and retrain F. Table 6.4 and 6.5 compare the performances of the three configurations, only for one run in this case.

Model		NSC Acc.	♯ Err.	Det. rate	Rej. rate
$\mathbf{IP_{fo}}$	CP	99.55	45	100.00	5.37
	VI	99.72	28	83.81	1.46
	HMC	99.03	97	96.70	9.68
$\mathbf{AP_{fo}}$	CP	99.64	36	100.00	4.97
	VI	99.30	70	97.17	5.36
	HMC	98.35	165	96.73	9.34
$\mathbf{CC_{fo}}$	CP	99.88	12	100.00	3.63
	VI	99.25	75	91.15	3.67
	HMC	98.16	184	98.04	7.75
$\mathbf{TWT_{fo}}$	CP	99.81	19	100.00	4.40
	VI	99.58	42	74.92	1.06
	HMC	98.13	187	93.30	6.56
$\mathbf{HC_{fo}}$	CP	99.43	57	97.21	6.12
	VI	97.37	263	84.24	13.55
	HMC	97.66	234	91.60	16.03
$\mathbf{SN_{fo}}$	CP	99.79	21	100.00	3.95
	VI	98.47	153	85.76	6.04
	HMC	98.69	131	99.24	21.84

Table 6.3: NSC accuracy, error detection rate and rejection rate in initial configuration. Each block denotes a different case study. CP indicates the frequentist approach, VI and HMC indicate the two inference techniques used in the Bayesian approach.

		NSC Acc.	♯ Err.	Det. rate	Rej. rate
Inverted Pendulum (IP_{fo})					
CP	Initial	99.79	21	100.00	5.24
	Active	99.87	13	100.00	3.12
	Passive	99.79	21	100.00	6.07
VI	Initial	99.58	42	80.95	1.68
	Active	99.58	42	85.71	1.31
	Passive	99.71	29	75.81	1.13
HMC	Initial	87.45	1255	100.00	24.72
	Active	99.15	85	87.06	4.17
	Passive	98.49	151	100.00	13.02
Artificial Pancreas (AP_{fo})					
CP	Initial	99.61	39	100.00	4.17
	Active	99.83	17	100.00	1.65
	Passive	99.62	38	100.00	5.48
VI	Initial	99.29	71	95.77	5.17
	Active	99.71	29	96.55	1.75
	Passive	99.47	53	100.00	3.29
HMC	Initial	98.95	105	98.09	8.32
	Active	99.38	62	90.32	4.17
	Passive	95.12	488	100.00	26.79
Cruise Controller (CC_{fo})					
CP	Initial	99.85	15	100.00	3.46
	Active	99.96	4	100.00	0.51
	Passive	99.88	12	100.00	5.15
VI	Initial	99.01	99	97.98	5.53
	Active	99.84	16	100.00	1.25
	Passive	99.74	26	100.00	1.46
HMC	Initial	97.22	278	99.64	8.41
	Active	99.47	53	94.34	3.14
	Passive	95.75	425	97.03	8.42

Table 6.4: Comparison of initial, active and passive approaches. Results are over a single run. Legend is as in Table 6.3.

		NSC Acc.	♯ Err.	Det. rate	Rej. rate
Triple Water Tank ($\mathbf{TWT_{fo}}$)					
CP	Initial	99.82	18	100.00	5.87
	Active	99.96	4	100.00	0.70
	Passive	99.81	19	100.00	4.43
VI	Initial	99.60	40	77.50	1.11
	Active	99.67	33	84.85	1.61
	Passive	99.50	50	75.40	20.32
HMC	Initial	97.50	250	96.80	5.04
	Active	99.20	80	95.00	3.70
	Passive	91.86	814	52.58	11.31
Helicopter ($\mathbf{HC_{fo}}$)					
CP	Initial	99.21	79	95.95	6.75
	Active	99.49	51	94.12	4.52
	Passive	99.40	60	95.00	5.92
VI	Initial	98.14	186	88.71	13.64
	Active	98.90	110	92.86	1.98
	Passive	98.66	134	87.54	9.94
HMC	Initial	97.74	226	89.82	14.71
	Active	97.74	223	73.01	7.06
	Passive	97.77	223	65.47	6.40
Spiking Neuron ($\mathbf{SN_{fo}}$)					
CP	Initial	99.61	39	100.00	4.17
	Active	99.83	17	100.00	1.65
	Passive	99.62	28	100.00	5.48
VI	Initial	98.18	182	91.21	7.91
	Active	98.20	180	91.11	6.26
	Passive	98.20	180	98.52	14.57
HMC	Initial	98.32	168	74.40	9.89
	Active	98.89	111	87.38	5.91
	Passive	98.21	179	74.86	14.85

Table 6.5: Comparison of initial, active and passive approaches. Results are over a single run. Legend is as in Table 6.3.

Frequentist approach. The average NSC accuracy over the six case studies is 99.68%. The rejection criterion recognizes well almost all the errors, with an average error detection rate of 99.53%, but the overall rejection rate in the initial configuration is around 5%, a non-negligible amount. Table 6.4 and 6.5 show that the passive learning approach provides little improvement: the NSC accuracy is similar to the initial one and the rejection rate is still relatively large. However, the active approach provides a significant improvement: the overall rejection rate and the number of errors made by the NSC falls dramatically while preserving the ability to detect almost all errors (with an error detection rate of 100%, except for the helicopter). In particular, rejection rates span from 3.46% to 6.75% with the initial rejection rule but drop to between 0.51% and 4.52% after active learning, and the average NSC accuracy increases from 99.68% (initial) to 99.82% (active).

Bayesian approach. The predictive distribution is approximated by samples of 100 observations. The BNN priors are chosen to be standard normal distributions. In the initial configuration, the NSC accuracy, averaged over all the case studies, is 98.95% with VI and 98.27 with HMC. The rejection criterion recognizes on average 86.18% of errors with VI and 95.27% with HMC. The overall rejection rate is approx. 5.19% with VI, spanning from 1.06% to 13.55%, and approx. 9.87% with HMC, spanning from 6.56% to 16.03%.

Table 6.4 and 6.5 show that, in the HMC framework, the passive learning approach happens to produce results that are even worse than the initial configuration. The reason might be that that once the training sets are extended with data that may come from a distribution different from $\mathcal{X}$, the set of hyper-parameters chosen to optimally solve the initial problem may become sub-optimal. Indeed, when the hyper-parameters were tuned again, specifically for the passive learning dataset, high performances were reached. For instance, in the TWT model, the initial HMC performance rates (obtained with proper hyper-parameter tuning) are: 97.5% accuracy, 96.8% recognition and 5.04% rejection. Passive learning (without re-tuning the hyper-parameters) causes a significant drop in performance: 91.86% accuracy, 52.58% recognition and 11.31% rejection. However, once the HMC hyper-parameters are tuned specifically for the passive learning dataset, the level of performance gets back to the initial one: 98.59% accuracy, 96.45%

recognition and 3.63% rejection.

On the other hand, active learning still yields improvements: the NSC accuracy rises from 98.27% to 99.30%, and the rejection rates, initially very high, are significantly reduced, which unfortunately causes a slight decrease in the error detection accuracy. The recognition rate falls from 95.27% to 91.68%.

We finally observe that VI outperforms inference via HMC, even though VI is not able to reach recognition rates as high as in the frequentist approach. In particular, on average, the initial VI approach yield an NSC accuracy of 98.95%, a rejection rate of 5.19% and a recognition rate of 86.18%. The passive results, as before, introduce only minor improvements, whereas active learning yields a significant reduction in the rejection rate (from 5.19% to 2.36%), an increase in the NSC accuracy (from 98.95% to 99.32%) and an increase in the overall recognition rate (from 86.175% to 91.85%).

Discussion. A likely reason why the Bayesian approach falls behind the frequentist one is that the former introduces several levels of approximation. Indeed the BNN is trained using either VI or HMC, two approximate inference techniques, resulting in an approximation of the true posterior distribution. Moreover, the resulting uncertainty measures are defined by statistics of said distribution (mean and variance in our case), which introduces an additional error as these measures do not retain full information about the BNN posterior. The latter error propagates as we apply GP classification for error detection, which produces another approximate solution.

However, it is not to say that the Bayesian approach does not work well overall. Indeed, Bayesian NPM is capable of recognizing always at least the 85% of the prediction errors and the accuracy of the predictive monitor is always well above 98%. As future work (see Section 6.7), we intend to explore the performance of the Bayesian solution in settings with noise and partial observability, i.e., settings where the Bayesian approach is expected to provide more robust performances compared to the frequentist one.

Another interesting aspect is that active learning seems to enhance the classifier confidence in its predictions, as demonstrated by an improved detection rate and a sensibly reduced rejection rate. This is the main advantage

of active learning, besides providing a higher state classifier accuracy.

In summary, the main conclusions from our experimental analysis are:

- Our reachability predictors attain high accuracies, consistently above 97.37% (above 99.43% for the frequentist case).
- The frequentist approach overall outperforms the Bayesian ones in all metrics and configurations, followed by VI.
- Error detection rates stay approximately constant after retraining (active or passive). The frequentist approach achieves staggering performance on this metric.
- The benefits of active learning are visible from an overall reduction of the rejection rate and an overall increase in the NSC accuracy.

CP over the error detection rule

Recall that our frequentist error detection rule r_f builds on CP to quantify the reliability of the NSC predictions. In principle, CP can be applied to derive prediction regions with statistical guarantees to any supervised learning model. The SVC r_f for error detection is no exception.

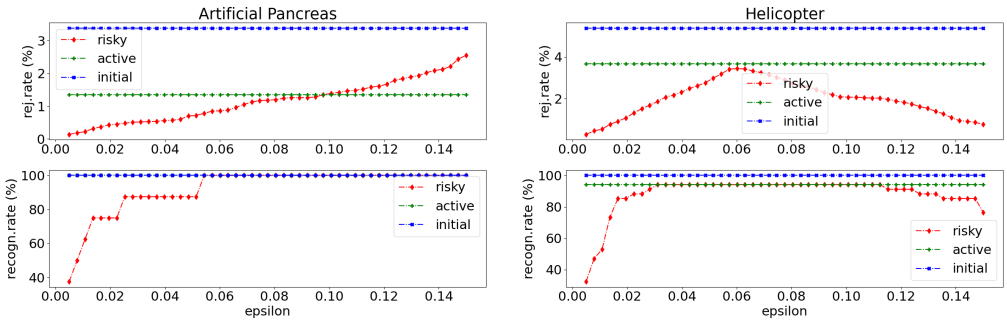


Figure 6.7: Rejection rate and recognition rate of initial rejection rule (initial), the rule obtained after one active learning iteration (active), and the "risky" rule obtained by applying CP to the latter (risk).

In this experiment, we show that we can apply CP to produce prediction regions Γ^ε for r_f, which, by definition, contain the correct rejection decision with probability $1-\varepsilon$. For this purpose, we apply CP to the SVC r_f^a obtained after one active learning iteration. In particular, we derive from Γ^ε a so-called *risky* rejection strategy, aimed at reducing the rejection rate: we reject only those points by which the prediction region for r_f^a contains only class 1, that is, when rejecting is the only plausible decision (according to CP). We report the results for two case studies, the helicopter and the artificial pancreas, to show how the performances of the risky rejection strategy compare to the ones obtained via active learning. These two models are representative of cases where active learning could sensibly reduce the rejection rate (artificial pancreas) and where instead it could not (helicopter).

Note that applying CP to r_f^a requires a new calibration set, i.e., a set of points from W_v^a not used to train r_f^a but rather to calibrate its predictions. The CP framework needs a few further adjustments, presented in Section 3.3.2 (*Mondrian approach*, *NCF for SVC*). Here the input space is the uncertainty domain U.

Choosing an optimal value for ε, i.e., one that yields high detection rates and low rejection rates is non-trivial and requires problem-specific tuning. In Figure 6.7, we compare the above introduced risky strategy against the initial one at different ε levels and after one active learning iteration (results are reported only for the HC and AP case studies). We observe that, with a properly tuned ε, we can achieve the same detection rate of the initial approach (this occurs for $\varepsilon \in [0.057, 0.1]$ in the AP model, and $\varepsilon \in [0.03, 0.11]$ in the HC model), but at the same time a lower rejection rate. For instance, a sweet spot that most reduces the rejection rate without sacrificing detection is $\varepsilon = 0.03$ for the HC and $\varepsilon = 0.057$ for the AP.

6.5.4 Results under Partial Observability

Datasets. For each case study, we generate both an independent and a sequential dataset.

- *Independent:* the train set consists of 50K independent sequences of states of length 32, the respective noisy measurements and the reach-

ability labels. The calibration and test set contains respectively 8.5K and 10K samples.

- *Sequential:* for the train set, 5K states are randomly sampled. From each of these states, we simulate a long trajectory. From each long trajectory, we obtain 100 sub-trajectories of length 32 in a sliding window fashion. The same procedure is applied to the test and calibration set, where the number of initial states is respectively 1K and 850.

Data are scaled to the interval $[-1, 1]$ to avoid sensitivity to different scales. While the chosen datasets are not too large, our approach would work well even with smaller datasets, resulting however in lower accuracy and higher uncertainty. In these cases, our proposed uncertainty-based active learning would represent the go-to solution as is designed for situations where data collection is particularly expensive.

End-to-end solution

We train a one-dimensional convolutional neural net (CNN) that learns a direct mapping from $\mathbf{y}_t$ to l_t, i.e., we solve a simple binary classification problem. This approach ignores the sequence of states $\mathbf{s}_t$. The canonical binary cross-entropy function can be considered as a loss function for the weights optimization process.

Two-step solution

A CNN regressor, referred to as Neural State Estimator (NSE), is trained to reconstruct the sequence of states $\hat{\mathbf{s}}_t$ from the sequence of noisy observations $\mathbf{y}_t$. This is combined with, a CNN classifier, referred to as Neural State Classifier (NSC), trained to predict the reachability label l_t from the sequence of states $\mathbf{s}_t$. The mean square error between the sequences of real states $\mathbf{s}_t$ and the reconstructed ones $\hat{\mathbf{s}}_t$ is a suitable loss function for the NSE, whereas for the NSC we use, once again, a binary cross-entropy function.

The network resulting from the combination of the NSE and the NSC maps the sequence of noisy measurements into the safety label, exactly as

required in Problem 4. However, the NSE inevitably introduces some errors in reconstructing $\mathbf{s}_t$. Such error is then propagated when the NSC is evaluated on the reconstructed state, $\hat{\mathbf{s}}_t$, as it is generated from a distribution different from $\mathcal{S}$, affecting the overall accuracy of the combined network. To alleviate this problem, we introduce a *fine-tuning* phase in which the weights of the NSE and the weights of the NSC are updated together, minimizing the sum of the two respective loss functions. In this phase, the NSC learns to classify correctly the state reconstructed by the NSE, $\hat{\mathbf{s}}_t$, rather than the real state $\mathbf{s}_t$, so to improve the task-specific accuracy.

Neural State Estimation. The two-step approach has an important additional advantage, the NSE. In general, any traditional state estimator could have been used. Nevertheless, non-linear systems make SE extremely challenging for existing approaches. On the contrary, our NSE reaches very high reconstruction precision (as demonstrated in the result section). Furthermore, because of the fine-tuning, it is possible to calibrate the estimates to be more accurate in regions of the state-space that are safety-critical.

Training details. Both approaches to PO-NPM consider sequences of states and observations of fixed length, thus one-dimensional CNNs are indeed a suitable architecture. In particular, the end-to-end classifier and the NSC share the same architecture: four convolutional layers with 128 filters of size 3, with Leaky-ReLU activation functions with slope 0.2 and, for regularization purposes, a drop-out with probability 0.2. The architecture terminates with two dense layers with 100 and 2 nodes respectively. The last layer has a ReLU activation function, to enforce the positivity of the class likelihood scores. On the other hand, the NSE architecture is composed of 5 convolutional layers with 128 filters of size 5, LeakyReLU activations with slope 0.2 and drop-out with probability 0.2. The last layer has the Tanh as activation function, so that the reconstructed states is bounded to the interval $[-1, 1]$. Adam [78] is the algorithm used to optimize every loss. In the end-to-end approach, the learning rate is set to 10^{-5} and it is trained for 200 epochs with batches of size 64. In the two-step approach, the learning rate is set to 10^{-6} when training NSC and NSE separately and to 10^{-7} for the combined fine-tuning phase. The NSE and the NSC are trained for 200 epochs on batches of size 64 and, finally, 100 epochs of fine-tuning are performed.

Offline computational costs. Training the end-to-end approach takes around 15 minutes. Training the two-step approach takes around 40 minutes: 9 for the NSE, 11 for the NSC and 20 minutes for the fine-tuning. Training the SVC takes from 0.5 to 10 seconds. Actively querying new data from a pool of 50K samples takes around 5 minutes.

Online computational costs. Once trained, the time needed to analyse the reachability of the current sequence of observations is the time needed to evaluate one (or two) CNN, which is almost negligible (in the order of microseconds on GPU). Making a single prediction takes around 7×10^{-7} seconds in the end-to-end scenario and 9×10^{-7} seconds in the two-step scenario. Computing values of confidence and credibility for a single point takes from 0.3 to 2 ms.

Results

Initial setting. Table 6.6 compares the performances of the two approaches to PO-NPM via predictive accuracy, detection rate, i.e. the percentage of prediction errors, either false-positives (FP) or false-negatives (FN), recognized by the error detection rule, and the overall rejection over the test set. We can observe how both methods work well despite PO, i.e., they reach extremely high accuracies and high detection rates. However, the two-step approach seems to behave slightly better than the end-to-end. As a matter of fact, accuracy is almost always greater than 99% with a detection rate close to 100.00. The average rejection rate is around 11% in the end-to-end scenario, and reduces to 9% in the two-step scenario, making the latter less conservative and thus more efficient from a computational point of view. These results come with no surprise, because, compared to the end-to-end one, the two-step approach leverages more information available in the dataset for training, that is the exact sequence of states.

Benefits of active learning. Table 6.7 presents the results after one iteration of active learning. Additional data were selected from a pool of 50K points, using the error detection rule as query strategy. We observe a slight improvement in the performance, mainly reflected in higher detection rates

End-to-end					
Model	**Acc.**	**Det.**	**FN**	**FP**	**Rej.**
$\mathbf{SN_{po}}$	97.72	94.30	79/88	136/140	11.30
$\mathbf{IP_{po}}$	96.27	93.48	148/155	153/167	27.32
$\mathbf{CVDP_{po}}$	99.19	100.00	30/30	51/51	5.75
$\mathbf{TWT_{po}}$	98.93	95.51	18/20	67/69	7.45
$\mathbf{LALO_{po}}$	98.88	99.11	66/66	45/46	7.39
$\mathbf{HC_{po}}$	99.63	100.00	19/19	15/15	8.47
Two-step					
Model	**Acc.**	**Det.**	**FN**	**FP**	**Rej.**
$\mathbf{SN_{po}}$	97.12	95.49	53/54	222/234	19.98
$\mathbf{IP_{po}}$	98.42	91.14	81/91	63/66	10.01
$\mathbf{CVDP_{po}}$	99.68	100.00	17/17	15/15	3.51
$\mathbf{TWT_{po}}$	98.93	96.26	52/56	51/51	10.46
$\mathbf{LALO_{po}}$	99.24	100.00	52/52	24/24	6.11
$\mathbf{HC_{po}}$	99.84	100.00	8/8	8/8	4.03

Table 6.6: **Initial results**: *Acc.* is the accuracy of the PO-NPM, *Det.* the detection rate, *Rej.* the rejection rate of the error detection rule and *FN* (*FP*) is the number of detected false negative (positive) errors.

and smaller rejection rates, with an average that reduces to 8% for the end-to-end and to 6% for the two-step.

Probabilisic guarantees. In our experiments, we measured the efficiency as the percentage of singleton prediction regions over the test set. Table 6.8 compares the empirical coverage and the efficiency of the CP prediction regions in the initial and active scenario for both the end-to-end and two-step classifiers. The confidence level is set to $(1 - \epsilon) = 95\%$. Fig. 6.8 shows coverage and efficiency for different significance levels (ranging from 0.01 to 0.1). CP provides theoretical guarantees on the validity, meaning empirical coverage matching the expected one of 95%, only in the initial setting. As a matter of fact, with active learning, we modify the data-generating distribution of the training and calibration sets, while the test set remains the same, i.e., sampled from the original data distribution. As a result, we observe (Table 6.8) that both methods in the initial setting are valid. In the active scenario, even if theoretical guarantees are lost, we obtain both better coverage and higher efficiency. This means that the increased coverage is not

End-to-end					
Model	**Acc.**	**Det.**	**FN**	**FP**	**Rej.**
$\mathbf{SN_{po}}$	98.06	94.87	81/88	104/107	9.80
$\mathbf{IP_{po}}$	99.47	87.91	150/166	119/140	15.44
$\mathbf{CVDP_{po}}$	99.10	95.55	43/46	43/44	4.81
$\mathbf{TWT_{po}}$	99.04	100.00	45/45	62/62	10.45
$\mathbf{LALO_{po}}$	98.79	96.69	87/90	30/31	6.88
$\mathbf{HC_{po}}$	99.86	100.00	5/5	9/9	2.35
Two-step					
Model	**Acc.**	**Det.**	**FN**	**FP**	**Rej.**
$\mathbf{SN_{po}}$	98.41	100.00	55/55	104/104	12.00
$\mathbf{IP_{po}}$	98.75	92.86	63/69	52/56	7.72
$\mathbf{CVDP_{po}}$	99.69	100.00	19/19	12/12	2.48
$\mathbf{TWT_{po}}$	99.07	94.62	44/49	44/44	6.20
$\mathbf{LALO_{po}}$	99.27	100.00	40/40	33/33	4.28
$\mathbf{HC_{po}}$	99.79	100.00	17/17	4/4	2.73

Table 6.7: **Active results** (1 iteration): *Acc.* is the accuracy of the PO-NPM, *Det.* the detection rate, *Rej.* the rejection rate of the error detection rule and *FN* (*FP*) is the number of detected false negative (positive) errors.

due to a more conservative predictor but to improved accuracy.

Table 6.9 shows values of coverage and efficiency for the two separate steps (state estimation and reachability prediction) of the two-step approach. Recall that the efficiency in the case of regression, and thus of state estimation, is given by the volume of the prediction region. So, the smaller the volume, the more efficient the regressor. The opposite holds for classifiers, where a large value of efficiency means tight prediction regions. It is interesting to observe how active learning makes the NSC reach higher coverages at the cost of more conservative prediction regions (lower efficiency), whereas the NSE coverage is largely unaffected by active learning (except for TWT). Reduction in NSC efficiency, differently from the two-step combined approach, is likely due to an adaptation of the method to deal with and correct noisy estimates. Such behaviour suggests that the difficulty in predicting the reachability of a certain state is independent of how hard it is to reconstruct that state[10].

[10]We select re-training points based on the uncertainty of the reachability predictor; if the SE performed badly on those same points, re-training would have led to a higher SE

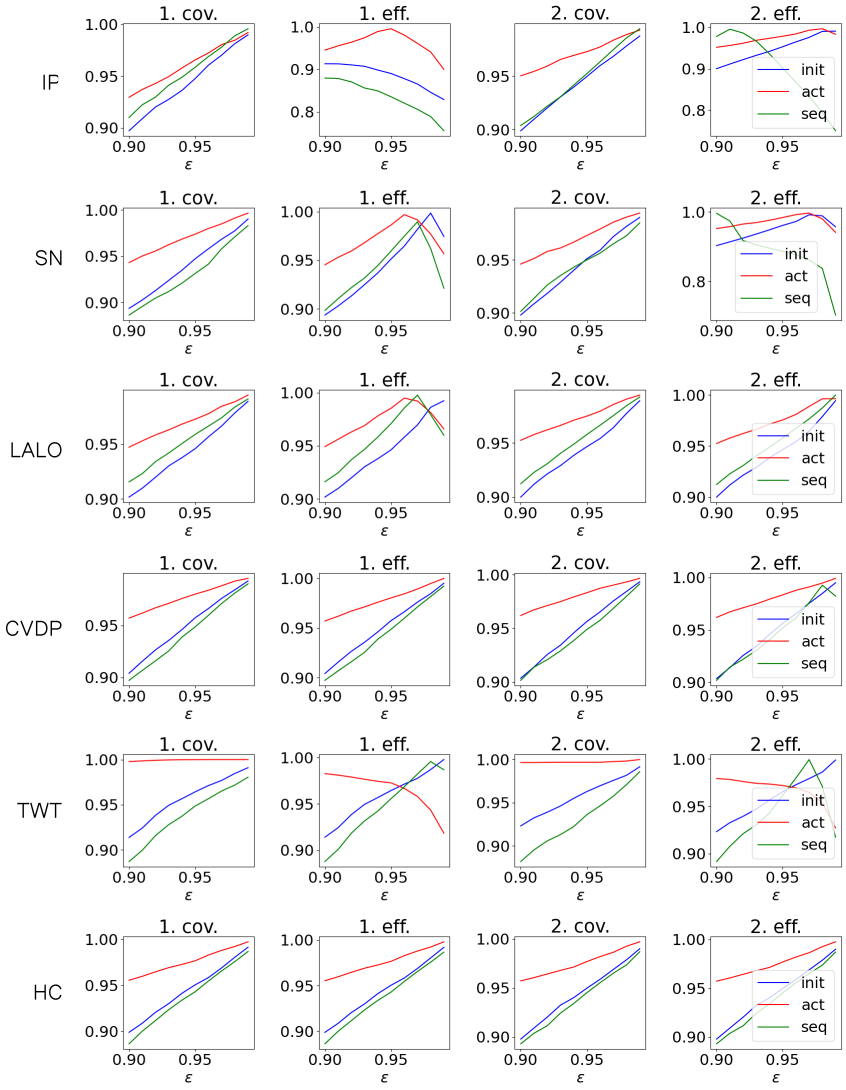


Figure 6.8: Coverage and efficiency of the PO-NSC for the initial, active and sequential configuration. 1. in the title denotes the end-to-end approach, whereas 2. denotes the two-step approach.

accuracy and hence, increased coverage.

	End-to-end				Two-step			
	initial		active		initial		active	
Model	**Cov.**	**Eff.**	**Cov.**	**Eff.**	**Cov.**	**Eff.**	**Cov.**	**Eff.**
$\mathbf{SN_{po}}$	95.12	95.70	97.19	98.50	94.80	99.54	97.32	98.37
$\mathbf{IP_{po}}$	95.30	89.31	96.60	99.62	94.85	94.92	97.28	97.88
$\mathbf{CVDP_{po}}$	95.73	95.73	98.00	98.02	95.63	95.63	98.31	98.34
$\mathbf{TWT_{po}}$	96.43	96.43	99.99	97.26	96.60	96.97	99.66	97.20
$\mathbf{LALO_{po}}$	94.59	94.61	97.28	98.52	94.66	94.66	97.48	97.55
$\mathbf{HC_{po}}$	95.03	95.03	97.65	97.65	94.97	94.97	97.69	97.69

Table 6.8: Coverage and efficiency for both the approaches to PO-NPM. Initial results are compared with results after one active learning iteration. Expected coverage 95%.

State estimator. We compare the performances of the NSE with two traditional state estimation techniques: Unscented Kalman Filters[11] (UKF) [114] and Moving Horizon Estimation[12] (MHE) [115]. In particular, for each point in the test set, we compute the relative error given by the norm of the difference between the real and reconstructed state trajectories divided by the maximum range of state values. The results, shown in Table 6.10 and Fig. 6.9, show how our neural network-based state estimator significantly outperforms both UKF and MHE in our case studies (full results are shown Appendix B). Moreover, unlike the existing SE approaches, our state estimates come with a prediction region that provides probabilistic guarantees on the expected reconstruction error, as shown in Fig. 6.9.

Sequential data. All the results presented so far consider a dataset Z'' of observation sequences generated by independently sampled initial states. However, we are interested in applying NPM at runtime to systems that are evolving in time. States will thus have a temporal correlation, meaning that we lose the exchangeability requirement behind the theoretical validity of CP regions. Table 6.11 shows the performance of predictor and error detection trained and tested on sequential data. In general, accuracy and detection rates are still very high (typically above 95%), but the results are on average worse than the independent counterpart. The motivation could be two-fold:

[11] pykalman library: https://pykalman.github.io/
[12] do-mpc library: https://www.do-mpc.com/en/latest/

	NSC				NSE			
	initial		active		initial		active	
Model	**Cov.**	**Eff.**	**Cov.**	**Eff.**	**Cov.**	**Eff.**	**Cov.**	**Eff.**
$\mathbf{SN_{po}}$	94.82	99.51	97.23	90.12	94.49	1.361	95.18	1.621
$\mathbf{IP_{po}}$	94.51	99.69	97.23	91.63	94.65	3.064	95.44	3.233
$\mathbf{CVDP_{po}}$	95.60	95.64	98.25	98.32	95.37	0.343	96.40	0.358
$\mathbf{TWT_{po}}$	96.68	96.98	98.72	95.61	95.07	0.770	100.00	1.366
$\mathbf{LALO_{po}}$	94.88	98.18	98.01	80.86	95.29	0.6561	95.36	0.8582
$\mathbf{HC_{po}}$	94.67	94.74	97.33	99.12	94.50	12.44	94.58	12.464

Table 6.9: Coverage and efficiency for the two steps of the two-step approach. NSC is a classifier, whereas NSE is a regressor. Initial results are compared with results after one active learning iteration. Expected coverage 95%.

Model	**Neural SE**	**UKH**	**MHE**
$\mathbf{SN_{po}}$	0.0119 ± 0.0233	0.5522 ± 0.5656	0.7139 ± 0.7442
$\mathbf{IP_{po}}$	0.0233 ± 0.0401	0.1987 ± 0.1285	0.1397 ± 0.1344
$\mathbf{CVDP_{po}}$	0.0040 ± 0.0039	0.0210 ± 0.0321	0.0337 ± 0.0763
$\mathbf{TWT_{po}}$	0.0093 ± 0.0097	0.0316 ± 0.0929	1.3285 ± 0.2032
$\mathbf{LALO_{po}}$	0.0081 ± 0.0071	0.0348 ± 0.0709	0.0351 ± 0.1023
$\mathbf{HC_{po}}$	0.0559 ± 0.0605	0.0832 ± 0.1065	0.1217 ± 0.1363

Table 6.10: Comparison of the relative errors (mean and standard deviation over the test set) of the state estimators: the NeuralSE is compared to a Unscented Kalman Filter (UKF) and a Moving Horizon Estimator (MHE).

on one side, it is reasonable to assume that a recurrent neural net would perform better on sequential data, compared to CNN, on the other, the samples contained in the sequential dataset are strongly correlated and thus they may cover only poorly the state space. The table also shows values of coverage and efficiency of both the end-to-end and the two-step approach. Even if theoretical validity is lost, we still observe empirical coverages that match the nominal value of 95%, i.e., the probabilistic guarantees are satisfied in practice.

Anomaly detection. The data-generating distribution at runtime is assumed to coincide with the one used to generate the datasets. However,

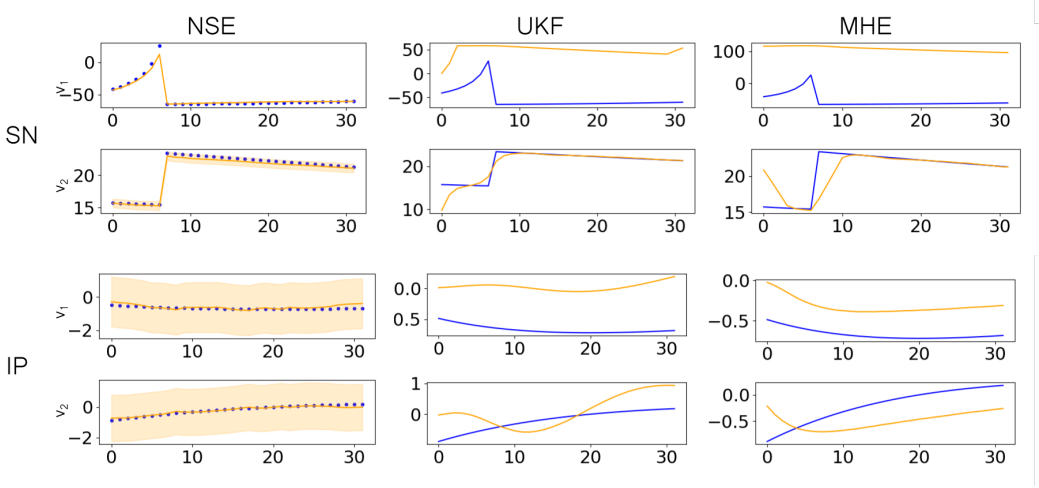


Figure 6.9: Comparison of different state estimators on a state of the SN (top) and IP (bottom) model. Blue is the exact state sequence, orange is the estimated one.

in practice, such distribution is typically unknown and subject to runtime deviations. Thus, we are interested in observing how the sequential PO-NPM behave when an anomaly takes place. In our experiments, we model an anomaly as an increase in the variance of the measurement noise, i.e. $\mathcal{W}' = \mathcal{N}(0, 0.25 \cdot I)$. Fig. 6.10 compares the performances with and without anomaly on each case study. We observe that the anomaly causes a drop in accuracy and error detection rate, which comes with an increase in the number of predictions rejected because deemed to be unreliable. These preliminary results show how an increase in the NPM rejection rate could be used as a significant measure to preemptively detect runtime anomalies.

6.6 Related Work

Several methods have been proposed for online reachability analysis that relies on separating the reachability computation into distinct offline and online phases. However, these methods are either limited to restricted classes of models [38, 116] or require handcrafted optimization of the HA's derivatives [117], or are efficient only for low-dimensional systems and simple dy-

End-to-end					
Model	**Acc.**	**Det.**	**Rej.**	**Cov.**	**Eff.**
$\mathbf{SN_{po}}$	94.96	85.83	19.74	93.93	97.73
$\mathbf{IP_{po}}$	94.17	91.08	31.74	95.31	84.32
$\mathbf{CVDP_{po}}$	98.97	99.12	7.97	94.88	94.92
$\mathbf{TWT_{po}}$	96.95	95.33	16.84	93.42	94.52
$\mathbf{LALO_{po}}$	98.99	97.75	7.18	95.93	97.08
$\mathbf{HC_{po}}$	99.57	100.00	3.89	94.29	94.29
Two-step					
Model	**Acc.**	**Det.**	**Rej.**	**Cov**	**Eff.**
$\mathbf{SN_{po}}$	90.37	81.93	26.59	95.01	88.66
$\mathbf{IP_{po}}$	91.47	98.01	30.81	95.23	90.23
$\mathbf{CVDP_{po}}$	98.33	98.20	9.89	94.89	95.19
$\mathbf{TWT_{po}}$	95.74	92.72	23.52	93.60	96.16
$\mathbf{LALO_{po}}$	99.26	100.00	5.37	95.78	95.80
$\mathbf{HC_{po}}$	99.64	97.22	3.84	94.51	94.52

Table 6.11: **Sequential results**: *Acc.* is the accuracy of the PO-NPM, *Det.* the detection rate, *Rej.* the rejection rate, *Cov.* the CP coverage and *Eff.* the CP efficiency.

namics [118].

In contrast, NSC [98] is based on learning DNN-based classifiers, is fully automated and has negligible computational cost at runtime. In [119, 120], similar techniques are introduced for neural approximation of Hamilton-Jacobi (HJ) reachability. Our methods for prediction rejection and active learning are independent of the class of systems and the machine-learning approximation of reachability, and thus can also be applied to neural approximations of HJ reachability.

In [121], Yel and others present a runtime monitoring framework that has similarities with our NPM approach, in that they also learn neural network-based reachability monitors (for UAV planning applications), but instead of using, as we do, uncertainty measures to pin down potentially erroneous predictions, they apply NN verification techniques [122] to identify input regions that might produce false negatives. Thus, their approach is complementary to our uncertainty-based error detection, but, due to the limitations of the un-

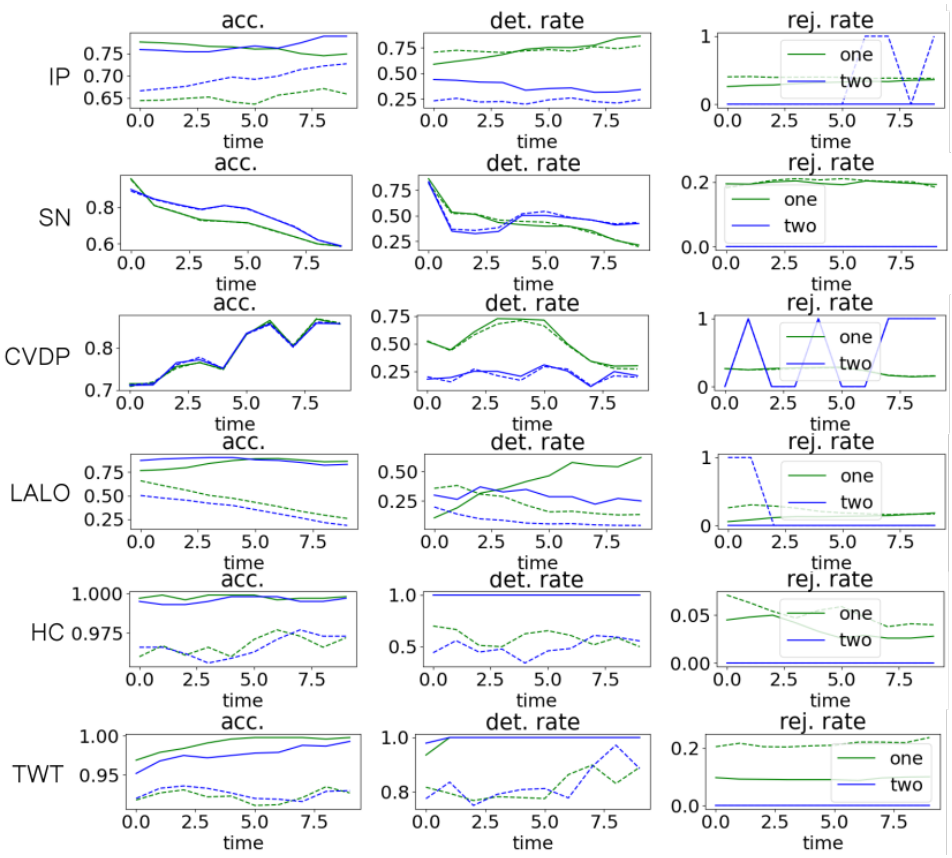


Figure 6.10: **Anomaly detection.** Blue lines denote the performances of the two-step approach. Green lines the end-to-end approach. Dashed lines denote the performances on observations with anomalies in the noise.

derlying verification algorithms, they can only support deterministic neural networks with sigmoid activations. On the contrary, our techniques support any kind of ML-based monitors, including probabilistic ones.

The work of [123, 124] addresses the predictive monitoring problem for stochastic black-box systems, where a Markov model is inferred offline from observed traces and used to construct a predictive runtime monitor for probabilistic reachability checking. In contrast to NPM, this method focuses on discrete-space models, which allows the predictor to be represented as a look-up table, as opposed to a neural network.

In [125], a method is presented for predictive monitoring of STL specifications with probabilistic guarantees. These guarantees derive from computing prediction intervals of ARMA/ARIMA models learned from observed traces. Similarly, we use CP which also can derive prediction intervals with probabilistic guarantees, with the difference that CP supports any class of prediction models (including auto-regressive ones). In [126], model predictions are used to forecast future robustness values of MTL specifications for runtime monitoring. However, no guarantee, statistical or otherwise, is provided for the predicted robustness. Deshmukh and others [127] have proposed an interval semantics for STL over partial traces, where such intervals are guaranteed to include the true STL robustness value for any bounded continuation of the trace. This approach can be used in the context of predictive monitoring but tends to produce over-conservative intervals.

A related approach to NPM is smoothed model checking (smMC) [128], where Gaussian processes [107] are used to approximate the satisfaction function of stochastic models, i.e., mapping model parameters into the satisfaction probability of a specification. Smoothed model checking leverages Bayesian statistics to quantify prediction uncertainty, but faces scalability issues as the dimension of the system increases. In contrast, computing our measure of prediction reliability is very efficient, because it is nearly equivalent to executing the underlying predictor. Similarly, in [129] the authors use conformal inference to approximate the robust satisfaction of a deterministic system with parametric uncertainty, producing a surrogate model with statistical guarantees. The main difference is that [129] solves a regression task over robustness values, whereas our approach, as well as smMC, solves a classification task over Boolean values.

In the field of computer security, a machine learning-based method for malware detection [103] is conceptually very similar to our NPM method. The authors develop a tool for assessing the performance of a classifier using the statistical guarantees of conformal predictions with a self-trained mechanism to filter out unreliable classification decisions.

Literature on uncertainty-based active learning in deep-learning models is small and sparse, mainly because deep learning methods rarely represent model uncertainty, and state-of-the-art deep learning techniques require large amounts of data, which makes active learning impractical. Several uncertainty-based acquisition functions (i.e., functions used to rank the in-

formativeness of new observations for active learning) are reviewed in [51]. In [130], a Deep Ensemble active learning method is proposed, where uncertainty is estimated from a stochastic ensemble of BNN models (obtained via MC-Dropout, another approximate Bayesian inference technique). Some applications of BNN in active learning are presented in [131, 132]. In these works, however, the decision threshold used to identify informative samples is always chosen empirically. An important contribution of our work is the automatic tuning of the decision rule.

A basic application of conformal predictors in active learning is presented in [133]. Our approach introduces three important improvements: a more flexible and meaningful combination of confidence and credibility values, automated learning of rejection thresholds (which are instead fixed in [133]), and refinement of the query strategy.

In [134], we presented a preliminary version of the frequentist approach. In [135] we added to it an automated and optimal method to select the rejection thresholds and an active learning framework.

To our knowledge, the only existing work to focus on PM and PO is [136], which combines Bayesian estimation with pre-computed reach sets to reduce the runtime overhead. While their reachability bounds are certified, no correctness guarantees can be established for the estimation step. Our work instead provides probabilistic guarantees as well as techniques for preemptive error detection. A related but substantially different problem is to verify signals with observation gaps using state estimation to fill the gaps [137, 138].

In [139] a model-based approach to predictive runtime verification is presented. However, PO and computational efficiency are not taken into account. A problem very similar to ours is addressed in [140], but for a different class of systems (MDPs).

Learning-based approaches for reachability prediction of hybrid and stochastic systems include [128, 98, 119, 120, 121, 141]. Of these, [121] develop, akin to our work, error detection techniques, but using neural network verification methods [122]. Such verification methods, however, do not scale well on large models and support only specific classes of neural networks. On the opposite, our uncertainty-based error detection can be applied to any ML-based predictive monitor. Learning-based PM approaches for temporal logic

properties [125] typically learn a time-series model from past observations and then use such model to infer property satisfaction. In particular, [125] provide (like we do) guaranteed prediction intervals, but (unlike our method) they are limited to ARMA/ARIMA models. Ma et al [142] use uncertainty quantification with Bayesian RNNs to provide confidence guarantees. However, these models are, by nature, not well-calibrated (i.e., the model uncertainty does not reflect the observed one [143]), making the resulting guarantees not theoretically valid[13].

PM is at the core of the Simplex architecture [39, 144] and recent extensions thereof [145, 146], where the PM component determines when to switch to the fail-safe controller to prevent imminent safety violations. In this context, our approach can be used to guarantee an arbitrarily small probability of wrongly failing to switch.

6.7 Conclusion

We have presented Neural Predictive Monitoring, an approach for runtime predictive monitoring of hybrid systems that complements reachability predictions with principled estimates of the prediction uncertainty. NPM uses these estimates to derive optimal rejection criteria that identify potentially erroneous predictions without knowing the true reachability values. We have further designed an active learning strategy that, leveraging such uncertainty-based rejection criteria, increases the accuracy of the reachability predictor and reduces the overall rejection rate. Our approach overcomes the computational footprint of reachability checking (infeasible at runtime) while improving on traditional runtime verification by being able to detect future violations in a preemptive way. We have devised two alternative solution methods for NPM. The first one follows a frequentist approach, with state classifiers expressed as deterministic DNNs and rejection rules expressed as Support Vector Classifiers, where the rejection rules are optimized to detect unreliable predictions from uncertainty measures derived via Conformal Prediction. The second one follows a Bayesian approach, with a probabilistic state classifier based on Bayesian Neural Networks, rejection rules given

[13]The authors develop a solution for Bayesian RNNs calibration, but the solution, in turn, is not guaranteed to produce well-calibrated models.

as Gaussian Process Classifiers, and uncertainty measures extracted from statistics of the BNN predictive distribution.

The strengths of our NPM technique are its effectiveness in identifying and rejecting prediction errors and its computational efficiency: executing the classifier and the rule take on the order of milliseconds. NPM's efficiency is not directly affected by the complexity of the system under analysis but only by the complexity of the underlying learning problem and classifier.

Our experimental evaluation demonstrates that the frequentist approach outperforms the Bayesian one: the state classifier is simpler to train and faster to evaluate, and the error detection criteria are more accurate. Regarding BNN inference, we found that VI scales better than HMC with respect to the dimension of the system. The assumptions on the prior and the necessary tuning of hyperparameters represent important drawbacks of the Bayesian techniques, which, however, tend to be more consistent than deterministic models in regions with no data observed: here the posterior distribution will typically have high variance.

We also presented an extended framework that works under the most realistic settings of noise and partially observability. We proposed two alternative solution strategies: an end-to-end solution, predicting reachability directly from raw observations, and a two-step solution, with an intermediate state estimation step. Both methods produce extremely accurate predictions, with the two-step approach performing better overall than the end-to-end version, and further providing accurate reconstructions of the true state. The online computational cost is negligible, making this method suitable for runtime applications. The method is equipped with an error detection rule to prevent reachability prediction errors, as well as with prediction regions providing probabilistic guarantees.

We demonstrated that error detection can be meaningfully used for active learning, thereby improving our models on the most uncertain inputs.

As future work, we intend to analyze the performance of the Bayesian approach under partial and noisy observations and compare it with the frequentist approach presented in this thesis. Moreover, we plan to extend the NPM approach to fully stochastic models, investigating the use of deep generative models for state estimation. We will further explore the use of

recurrent or attention-based architectures in place of convolutional ones to improve performance for sequential data.

Chapter 7

Conclusions

7.1 Concluding Remarks

In this thesis, we presented efficient tools to synthesize and monitor extremely complex cyber-physical systems. In particular, we developed a tool providing fast simulation and, possibly, an automatic and data-driven abstraction of stochastic CPS dynamics (Chapter 4). We developed a tool to synthesize a controller that robustly enforces safety over highly uncertain environments (Chapter 5). Finally, we developed a tool to efficiently monitor, in a preemptive fashion, the runtime evolution of a complex CPS (Chapter 6).

To summarize, the thesis is divided into two parts: the first part (Chapter 2 and 3) introduces the background material and the second part (Chapter 4, 5 and 6) presents the research contributions. These latter can be summarized as follows.

Chapter 4 presents a technique to speed up the simulation process of stochastic CRNs. The goal is to alleviate the scalability issues present in complex and challenging simulation frameworks. The rationale is to use a conditional Wasserstein GAN (cWGAN) to learn an abstract model from a pool of simulated trajectories. The cWGAN learns to generalize over the entire initial state space, so that, for each new initial state, the cWGAN imitated the behaviour of the trajectories present in the training set. Our experimental results show high reconstruction accuracy and a great improvement

in the computational efficiency compared to traditional stochastic simulation algorithms. The main reason is that the computational cost of cWGAN simulation is no more related to the complexity of the underlying CRN.

Chapter 5 presents an adversarial learning technique, whose training architecture is inspired by GAN, to train a CPS controller to stay robustly safe against complex real-world scenarios. The possibly adversarial environment is represented as a deep neural network and the controller is itself a neural network that learns to protect itself from the attacks of the environment. The technique is tested on a vehicle platooning and on a cart-pole problem with additional stochastic components. In both cases, our approach has been able to enforce the safety of the model, while also gaining insights into possible adverse configurations of the environment.

Chapter 6 presents an approximate solution for making the predictive monitoring of hybrid systems applicable at runtime. The reachability predictions are enriched with principled estimates of the prediction uncertainty that are then used to automatically identify potentially erroneous predictions. These latter are recognized as untrustworthy and thus rejected. This uncertainty-based rejection criterion can be used as a query strategy for an active learning phase that could increase the accuracy of the reachability predictor and reduce the overall rejection rate. Two alternative solution methods for NPM are devised: a full frequentist approach and a fully Bayesian one. We also presented an extended framework that works under the most realistic settings of noise and partially observability. Our experimental evaluation shows how NPM produces extremely accurate predictions and how NPM is suitable for runtime applications as the online computational cost is negligible as the computational efficiency of NPM is not directly affected by the complexity of the system under analysis but only by the complexity of the underlying classifier. NPM predictions are also equipped with an error detection rule to prevent reachability prediction errors, as well as with prediction regions providing probabilistic guarantees.

The main idea for the future is to combine these three ingredients so that we can analyse the behaviour of extremely complex systems. Consider, for instance, a CPS with dynamics so complex to become uninterpretable and thus unknown. The model abstraction tool can be used to learn a representation of such a system from a pool of observed realizations. The accurate abstraction could then be used to efficiently simulate new, previously unobserved,

trajectories. Once the dynamics of the system is known, we can synthesize a controller that manipulates some controllable variables of the system, so that the system can meet a desired behavioural requirement. The controller would be robust to uncertainties, meaning that it takes into account some of the inevitable modeling uncertainty coming from the abstraction procedure. Finally, the hybrid automata coming from the combination of the abstract model and of the controller can be analysed efficiently using the neural predictive monitoring approach. By doing so, we can check that the running system is actually moving in a safe region.

However, every tool provides an approximate solution that introduces inevitable errors in each step. Moreover, these errors will propagate along the combined system. For these reasons, it is extremely important to leverage the uncertainty quantification techniques presented in Section 3.3. By doing so, we can reduce the number of errors, by actively refining the synthetic training sets, we can detect untrustworthy solutions and can provide statistical guarantees over the performances of the combined approximate solution.

Bibliography

[1] Francesca Cairoli, Ginevra Carbone, and Luca Bortolussi. Abstraction of markov population dynamics via generative adversarial nets. *CoRR*, abs/2106.12981, 2021.

[2] Luca Bortolussi, Francesca Cairoli, Ginevra Carbone, Francesco Franchina, and Enrico Regolin. Adversarial learning of robust and safe controllers for cyber-physical systems. *IFAC-PapersOnLine*, 54(5):223–228, 2021.

[3] Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, Scott A Smolka, and Scott D Stoller. Neural predictive monitoring and a comparison of frequentist and bayesian approaches. *International Journal on Software Tools for Technology Transfer*, 23(4):615–640, 2021.

[4] Francesca Cairoli, Luca Bortolussi, and Nicola Paoletti. Neural predictive monitoring under partial observability. In *International Conference on Runtime Verification*, pages 121–141. Springer, 2021.

[5] Luiz Eduardo G Martins and Tony Gorschek. Requirements engineering for safety-critical systems: overview and challenges. *IEEE software*, 34(4):49–57, 2017.

[6] Stacy Nelson. Certification processes for safety-critical and mission-critical aerospace software. Technical report, 2003.

[7] Andrew Kornecki and Janusz Zalewski. Certification of software for real-time safety-critical systems: state of the art. *Innovations in Systems and Software Engineering*, 5(2):149–161, 2009.

[8] Fiona C Saunders, Andrew W Gale, and Andrew H Sherry. Conceptualising uncertainty in safety-critical projects: A practitioner perspective. *International Journal of Project Management*, 33(2):467–478, 2015.

[9] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[10] Aws Albarghouthi et al. Introduction to neural network verification. *Foundations and Trends® in Programming Languages*, 7(1–2):1–157, 2021.

[11] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[12] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[13] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, and Alois Knoll. A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*, 2022.

[14] Steve Howes, Ivan Mohler, and Nenad Bolf. Multivariable identification and pid/apc optimization for real plant application. In *ACHEMA*, 2018.

[15] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of computer and system sciences*, 57(1):94–124, 1998.

[16] Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International conference on automated deduction*, pages 208–214. Springer, 2013.

[17] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[18] J. R Norris. *Markov chains*. Cambridge University Press, Cambridge, UK; New York, 1998.

[19] Daniel T. Gillespie and L. Petzold. Numerical simulation for biochemical kinetics. *Systems Modelling in Cellular Biology*, pages 331–354, 2006.

[20] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[21] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *FORMATS*. Springer Berlin Heidelberg, 2010.

[22] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for stl. In *International Conference on Computer Aided Verification*, pages 264–279. Springer, 2013.

[23] Jan Lunze and Françoise Lamnabhi-Lagarrigue. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009.

[24] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Falsification of ltl safety properties in hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 368–382. Springer, 2009.

[25] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancic, Aarti Gupta, and George Pappas. Monte-carlo techniques for the falsification of temporal properties of non-linear systems. *Hybrid Systems: Computation and Control*, pages 211–220, 2010.

[26] Masaki Waga. Falsification of cyber-physical systems with robustness-guided black-box checking. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–13, 2020.

[27] K Larsen, P Pettersson, and W Yi. Uppaal. *Uppsala University, Sweden and Aalborg University, Denmark.[Online]. Available: http://www. uppaal. com*, 1997.

[28] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[29] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*, pages 258–273. Springer, 2005.

[30] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

[31] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, and Stefan Kowalewski. H y p ro: A c++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 288–294. Springer, 2017.

[32] Luca Benvenuti, Davide Bresolin, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Emanuele Mazzi, Alberto Sangiovanni-Vincentelli, and Tiziano Villa. Reachability computation for hybrid systems with ariadne. *IFAC Proceedings Volumes*, 41(2):8960–8965, 2008.

[33] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. JuliaReach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.

[34] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, pages 167–170. Springer, 2010.

[35] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms*

for the Construction and Analysis of Systems, pages 254–257. Springer, 2011.

[36] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2e2: A verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 68–82. Springer, 2015.

[37] Stanley Bak and Parasara Sridhar Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 173–178, 2017.

[38] Xin Chen and Sriram Sankaranarayanan. Model predictive real-time monitoring of linear systems. In *Real-Time Systems Symposium (RTSS), 2017 IEEE*, pages 297–306. IEEE, 2017.

[39] Lui Sha. Using simplicity to control complexity. *IEEE Software*, 18(4):20–28, 2001.

[40] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*, pages 135–175. Springer, 2018.

[41] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Massachusetts, USA:, 2017.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[44] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[45] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[47] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[48] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[49] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[50] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[51] Yarin Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016.

[52] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[53] Aad W Van der Vaart. *Asymptotic statistics*, volume 3. Cambridge university press, 2000.

[54] Dieter Rasch, Jurgen Pilz, LR Verdooren, and Albrecht Gebhardt. *Optimal experimental design with R*. Chapman and Hall/CRC, 2011.

[55] Pascal Massart. The tight constant in the dvoretzky-kiefer-wolfowitz inequality. *The annals of Probability*, pages 1269–1283, 1990.

[56] Radford M Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[57] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[58] Giacomo Deodato, Christopher Ball, and Xian Zhang. Bayesian neural networks for cellular image classification and uncertainty analysis. *bioRxiv*, page 824862, 2019.

[59] Vineeth Balasubramanian, Shen-Shyang Ho, and Vladimir Vovk. *Conformal prediction for reliable machine learning: theory, adaptations and applications*. Newnes, 2014.

[60] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

[61] Harris Papadopoulos. Inductive conformal prediction: Theory and application to neural networks. In *Tools in artificial intelligence*. InTech, 2008.

[62] Paolo Toccaceli and Alexander Gammerman. Combination of inductive mondrian conformal predictors. *Machine Learning*, 108(3):489–510, 2019.

[63] Alexander Gammerman and Vladimir Vovk. Hedging predictions in machine learning. *The Computer Journal*, 50(2):151–163, 2007.

[64] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(Mar):371–421, 2008.

[65] Vineeth Nallure Balasubramanian, R Gouripeddi, Sethuraman Panchanathan, J Vermillion, A Bhaskaran, and RM Siegel. Support vector machine based conformal predictors for risk of complications following a coronary drug eluting stent procedure. In *2009 36th Annual Computers in Cardiology Conference (CinC)*, pages 5–8. IEEE, 2009.

[66] Yaniv Romano, Evan Patterson, and Emmanuel J Candès. Conformalized quantile regression. *arXiv preprint arXiv:1905.03222*, 2019.

[67] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

[68] Yang Cao, Daniel T Gillespie, and Linda R Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of chemical physics*, 124(4):044109, 2006.

[69] Jürgen Pahle. Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. *Briefings in bioinformatics*, 10(1):53–64, 2009.

[70] Luca Bortolussi and Luca Palmieri. Deep abstractions of chemical reaction networks. In *International Conference on Computational Methods in Systems Biology*, pages 21–38. Springer, 2018.

[71] Tatjana Petrov and Denis Repin. Automated deep abstractions for stochastic chemical reaction networks. *arXiv preprint arXiv:2002.01889*, 2020.

[72] Luca Bortolussi and Francesca Cairoli. Bayesian abstraction of markov population models. In *International Conference on Quantitative Evaluation of Systems*, pages 259–276. Springer, 2019.

[73] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[74] Timo R Maarleveld, Brett G Olivier, and Frank J Bruggeman. Stochpy: a comprehensive, user-friendly tool for simulating stochastic biological processes. *PloS one*, 8(11):e79345, 2013.

[75] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[76] Konstantinos Koutroumbas and Sergios Theodoridis. *Pattern recognition*. Academic Press, 2008.

[77] Phillip Pope, Chen Zhu, Ahmed Abdelkader, Micah Goldblum, and Tom Goldstein. The intrinsic dimension of images and its impact on learning. *arXiv preprint arXiv:2104.08894*, 2021.

[78] Yoshua Bengio and MONTREAL CA. Rmsprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390*, 2015.

[79] Gábor J Székely and Maria L Rizzo. Energy statistics: A class of statistics based on distances. *Journal of statistical planning and inference*, 143(8):1249–1272, 2013.

[80] Håkan LS Younes and Reid G Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368–1409, 2006.

[81] Volodymyr et al. Mnih. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[82] Guy Avni, Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, Bettina Könighofer, and Stefan Pranger. Run-time optimization for learned controllers through quantitative games. In *CAV 2019*, pages 630–649. Springer, 2019.

[83] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *ICML*, pages 2817–2826. PMLR, 2017.

[84] Anand Balakrishnan and Jyotirmoy V Deshmukh. Structured reward shaping using signal temporal logic specifications. In *2019 IEEE/RSJ IROS*, pages 3481–3486. IEEE, 2019.

[85] Alper Kamil Bozkurt, Yu Wang, Michael Zavlanos, and Miroslav Pajic. Model-free reinforcement learning for stochastic games with linear temporal logic objectives. *arXiv preprint arXiv:2010.01050*, 2020.

[86] Wenliang Liu, Noushin Mehdipour, and Calin Belta. Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints. *IEEE Control Systems Letters*, 2021.

[87] Xiao Li, Cristian Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. *CoRR*, abs/1612.03471, 2016.

[88] Razvan V Florian. Correct equations for the dynamics of the cart-pole system. *Center for Cognitive and Neural Studies (Coneural), Romania*, 2007.

[89] Carlos Aguilar-Ibáñez, Julio Mendoza-Mendoza, and Jorge Dávila. Stabilization of the cart pole system: by sliding mode control. *Nonlinear Dynamics*, 78(4):2769–2777, 2014.

[90] Yang Liu, Hongnian Yu, Sam Wane, and Taicheng Yang. On tracking control of a pendulum-driven cart-pole underactuated system. *IJMIC*, 4(4):357–372, 2008.

[91] Savinay Nagendra, Nikhil Podila, Rashmi Ugarakhod, and Koshy George. Comparison of reinforcement learning algorithms applied to the cart-pole problem. In *ICACCI 2017*, pages 26–32. IEEE, 2017.

[92] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[93] Jia-Jun Wang. Simulation studies of inverted pendulum based on pid controllers. *Simulation Modelling Practice and Theory*, 19:440–449, 01 2011.

[94] Christopher Edwards and Sarah Spurgeon. *Sliding mode control: theory and applications*. Crc Press, 1998.

[95] Lejla Banjanovic-Mehmedovic, Ivana Butigan, Fahrudin Mehmedovic, and Mehmed Kantardzic. Hybrid automaton based vehicle platoon modelling and cooperation behaviour profile prediction. *Tehnicki vjesnik - Technical Gazette*, 25(3), Jun 2018.

[96] Dongyao Jia, Kejie Lu, Jianping Wang, Xiang Zhang, and Xuemin Shen. A survey on platoon-based vehicular cyber-physical systems. *IEEE Communications Surveys & Tutorials*, 18(1):263–284, 2016.

[97] Massimo Zambelli and Antonella Ferrara. Robustified distributed model predictive control for coherence and energy efficiency-aware platooning. In *ACC 2019*. IEEE, 2019.

[98] Dung Phan, Nicola Paoletti, Timothy Zhang, Radu Grosu, Scott A Smolka, and Scott D Stoller. Neural state classification for hybrid systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 422–440. Springer, 2018.

[99] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[100] Matthias Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[101] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[102] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[103] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, 2017.

[104] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

[105] Ulf Brefeld, Peter Geibel, and Fritz Wysotzki. Support vector machines with example dependent costs. In *European Conference on Machine Learning*, pages 23–34. Springer, 2003.

[106] Rukshan Batuwita and Vasile Palade. *Class Imbalance Learning Methods for Support Vector Machines*, chapter 5, pages 83–99. John Wiley & Sons, Ltd, 2013.

[107] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[108] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6), 2017.

[109] Nicola Paoletti, Kin Sum Liu, Scott A Smolka, and Shan Lin. Data-driven robust control for type 1 diabetes under meal and exercise uncertainties. In *International Conference on Computational Methods in Systems Biology*, pages 214–232. Springer, 2017.

[110] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Alexandre Donze, Georgios Fainekos, Goran Frehse, Logan Mathesen, Claudio Menghi, Giulia Pedrinelli, Marc Pouzet, et al. Arch-comp 2020 category report: Falsification. *EPiC Series in Computing*, 2020.

[111] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[112] Dustin Tran, Alp Kucukelbir, Adji B Dieng, Maja Rudolph, Dawen Liang, and David M Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

[113] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

[114] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[115] Douglas A Allan and James B Rawlings. Moving horizon estimation. In *Handbook of Model Predictive Control*, pages 99–124. Springer, 2019.

[116] Hansol Yoon, Yi Chou, Xin Chen, Eric Frew, and Sriram Sankaranarayanan. Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for uavs. In *International Conference on Runtime Verification*, pages 349–367. Springer, 2019.

[117] Stanley Bak, Taylor T Johnson, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 138–148. IEEE, 2014.

[118] Gerald Sauter, Henning Dierks, Martin Fränzle, and Michael R Hansen. Lightweight hybrid model checking facilitating online prediction of tem-

poral properties. In *Proceedings of the 21st Nordic Workshop on Programming Theory*, pages 20–22, 2009.

[119] Badis Djeridane and John Lygeros. Neural approximation of PDE solutions: An application to reachability computations. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 3034–3039. IEEE, 2006.

[120] Vicenc Rubies Royo, David Fridovich-Keil, Sylvia Herbert, and Claire J Tomlin. Classification-based approximate reachability with guarantees applied to safe trajectory tracking. *arXiv preprint arXiv:1803.03237*, 2018.

[121] E. Yel, T. J. Carpenter, C. Di Franco, R. Ivanov, Y. Kantaros, I. Lee, J. Weimer, and N. Bezzo. Assured runtime monitoring and planning: Toward verification of neural networks for safe autonomous operations. *IEEE Robotics & Automation Magazine*, 27(2):102–116, 2020.

[122] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

[123] Reza Babaee, Arie Gurfinkel, and Sebastian Fischmeister. Predictive run-time verification of discrete-time reachability properties in black-box systems using trace-level abstraction and statistical learning. In *International Conference on Runtime Verification*, pages 187–204. Springer, 2018.

[124] Reza Babaee, Vijay Ganesh, and Sean Sedwards. Accelerated learning of predictive runtime monitors for rare failure. In *International Conference on Runtime Verification*, pages 111–128. Springer, 2019.

[125] Xin Qin and Jyotirmoy V Deshmukh. Predictive monitoring for signal temporal logic with probabilistic guarantees. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 266–267. ACM, 2019.

[126] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. On-line monitoring for temporal logic robustness. In *International Conference on Runtime Verification*, pages 231–246. Springer, 2014.

[127] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30, 2017.

[128] Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. Smoothed model checking for uncertain continuous-time Markov chains. *Information and Computation*, 247:235–253, 2016.

[129] Chuchu Fan, Xin Qin, Yuan Xia, Aditya Zutshi, and Jyotirmoy Deshmukh. Statistical verification of autonomous systems using surrogate models and conformal inference. *arXiv preprint arXiv:2004.00279*, 2020.

[130] Remus Pop and Patric Fulop. Deep ensemble bayesian active learning: Addressing the mode collapse issue in monte carlo dropout via ensembles. *arXiv preprint arXiv:1811.03897*, 2018.

[131] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.

[132] Feras Dayoub, Niko Sunderhauf, and Peter I Corke. Episode-based active learning with bayesian neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 26–28, 2017.

[133] Lázaro Emílio Makili, Jesús A Vega Sánchez, and Sebastián Dormido-Canto. Active learning using conformal predictors: application to image classification. *Fusion Science and Technology*, 62(2):347–355, 2012.

[134] Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, and Scott D Stoller. Conformal predictions for hybrid system state classification. In *From Reactive Systems to Cyber-Physical Systems*, pages 225–241. Springer, 2019.

[135] Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, Scott A Smolka, and Scott D Stoller. Neural predictive monitoring. In *International Conference on Runtime Verification*, pages 129–147. Springer, 2019.

[136] Yi Chou, Hansol Yoon, and Sriram Sankaranarayanan. Predictive runtime monitoring of vehicle models using bayesian estimation and reachability analysis. In *Intl. Conference on Intelligent Robots and Systems (IROS)*, 2020.

[137] Scott D Stoller, Ezio Bartocci, Justin Seyster, Radu Grosu, Klaus Havelund, Scott A Smolka, and Erez Zadok. Runtime verification with state estimation. In *International conference on runtime verification*, pages 193–207. Springer, 2011.

[138] Kenan Kalajdzic, Ezio Bartocci, Scott A Smolka, Scott D Stoller, and Radu Grosu. Runtime verification with particle filtering. In *International Conference on Runtime Verification*, pages 149–166. Springer, 2013.

[139] Srinivas Pinisetty, Thierry Jéron, Stavros Tripakis, Yliès Falcone, Hervé Marchand, and Viorel Preoteasa. Predictive runtime verification of timed properties. *Journal of Systems and Software*, 132:353–365, 2017.

[140] Sebastian Junges, Hazem Torfah, and Sanjit A Seshia. Runtime monitors for markov decision processes. In *International Conference on Computer Aided Verification*, pages 553–576. Springer, 2021.

[141] Wolfgang Granig, Stefan Jakšić, Horst Lewitschnig, Cristinel Mateis, and Dejan Ničković. Weakness monitors for fail-aware systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 283–299. Springer, 2020.

[142] Meiyi Ma, John A. Stankovic, Ezio Bartocci, and Lu Feng. Predictive monitoring with logic-calibrated uncertainty for cyber-physical systems. *CoRR*, abs/2011.00384v2, 2020.

[143] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In *International Conference on Machine Learning*, pages 2796–2804. PMLR, 2018.

[144] Taylor T Johnson, Stanley Bak, Marco Caccamo, and Lui Sha. Real-time reachability for verified simplex design. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(2):1–27, 2016.

[145] Dung T Phan, Radu Grosu, Nils Jansen, Nicola Paoletti, Scott A Smolka, and Scott D Stoller. Neural simplex architecture. In *NASA Formal Methods Symposium*, pages 97–114. Springer, 2020.

[146] Usama Mehmood, Scott D Stoller, Radu Grosu, Shouvik Roy, Amol Damare, and Scott A Smolka. A distributed simplex architecture for multi-agent systems. *arXiv preprint arXiv:2012.10153*, 2020.

Appendices

Appendix A

Model Abstraction

The figures below show quantifications of the reconstruction accuracy of the learned abstraction for the two-dimensional models (eSIRS, Toggle Switch and MAPK). Five different measures of error are considered: the Wasserstein distance among the two one-dimensional distributions, the absolute and relative difference among the two means and the absolute and relative difference among the two variances. For each model, the landscapes show these five different quantities at three different time steps: step t_1, step $t_{H/2}$ and step t_H. In particular, Fig. A.1-A.3 show the landscapes for the eSIRS model, Fig. A.4-A.6 for the Toggle Switch model and Fig. A.7 for the MAPK model.

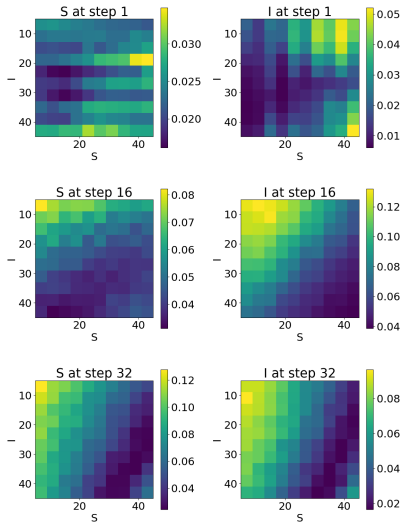
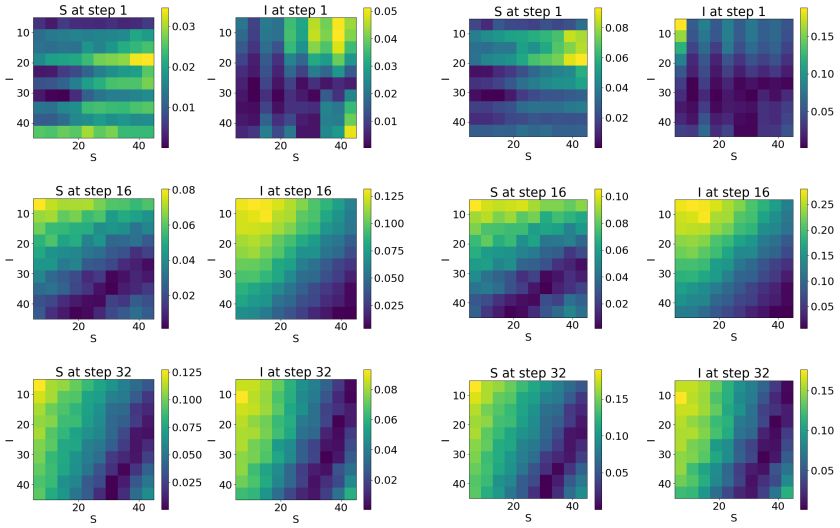
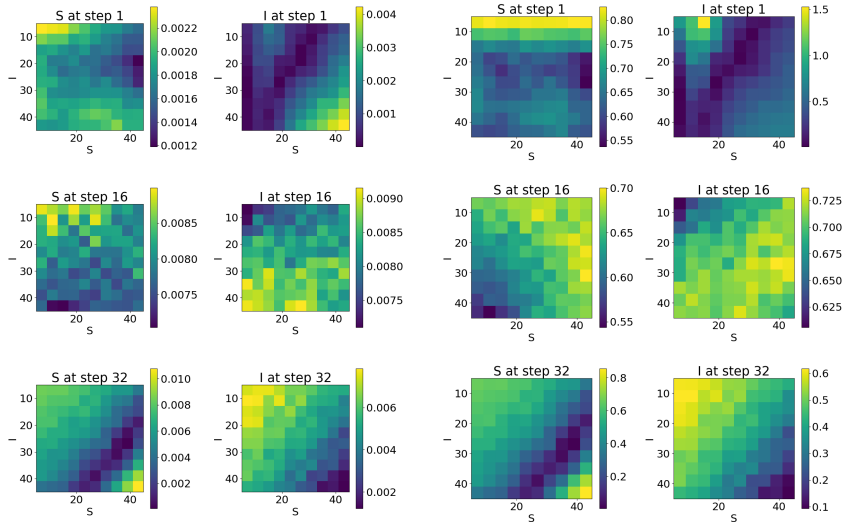


Figure A.1: Wasserstein distance landscapes for the two-dimensional **eSIRS** model.



(a) Means abs. err. (b) Means rel. err.

Figure A.2: Landscapes of the mean distance, absolute (a) and relative (b) for the two-dimensional **eSIRS** model.



(a) Variances abs. err. (b) Variances rel. err.

Figure A.3: Landscapes of the variance distance, absolute (a) and relative (b) for the two-dimensional **eSIRS** model.

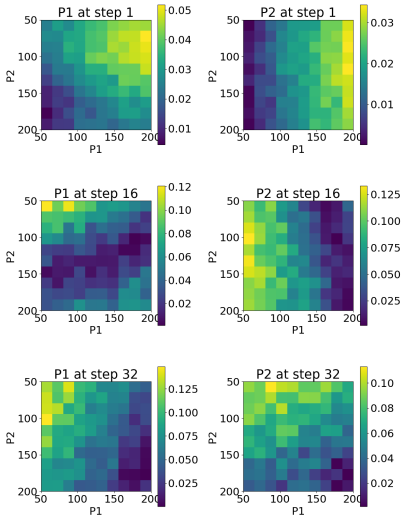
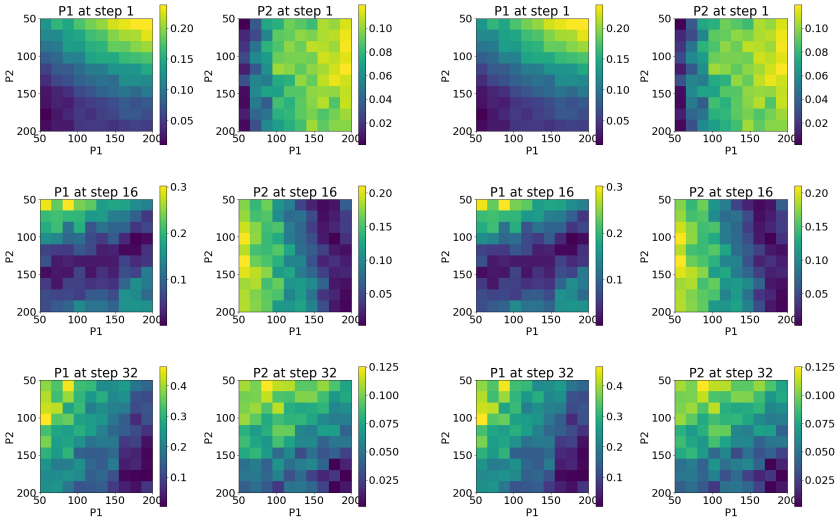
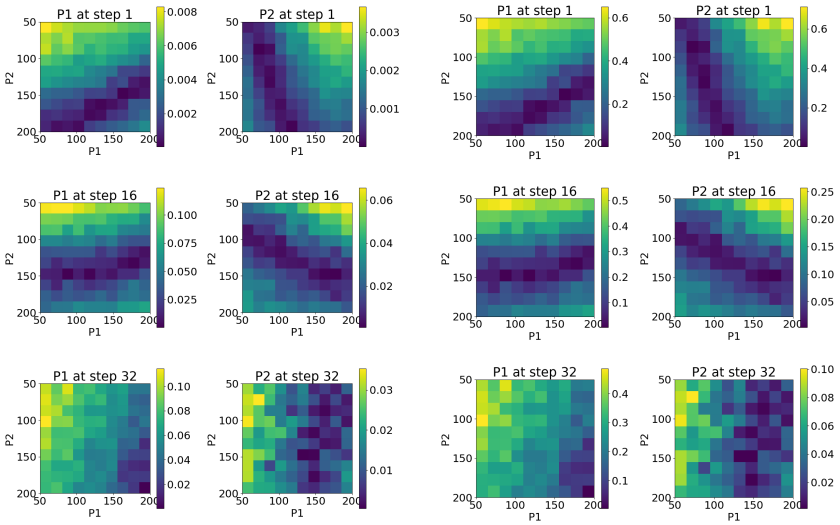


Figure A.4: Wasserstein distance landscapes for the two-dimensional **Toggle Switch** model.



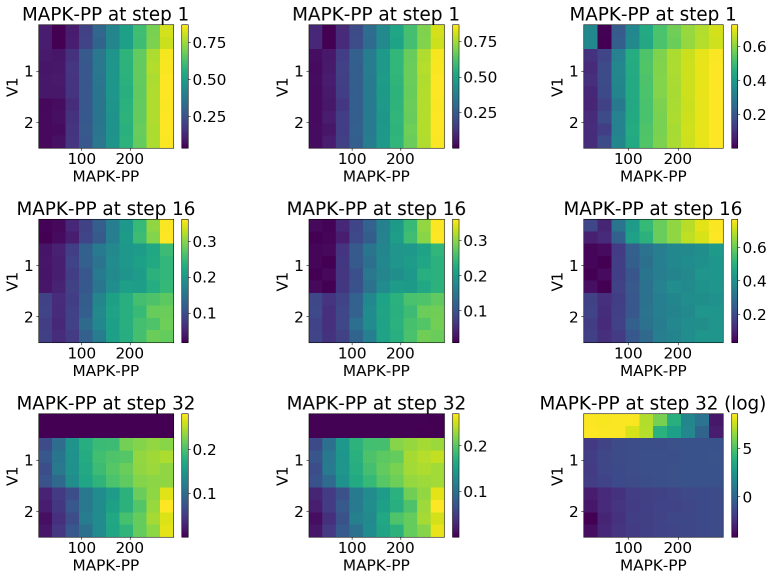
(a) Means abs. err. (b) Means rel. err.

Figure A.5: Landscapes of the mean distance, absolute (a) and relative (b) for the two-dimensional **Toggle Switch** model.

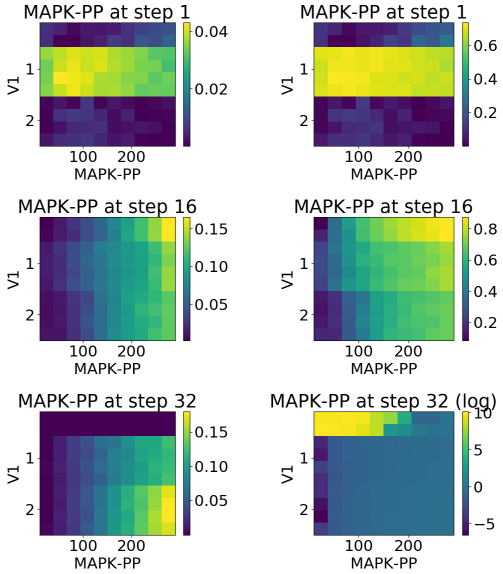


(a) Variances abs. err. (b) Variances rel. err.

Figure A.6: Landscapes of the variance distance, absolute (a) and relative (b) for the two-dimensional **Toggle Switch** model.



(a) Wass. dist. (b) Means abs. err. (c) Means rel. err.



(d) Variances abs. err. (e) Variances rel. err.

Figure A.7: Histogram distance landscapes for the **MAPK** model.

Appendix B

Neural Predictive Monitoring

The figures below compare the performances (over all the PO case studies) of the Neural State Estimator (NSE) against two traditional state estimation techniques: Unscented Kalman Filters (UKF) and Moving Horizon Estimation (MHE). In particular, Fig. B.1-Fig B.3 show a comparison over three test points of the SN_{po} model, Fig. B.4-Fig B.6 over three test points of the IP_{po} model, Fig. B.7-Fig B.9 over three test points of the TWT_{po} model, Fig. B.10-Fig B.12 over three test points of the $CVDP_{po}$ model, Fig. B.13-Fig B.15 over three test points of the $LALO_{po}$ model and Fig. B.16 over a single test point of the HC_{po} model. In these plots each column represents a different test point and each row represents a variable of the state space.

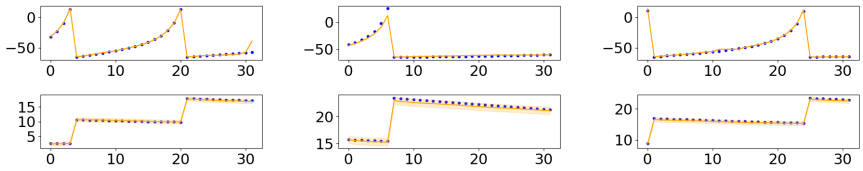


Figure B.1: SN_{po}: **Neural State Estimator**. Each column is a different test point and each row is a variable of the state space.

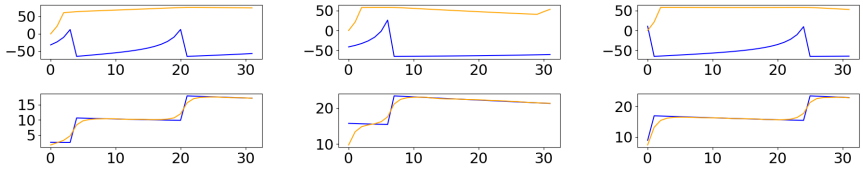


Figure B.2: SN_{po}: **Unscented Kalman Filters**.

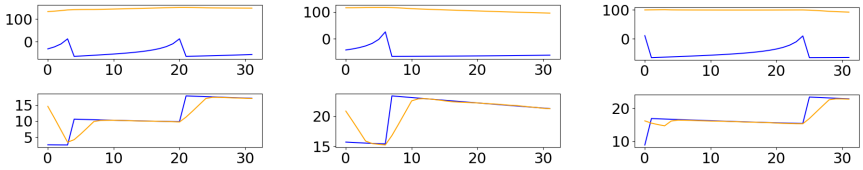


Figure B.3: SN_{po}: **Moving Horizon Estimate**.

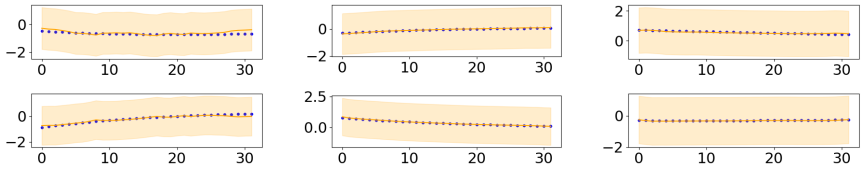


Figure B.4: IP_{po}: **Neural State Estimator**

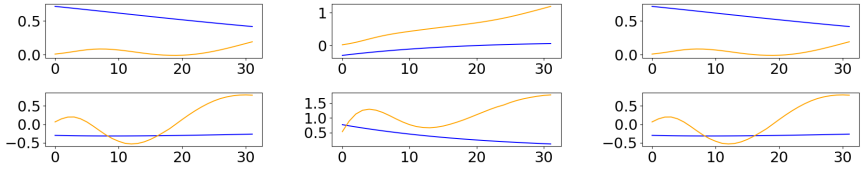


Figure B.5: IP_{po}: **Unscented Kalman Filters**

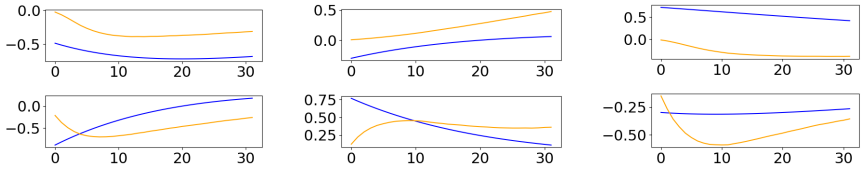


Figure B.6: IP_{po}: **Moving Horizon Estimate**.

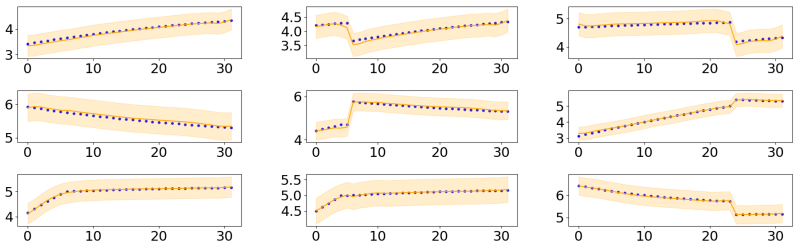


Figure B.7: TWT_{po}: **Neural State Estimator**.

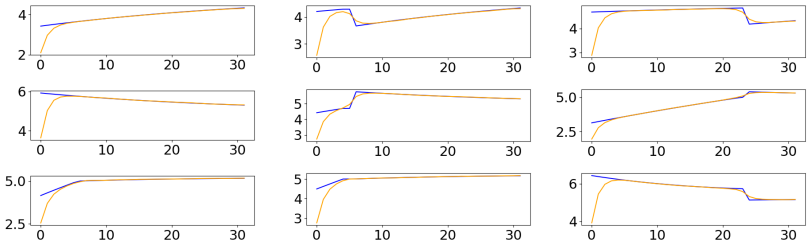


Figure B.8: TWT_{po}: **Unscented Kalman Filters**.

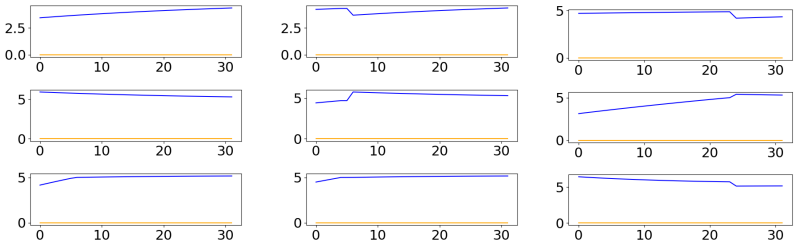


Figure B.9: TWT_{po}: **Moving Horizon Estimate**.

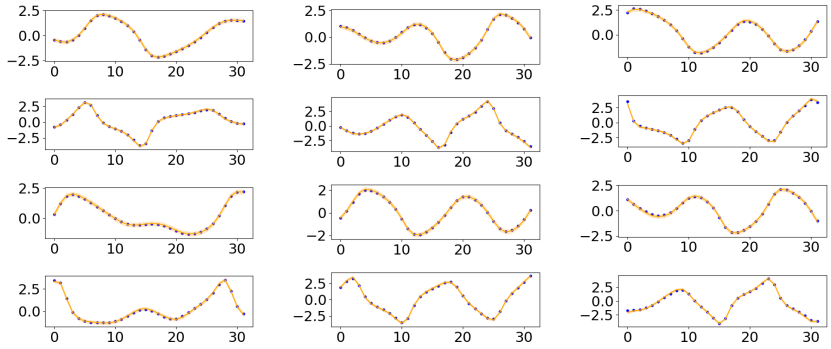


Figure B.10: $CVDP_{po}$: **Neural SE**.

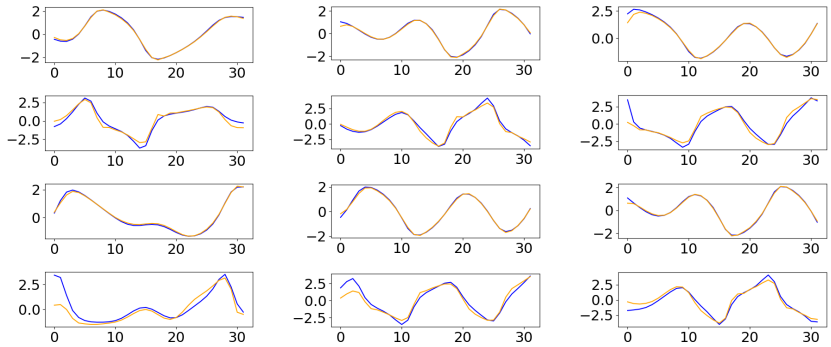


Figure B.11: $CVDP_{po}$: **Unscented Kalman Filters**.

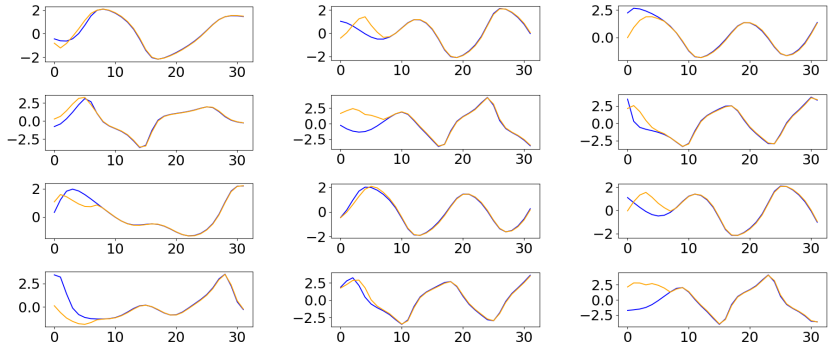


Figure B.12: $CVDP_{po}$: **Moving Horizon Estimate**.

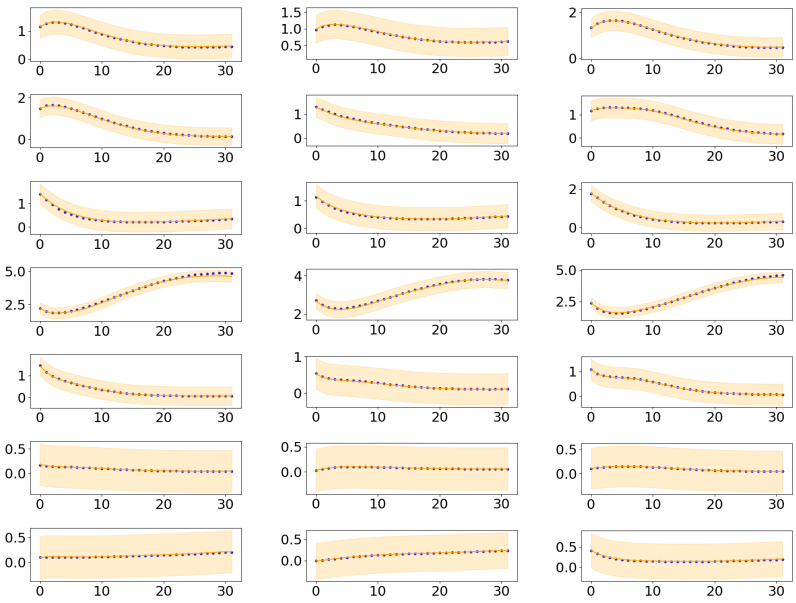


Figure B.13: $LALO_{po}$: **Neural State Estimator**

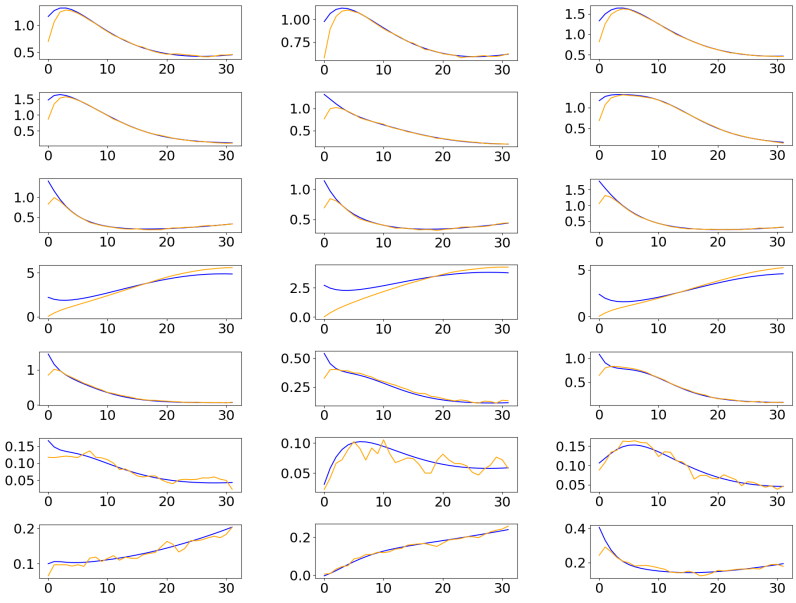


Figure B.14: $LALO_{po}$: **Unscented Kalman Filters**.

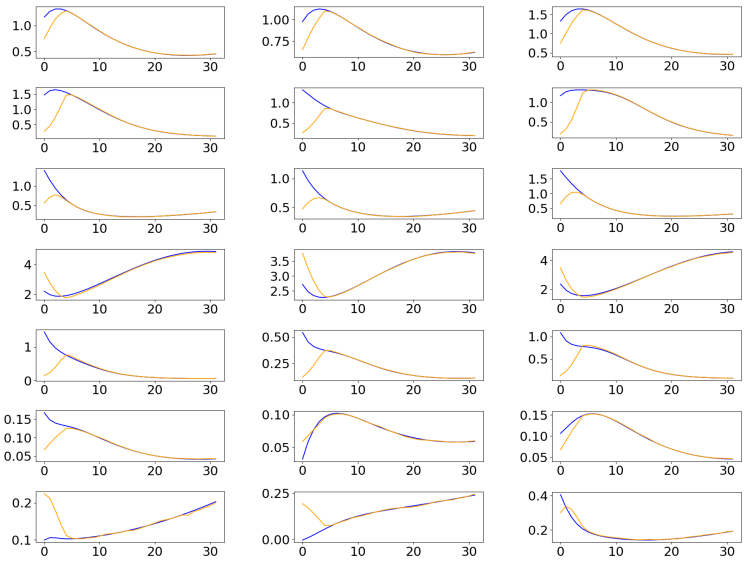


Figure B.15: $LALO_{po}$: **Moving Horizon Estimate**.

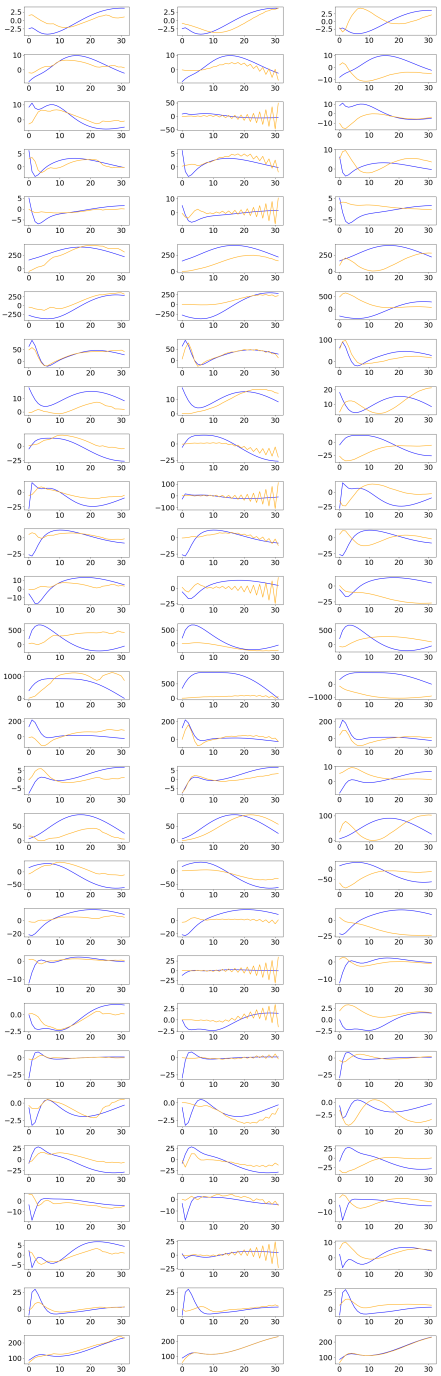


Figure B.16: HC_{po}: **Neural SE** (left) vs **UKF** (middle) vs **MHE** (right).