Efficient approximations of model predictive control laws via deep learning

Zur Erlangung des akademischen Grades eines

Dr.-Ing.

von der Fakultät Bio- und Chemieingenieurwesen
der Technischen Universität Dortmund
genehmigte Dissertation

vorgelegt von

M.Sc. Benjamin Karg

aus

Burglengenfeld

Tag der mündlichen Prüfung: 23.03.2023

1. Gutachter: Prof. Dr.-Ing. Sergio Lucia
2. Gutachter: Prof. Dr. Ali Mesbah

Dortmund 2023

Acknowledgements

The past few years have been a true roller coaster of events including changing the university, moving to a new city (basically twice) and trying to cope with the very special challenges created by a pandemic hitting a society that was barely prepared for it. Or in other words: it never got boring. Despite all the difficulties that arose, I learned many lessons both on a personal and academic level and I want to thank those people around me who supported me in many different ways.

First, I would like to thank Prof. Sergio Lucia for giving me the opportunity of writing my PhD thesis under his guidance. He has accompanied me on my academic journey from the humble beginnings writing my master thesis, to my first paper and up to the biggest challenge so far, my dissertation. The various discussions and his clear feedback helped me to not only dig deeper into the topic, but also make sure to dig into the right direction, which finally resulted in me becoming a better researcher. Further, I would like to thank Prof. Ali Mesbah for agreeing to be the external referee of my thesis and for the insightful feedback that he provided.

I am also very happy to have had the opportunity to cooperate with other researchers from various institutes who helped me widen my horizon and to explore new topics related to my research. In this context I want to especially thank Prof. Teodoro Alamo and Christian Utama.

I could not have achieved writing this thesis without the many great people I met during my years at TU Berlin, the Einstein Center Digital Future and TU Dortmund. I would like to thank my colleagues with whom I had the pleasure to work together, to share lunch meals and to have many great discussions about work-related topics as well as topics ranging from climate politics, ranking the sands of the world and pidgeon towers (yep, pidgeon towers), but also personal concerns. In this regard, I want to especially mention Felix Fiedler with whom I have happily shared an office for many years and who was always willing to listen to my ideas, to comment on them, and to play some table tennis to reset the brain.

Finally, I would like to express deep gratitude towards my family and friends for always supporting me in all the ways possible, helping me to keep my head up in difficult and stressful times and especially for all the quality time spent together which filled my heart with joy and content.

Abstract

Model predictive control (MPC) has established itself as the standard method for the control of complex nonlinear systems due to its ability to directly consider constraints and uncertainties while optimizing a control objective. However, the application of MPC requires repeatedly solving an optimal control problem online which can be computationally prohibitive, especially for large systems, for systems with very high control sampling rates and for the implementation on embedded hardware.

This thesis presents deep neural networks (DNNs) as a means of enabling the implementation of MPC algorithms when computation power is limited. The expressive capabilities of DNNs are leveraged to closely approximate the control law implicitly defined by the MPC problem. For the online application only evaluating the DNN, an explicit function consisting of simple arithmetic operations, is required. This results in speed-ups of several orders of magnitude in comparison to solving the MPC problem online.

Throughout the thesis, we shed light onto various aspects that enable and motivate the usage of DNNs as safe approximate MPC laws. Approaches to modify a DNN after an initial learning phase such that the closed-loop performance is improved are proposed. Further, methods that enable the analysis of the closed-loop behavior to obtain both deterministic and probabilistic guarantees on the online operation regarding safety, performance and stability are presented.

The efficacy of the proposed approaches is investigated for a wide range of case studies including a polymerization reactor of industrial complexity. The analysis shows that the DNN controllers do not only outperform other approximate MPC approaches in terms of control performance, memory footprint and evaluation times, but that DNN controllers can even outperform the exact optimization-based MPCs when ideas from reinforcement learning are used. Further it is shown that the DNN controllers can be deployed on embedded hardware such as microcontrollers with small effort.

Kurzzusammenfassung

Modelprädiktive Regelung (kurz: MPC) hat sich als die Standard-Methode zur Regelung von komplexen nichtlinearen System etabliert, da es erlaubt Beschränkungen und Unsicherheiten direkt zu berücksichtigen und zeitgleich eine Zielfunktion zu optimieren. Dafür muss jedoch wiederholt ein Optimierungsproblem online gelöst werden. Dies verhindert die Anwendung von MPC, wenn die Hardware nicht die nötige Rechenleistung aufbringt zur Berechnung einer Lösung innerhalb eines Zeitschritts.

In dieser Dissertation werden tiefe neuronale Netzwerke (kurz: DNNs) eingeführt als eine Möglichkeit, um MPC-Algorithmen auf leistungsschwacher Hardware zu realisieren. Dabei werden die repräsentativen Fähigkeiten von tiefen neuronalen Netzwerken genutzt, um das Regelgesetz, das implizit vom MPC-Optimierungsproblem definiert wird, zu approximieren. Der approximative DNN-Regler kann mehrere Grössenordnungen schneller evaluiert werden als der optimierungs-basierte Regler, da das DNN eine explizite Funktion bestehend aus simplen arithmetischen Operationen darstellt.

Im Laufe der Dissertation werden verschiedenste Aspekte, die die Nutzung von DNNs als effiziente approximative MPC-Regler ermöglichen und motivieren, vorgestellt. Dazu gehören Methoden zur Modifizierung eines gelernten DNN-Reglers, sodass die Performance optimiert wird, und Ansätze zur Analyse des geschlossenen Regelkreises, die es ermöglichen sowohl probabilistische als auch deterministische Garantien bezüglich Sicherheit, Performance und Stabilität zu erhalten.

Die Effektivität des vorgestellten Ansatzes wird für eine Vielzahl an Fallstudien untersucht, unter anderem für einen Polymerisationsreaktor von industrieller Komplexität. Die Untersuchungen zeigen, dass DNNs nicht nur andere approximative MPC Methoden übertreffen bezüglich Regel-Performance, Speicherbedarf und Ausführzeiten, sondern auch den ursprünglichen exakten MPC-Ansatz übertreffen können, wenn Ideen aus dem Bereich des bestärkenden Lernens genutzt werden. Zusätzlich wird gezeigt, dass die gelernten Regler mit wenig Aufwand auf eingebetteten Plattformen wie Micro-Controllern implementiert werden können.

Contents

Notation

Numbers and Arrays

a	A scalar (integer or real) or a vector
A	A matrix
$A \succ 0$	Square matrix A is positive definite
$A \succeq 0$	Square matrix A is positive semi-definite
$\text{diag}(a)$	A square, diagonal matrix with diagonal entries given by a
I	Identity matrix with dimensionality implied by context

Systems

A	System matrix of state state-space representation
B	Input matrix of state-space representation
d	Disturbance/uncertainty
E	Disturbance/uncertainty matrix of state-space representation
$f(\cdot)$	Discrete-time system equation
$g(\cdot)$	General nonlinear constraints
$h(\cdot)$	Measurement equation
u	control input
x	State of a system
x_{est}	State estimate

y	Measurement

Controllers

$\mu(\cdot)$	General state-feedback controller
$\mu_{\text{ideal}}(\cdot)$	Optimal robust state-feedback controller
$\mu_{\text{exp}}(\cdot)$	Exact explicit model predictive controller
$\mu_{\text{mpc}}(\cdot)$	General model predictive controller
$\mu_{\text{ms}}(\cdot)$	Multi-stage model predictive controller
$\mu_{\text{nn}}(\cdot;\theta)$	Neural network controller
$\mu_{\text{poly}}(\cdot;\theta_{\text{poly}})$	Approximate explicit model predictive controller based on polynomials
$\mu_{\text{synth}}(\cdot;\theta_{\text{synth}})$	Approximate explicit model predictive controller based on the partitioning of an explicit MPC of lower complexity
θ	Parameters of the neural network control policy $\mu_{\text{nn}}(\cdot)$
θ_{poly}	Parameters of the control policy $\mu_{\text{poly}}(\cdot)$
θ_{synth}	Parameters of the control policy $\mu_{\text{synth}}(\cdot)$

Sets

$\mathcal{A}$	A set
$\mathcal{A} \setminus \mathcal{B}$	Set subtraction, i.e., the set containing the elements of $\mathcal{A}$ that are not in $\mathcal{B}$
$\mathcal{A} \oplus \mathcal{B}$	Minkowski sum, i.e., all elements that are the sum of an element of $\mathcal{A}$ and an element of $\mathcal{B}$
$\lvert\mathcal{A}\rvert$	Cardinality of set $\mathcal{A}$, i.e. number of elements in $\mathcal{A}$
$\text{Conv}(\cdot)$	Convex hull of a set

$\mathcal{D}$	Set containing all uncertainties and disturbances d (at least with high probability)
Δ	Set containing the approximation error $\delta(x)$ (at least with high probability)
$\mathbb{N}$	The set of all non-negative integers including ∞
$\mathbb{N}^+$	The set of all positive integers including ∞
$\mathbb{N}_{[a]}$	The set of all integers from 0 to a
$\mathbb{N}^+_{[a]}$	The set of all integers from 1 to a
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	The vector of real numbers with n entries
$\mathbb{R}^{n\times m}$	The matrix of real numbers with n rows and m columns
$\mathcal{U}$	Admissible control input space
$\mathcal{X}$	Admissible state space
$\mathcal{X}_\mathsf{f}$	Terminal set

Indexing

$a^{(i)}$	Element i of vector a, with indexing starting at 1.
$A^{(i,j)}$	Element i, j of matrix A
$A^{(i)}$	Row i of matrix A

Linear Algebra Operations

a^T	Transpose of vector a
A^T	Transpose of matrix A
$A \odot B$	Element-wise (Hadamard) product of A and B

Calculus

$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to x

Probability

beta(α, β)	Beta distribution with shape parameters $\alpha > 0$ and $\beta > 0$
$\mathbb{E}[\cdot]$	Expected value of a random variable
normal(α, β)	Normal distribution with mean value α and variance β
pareto(α)	Pareto distribution with smallest possible value α
Pr$\{\cdot\}$	Probability of an event
$x \sim \chi$	Random variable x has distribution χ

Neural networks

b_l	Bias of the l-th layer
Γ	Fixed activation pattern
$\mathcal{H}_\Gamma$	Polytopic region defined by a fixed activation pattern Γ
L	Number of hidden layers
$\lambda_l(\cdot)$	Affine function of the l-th layer, i.e. $\lambda_l(x) = W_l x + b_l$, where x is the output of the previous layer
$\lambda_{\mathsf{p}}(\cdot)$	Perceptron
$\mu_{\mathsf{nn}}(\cdot; \theta, n_0, n_1, \ldots, n_{L+1})$	Feedforward deep neural network with n_0 input neurons , n_{L+1} output neurons and n_l neurons
n_l	Number of neurons in the l-th layer
$\Psi(x)$	Activation pattern for a state x

$\mathrm{ReLU}(x)$	rectified linear unit, $\frac{1}{1+\exp(-x)}$
$\sigma_l(x)$	Nonlinear activation function
$\tanh(x)$	hyperbolic tangent, $\frac{e^x - e^x}{e^{-x} + e^{-x}}$
W_l	Weights of the l-th layer
$\zeta(\cdot)$	Activation-based description of a neural network controller
z_l	Output of the l-th layer

Functions and Operators

$f \circ g$	Composition of the functions f and g
$f_{\mathrm{PWA}}(\cdot)$	Piecewise affine function
$\prod_{k=1}^{r} f$	r-time composition of the function f
$f(x;\theta)$	A function of x parametrized by θ. (Sometimes we write $f(x)$ and omit the argument θ for the sake of readability)
$J(\cdot)$	Objective function
$\ell(x,u)$	Stage cost
$\ell_{\mathrm{cl}}(x,\mu)$	Closed loop stage cost w.r.t. controller $\mu(\cdot)$, i.e. $\ell(x,\mu(x))$
$\ell_{\mathrm{f}}()$	terminal cost
$\ln x$	Natural logarithm of x
$\mathcal{R}(\cdot)$	Computes the reachable set for a set of initial conditions
$\mathcal{R}_r(\cdot)$	Computes the r-step reachable set for a set of initial conditions
$\mathcal{P}(\cdot)$	Computes the preimage of a set in the state space
$\mathcal{P}_r(\cdot)$	Computes the r-step preimage of a set in the state space
$\mathcal{Z}(\cdot)$	Computes the output set of a controller

Data sets and Distributions

χ_d	Probability distribution of the uncertain parameter-s/disturbances
χ_{init}	Probability distribution of the initial states of a control task
χ_w	Probability distribution of the parameters fully defining a closed loop trajectory
$\mathcal{S}$	Set of scenarios for reinforcing a neural network controller
$\mathcal{T}$	Data set for training a parametric controller, e.g. a deep neural network
$\mathcal{V}$	Data set for verifying a controller

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In times of rising emissions that drive the climate change and soaring energy prices, advanced control and optimization are crucial tools for the development of the efficient systems and processes of the future. In recent works, see [1] for an overview, it was shown that leveraging state-of-the-art control approaches can lead to significantly reduced energy consumption, wear and emissions. One example that illustrates the potential benefit of applying advanced methods is the control of the heating, ventilation and air conditioning system of a data center, whose energy consumption could be reduced by 9 % [2] by using an advanced approach, i.e. model predictive control (MPC), instead of PID controllers. But also the locomotion of quadruped robots at moderate velocities up to 3 m s^{-1} can be achieved via MPC [3].

In MPC, a mathematical model is used to predict the future behavior of the system. Based on this prediction, an optimization problem is solved to derive the optimal control inputs to be applied to the system. The major drawback of these advanced optimization-based approaches is the necessity to solve a challenging optimization problem online which can be prohibitive, especially if the computational hardware is limited, or when the control sampling rate is very high.

To counteract this drawback, a lot of research effort has been devoted to the development of methods that pre-compute the solutions of the optimization problems offline and store the result on the control hardware and such avoid online optimization. In online operation, the stored information can be used like a look-up table to retrieve the optimal control input. But this class of approaches are only applicable to small-scale systems due to the exponential growth of the look-up tables with respect to the system size and the subsequent computationally prohibitive offline computation phase and significant memory requirements for the online implementation.

Inspired by the recent successes of deep learning to tackle a broad range of tasks, in this thesis deep neural networks are explored as a means to alleviate the disadvantages of the existing MPC techniques.

1.2 Scope of the thesis

The goal of this thesis is to introduce methods that broaden the range of applications for which advanced control strategies can be deployed. The central challenge of implementing an advanced controller such as MPC is the requirement to solve a complex nonlinear optimization problem online in every control instant which might require potent hardware and specific software, especially in case of operating uncertain and nonlinear systems. Hence, the developed approach should avoid the need for online optimization. An online optimization-free advanced control strategy would facilitate the implementation due to the absence of complex optimization algorithms and and the lower computational requirements enable the realization of MPC for fast systems and cheap and computationally limited hardware. The deployed controllers should ensure the robust satisfaction of process constraints at all times. Further, guarantees on performance are desired as well as a stabilizing behavior in case of regularization tasks.

These objectives are accomplished primarily by leveraging the recent advances in the field of deep learning. The expressive capabilities of deep neural networks allow to closely imitate the behavior of robust nonlinear model predictive control approaches while only requiring a fraction of the computational power and memory. By combining ideas from output range analysis and well-established control theoretic properties or utilizing probabilistic tools, it is possible to provide guarantees on the closed-loop operation.

The performance of the proposed methods will be analyzed via various challenging case studies that will highlight its advantages in comparison to existing approaches that were developed to render the usage of MPC methods without online optimization possible.

1.3 Structure and contribution

The thesis consists of three main parts which cover the theoretical basics of deep learning-based MPC, deal specifically with methods to design and analyze the control of linear systems with neural network controllers and approaches that are more suited for the operation of nonlinear systems. Each part will be introduced in detail in the following before listing which sections are based on publications derived from the research phase of the thesis.

1.3.1 Part I: Background

The first part introduces the challenges which are being tackled throughout the thesis and the theoretical foundations which are used to derive the results presented in the manuscript.

In Chapter 2, the general robust control problem and the corresponding ideal MPC controller are introduced. Further, an overview of robust MPC techniques is provided and a tractable approximation of the ideal controller, namely multi-stage MPC, and an MPC approach that avoids online optimization for systems of moderate size and complexity are presented. The contents of the chapter are derived from [4, 5].

In Chapter 3, neural networks are introduced as a means of enabling the efficient and optimization-free advanced control of complex systems. The introduction includes a short overview of the history of deep learning from the early beginnings in the middle of the 20th century to current achievements and presents the structure of the feedforward neural networks used in this work. Further, it is described how multi-stage MPC can be used to generate the necessary data sets to obtain neural networks via deep learning that closely imitate the behavior of the optimization-based MPCs. Parts of this chapter are based on the publication [5].

1.3.2 Part II: Deep learning-based control for linear systems

The second part presents results that motivate the usage of deep neural networks as controllers, in particular for linear systems, and presents methods that allow the offline verification of learned controllers regarding safety and performance before deployment.

Chapter 4 shows that for a common class of model predictive control problems, the solution can be exactly represented by deep neural networks with rectified linear unit activations. The chapter is mainly based on the results published in [5].

In Chapter 5, it is highlighted that the main advantage of using deep neural networks is not their capability to exactly represent the solution of an MPC problem for linear systems, but their ability to closely approximate the solution in an efficient manner. This enables the simple and fast application of advanced control methods where online MPC is too slow or computationally challenging. As for the previous chapter, the results of Chapter 5 are based on the results from [5].

For uncertain linear systems, it is possible to obtain deterministic guarantees on closed-loop constraint satisfaction, performance and stability, as shown in Chapter 6. The chapter is based on the contributions published in [6, 7].

1.3.3 Part III: Deep learning-based control for nonlinear systems

The third part deals with the more realistic and challenging case of nonlinear systems where only incomplete state information is available and state estimation strategies need to be applied to enable feedback control.

Chapter 7 deals with the case of output-feedback where not only the problem of learning a controller, but also the learning of a state estimator and the direct mapping from previous outputs and measurements to the optimal control input is investigated. The results used in this chapter were obtained in [8].

Chapter 8 makes use of the explicit nature of learning-based controllers and detailed models to improve and adapt the learning-based policy by leveraging ideas from reinforcement learning. The results have originally been published in [9].

Due to the complex behavior of the uncertain nonlinear systems and the additional uncertainty introduced by noise and measurements, obtaining deterministic guarantees as presented for the linear case in Chapter 6 can be intractable. Therefore we resort to methods that provide probabilistic guarantees on safety and performance which are presented in Chapter 9. The contents of the chapter are based on the publication [4].

1.3.4 Part IV: Conclusions and future work

The findings and main results of the thesis are summarized in Chapter 10 by recapitulating how the goal of realizing an advanced optimization-free controller was reached. Finally, directions for future research are provided that might further facilitate the implementation of efficient learning-based controllers and that might improve the performance of the deployed deep neural network controllers.

1.4 Declaration on the reproduction of pre-published content

Parts of this thesis are based on already published and peer-reviewed works. An overview on which publications have been used in which section of the thesis is given the following and the corresponding publications are listed in the next paragraph.

Chapter 2	2.2	in parts from (modified)	(D)
	2.4	in parts from (modified)	(D)
	2.5	in parts from (modified)	(A)
Chapter 3	3.2	in parts from (modified)	(A)
	3.3	in parts from (modified)	(A)
Chapter 4	4.1	in parts from (modified)	(A)
	4.1	in parts from (modified)	(A)
Chapter 5	5.1	in parts from (modified)	(A)
	5.2	in parts from (modified)	(A)
Chapter 6	6.1	in parts from (modified)	(B), (F)
	6.2	in parts from (modified)	(B), (F)
	6.3	in parts from (modified)	(B), (F)
	6.5	in parts from (modified)	(B)
	6.7	in parts from (modified)	(B), (F)
Chapter 7	7.1	in parts from (modified)	(C)
	7.2	in parts from (modified)	(C)
	7.3	in parts from (modified)	(C)
	7.4	in parts from (modified)	(C)
Chapter 8	8.1	in parts from (modified)	(E)
	8.2	in parts from (modified)	(E)
	8.3	in parts from (modified)	(E)
	8.4	in parts from (modified)	(E)

	8.5	in parts from (modified)	(E)
Chapter 9	9.1	in parts from (modified)	(D)
	9.2	in parts from (modified)	(D)
	9.3	in parts from (modified)	(D)
	9.4	in parts from (modified)	(D)
	9.5	in parts from (modified)	(D)

Publications

During the research phase for this thesis, various works have been published to present intermediate results in journals and at conferences. The following publications have been essential for the thesis:

(A) Karg, B.; Lucia, S. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics* **2020**, 50 (9), 3866-3878.

(B) Karg, B.; Lucia, S. Stability and feasibility of neural network-based controllers via output range analysis. In *Proceedings of the Conference on Decision and Control* 2020, 4947-4954.

(C) Karg, B.; Lucia, S. Approximate moving horizon estimation and robust nonlinear model predictive control via deep learning. *Computers & Chemical Engineering* **2021**, 148, 107266.

(D) Karg, B.; Alamo, T.; Lucia, S. Probabilistic performance validation of deep learning-based robust NMPC controllers. *International Journal of Robust and Nonlinear Control*, **2021**, 31 (18), 8855-8876.

(E) Karg, B.; Lucia, S. Reinforced approximate robust nonlinear model predictive control. In *Proceedings of the International Conference on Process Control* 2021, 149-156.

(F) Karg, B.; Lucia, S. Guaranteed safe control of systems with parametric uncertainties via neural network controllers. In *Proceedings of the Conference on Decision and Control*, 2022.

Further works that were published throughout the research phase of this thesis, but are not central to thesis, are given in the following list:

- Lucia, S.; Karg, B. A deep learning-based approach to robust nonlinear model predictive control. *IFAC - PapersOnLine* **2018**, 51 (20), 511-516.

- Karg, B.; Lucia, S. Towards low-energy, low-cost and high-performance IoT-based operation of interconnected systems. In *Proceedings of the World Forum on Internet of Things* 2018, 706-711.

- Karg, B.; Lucia, S. Deep learning-based embedded mixed-integer model predictive control. In *Proceedings of the European Control Conference* 2018, 2075-2080.
- Karg, B.; Lucia, S. Learning-based approximation of robust nonlinear predictive control with state estimation applied to a towing kite. In *Proceedings of the European Control Conference* 2019, 16-22.
- Lucia, S.; Navarro, D.; Karg, B.; Sarnago, H. Lucia, O. Deep learning-based model predictive control for resonant power converters. *IEEE Transactions on Industrial Informatics* **2020**, 17 (1), 409-420.
- Karg, B.; Lucia, S. Model predictive Control for the Internet of Things. In *Recent Advances in Model Predictive Control*, Springer, 2021; pp 165-189.
- Utama, C.; Karg, B.; Meske, C.; Lucia, S. Explainable artificial intelligence for deep learning-based model predictive controllers. In *Proceedings of the International Conference on System Theory, Control and Computing* 2022, 464-471.

Part I

Background

Chapter 2

Model predictive control

Model predictive control (MPC) is a popular advanced control technique that can deal with nonlinear systems and constraints while considering general control goals that go beyond conventional set-point tracking tasks. More specifically, if robust MPC is considered, it is possible to handle uncertainties that arise in the context of parameter variations, disturbances and plant-model mismatch. Despite these advantageous properties which made MPC a popular advanced control method, one major obstacle needs to be overcome to enable the application of MPC for a broader range of systems. The resulting computational complexity of the non-convex optimization problem - which needs to be solved online in every control interval - renders designing and implementing a nonlinear model predictive controller a challenging and sometimes even infeasible task.

In this chapter, we will specify the control performance that an ideal robust controller should provide and present methods that approximate this ideal controller in a more tractable fashion. First, the general principle of model predictive control is explained in Section 2.1, before introducing the optimization problem that encodes the ideal behavior of a robust, but usually intractable control policy in Section 2.2. Approaches and basic techniques that enable the formulation of tractable robust optimal control problems are presented in Section 2.3 before presenting multi-stage MPC, which will be used extensively throughout the thesis, in detail in Section 2.4. A method that further reduces the computational requirements for applying MPC schemes by completely avoiding online optimization, namely explicit MPC, is presented in Section 2.5. The contents of this chapter are largely based on the publication [4].

2.1 Basic principle of model predictive control

Model predictive control is a control approach where a mathematical model is used to predict the future evolution of a system considering the current state of the system. Based on the future evolution, the control objective and process constraints, an optimization problem is formulated whose optimal solution provides either a control input sequence or a policy that results in the optimal future behavior. The first element of the sequence is applied to the real system, the new current state of the system is obtained via measurements and the whole process is repeated at every control instance. The MPC control strategy is illustrated in Fig. 2.1.

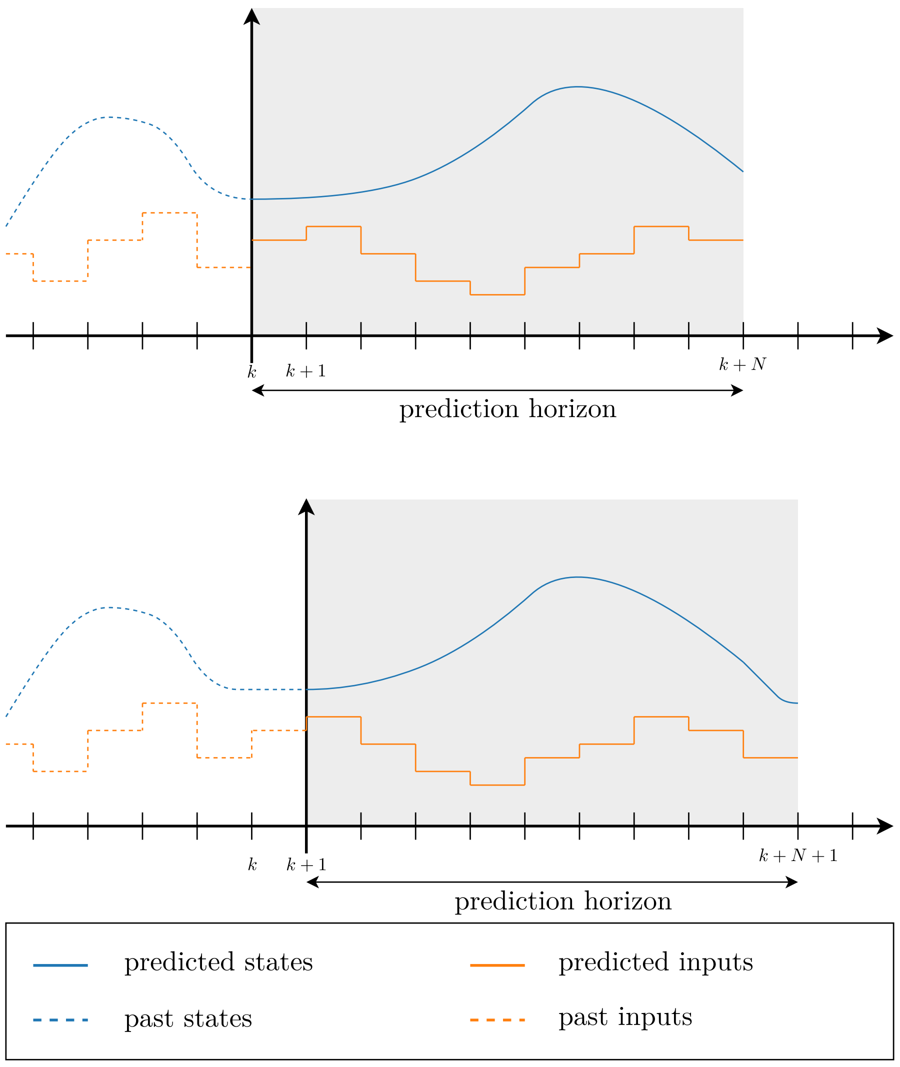


Figure 2.1: Two subsequent MPC steps.

The quality of the computed inputs is determined by the accuracy of the prediction model. However, each mathematical model suffers from errors arising from unmodelled dynamics, parameter variations or process noise and measurement noise. It is therefore crucial to consider robust formulations that are capable of dealing with these uncertainties.

2.2 Robust model predictive control problem

Handling uncertainty in the context of model predictive control is the main goal of robust MPC. In this thesis, we consider discrete-time nonlinear time-invariant systems that are affected by uncertainties:

$$x^+ = f(x, u, d), \tag{2.1}$$

where $x \in \mathbb{R}^{n_x}$ are the states, $u \in \mathbb{R}^{n_u}$ are the inputs, and $d \in \mathbb{R}^{n_d}$ are uncertain parameters and/or disturbances. Considering the uncertain parameters d, which are contained in a set $\mathcal{D} \subseteq \mathbb{R}^{n_d}$, is vital due to unavoidable plant-model mismatch. By deploying a feedback controller $\mu : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$ the closed-loop behavior is described by:

$$x^+ = f(x, \mu(x), d). \tag{2.2}$$

One challenge of robust control is to ensure the satisfaction of the constraints for all possible realizations of the uncertain values $d \in \mathcal{D}$ in closed-loop operation. In order to formulate the constraints in the optimal control problem considering the state, reachable sets can be leveraged. Based on the the system model $f(\cdot)$, the controller $\mu(\cdot)$, a set of initial conditions $\mathcal{X}_\text{s} \subseteq \mathbb{R}^{n_x}$ and the uncertainty set $\mathcal{D}$, the reachable set of the closed-loop (2.2) is defined as:

$$\mathcal{R}(\mathcal{X}_\text{s}, \mathcal{D}, \mu) := \left\{x^+ \in \mathbb{R}^{n_x} \,|\, x^+ = f(x, \mu(x), d)\, \forall (x, d) \in \mathcal{X}_\text{s} \times \mathcal{D}\right\}. \tag{2.3}$$

The reachable set operator $\mathcal{R}(\cdot)$ can also be applied repeatedly to obtain multi-step reachable sets. There are several methods to compute such reachable sets. In the case of linear systems and polytopic uncertainty sets, the consideration of the vertices of the uncertainty set and their propagation along the prediction horizon is enough to compute an exact reachable set. In the nonlinear case, linearization techniques [10] or ODE bounding techniques [11] can be used to obtain guaranteed over-approximations. Another possibility for the propagation of uncertainty is to resort to probabilistic reachable sets as done in [12, 13].

For constraints that are affected by the control input, we make use of output sets. An output set is the entirety of control inputs that a controller provides for a set of states. Output sets are defined - in a similar manner as reachable sets - with respect to a set of initial conditions $\mathcal{X}_\text{s}$ and for a control policy $\mu(\cdot)$ via:

$$\mathcal{Z}(\mathcal{X}_\text{s}, \mu) := \{u \in \mathbb{R}^{n_u} \,|\, u = \mu(x)\ \forall\, x \in \mathcal{X}_\text{s}\}. \tag{2.4}$$

The desired specifications for an ideal robust control policy can then be cast as an optimization problem based on the reachable sets and output sets:

$$\begin{aligned}
\underset{\mu_\text{ideal}(\cdot)}{\text{minimize}} \quad & J(\mathcal{X}, \mathcal{D}, \mu_\text{ideal}, N) && \text{(2.5a)}\\
\text{subject to} \quad & \text{for all } x_\text{init} \in \mathcal{X}: \\
& \quad \mathcal{X}_0 = \{x_\text{init}\}, && \text{(2.5b)}\\
& \quad \text{for all } k \in \mathbb{N}_{[N-1]}: \\
& \quad\quad \mathcal{X}_{k+1} = \mathcal{R}(\mathcal{X}_k, \mathcal{D}, \mu_\text{ideal}), && \text{(2.5c)}\\
& \quad\quad \mathcal{U}_k = \mathcal{Z}(\mathcal{X}_k, \mu_\text{ideal}), && \text{(2.5d)}\\
& \quad\quad \mathcal{X}_k \subseteq \mathcal{X}, && \text{(2.5e)}\\
& \quad\quad \mathcal{U}_k \subseteq \mathcal{U}, && \text{(2.5f)}\\
& \quad\quad g_\text{cl}(\mathcal{X}_k, \mathcal{D}, \mu_\text{ideal}) \leq 0, && \text{(2.5g)}\\
& \quad \mathcal{X}_N \in \mathcal{X}_\text{f}, && \text{(2.5h)}
\end{aligned}$$

where N is the prediction horizon and (2.5a) is the objective function. The state constraints (2.5e) and control input constraints (2.5f) are expressed via the sets $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ and $\mathcal{U} \subseteq \mathbb{R}^{n_u}$, respectively, and the closed-loop mixed constraints as an inequality (2.5g). The mixed constraints mean that the condition $g(x_k, u_k, d_k) \leq 0$ has to be satisfied for all $(x_k, d_k) \in \mathcal{X}_k \times \mathcal{D}$ and corresponding $u_k = \mu_{\text{ideal}}(x_k)$. The terminal constraint (2.5h) with $\mathcal{X}_\text{f} \subseteq \mathbb{R}^{n_x}$ can be designed such that safety and stability is ensured. Solving the ideal robust NMPC problem defined in (2.5), one obtains a receding horizon policy $\mu_{\text{ideal}}(\cdot)$ which provides the optimal control input based on the initial state x_{init}. But obtaining a solution for (2.5) is usually intractable mainly because the exact reachable and output sets might be unknown and the ideal feedback law $\mu_{\text{ideal}}(\cdot)$ is difficult to determine for all of $\mathcal{X}$. However, various approaches that compute an approximate solution to (2.5) based on simplifications exist. In the following several approaches and techniques for the control of uncertain systems are shortly presented.

2.3 Robust control techniques

The first robust MPC approach was based on min-max optimization, a robust optimization technique based on the findings in [14], and was introduced in [15]. In min-max MPC, the objective is minimized for the worst-case realization of the uncertainties while assuring that in no case the constraints are violated. The main drawback of this approach is that the control problem is formulated in open-loop fashion. This leads to very conservative results or renders the optimization problem infeasible in the case of larger uncertainties.

Closed-loop robust MPC tackles the problem of conservativeness by using the idea that new information will be available in the future and hence the decisions can be updated. In [16], the robust control problem is solved by optimizing over control policies instead of optimizing over control inputs, leading however to intractable formulations in the general case. An alternative and common approach to include recourse in the problem formulation is to consider various scenarios, i.e. sequences of future realizations of the uncertainties, to consider a point-wise approximation of the true reachable sets. Since in each interval a new optimal control input will computed based on the current state, the future evolution of the system can described by a tree structure, the so-called scenario tree. In [17] the scenario tree is used in a min-max approach which results in a significantly less conservative performance in comparison to [15]. However, the conservativeness of scenario tree-based approaches can be further reduced when the probability distributions of the uncertainties are known. One can then minimize the expected value of the cost [18] - instead of the worst-case cost - or design a cost function which weighs the impact of each scenario according to the probability of realization [19]. Due to these probabilistic considerations, the latter two methods can be accounted to the family of stochastic MPC approaches.

Another common technique of stochastic MPC approaches is to formulate the process constraints as chance constraints. Instead of requiring the constraints to be satisfied for all possible realizations of the uncertainty, the constraints only need to be satisfied with a pre-specified probability [20]. This can render the application of robust MPC methods possible when uncertainties and disturbances can be very large or follow unbounded probability distributions. However, the scenario-based and stochastic MPC methods result in challenging optimization problems because various scenarios need to be considered.

Tube-based approaches [21] simplify the challenge for robust control by decomposing the robust MPC problem into a nominal MPC and an ancillary controller. The ancillary controller, often realized in the form of a an affine feedback, makes sure that the real uncertain system stays within an invariant tube around the trajectory planned by the nominal MPC. The invariant tube is further used to tighten the constraints of the nominal MPC such that robust constraint satisfaction of the overall system is achieved. In the simplest version, the complexity of tube-based MPC is the same as that of an MPC for certain systems. However, if an increased performance is desired, the complexity grows as presented in [22] or [23] and in presence of large uncertainties, the behavior of tube-based MPC can be conservative. To mitigate the drawback of tube-base MPC and MPC formulations that use a scenario tree, methods have been developed that combine the two methods [24]. This allows to trade-off computational complexity and performance by considering the dominant uncertainties in a scenario tree and by counteracting the effect of smaller uncertainties and disturbances with the ancillary controller.

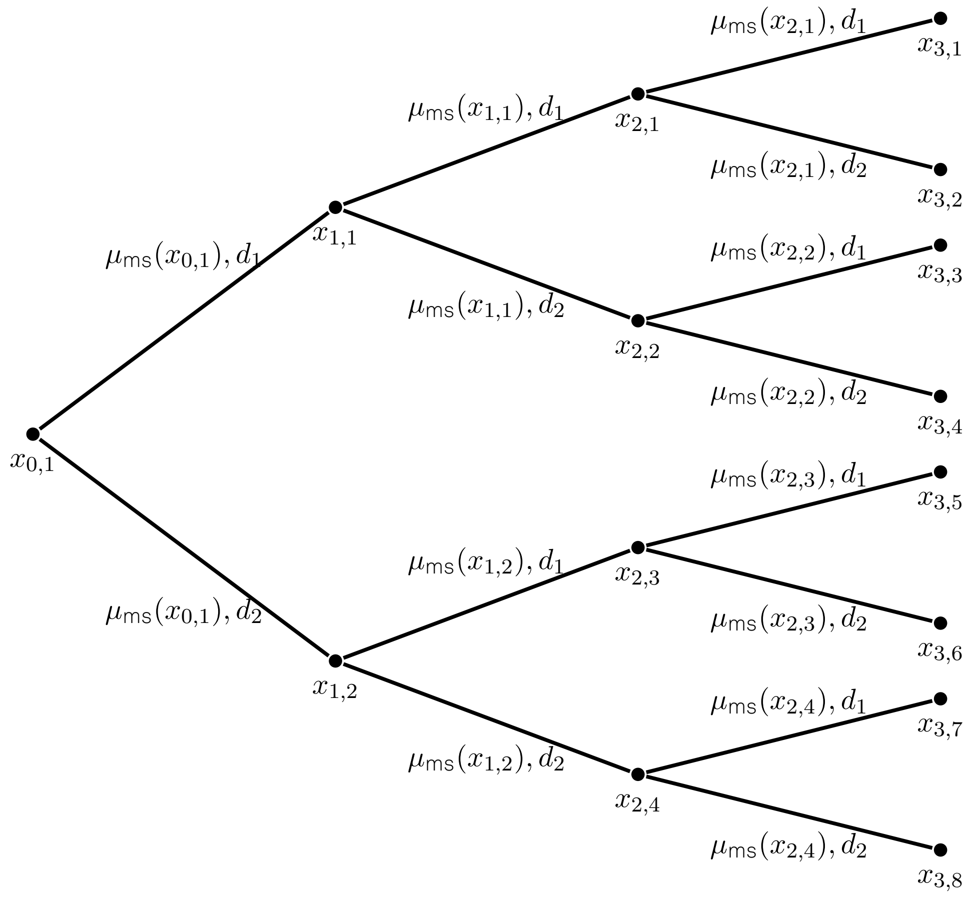
In this thesis, we will leverage an MPC formulation that relies on a scenario tree and which is usually called multi-stage MPC [19]. We choose multi-stage MPC to approximate (2.5) for two reasons. First, multi-stage often exerts an improved performance in practice because the feedback structure is not restricted to be affine, as usually done in tube-based MPC and in other robust approaches [25]. And second, the complex multi-stage optimization problem formulation will only be used in the offline phase - where computational limitations are of no concern- to obtain the safe and efficient deep learning-based neural network controllers whose design is the central goal of the thesis.

2.4 Multi-stage MPC

Multi-stage MPC [26] or scenario tree-based MPC [17, 18, 27] approximates the policy defined in (2.5) by considering a finite amount of possible realizations of the uncertainties in a tree structure. This means that the true uncertainty set is approximated by a discrete number of uncertainty realizations:

$$\mathcal{D} \approx \tilde{\mathcal{D}} = \{d_1, \dots, d_s\}, \tag{2.6}$$

where s is the number of possible realizations of the uncertainty that are considered in the tree. The considered realizations mean that each node branches s times at each control instant which results in s^k nodes at stage k. A reasonable strategy to build a scenario tree is to consider as branches all possible combinations of the extreme values of the uncertainty. A exemplary scenario tree where $s = 2$ possible uncertainty realizations d_1 and d_2 are considered is illustrated in Fig. 2.2. The state $x_{0,1}$ represents the current state of the system and the possible evolutions of the system until a prediction horizon of $N = 3$ are formulated. To correctly model the temporal causality of the feedback, the control inputs applied on branches originating from the same state need to be equal.


$\mu_{\mathrm{ms}}(x_{2,1}), d_1$
$x_{3,1}$
$\mu_{\mathrm{ms}}(x_{1,1}), d_1$
$x_{2,1}$
$\mu_{\mathrm{ms}}(x_{2,1}), d_2$
$x_{3,2}$
$x_{1,1}$
$\mu_{\mathrm{ms}}(x_{0,1}), d_1$
$\mu_{\mathrm{ms}}(x_{2,2}), d_1$
$x_{3,3}$
$\mu_{\mathrm{ms}}(x_{1,1}), d_2$
$x_{2,2}$
$\mu_{\mathrm{ms}}(x_{2,2}), d_2$
$x_{3,4}$
$x_{0,1}$
$\mu_{\mathrm{ms}}(x_{2,3}), d_1$
$x_{3,5}$
$\mu_{\mathrm{ms}}(x_{1,2}), d_1$
$x_{2,3}$
$\mu_{\mathrm{ms}}(x_{0,1}), d_2$
$\mu_{\mathrm{ms}}(x_{2,3}), d_2$
$x_{3,6}$
$x_{1,2}$
$\mu_{\mathrm{ms}}(x_{2,4}), d_1$
$x_{3,7}$
$\mu_{\mathrm{ms}}(x_{1,2}), d_2$
$x_{2,4}$
$\mu_{\mathrm{ms}}(x_{2,4}), d_2$
$x_{3,8}$

$$\tilde{\mathcal{X}}_0 = \{x_{0,1}\}; \qquad \tilde{\mathcal{X}}_1 = \{x_{1,1}, x_{1,2}\}; \qquad \tilde{\mathcal{X}}_2 = \{x_{2,1}, \ldots, x_{2,4}\}; \quad \tilde{\mathcal{X}}_3 = \{x_{3,1}, \ldots, x_{3,8}\};$$

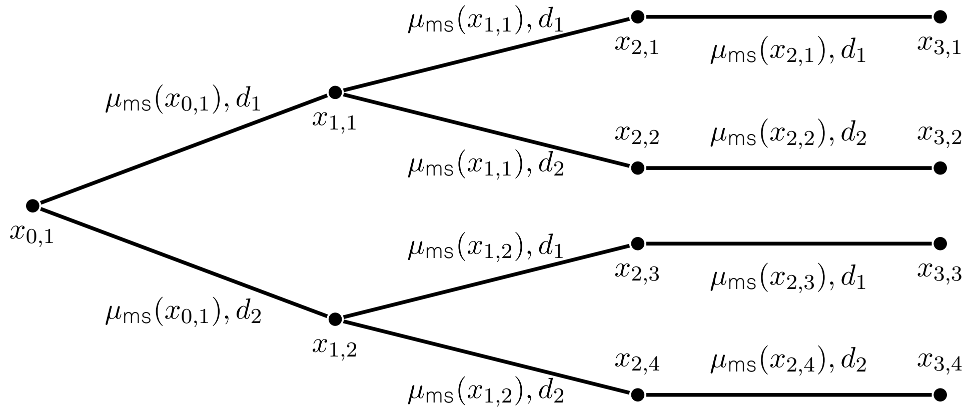
Figure 2.2: Scenario tree representation when considering 2 realizations of the uncertainty $\tilde{\mathcal{D}} = \{d_1, d_2\}$ and a prediction horizon of $N = 3$.

Using a scenario tree formulation, an approximation of the true reachable set can be obtained as the convex hull of the set of all the nodes at a given stage, i.e.:

$$\mathcal{X}_k \approx \mathrm{Conv}(\tilde{\mathcal{X}}_k) = \mathrm{Conv}\left(\bigcup_{i=1}^{s^k} x_{k,i}\right), \tag{2.7}$$

where Conv$(\cdot)$ denotes the convex hull of a set and $x_{k,i}$ denotes the node i of the tree at stage k as depicted in Figure 2.2.

In the linear case with polytopic uncertainty, including the extreme values of the uncertainty in $\tilde{\mathcal{D}}$ guarantees an exact representation of the actual reachable set. In the nonlinear case it is only an approximation and therefore we focus on the point-wise approximation $\tilde{\mathcal{X}}$.


$\mu_{\text{ms}}(x_{1,1}), d_1$
$x_{2,1}$
$\mu_{\text{ms}}(x_{2,1}), d_1$
$x_{3,1}$
$\mu_{\text{ms}}(x_{0,1}), d_1$
$x_{1,1}$
$x_{2,2}$
$\mu_{\text{ms}}(x_{2,2}), d_2$
$x_{3,2}$
$\mu_{\text{ms}}(x_{1,1}), d_2$
$x_{0,1}$
$\mu_{\text{ms}}(x_{1,2}), d_1$
$x_{2,3}$
$\mu_{\text{ms}}(x_{2,3}), d_1$
$x_{3,3}$
$\mu_{\text{ms}}(x_{0,1}), d_2$
$x_{1,2}$
$x_{2,4}$
$\mu_{\text{ms}}(x_{2,4}), d_2$
$x_{3,4}$
$\mu_{\text{ms}}(x_{1,2}), d_2$

$\tilde{\mathcal{X}}_0 = \{x_{0,1}\}$; $\tilde{\mathcal{X}}_1 = \{x_{1,1}, x_{1,2}\}$; $\tilde{\mathcal{X}}_2 = \{x_{2,1}, \dots, x_{2,4}\}$; $\tilde{\mathcal{X}}_3 = \{x_{3,1}, \dots, x_{3,4}\}$;

Figure 2.3: Scenario tree representation when considering 2 realizations of the uncertainty $\tilde{\mathcal{D}} = \{d_1, d_2\}$, a prediction horizon of $N = 3$ and a robust horizon of $N_{\text{robust}} = 2$.

The optimization problem that should be solved at each sampling time to obtain the multi-stage NMPC feedback policy $\mu_{\text{ms}}(\cdot)$ is formulated via (2.3) and (2.4) and given by:

$$\underset{\mu_{\text{ms}}(\cdot)}{\text{minimize}} \quad \ell_{\text{f}}(\mathcal{X}_N) + \sum_{k=0}^{N-1} \ell_{\text{cl}}(\mathcal{X}_k, \mu_{\text{ms}}), \tag{2.8a}$$

$$\text{subject to} \quad \tilde{\mathcal{X}}_0 = \{x_{\text{init}}\}, \tag{2.8b}$$

for all $k \in \mathbb{N}^+_{[N-1]}$:

$$\tilde{\mathcal{X}}_{k+1} = \mathcal{R}(\mathcal{X}_k, \tilde{\mathcal{D}}, \mu_{\text{ms}}) \tag{2.8c}$$

$$\tilde{\mathcal{U}}_k = \mathcal{Z}(\tilde{\mathcal{X}}_k, \mu_{\text{ms}}) \tag{2.8d}$$

$$\tilde{\mathcal{X}}_k \subseteq \mathcal{X}, \tag{2.8e}$$

$$\tilde{\mathcal{U}}_k \subseteq \mathcal{U}, \tag{2.8f}$$

$$g_{\text{cl}}(\tilde{\mathcal{X}}_k, \tilde{\mathcal{D}}, \mu_{\text{ms}}) \leq 0, \tag{2.8g}$$

$$\tilde{\mathcal{X}}_N \subseteq \mathcal{X}_f, \tag{2.8h}$$

where (2.8a) approximates (2.5a) based on the closed-loop stage cost $\ell(\cdot)$ and the terminal cost $\ell_{\text{f}}(\cdot)$. As can be seen from Fig. 2.2, the number of scenarios that need to be considered in the scenario tree grows exponentially with the prediction horizon which subsequently increases the complexity of (2.8). A common simplifying assumption is to consider that the tree branches only up to a given stage (usually called robust horizon) as portrayed in Fig. 2.3, which reduces the online computational requirements. While this simplification introduces further errors in the approximation of the reachable sets at stages beyond the robust horizon, it achieves good results in practice [19]. It is also possible to achieve stability and robust constraint satisfaction guarantees for a multi-stage MPC formulation [18, 28, 29], but its computational complexity grows exponentially with the dimension of the uncertainty space.

Despite the simplifications introduced in multi-stage NMPC to render the ideal robust model predictive control problem (2.5) tractable, (2.8) remains a challenging optimization problem. For this reason, approaches have been developed that completely remove the necessity to solve an optimization problem online for a certain class of MPC problems.

2.5 Explicit MPC

The basic idea of explicit MPC approaches is - as the name indicates - to derive an explicit expression for the solution of an MPC problem. Consider the discrete linear time-invariant (LTI) system:

$$x^+ = Ax + Bu, \tag{2.9}$$

where $A \in \mathbb{R}^{n_x \times n_x}$ is the system matrix, $B \in \mathbb{R}^{n_x \times n_u}$ is the input matrix and the pair (A, B) is controllable. The system description (2.9) can be for example derived by linearizing a system with minor uncertainties at an operating point.

Using a standard quadratic cost function, as is common in regularization and tracking tasks, the following constrained finite time optimal control problem with a horizon of N steps should be solved at each sampling time to obtain the MPC feedback law:

$$\begin{aligned}
&\underset{\mathbf{x},\mathbf{u}}{\text{minimize}} && x_N^T P x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k && \text{(2.10a)}\\
&\text{subject to} && x_0 = x_{\text{init}}, && \text{(2.10b)}\\
& && \text{for } k \in \mathbb{N}_{[N-1]}: && \text{(2.10c)}\\
& && \quad x_{k+1} = Ax_k + Bu_k, && \text{(2.10d)}\\
& && \quad x_k \in \mathcal{X}, && \text{(2.10e)}\\
& && \quad u_k \in \mathcal{U}, && \text{(2.10f)}\\
& && x_N \in \mathcal{X}_{\text{f}}, && \text{(2.10g)}
\end{aligned}$$

where $\mathbf{u} = [u_0^T, \dots, u_{N-1}^T]^T$ is a sequence of control inputs and $\mathbf{x} = [x_0^T, \dots, x_N^T]^T$ is the corresponding state trajectory. The weighting matrices in the objective function (2.10a) are $P \in \mathbb{R}^{n_x \times n_x}$, $Q \in \mathbb{R}^{n_x \times n_x}$ and $R \in \mathbb{R}^{n_u \times n_u}$ and chosen such that $P \succeq 0$ and $Q \succeq 0$ are positive semidefinite, and $R \succ 0$ is positive definite. The state, terminal and input constraints are bounded polytopic sets $\mathcal{X} := \{x \in \mathbb{R}^{n_x} \,|\, C_x x \leq c_x\}$, $\mathcal{X}_{\text{f}} := \{x \in \mathbb{R}^{n_u} \,|\, C_{\text{f}} u \leq c_{\text{f}}\}$ and $\mathcal{U} := \{u \in \mathbb{R}^{n_u} \,|\, C_u u \leq c_u\}$ defined by the matrices $C_x \in \mathbb{R}^{n_\mathcal{X} \times n_x}$, $C_f \in \mathbb{R}^{n_{\text{f}} \times n_x}$, $C_u \in \mathbb{R}^{n_\mathcal{U} \times n_u}$ and the vectors $c_x \in \mathbb{R}^{n_\mathcal{X}}$, $c_f \in \mathbb{R}^{n_{\text{f}}}$ and $c_u \in \mathbb{R}^{n_\mathcal{U}}$. The terminal cost defined by P as well as the terminal set $\mathcal{X}_{\text{f}}$ are usually chosen in such a way that stability of the closed-loop system and recursive feasibility of the optimization problem are guaranteed [30].

The MPC problem (2.10) is a quadratic program whose solution is a piecewise affine (PWA) function in the state space defined on polytopes. The optimal solution only depends on the current state x_{init} of the system [31]. Explicit MPC exploits this property by precomputing the solution of (2.10) and storing the resulting function that fully defines the MPC feedback law. The online evaluation of the explicit MPC law is then reduced to finding the polytopic region in which the system is currently located and applying the corresponding affine feedback law.

To obtain the PWA description of the solution of the MPC problem, (2.10) is reformulated as a multi-parametric programming problem as given in [31, 32] that only depends on the current system state x_{init}:

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \mathbf{u}^T F \mathbf{u} + x_{\text{init}}^T G \mathbf{u} + x_{\text{init}}^T H x_{\text{init}} && (2.11\text{a}) \\ \text{subject to} \quad & C_c \mathbf{u} \leq T x_{\text{init}} + c_c, && (2.11\text{b}) \end{aligned}$$

where $F \in \mathbb{R}^{Nn_u \times Nn_u}$, $G \in \mathbb{R}^{n_x \times Nn_u}$, $H \in \mathbb{R}^{n_x \times n_x}$, $C_c \in \mathbb{R}^{Nn_{\text{ineq}} \times Nn_u}$, $T \in \mathbb{R}^{Nn_{\text{ineq}} \times n_x}$, $c_c \in \mathbb{R}^{Nn_{\text{ineq}}}$ and n_{ineq} is the total number of inequalities in (2.10).

The PWA solution of the multi-parametric quadratic programming problem (2.11) [31] is given by:

$$\mu_{\text{exp}}(x_{\text{init}}) = \begin{cases} K_1 x_{\text{init}} + m_1, & \text{if} \quad x_{\text{init}} \in \mathcal{H}_1, \\ & \vdots \\ K_{n_{\text{r}}} x_{\text{init}} + m_{n_{\text{r}}}, & \text{if} \quad x_{\text{init}} \in \mathcal{H}_{n_{\text{r}}}, \end{cases} \tag{2.12}$$

with n_{r} regions, $K_i \in \mathbb{R}^{Nn_u \times n_x}$ and $m_i \in \mathbb{R}^{Nn_u}$. Each region $\mathcal{H}_i$ is described by a polytope

$$\mathcal{H}_i = \{x \in \mathbb{R}^{n_x} \mid H_i x \leq h_i\}, \quad i \in \mathbb{N}^+_{[n_r]}, \tag{2.13}$$

where $H_i \in \mathbb{R}^{c_i \times n_x}$ and $h_i \in \mathbb{R}^{c_i}$ describe the c_i half-spaces of the i-th region. The formulation (2.12) is defined on the bounded polytopic partition $\mathcal{X}_\Omega = \cup_{i=1}^{n_{\text{r}}} \mathcal{H}_i$ with $\text{int}(\mathcal{H}_i) \cap \text{int}(\mathcal{H}_j) = \emptyset$ for all $i \neq j$, where $\text{int}(\cdot)$ denotes the interior of a set.

However, explicit MPC can only be applied to systems of moderate size and complexity. The main drawback of the explicit MPC formulation is that the number of regions for an exact representation can grow exponentially with respect to the horizon and number of constraints [31], which might render computing the PWA function infeasible or might result in stark memory requirements. The memory needed to store the explicit MPC controller (2.12) can be approximated as

$$\text{MEM}_{\mu_{\text{exp}}} = \alpha_{\text{bit}} \left(\left(n_{\text{h}} \left(n_x + 1 \right) \right) + n_{\text{uf}} \left(n_x n_u + n_u \right) \right), \tag{2.14}$$

where n_{h} is the number of unique hyperplanes , n_{uf} is the number of unique feedback laws and α_{bit} is the memory necessary to store a real number. It is only required to consider the unique elements because many hyperplanes are shared by neighboring regions and the feedback laws can be identical for two or more polytopic regions. Since for the actual implementation of the explicit MPC law only the input of the first time step is needed, only the first n_u rows of K_j and m_j for $j \in \mathbb{N}^+_{[n_{\text{uf}}]}$ have to be stored which equals $n_x n_u + n_u$ numbers per unique feedback law. Some approaches try to counteract the massive memory requirements by finding a more efficient representation of the control law, e.g. by eliminating redundant regions [33] or by using different number representations [34], but these methods have a negligible impact.

Another major disadvantage of explicit MPC concerns the online application and results also from the exponential growth of the polytopic region. The effort required for identifying the region in which the system state currently resides in order to apply the correct affine feedback law can be significant. One method to reduce the complexity of the so-called point location problem is the usage of binary search trees [35–39] which introduce a tree structure where the nodes represent unique hyperplanes. At each node, it is checked on which side of the hyperplane the state is until a leaf node is reached. At the leaf node, a unique feedback law is identified and evaluated. This method renders the online computational time logarithmic in the number of regions, but precomputation times can be prohibitive or intractable for larger problems [40].

Since the application of explicit MPC is limited to systems of moderate size and complexity, despite avoiding online optimization, a less restricted approach based on artificial neural networks to obtain optimization-free MPC controllers will be presented in the next chapter.

Chapter 3

Artificial neural networks

The field of machine learning has rapidly evolved in the recent decade and enabled machines to excel in tasks that were thought of being only accessible to the human mind. The crucial factor for this fast development besides the improved hardware are the advances made in deep learning. Examples for the outstanding achievements are the successful categorization of high-resolution image data sets consisting of more than a million images into 1000 categories [41], learning to play Atari games by directly feeding the pixel output to a deep neural network [42] and the superhuman performance in the game of Go [43, 44], a highly complex strategic board game. In the following, the historical precursor of neural networks will be introduced in Section 3.1 before the structure and mathematical description of the deep neural networks used throughout this thesis is explained in Section 3.2. The basic learning paradigm - imitation learning - is presented in Section 3.3 that allows to adapt the parameters of a deep neural network such that a desired input-output behavior is obtained. In Section 3.4, it is presented how deep learning has been leveraged in the context of MPC and how imitation learning can be used in combination with robust MPC formulations to obtain broadly applicable explicit controllers.

3.1 Perceptron

The predecessor of the modern deep neural networks is the perceptron - a classifier - which was introduced in 1943 by McCulloch and Pitts [45]. The basic idea of the perceptron is to discriminate between two classes based on threshold logic. The perceptron, depicted in Fig. 3.1, provides a binary output based on a fixed threshold $\bar{b} \in \mathbb{R}$:

$$\lambda_{\text{p}}(x) = \begin{cases} 1, & \text{if } w^T x > \bar{b}, \\ 0, & \text{else}, \end{cases} \tag{3.1}$$

where $x \in \mathbb{R}^{n_x}$ is the input to the perceptron and $w \in \mathbb{R}^{n_x}$ is the weight vector. In 1957 Frank Rosenblatt developed a learning rule [46] - inspired by the model for biological learning proposed by Hebb [47] - to find the optimal weights for a perceptron. Despite the neurological inspiration, the derived classifier from (3.1) is just a linear one. The ability to handle more complex and nonlinear data sets is only obtained when at least a second layer of perceptrons is introduced.

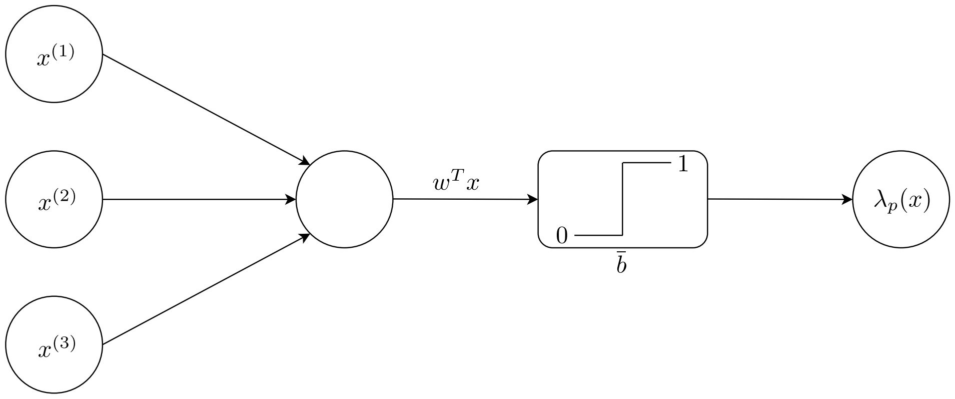


Figure 3.1: Perceptron with 3 inputs.

3.2 Feed-forward neural networks

A feed-forward neural network is defined as a sequence of layers, each consisting of several perceptrons (in this context also called neurons), which transforms input data in a highly nonlinear fashion. The neural network function is a mapping $\mu_{\mathrm{nn}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ of the form:

$$\mu_{\mathrm{nn}}(x; \theta, n_0, \ldots, n_{L+1}) = \lambda_{L+1} \circ \prod_{l=1}^{L} \left(\sigma_l \circ \lambda_l\left(z_{l-1}\right)\right), \tag{3.2}$$

combining in each (hidden) layer an affine function:

$$\lambda_l(z_{l-1}) = W_l z_{l-1} + b_l, \tag{3.3}$$

that processes the output of the previous layer z_{l-1}, with a nonlinear activation function $\sigma_l(\cdot)$. The symbol $\circ$ denotes a function composition, e.g.

$$\sigma_l \circ \lambda_l(z_{l-1}) = \sigma_l(\lambda_l(z_{l-1})),$$

and $\sum_{l=1}^{L}$ stands in this case for the L-time function composition. The output of each layer is defined as:

$$z_l = \sigma_l\left(W_l z_{l-1} + b_l\right), \tag{3.4}$$

for $l \in \mathbb{N}^{+}_{[L+1]}$ and as $z_0 = x$ for the initial input layer. The parameter

$$\theta = \{W_1, b_1, \ldots, W_{L+1}, b_{L+1}\}$$

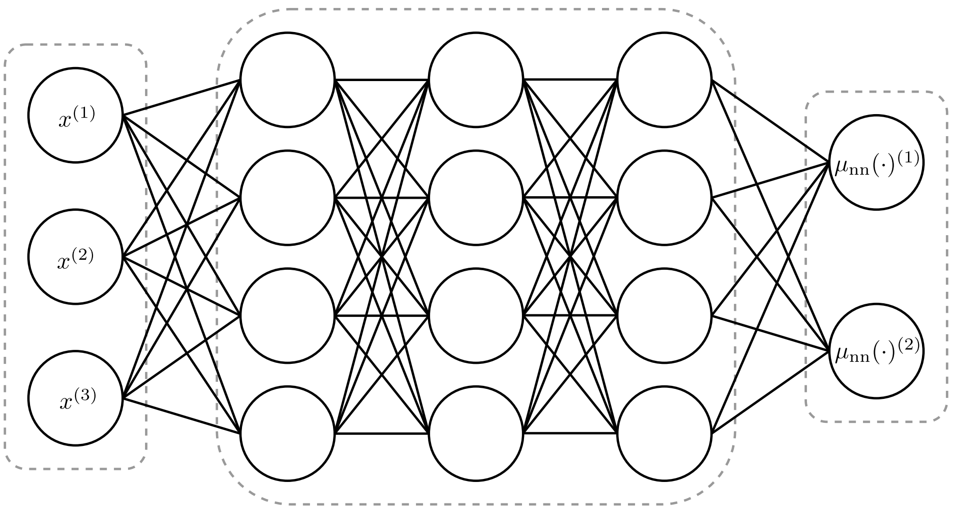

$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$\mu_{\text{nn}}(\cdot)^{(1)}$
$\mu_{\text{nn}}(\cdot)^{(2)}$

Figure 3.2: Deep neural network with $n_0 = 3$ inputs, $n_{L+1} = 2$ outputs and three hidden layers with $n_1 = n_2 = n_3 = 4$ neurons each.

contains all the weights $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and biases $b_l \in \mathbb{R}^{n_l}$ of the affine functions of each layer $l \in \mathbb{N}^+_{[L+1]}$. Each layer (3.3) of a deep neural network can be seen as a stack of perceptrons with

$$W_l = \begin{bmatrix} w_1 \\ \vdots \\ w_{n_l} \end{bmatrix} \text{ and } b_l = \begin{bmatrix} -\bar{b}_1 \\ \vdots \\ -\bar{b}_{n_l} \end{bmatrix},$$

which does provide a non-binary output that depends on the chosen activation function. The number of hidden layers is given by L and each layer contains n_l, $l \in \mathbb{N}^+_{[L]}$, neurons. If $L \geq 2$, $\mu_{\text{nn}}(\cdot)$ is described as a *deep* neural network and if $L = 1$ as a *shallow* neural network. A deep neural network with three hidden layers, four neurons per hidden layer, and three input and two output neurons is portrayed in Fig. 3.2. Common choices for the nonlinear activation function $\sigma(\cdot)$ are rectifier linear units (ReLUs), which compute the element-wise maximum between zero and the affine function of the current layer l:

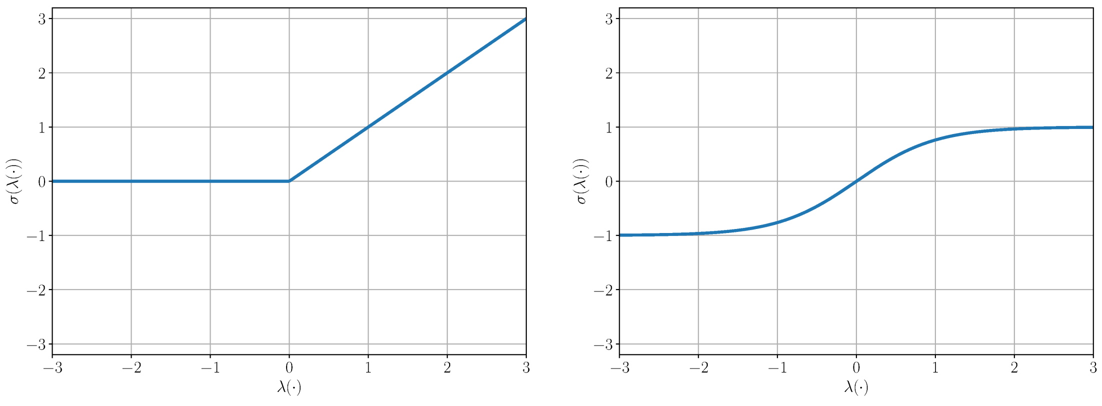
$$\text{ReLU}(\lambda_l(z_l)) = \max(0, \lambda_l(z_l)), \tag{3.5}$$

and the hyperbolic tangent:

$$\tanh(\lambda_l(z_l)) = \frac{e^{\lambda_l(z_l)} - e^{\lambda_l(z_l)}}{e^{-\lambda_l(z_l)} + e^{-\lambda_l(z_l)}}. \tag{3.6}$$

The two activation functions are visualized in Fig. 3.3.

In this thesis, only feed-forward neural networks of the form (3.2) are considered. Alternative topologies such as recurrent neural networks [48, 49], where inter-



(a) ReLU (b) tanh

Figure 3.3: Illustration of rectifier linear unit (a) and the hyperbolic tangent (b) which are two common choices for the activation function $\sigma(\cdot)$.

nal states exist that behave like a memory, and convolutional neural networks, that are mainly used in image recognition tasks [50, 51], are out of the scope of this work.

While the neural network structure (3.2) can potentially approximate any continuous function arbitrarily close by choosing appropriate parameters, as stated in various universal approximation theorems [52–58], no constructive approach to obtain the corresponding parameters is known. Therefore, data-based approaches are leveraged to compute (near-) optimal parameters, as presented in the next section.

3.3 Training artificial neural networks

The goal of any machine learning approach, and hence also deep learning, is to find a parametric description of a (potentially unknown) function $\mu_{\text{goal}} : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$. The function $\mu_{\text{goal}}(\cdot)$ can either represent a classification task, e.g. deciding what kind of animal can be seen on a picture, or a regression task like approximating a control policy. The only requirement in order to apply deep learning is that it is possible to evaluate the goal function to obtain input-output samples $(x, \mu_{\text{goal}}(x))$.

After fixing the structure of a feedforward neural network by choosing the number of hidden layers and the number of neurons in each hidden layer, the parameter θ needs to be chosen such that the resulting network can solve the task at hand. Strategies to optimize the neural network architecture, e.g. Bayesian optimization [59, 60], are not considered in this thesis. By evaluating $\mu_{\text{goal}}(\cdot)$ for n_{tr} different inputs, the data set

$$\mathcal{T} = \left\{(x_1, \mu_{\text{goal}}(x_1)), (x_2, \mu_{\text{goal}}(x_2)), \ldots, (x_{n_{\text{tr}}}, \mu_{\text{goal}}(x_{n_{\text{tr}}}))\right\}$$

is generated. The data set $\mathcal{T}$ is then used to define a loss function that allows to compute parameters for the neural network that result in a similar input-output behavior of the deep neural network in comparison to the original function. For regression tasks, the loss function is commonly defined as the mean squared error (MSE) between the output of the neural network and the output of the goal function:

$$\text{MSE}(\mathcal{T};\theta) = \sum_{i=1}^{n_{\text{tr}}} \left|\left|\mu_{\text{nn}}(x_i;\theta,\cdot) - \mu_{\text{goal}}(x_i)\right|\right|_2^2. \tag{3.7}$$

Due to the highly nonlinear nature of the networks, developing methods to obtain (near-) optimal weights and biases θ^* by solving

$$\theta^* = \arg\min_{\theta} \quad \text{MSE}(\mathcal{T};\theta) \tag{3.8}$$

has been a challenging task.

The backbone of modern approaches to solve (3.8) are based on backpropagation. Backpropagation uses the loss (3.7) and the chain-rule of differentiation to compute the gradients of the weights and biases with respect to the loss (3.7). The gradient information is then used in gradient-based optimization algorithms to update the parameters of the neural network. The general principles of backpropagation were already explored in the Sixties, see e.g. [61], but the first work in which the use of the backpropagation algorithm was explicitly proposed for the adaption of the parameters of a neural network was published in 1974 [62] and further popularized in 1986 [63].

The above described training procedure, usually called imitation learning, illustrates why obtaining an exact neural network representation of the function to be approximated is usually intractable, even if the expressiveness of the neural network structure is theoretically sufficient. First, the parameters are computed from samples, meaning that the goal function is only approximated point-wise. Second, due to the highly nonlinear nature of artificial neural networks and the high-dimensional solution space, i.e. the parameter space, solving (3.8) globally is usually intractable and computationally less expensive gradient-based methods are exploited.

However, the near-optimal parameters obtained by leveraging the gradient-based methods provide good results in practice and thus deep learning has been used in various ways in the context of MPC.

3.4 Deep learning-based MPC

The recent successes of deep learning have sparked a surge in research interest in the field of control engineering on how to leverage the capabilities of deep neural networks. Hence, many approaches, despite being very different in nature, have been coined (deep) learning-based MPC. In order to avoid any confusion, an overview of model predictive control strategies that use deep learning is provided before presenting the in this thesis mainly used approach of obtaining deep neural networks that imitate MPC laws. For a detailed survey of general machine learning-based approaches in the field of robust control, see [64].

3.4.1 Overview of deep learning-based MPC strategies

One class of methods apply deep learning for system identification [65] tasks. The major of advantages of these approaches is that the cumbersome process of developing a first-principles model is avoided. Instead, the ubiquitous sensing capabilities of modern plants and therefrom resulting data can be used to obtain an accurate system model in form of an artificial neural network that either represents the whole model or difficult to model (nonlinear) dynamics [66, 67]. The identified model is then used as the prediction model in a predictive control scheme. However, the usage of Gaussian Processes [68] for system identification purposes is more common than the usage of neural networks because Gaussian processes inherently provide uncertainty measures which can be leveraged in a robust or stochastic MPC formulation [69, 70]. However, in this work we are interested in leveraging the strengths of deep learning to obtain deep neural network controllers.

When deep neural networks are considered to directly approximate the solution of MPC problems, two main approaches for obtaining learning-based neural network controllers can be distinguished. The first approach is imitation learning, where the neural network mimics the behavior of a demonstrator, often an optimization-based controller like MPC [5, 71, 72]. The second method is reinforcement learning, where the controller interacts with the system and adjusts its policy based on a received reward to find an optimal policy [9, 73–77]. The results obtained with both methods showed that the learning-based controllers exhibit competitive performance in comparison to optimization-based control methods while only requiring a fraction of the computational and memory requirements [5, 78].

Despite such promising results, obtaining guarantees for the safety and performance are central challenge because the learning-based controller do not necessarily inherit the same properties from the MPC demonstrator due to approximation errors. Generally, three approaches for analyzing safety properties of neural network controlled systems are followed.

The first class of approaches is based on leveraging probabilistic methods to bound the approximation error of the neural network controller with respect to the optimal controller [72, 79] or to directly evaluate the closed-loop performance [4]. These approaches can also be applied for nonlinear systems, but no deterministic safety guarantees can be provided. One advantage of these probabilistic methods is, that they do not require an additional measure to apply the neural network controllers online and hence are also explored in this thesis.

The second class of approaches is combining the control signal of a neural network controller with optimization-based methods to guarantee safety, e.g. using projection onto the feasible set [71, 80], using an MPC-based backup controller [81] or by taking the output of the neural network controller to warm start an active-set solver [78]. While these approaches guarantee the deployment of safe control inputs, they still require to solve a potentially challenging optimization problem online, at least when the neural network controller provides an unsafe control input. Since one goal of this thesis is to develop methods that enable the application of sophisticated control algorithms without online optimization, these kind of approaches are not considered.

The third class of approaches, to which some techniques proposed in this thesis belong to, is the offline deterministic verification of a learned neural network controller. Most approaches in this category are based on output-range analysis techniques, see [82] for an overview. Output range analysis methods approximate the set of outputs that a neural network generates for a given set of inputs. Output range analysis has been originally used in classification tasks to robustify the decisions against adversarial perturbations [83, 84] and were adapted in the control field to verify closed-loop properties such as reachability of target sets and to identify robust positive invariant sets of the closed loop [6, 84, 85]. An alternative deterministic verification approach is based on directly comparing the piece-wise affine functions of the explicit solution of certain MPC problems and an approximate neural network controller with rectified linear unit activations. If the maximum approximation error and the difference in Lipschitz constants between the MPC solution and the neural network approximation is sufficiently small, stability and safety within a region of interest can be guaranteed [86]. The main advantage of these approaches is that the neural network controller can be directly applied in the closed-loop after the verification procedure with guarantees on safety and performance without any additional measures. The main disadvantage is that the offline computational load can be significant, as it requires solving complex and often non-convex optimization problems. However, before any of the safety analysis methods can be applied, the neural network controllers need to be designed, e.g. by following the approach presented in the subsection.

3.4.2 Approximating MPC laws with deep neural networks

The usage of artificial neural networks as explicit MPC-like controllers has already been proposed as early as in the mid 90-ies [87, 88], but technical limitations have hindered the broad adoption of this approach. Apart from very restricted computational resources the lack of specialized deep learning toolboxes - such as Keras [89], Tensorflow [90] and PyTorch [91] - which enable the straight-forward usage of state-of-the-art solvers [92, 93] were far from being available. Modern-day technology facilitated the process of obtaining a deep neural network controller that closely approximates the behavior of a robust MPC controller. The approximate deep learning-based controllers are obtained following three steps.

First, one needs to design the MPC for the system at hand, e.g. a multi-stage NMPC formulation (2.8) as done throughout this thesis. It is recommended, but not necessary, to a priori consider the approximation error of the learning-based controllers by modelling it as part of the uncertainties $d \in \mathcal{D}$ as proposed in [72]. The resulting MPC is then used to generate the training data set

$$\mathcal{T} = \{(x_{\text{init},1}, \mu_{\text{ms}}(x_{\text{init},1})), (x_{\text{init},2}, \mu_{\text{ms}}(x_{\text{init},2})), \ldots, (x_{\text{init},n_{\text{tr}}}, \mu_{\text{ms}}(x_{\text{init},n_{\text{tr}}}))\} \tag{3.9}$$

by solving (2.8) for many feasible initial states $x_{\text{init},i} \in \mathcal{X}$, $i \in \mathbb{N}^{+}_{[n_{\text{tr}}]}$. Note that collecting the samples from closed-loop trajectories usually results in a more efficient learning [8, 78].

The second step concerns finding control parameters θ of the deep neural network $\mu_{\text{nn}}(x;\theta,\cdot)$ that minimize the approximation error defined in terms of the MSE (3.7) based on the training data set $\mathcal{T}$, i.e.

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \left|\left|\mu_{\text{nn}}(x_{\text{init},i};\theta,\cdot) - \mu_{\text{ms}}(x_{\text{init},i})\right|\right|^2. \tag{3.10}$$

The optimization problem (3.10) is then usually solved by applying mini-batch stochastic gradient descent or related approaches.

The third and final step is the implementation of the neural network on the control hardware. One central advantage of neural network controllers is their simplicity as the evaluation consists only of matrix-vector multiplications, vector additions and simple nonlinear computations for the activation functions. This enables the straightforward deployment of advanced MPC strategies for systems with extremely high control sampling rate, such as power electronics [94] and plasma jets [95], and on hardware with very limited computational abilities such as microcontrollers [4, 96] and field programmable gate arrays [94, 97, 98].

In the remainder of this thesis, neural network controllers that were developed as described above will be investigated. Theoretical connections between MPC laws and neural networks will be established, approaches to modify the weights such that an improved closed-loop performance is obtained and methods to derive probabilistic and deterministic guarantees regarding the online operation, without installing any additional safety measures, will be presented.

Part II

Deep learning-based control for linear systems

Chapter 4

Exact representation of model predictive control laws via deep learning

A central motivation for the usage of deep neural network networks as explicit controllers is that an exact representation of MPC laws can be achieved not only in the limit - as stated in the universal approximation theorem [52] - but also in a constructive way, as will be shown in this chapter.

First, the expressive capabilities of deep neural networks with ReLU activations will be investigated in Section 4.1 before showing how these networks can exactly represent MPC laws in Section 4.2. The contributions of this chapter are based on the findings published in [5].

4.1 Expressive capabilities of deep neural networks

In Section 2.5, it was shown that the MPC law for a linear system often constitutes a piecewise affine function in the state space. The same is true for deep neural networks that use ReLUs (3.5) as the activation function.

Lemma 4.1. *[99] Every neural network $\mu_{\text{nn}}(x;\theta,n_0,\ldots,n_{L+1})$ with input $x \in \mathbb{R}^{n_x}$ defined as in* (3.2) *with ReLUs* (3.5) *as activation functions and $n_l \geq n_x$ for all $l \in \mathbb{N}_{[L]}^{+}$ represents a piecewise affine function. In addition, a lower bound on the maximum number of affine regions that the neural network represents is given by the following expression:*

$$\left(\prod_{l=1}^{L-1} \left\lfloor \frac{n}{n_x} \right\rfloor^{n_x}\right) \sum_{j=0}^{n_x} \binom{L}{j},$$

where $n = \min\{n_1,\ldots,n_L\}$.

Proof of Lemma 4.1. The neural network $\mu_{\text{nn}}(x;\theta,n_0,\ldots,n_{L+1})$ is a piecewise affine function because it only contains compositions of affine transformations with a piecewise affine function (ReLUs). For the derivation of the maximum number of regions, see [99]. □

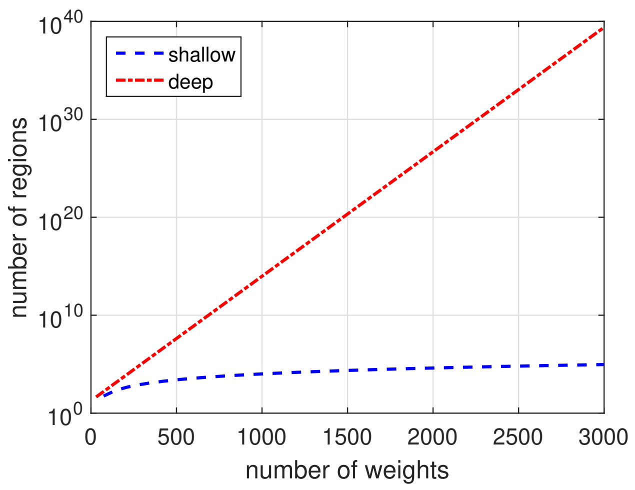


Figure 4.1: Number of regions with respect to the number of weights a neural network can represent. The parameters for this plot were chosen to $n_x = n_0 = 2, n_1, \dots, n_L = 10$, $L \in \mathbb{N}_{[50]}$ and $n_u = n_{L+1} = 4$. ©[2020] IEEE.

Lemma 4.1 gives clear insights about why deep networks are a promising candidate to obtain explicit controllers and why they obtain a better performance to approximate complex functions when compared to shallow networks as often observed in practice. In particular, Lemma 4.1 implies that the number of affine regions that a neural network can represent grows exponentially with the number of layers L as long as the number of neurons in each hidden layer n_l, $l \in \mathbb{N}_{[L]}^{+}$, is not smaller than the number of inputs n_x. The bound of Lemma 4.1 can be slightly improved if $n_l \geq 3n_x$ for all $l \in \mathbb{N}_{[L]}^{+}$ as shown in [100]. Despite the fact that the expressive capacity, i.e. the maximum number of linear regions that a deep neural network can theoretically represent, is usually not reached in practice [101–104], deep neural networks achieve in general better learning results in comparison to shallow neural networks with comparable expressive capacity [103].

The number of parameters contained in θ that are necessary to fully describe the neural network $\mu_{\text{nn}}(x; \theta, n_0, \dots, n_{L+1})$ are determined by the dimensions of the weights and biases at each layer. Assuming that storing each number requires α_{bit} bits, the total amount of memory necessary to store the neural network $\mu_{\text{nn}}(x; \theta, n_0, \dots, n_{L+1})$ can be computed as:

$$\text{MEM}_{\mu_{\text{nn}}} = \alpha_{\text{bit}} \sum_{l=1}^{L+1} \left((n_{l-1} + 1)\, n_l \right). \tag{4.1}$$

Since $\text{MEM}_{\mu_{\text{nn}}}$ only grows linearly with respect to the number of layers L, deep ReLU networks can represent exponentially many more linear regions than shallow ones for a fixed amount of memory. This fact can be clearly seen in Fig. 4.1. We believe that this observation, while somewhat obvious, is a very powerful result with important implications in control theory and constitutes one of the central motivations for the usage of deep neural networks as explicit controllers.

4.2 Exact representation of MPC laws via deep neural networks

This section shows how to design a deep neural network that can exactly represent an explicit MPC feedback law (2.12) of the form $\mu_{\text{exp}} : [0,1]^{n_x} \rightarrow \mathbb{R}^{+,n_u}$ by only mapping the current state x_{init} to the first optimal control input u_0^*. Considering only the control input of the first time step is sufficient, because after its application a new control input trajectory $\mathbf{u}^*$ is computed in the MPC setting.

We make use of two lemmas to derive specific bounds for a deep neural network to be able to exactly represent any explicit MPC law $\mu_{\text{exp}}(\cdot)$. The following lemma from [105] is used.

Lemma 4.2. *Every scalar piecewise affine function $f_{\text{PWA}}(x) : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ can be written as the difference of two convex piecewise affine functions:*

$$f_{\text{PWA}}(x) = \gamma(x) - \eta(x), \tag{4.2}$$

where $\gamma(x) : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ has r_γ regions and $\eta(x) : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ has r_η regions.

Proof of Lemma 4.2. See [105] or [106]. □

The following Lemma, presented in [107], gives specific bounds for the structure that a deep neural network should have to be able to exactly represent a convex piecewise affine function.

Lemma 4.3. *A convex piecewise affine function $f_{\text{PWA}} : [0,1]^{n_x} \rightarrow \mathbb{R}^+$ defined as the point-wise maximum of N_{aff} affine functions:*

$$f(x) = \max_{i \in \mathbb{N}^+_{[N_{\text{aff}}]}} f_i(x),$$

can be exactly represented by a deep ReLU network with width $n_l = n_x + 1$ for all $l \in \mathbb{N}^+_{[L]}$ and depth $L = N_{\text{aff}}$.

Proof of Lemma 4.3. See Theorem 2 from [107]. □

One of the main contributions of this chapter is given in the following theorem, which states that any explicit MPC law of the form (2.12) with $\mu_{\text{exp}}(x) : [0,1]^{n_x} \rightarrow \mathbb{R}^{+,n_u}$ can be represented by a deep ReLU neural network with a predetermined size.

Theorem 4.1. *There always exist parameters $\theta_{\gamma,i}$ and $\theta_{\eta,i}$ for $2n_u$ deep ReLU neural networks with depth $r_{\gamma,i}$ and $r_{\eta,i}$, $i \in \mathbb{N}^+_{[n_u]}$, and width $n_j = n_x + 1$ for all $j \in \mathbb{N}^+_{[r_{\gamma,i}]}$ and $j \in \mathbb{N}^+_{[r_{\eta,i}]}$, respectively, such that the vector of neural networks defined by*

$$\begin{bmatrix} \mu_{\text{nn}}(x; \theta_{\gamma,1}, n_0, \ldots, n_{r_{\gamma,1}+1}) - \mu_{\text{nn}}(x; \theta_{\eta,1}, n_0, \ldots, n_{r_{\eta,1}+1}) \\ \vdots \\ \mu_{\text{nn}}(x; \theta_{\gamma,n_u}, n_0, \ldots, n_{r_{\gamma,n_u}+1}) - \mu_{\text{nn}}(x; \theta_{\eta,n_u}, n_0, \ldots, n_{r_{\eta,n_u}+1}) \end{bmatrix} \tag{4.3}$$

can exactly represent an explicit MPC law $\mu_{\text{exp}}(x) : [0,1]^{n_x} \rightarrow \mathbb{R}^{+,n_u}$.

Proof of Theorem 4.1. Every explicit MPC law $\mu_{\text{exp}}(x) : [0,1]^{n_x} \to \mathbb{R}^{+,n_u}$ can be split into one explicit MPC law per output dimension:

$$\mu_{\text{exp}}^{(i)}(x) : [0,1]^{n_x} \to \mathbb{R}^{+} \quad \forall i \in \mathbb{N}_{[n_u]}^{+}. \tag{4.4}$$

Applying Lemma 4.2 to all n_u MPC laws, each one of them can be decomposed into two convex scalar PWA functions:

$$\mu_{\text{exp}}^{(i)}(x) = \gamma_i(x) - \eta_i(x) \quad \forall i \in \mathbb{N}_{[n_u]}^{+}, \tag{4.5}$$

where each $\gamma_i(x)$ and each $\eta_i(x)$ are composed of r_{γ_i} and r_{η_i} affine regions. The explicit MPC law $\mu_{\text{exp}}(x) : [0,1]^{n_x} \to \mathbb{R}^{+,n_u}$ can thus be vectorized as

$$\mu_{\text{exp}}(x) = \begin{bmatrix} \gamma_1(x) - \eta_1(x) \\ \vdots \\ \gamma_{n_u}(x) - \eta_{n_u}(x) \end{bmatrix}. \tag{4.6}$$

According to Lemma 4.3, it is always possible to find parameters $\theta_{\gamma,i}, \theta_{\eta,i}$ for deep ReLU networks with width $n_l = n_x + 1$ for all $l \in \mathbb{N}_{[L]}^{+}$, depth L not larger than $r_{\gamma,i}$ and $r_{\eta,i}$, respectively, that can exactly represent the scalar convex functions $\gamma_i(x)$ and $\eta_i(x)$. This holds because any convex affine function with N_{aff} regions can be described as the point-wise maximum of N_{aff} scalar affine functions. This means that each component of the transformed explicit MPC law can be written as:

$$\gamma_i(x) - \eta_i(x) = \mu_{\text{nn}}(x; \theta_{\gamma,i}, n_0, \ldots, , n_{r_{\gamma,i}+1}) - \mu_{\text{nn}}(x; \theta_{\eta,i}, n_0, \ldots, n_{r_{\eta,i}+1}), \tag{4.7}$$

for all $i \in \mathbb{N}_{[n_u]}^{+}$. Substituting (4.7) in (4.6) results in (4.3). □

Theorem 4.1 requires that the piecewise affine function maps the unit hypercube to the space of non-negative real numbers. Any explicit MPC law (2.12) can be written in this form, as long as the invertible affine transformations defined in Assumption 4.1 exist. This result is formalized in Corollary 4.1.

Assumption 4.1. *There exist two invertible affine transformations $\mathcal{A}_x : \mathcal{X}_\Omega \to [0,1]^{n_x}$ and $\mathcal{A}_u : \mathcal{U} \to \mathbb{R}^{+,n_u}$ for an explicit control law* (2.12) *$\mu_{\text{exp,orig}} : \mathcal{X}_\Omega \to \mathcal{U}$ such that*

$$\mu_{\text{exp,orig}}(x) = \mathcal{A}_u^{-1} \circ \mu_{\text{exp}}(\hat{x}), \tag{4.8}$$

where $\hat{x} = \mathcal{A}_x \circ x$. The affine transformations $\mathcal{A}_x$ and $\mathcal{A}_u$ always exist, when $\mathcal{X}_\Omega$ and $\mathcal{U}$ are compact sets, as it is standard in control applications.

Corollary 4.1. *If for a given explicit MPC solution* (2.12) *$\mu_{\text{exp,orig}} : \mathcal{X}_\Omega \to \mathcal{U}$, there exist two invertible affine transformations such that Assumption 4.1 holds, then Theorem 4.1 can be applied to the transformed MPC solution $\mu_{\text{exp}}(\hat{x}) : [0,1]^{n_x} \to \mathbb{R}^{+}, n_u$ (4.8). Hence, such an explicit MPC solution of the form* (2.12) *can be exactly represented by two invertible affine transformations and $2n_u$ deep ReLU networks with width and depth as defined in Theorem 4.1.*

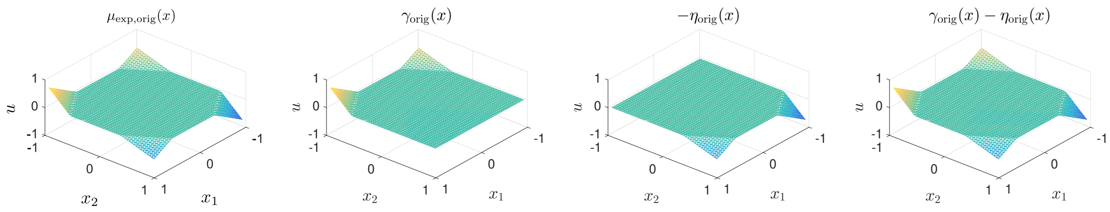


Figure 4.2: PWA explicit MPC law $\mu_{\text{exp,orig}}(x)$ (left plot). Decomposition of $\mu_{\text{exp,orig}}(x)$ into convex function $\gamma_{\text{orig}}(x) = \mathcal{A}_u^{-1} \circ \gamma(\hat{x})$ and concave function $-\eta_{\text{orig}}(x) = -\mathcal{A}_u^{-1} \circ \eta(\hat{x})$ with $\hat{x} = \mathcal{A}_x \circ x$ (middle plots). The resulting exact representation $\gamma_{\text{orig}}(x) - \eta_{\text{orig}}(x) = \mathcal{A}_u^{-1} \circ (\mu_{\text{nn}}(\hat{x}; \theta_\gamma, n_0, \dots, n_{r_\gamma}) - \mu_{\text{nn}}(\hat{x}; \theta_\eta, n_0, \dots, , n_{r_\eta}))$ via two deep neural networks is shown on the right plot. ©[2020] IEEE.

The proof presented in [106] for the decomposition of a PWA function into the difference of two PWA functions is constructive, which means that Theorem 4.1 gives explicit bounds for the construction of neural networks that can exactly represent any explicit MPC of the form (2.12) with $\mu_{\text{exp}}(x) : [0,1]^{n_x} \to \mathbb{R}^{+,n_u}$, considering only the first step of the optimal control input sequence.

We illustrate Theorem 4.1 with a small example of an oscillator with the dynamics (2.9) and the discrete system matrices

$$A = \begin{bmatrix} 0.5403 & 0.8415 \\ 0.8415 & 0.5403 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} -0.4597 \\ 0.8415 \end{bmatrix}.$$

We chose the tuning parameters for (2.10) to $P = 0$, $R = 1$, $Q = 2I$ and the horizon to $N = 1$. The state constraints are given by $|x_i| \leq 1$ for $i \in \mathbb{N}_{[2]}^{+}$ and input constraints by $|u| \leq 1$. We used the toolbox MPT3 [108] to compute the explicit MPC controller which has 5 regions and is illustrated in the left plot of Fig. 4.2. By applying two invertible affine transformations (4.8), the algorithm given in [106] is used to decompose the explicit MPC controller into the convex function $\gamma_{\text{orig}}(x) = \mathcal{A}_u^{-1} \circ \gamma(\hat{x})$ and the concave function $-\eta_{\text{orig}}(x) = -\mathcal{A}_u^{-1} \circ \eta(\hat{x})$ with $\hat{x} = \mathcal{A}_x \circ x$, depicted in the middle plots of Fig. 4.2. Both functions consist of $r_\gamma = r_\eta = 3$ regions. According to Theorem 4.1, two neural networks $\mu_{\text{nn}}(\hat{x}; \theta_\gamma, 2, 3, 3, 3, 1)$ and $\mu_{\text{nn}}(\hat{x}; \theta_\eta, 2, 3, 3, 3, 1)$ with widths $n_i = n_x + 1 = 3$ for all $i \in \mathbb{N}_{[r_\gamma]}^{+}$ and $i \in \mathbb{N}_{[r_\eta]}^{+}$, respectively, and depths $r_\gamma = r_\eta = 3$ are used to represent the two convex functions. The parameter values of the networks θ_γ and θ_η are computed as the minimizers of the mean squared error defined by:

$$\theta_\gamma = \underset{\theta_\gamma}{\text{argmin}} \quad \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} ||\mu_{\text{nn}}(\hat{x}_i; \theta_\gamma, n_0, \dots, n_{r_\gamma+1}) - \gamma(\hat{x}_i)||_2^2, \tag{4.9}$$

based on $n_{\text{tr}} = 1000$ randomly chosen sampling points, collected in the set $\mathcal{T}_x = \{x_1, x_2, \dots, x_{n_{\text{tr}}}\}$, for the functions $\gamma(\hat{x})$ (and analogously for $\eta(\hat{x})$). The learned representation of the neural networks

$$\gamma_{\text{orig}}(x) - \eta_{\text{orig}}(x) = \mathcal{A}_u^{-1} \circ (\mu_{\text{nn}}(\hat{x}; \theta_\gamma, n_0, \dots, n_{r_\gamma+1}) - \mu_{\text{nn}}(\hat{x}; \theta_\eta, n_0, \dots, n_{r_\eta+1}))$$

is shown in the right plot of Fig. 4.2, which is the same function as the original explicit MPC controller. The training procedure is considered finished when the maximum error:

$$e_{\text{prox}} = \max_{\hat{x} \in \hat{\mathcal{T}}_x} \quad |\mu_{\text{exp}}(\hat{x}) - (\gamma(\hat{x}) - \eta(\hat{x}))|,$$

where $\hat{\mathcal{T}}_x = \{\mathcal{A}_x \circ x_1, \mathcal{A}_x \circ x_2, \ldots, \mathcal{A}_x \circ x_{n_{\text{tr}}}\}$, is less than 0.001, which we consider to be an exact representation of the transformed explicit MPC law. The study of the sample complexity of random sampling points n_{tr} that are necessary to obtain a given error e_{prox} is an interesting research topic, but it is out of the scope of this thesis.

This chapter showed that explicit MPC laws (2.12) can be exactly represented by neural networks with ReLU activations and that deep neural networks might be a very efficient way to do so in terms of memory footprint and evaluation complexity. In the next chapter we will analyze the performance of deep neural networks in comparison to other approximate explicit MPC techniques.

Chapter 5

Efficient approximation of MPC contral laws via deep learning

The previous chapter outlined two main connections between deep learning and explicit MPC. The first one is that deep neural networks can exactly represent an explicit MPC law, and not only approximate it arbitrarily well for an increasing number of neurons, as it is known from universal approximation theorems such as [55]. The second connection is that, as shown in Lemma 4.1, the number of linear regions that deep neural networks can represent grows exponentially with the number of layers.

While it is possible to know the structure of a network that can exactly represent a given explicit MPC function, constructing such a network may be prohibitive in practice due to the potentially large number of required hidden layers. We therefore believe that the use of deep networks to achieve efficient approximations - instead of exact representations - is the more promising idea.

In this chapter, we present existing approximate explicit MPC methods in Section 5.1, before highlighting the efficacy of deep neural networks as approximate MPC laws by comparing it to other approximate explicit MPC approaches in Section 5.2. The results of this chapter are based on the publication [5].

5.1 Approximate explicit MPC

As pointed out in Section 2.5, the main drawback of exact explicit MPC is the potentially exponential growth of polyhedral regions with respect to the number of constraints and the horizon resulting in a demanding, potentially intractable offline computation phase [40]. And if the explicit solution can be computed, the memory footprint can be large and solving the point location problem in order to apply the correct feedback law can be very challenging. Several methods have been developed to mitigate these issues.

Apart from methods that find an optimized exact representation [33, 34], which in most cases provide only minor simplifications, approaches that try to find efficient approximations of the exact solution, see [109] for a review, allow to more significantly reduce the complexity of the resulting controller. The majority of the approximate explicit MPC methods try to define a more tractable partitioning - in comparison to standard explicit MPC (2.12) - on which the affine feedbacks are defined by using simplices [36, 110–112], lattice representation [113], a hypercubical partitioning [37] or using the simpler partitioning of an explicit controller with a short horizon whose affine feedback laws are adapted such that it closely approximates the performance of the MPC law with a long horizon [114].

All of the above mentioned approaches have in common that they mitigate the drawbacks of explicit MPC, but do not provide a definitive solution to them. The occurrence of highly complex partitionings and the necessity of solving the point location problem can be completely removed if a parametric approximation over the full state space is used as in [115], where a stabilizing polynomial approximation of the MPC law is proposed, or in [88, 116], where approximations based on neural networks are presented. Inspired by the absence of the point location problem for parametric approaches and the representative capabilities of deep neural networks, illustrated in Section 4.1, we will highlight why using deep neural networks is a powerful method to obtain resource-efficient and easy-to-apply approximate MPC laws.

5.2 Efficacy of deep learning-based explicit MPC

The potential of deep neural networks as an approximate explicit MPC approach is illustrated with a simulation example modified from [117] and the classic example of the inverted pendulum on a cart. The MPC problems can be expressed as a linear MPC problem with a quadratic objective and polytopic constraints (2.10) and the goal is in both cases to steer the system to the origin. We will compare the proposed deep learning based MPC approach to three different approximate explicit MPC methods and to the exact explicit MPC in terms of closed-loop performance and memory efficiency.

5.2.1 Approximation methods

The goal of approximate explicit MPC methods is to imitate the closed-loop behavior of the exact MPC as closely as possible while reducing the effort in offline precomputation and online application. We consider as the baseline an MPC formulation with a prediction horizon of N_{max} and use the resulting control law $\mu_{\text{exp},N_{\text{max}}}(\cdot)$ to generate the data set

$$\mathcal{T} = \{(x_{\text{init},1}, \mu_{\text{exp},N_{\text{max}}}(x_{\text{init},1})), (x_{\text{init},2}, \mu_{\text{exp},N_{\text{max}}}(x_{\text{init},2})), \dots, (x_{\text{init},n_{\text{tr}}}, \mu_{\text{exp},N_{\text{max}}}(x_{\text{init},n_{\text{tr}}}))\}$$

required to derive the approximate methods which are presented in the following.

The first two considered approximation methods both use artificial neural networks (3.2) as presented in Section 3.4.2. However, we distinguish here between *shallow* neural networks with only a single hidden layer and *deep* neural networks with several hidden layers to emphasize the greater expressiveness of deep neural networks, as expressed in Lemma 4.1. The optimal parameters for the neural networks are obtained by solving:

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{n_{\text{tr}}} \left|\left|\mu_{\text{nn}}(x_{\text{init},i}, \theta, \cdot)) - \mu_{\text{exp},N_{\text{max}}}(x_{\text{init},i})\right|\right|_2^2, \tag{5.1}$$

and the memory requirements for storing the resulting controllers is given by (4.1).

Table 5.1: Summary of algorithms used, including exact explicit solution $\mu_{\text{exp},N}$ and the approximations methods. ©[2020] IEEE.

Method	Param.	Explanation
$\mu_{\text{exp},N_{\text{max}}}(x)$	N_{max}	prediction horizon
$\mu_{\text{nn},n,L}(x;\theta)$	θ	weights and biases $\{W_l, b_l\}$ for all layers
	n	neurons in each hidden layer
	L	number of hidden layers
$\mu_{\text{poly},p}(x;\theta_{\text{poly}})$	p	degree of the polynomial
	θ_{poly}	coefficients $\{a_{1,0,\dots,0},\dots,a_{n_u,p,\dots,p}\}$ for all terms
$\mu_{\text{synth},N}(x;\theta_{\text{synth}})$	N	prediction horizon
	θ_{synth}	affine transformations $\{K_i, m_i\}$ for all regions

The first non-deep learning approach to approximate the explicit controller uses multi-variate polynomials of the form $\mu_{\text{poly},p} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ with degree p:

$$\mu_{\text{poly},p}(x;\theta_{\text{poly}}) = \begin{bmatrix} \sum\limits_{i_1=0}^{p} \cdots \sum\limits_{i_{n_x}=0}^{p} \left(a_{1,i_1,\dots,i_{n_x}} \prod_{j=1}^{n_x} x_j^{i_j} \right) \\ \vdots \\ \sum\limits_{i_1=0}^{p} \cdots \sum\limits_{i_{n_x}=0}^{p} \left(a_{n_u,i_1,\dots,i_{n_x}} \prod_{j=1}^{n_x} x_j^{i_j} \right) \end{bmatrix} \tag{5.2}$$

where $\theta_{\text{poly}} = \{a_{1,0,\dots,0},\dots,a_{n_u,p,\dots,p}\}$ contains all coefficients. The coefficients of the polynomials are computed by solving:

$$\underset{\theta_{\text{poly}}}{\text{minimize}} \; \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} ||\mu_{\text{poly},p}(x_{\text{init},i};\theta_{\text{poly}}) - \mu_{\text{exp},N_{\text{max}}}(x_{\text{init},i})||^2. \tag{5.3}$$

The memory footprint of a multi-variate polynomial is given by

$$\text{MEM}_{\mu_{\text{poly},p}} = \alpha_{\text{bit}} n_u (p+1)^{n_x}. \tag{5.4}$$

The second non-deep learning method is similar to the approach in [114]. We use the partitioning of an explicit MPC law with a shorter horizon $N \leq N_{\text{max}}$ - and therefore less regions that define the resulting piecewise affine function - and adapt the parameters

$$\theta_{\text{synth}} = \{K_1, m_1, \dots, K_{n_r}, m_{n_r}\}$$

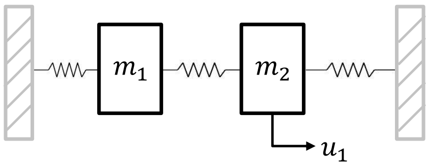


Figure 5.1: Chain of masses connected via springs. ©[2020] IEEE.

such that the performance of the explicit MPC law defined with a horizon of N_{max} is approximated by solving the following optimization problem:

$$\underset{\theta_{\text{synth}}}{\text{minimize}} \ \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} ||\mu_{\text{synth},N}(x_{\text{init},i};\theta_{\text{synth}}) - \mu_{\text{exp},N_{\text{max}}}(x_{\text{init},i})||^2. \tag{5.5}$$

The optimized formulations $\mu_{\text{synth},N} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ are then given by:

$$\mu_{\text{synth},N}(x;\theta_{\text{synth}}) = \begin{cases} K_1 x + m_1, & \text{if} \quad x \in \mathcal{H}_1, \\ & \vdots \\ K_r x + m_r, & \text{if} \quad x \in \mathcal{H}_r. \end{cases} \tag{5.6}$$

The memory footprint of the optimized explicit MPC can be estimated as done for the standard explicit MPC (2.14). All the considered controllers are summarized in Table 5.1.

5.2.2 Case studies

Two examples to investigate the various introduced approximate MPC approaches are introduced. The control goal for both case studies is to minimize the average settling time (AST). The AST is defined as the time necessary to steer all states to the origin. A state is considered to be at the origin when $|x_i| \leq 1 \times 10^{-2}$ for all $i \in \mathbb{N}^+_{[n_x]}$. It is assumed that all states of the systems can be measured.

Oscillating Masses (OM)

The first example represents two horizontally oscillating masses interconnected via a spring where each one is connected via a spring to a wall. The system is illustrated in Fig. 5.1. Both masses can only move horizontally and have a weight of 1 kg and each spring has a constant of 1 N m^{-1}. The states of each mass are its position, limited to $|s| \leq 4$ m, and its velocity v, limited to $|v| \leq 10$ m s^{-1}. A force limited by $|u| \leq 0.5$ N can be applied to the right mass.

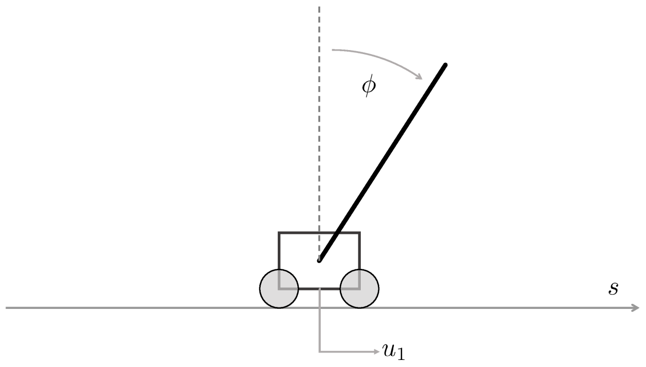


Figure 5.2: Inverted pendulum on a cart. ©[2020] IEEE.

The state vector is given by $x = [s_1, v_1, s_2, v_2]^T$ and the system matrices are discretized with first-order hold and a sampling time of 0.5 s resulting in:

$$A = \begin{bmatrix} 0.763 & 0.460 & 0.115 & 0.020 \\ -0.899 & 0.763 & 0.420 & 0.115 \\ 0.115 & 0.020 & 0.763 & 0.460 \\ 0.420 & 0.115 & -0.899 & 0.763 \end{bmatrix}, \quad B = \begin{bmatrix} 0.014 \\ 0.063 \\ 0.221 \\ 0.367 \end{bmatrix}.$$

The benchmark horizon was $N_{\text{max}} = 7$ corresponding to 2317 polytopic regions that describe the explicit solution. The exact explicit controller $\mu_{\text{exp},7}$ was used to generate $n_{\text{tr}} = 25952$ training samples.

Inverted pendulum on cart (IP)

The second example is the inverted pendulum on a cart, illustrated in Fig. 5.2. The goal is to keep the pole erected and the cart in the central position. The states are the angle of the pole Φ, its angular speed $\dot{\Phi}$, the position of the cart s and the speed of the cart $\dot{s}$. All states $x = [\Phi, s, \dot{\Phi}, \dot{s}]^T$ are constrained to $|x|^T \leq [1, 1.5, 0.35, 1.0]^T$. The force $|u| \leq 1$ N is directly applied to the cart .

Euler-discretization with a sampling time of 0.1 s was used to obtain the discrete system matrices:

$$A = \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 0.9818 & 0.2673 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & -0.0455 & 3.1182 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0.1818 \\ 0 \\ 0.4546 \end{bmatrix}.$$

For this example the explicit benchmark solution was computed with horizon $N_{\text{max}} = 10$ resulting in a PWA function consisting of 1638 polyhedral regions and $n_{\text{tr}} = 88341$ samples were generated to train the approximate controllers.

Table 5.2: Comparison of the relative average settling time (rAST) for 10000 simulation runs and memory footprint MEM$_{(\cdot)}$ for different controllers for the oscillating masses (OM) and inverted pendulum on cart (IP) example. ©[2020] IEEE.

OM	$\mu_{\text{exp},7}$	$\mu_{\text{synth},6}$	$\mu_{\text{synth},3}$	$\mu_{\text{poly},3}$	$\mu_{\text{nn},6,6}$	$\mu_{\text{nn},43,1}$
rAST [-]	1	1.020	1.113	1.407	1.015	1.125
MEM$_{(\cdot)}$ [kB]	691.9	431.8	38.3	2.00	1.93	2.02

IP	$\mu_{\text{exp},10}$	$\mu_{\text{synth},7}$	$\mu_{\text{synth},6}$	$\mu_{\text{poly},3}$	$\mu_{\text{nn},10,6}$	$\mu_{\text{nn},120,1}$
rAST [-]	1	1.897	2.273	2.276	1.038	1.060
MEM$_{(\cdot)}$ [kB]	444.7	191.5	137.1	2.00	4.77	5.63

5.2.3 Performance

The explicit MPC solutions $\mu_{\text{exp},7}(\cdot)$ (OM) and $\mu_{\text{exp},10}(\cdot)$ (IP) were used to generate the training data for the approximation via neural networks (5.1), polynomials (5.2) and simplified explicit MPC (5.6). Since both case studies include box input constraints, a simple saturation was used to guarantee satisfaction of the input constraints for the approximate controllers. For the rest of this section, the dependency of the controllers on the state and on the parameters are dropped for the sake of readability. Additionally, neural networks $\mu_{\text{nn}}(x; \theta, n_0, \ldots, n_{L+1})$ will be referred to by $\mu_{\text{nn},n,L}$ as the width of the hidden layers was chosen to be constant, i.e. $n_l = n$ for all $l \in \mathbb{N}^+_{[L]}$.

We investigated both examples OM and IP by simulating closed-loop trajectories starting from randomly chosen initial values within the feasible state space. For each initial value, the exact MPC controller $\mu_{\text{exp},N_{\text{max}}}$ and the approximate methods were applied and the corresponding average AST over all simulations computed. For both case studies, the evaluation led to similar results, as it can be seen in Table 5.2. The average ASTs are given relative to the average AST of the exact explicit controller with horizon N_{max} - as this performance is considered as the benchmark - and denoted as rAST.

The proposed deep neural networks $\mu_{\text{nn},6,6}$ and $\mu_{\text{nn},10,6}$ only use $0.23\,\%$ and $1.07\,\%$ of the memory of the optimal solutions $\mu_{\text{exp},7}$ and $\mu_{\text{exp},10}$ while reaching an average AST that is only $1.5\,\%$ and $3.8\,\%$ longer than the exact solution. The deep neural network clearly achieves the best trade-off between performance and memory requirements. It is interesting to see that the shallow networks $\mu_{\text{nn},43,1}$ and $\mu_{\text{nn},120,1}$ with a slightly larger memory footprint than the deep networks achieve considerably worse performance. The results further show that a naive polynomial approximation of the explicit MPC does not lead to good results as the performance that can be achieved with no more than 2 kB is significantly worse than the other approximation methods. Even if the optimized explicit MPC with the finest partition $\mu_{\text{synth},6}$ and $\mu_{\text{synth},7}$ is compared to

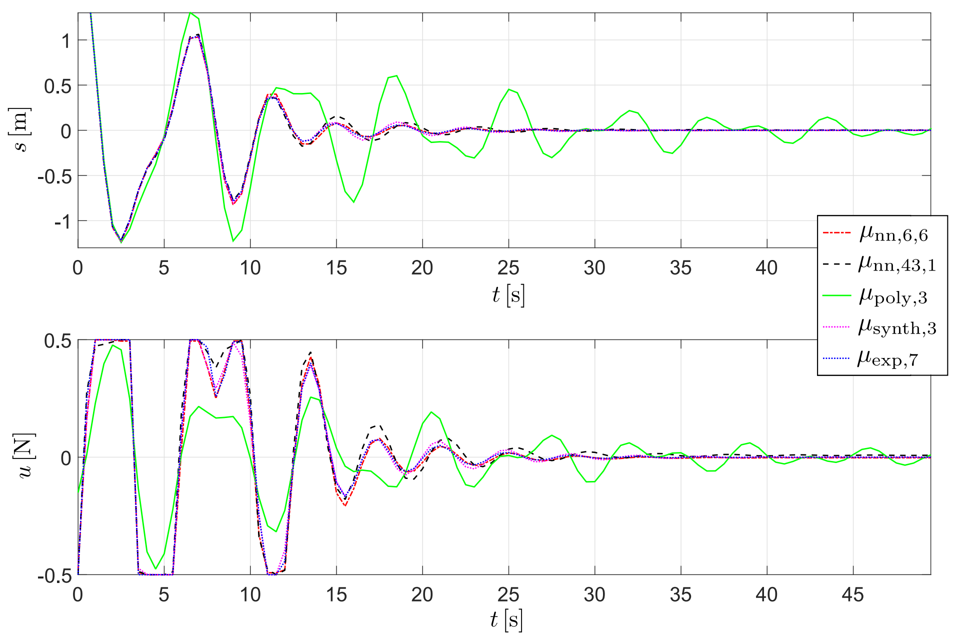


Figure 5.3: Position of the first mass (top plot) and control inputs (bottom plot) for different control strategies for one exemplary closed-loop simulation of the oscillating masses. ©[2020] IEEE.

the benchmark, the proposed deep neural network performs slightly better for OM and clearly better for IP while having a much smaller memory footprint.

Fig. 5.3 and Fig. 5.4 show an example of the closed-loop trajectories obtained for each type of controller for the two examples. It can be clearly seen that the polynomial approximation with degree $p = 3$ cannot properly approximate the explicit controller. The best results, which are almost identical to the exact explicit controller $\mu_{\text{exp},7}$ and $\mu_{\text{exp},10}$, are obtained by the deep neural networks $\mu_{\text{nn},6,6}$ and $\mu_{\text{nn},10,6}$.

One method to reduce the memory requirements and especially the evaluation time of the explicit MPC solutions [35] compared to a standard explicit MPC implementation are binary search trees. However, we have not included the corresponding results for binary search trees for the given examples since computing them for $N \geq 3$ with the toolbox MPT3 [108] was intractable. For shorter horizons, the binary search tree led to a memory footprint reduction around $25\,\%$. For instance for the inverted pendulum on a cart with horizon $N = 2$ the binary search tree led to a reduction of $21.9\,\%$, but it was not possible to solve problems with longer horizons.

The simulation results show that the proposed deep learning-based approximate explicit MPC achieves better performance than other approximate explicit MPC methods with significantly smaller memory requirements. The neural network controller even shows similar performance as the exact explicit MPC approach while being significantly simpler to implement, being applicable for system sizes at which explicit MPC fails due to the exponential growth of the linear regions, and also being applicable for nonlinear systems, as will be shown in Part III. This means that deep learning-

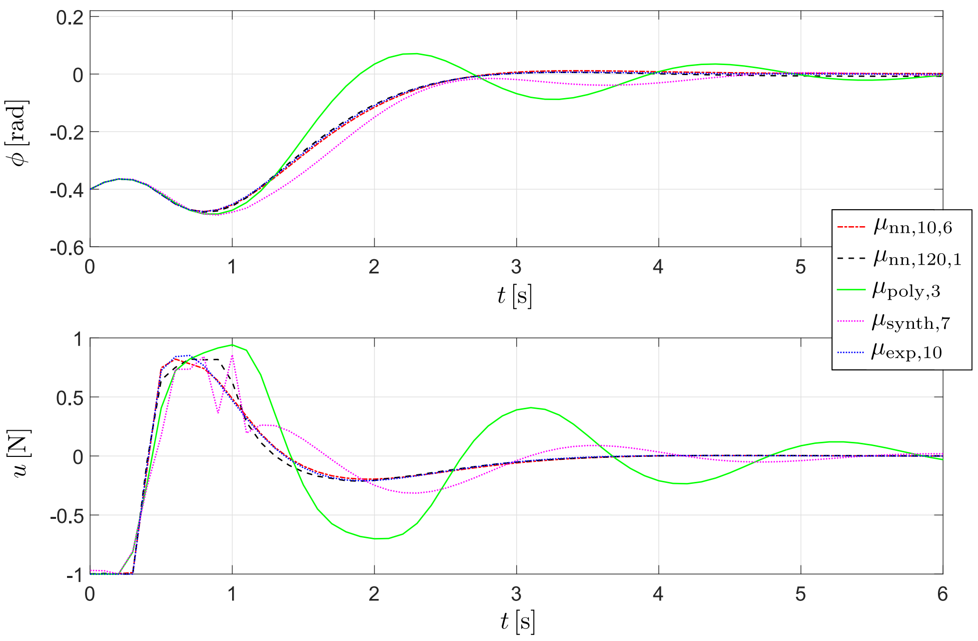


Figure 5.4: Position of the pendulum (top plot) and control input (bottom plot) for different control strategies for one exemplary closed-loop simulation of the inverted pendulum on a cart. ©[2020] IEEE.

based approximate MPC is the most promising option to enable advanced control without online optimization for a broad range of control problems. Additionally, it is possible to deterministically verify the safety and stability and to obtain guaranteed performance bounds for the closed-loop operation, as will be shown in the following chapter.

Chapter 6

Deterministic guarantees for neural network controllers

In the previous chapters, we showed the efficacy of using deep neural networks as explicit controllers to approximate the behavior of MPC policies. However, the learned controllers do not necessarily inherit the same properties regarding safety and performance from the imitated MPC due to approximation errors. In this chapter, methods are presented that enable to verify if deterministic guarantees regarding safety and performance hold for uncertain linear systems which are controlled by neural network controllers.

In the first two sections, the central concepts used throughout this chapter to obtain deterministic guarantees of the closed-loop operation of neural network controlled systems are introduced. These concepts are the activation-based description of ReLU networks, presented in Section 6.1 and output range analysis of neural networks, presented in Section 6.2. First, these concepts are used in Section 6.3 to verify if a learned controller guarantees safe application within a candidate set in the state space and in Section 6.4 is presented how these candidate sets can be obtained. Section 6.5 shows how to prove asymptotically stabilizing closed-loop behavior and how a neural network controller can be modified, such that it ensures optimal behavior in the neighborhood of an equilibrium. Finally, the concepts are used in Section 6.6 to compute worst-case performance bounds. The general approach to obtain deterministic guarantees of the online operation, consisting of the various methods introduced in the previous sections, is summarized in Fig. 6.1 and applied for the case study of a double integrator in Section 6.7 . The results in this chapter are largely based on the publications [6, 7].

6.1 Activation-based description of ReLU networks

Neural networks constitute highly nonlinear and therefore complex to analyze functions. In order to deterministically investigate the input-output behavior, it can be helpful to leverage an alternative description of artificial neural networks. When neural networks with ReLU activations (3.5) in the hidden layers are considered, it is possible to model the input-output behavior of the learned controller with binary variables representing the activations of the neurons. A neuron is considered active if its output

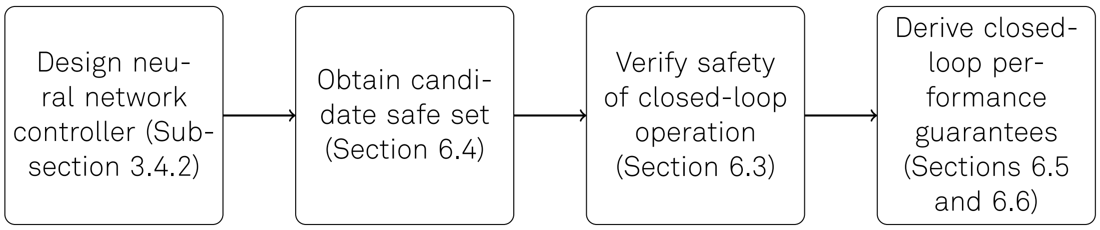

Design neural network controller (Subsection 3.4.2)
Obtain candidate safe set (Section 6.4)
Verify safety of closed-loop operation (Section 6.3)
Derive closed-loop performance guarantees (Sections 6.5 and 6.6)

Figure 6.1: Scheme for certifying safety of neural network controllers with performance guarantees based on output-range analysis.

is greater than zero, e.g. $\sigma_l(\cdot)^{(j)} > 0$ for the j-th neuron in the l-th layer, and inactive else. The activation pattern indicates for all hidden layers which neurons are active for a given input to the network and can be defined as:

$$\Psi(x) = \{v_1(x), \ldots, v_L(x)\}, \tag{6.1}$$

where the activations in each layer are computed via:

$$v_l^{(i)}(x) = \begin{cases} 1 & \text{if } (\sigma_l \circ \lambda_l(z_{l-1}))^{(i)} > 0, \\ 0 & \text{else.} \end{cases} \tag{6.2}$$

The resulting activation-based description of the neural network is then given by:

$$\zeta(x, \Psi(x); \theta) = \lambda_{L+1} \circ \prod_{l=1}^{L} \left(v_l(x) \odot \lambda_l(z_{l-1})\right), \tag{6.3}$$

which is equivalent to the neural network description (3.2), i.e. $\zeta(x, \Psi(x); \cdot) = \mu(x; \cdot)$.

Neural networks with ReLU activations belong to the class of piecewise affine functions as pointed out in Chapter 4. The neurons in the hidden layer can be seen as hyperplanes [107] which in sum describe the polytopic partitioning of the input space - which in case of neural network controllers is the state space $\mathbb{R}^{n_x}$ - on which the affine functions are defined. Consider an activation pattern $\Gamma = \Psi(x_{\text{fix}})$ for a state $x_{\text{fix}} \in \mathbb{R}^{n_x}$. This activation pattern implicitly describes a polytopic region in the state space in which x_{fix} resides via:

$$\mathcal{H}_\Gamma \coloneqq \{x \in \mathbb{R}^{n_x} \,|\, \Gamma = \Psi(x_{\text{fix}})\}, \tag{6.4}$$

i.e. the union of states which share the same activation pattern. The activation pattern Γ further defines a collection of hyperplanes via:

$$H_{\text{col}}(\Gamma) \coloneqq \{-W_{\Gamma,l}^{(i)} x \le b_{\Gamma,l}^{(i)} \,|\, \forall\, v_l^{(i)} = 1, \forall\, l \in \mathbb{N}_{[L]}^{+}\}. \tag{6.5}$$

The matrices $W_{\Gamma,l}$ and vectors $b_{\Gamma,l}$ describe the state-dependent affine function:

$$\begin{aligned} z_l &= W_{\Gamma,l} x + b_{\Gamma,l} \\ &= \prod_{i=0}^{l-1} (v_{l-i} \odot W_{l-i}) \circ x + \sum_{i=1}^{l} \left(\prod (v_j \odot W_j) \circ v_i \odot b_i \right), \end{aligned} \tag{6.6}$$

that the output of the l-th layer (3.4) represents for all $x \in \mathcal{H}_\Gamma$. By formulating a matrix which contains all unique hyperplanes of (6.5), the minimum half-space representation of the corresponding polytopic region (6.4) is derived:

$$\mathcal{H}_\Gamma := \{x \in \mathbb{R}^{n_x} \mid Hx \leq h\}, \tag{6.7}$$

where $H \in \mathbb{R}^{n_H \times n_x}$, $h \in \mathbb{R}^{n_H}$ and each row in H and h corresponds to one of the n_H non-redundant elements in (6.5), e.g. $H^{(j)} = -W^{(j)}_{\Gamma,l}$ and $h^{(j)} = b^{(j)}_{\Gamma,l}$, if $v^{(j)}_l = 1$. The parametric description of the ReLU networks can be used for output range analysis.

6.2 Output range analysis

The central idea of output range analysis for neural networks is to verify if the output set of a neural network (2.4), for improved readability of this chapter defined as:

$$\mu_{\text{nn}}(\mathcal{X}_\text{s}) = \mathcal{Z}(\mathcal{X}_\text{s}, \mu_{\text{nn}}) := \{u \in \mathbb{R}^{n_u} \mid u = \mu_{\text{nn}}(x; \cdot)\ \forall x \in \mathcal{X}_\text{s}\}, \tag{6.8}$$

belongs to a target set $\mathcal{U}_\text{target} \subseteq \mathbb{R}^{n_u}$ for a given set of inputs $\mathcal{X}_\text{s} \subseteq \mathbb{R}^{n_x}$. Output range analysis enables answering the question by computing an over-approximation $\tilde{\mathcal{U}}_\text{out} \subseteq \mathbb{R}^{n_u}$ of $\mu_{\text{nn}}(\mathcal{X}_\text{s})$ and checking if $\tilde{\mathcal{U}}_\text{out} \subseteq \mathcal{U}_\text{target}$. The idea followed in this chapter is to incorporate the system dynamics in such an output range analysis method to rigorously investigate the closed-loop properties of a neural network controlled system.

From the many existing output range analysis approaches, see [82] for a survey, we chose to leverage the one presented in [83]. The approach encodes the input-output behavior of a ReLU network as a mixed-integer linear problem (MILP). The MILP formulation is preferred over approaches that use a relaxed convex problem formulation (e.g. [84, 118]) because the MILP describes ReLU networks exactly and not only approximately.

The basic principle of the MILP formulation is exemplified for a hidden layer:

$$z_l = \max\{0, W_l z_{l-1} + b_l\}, \tag{6.9}$$

consisting of a single neuron ($n_l = 1$). Assume that the output of the previous layer z_{l-1} can take values in $[z_\text{lb}, z_\text{ub}]$. Due to the max operator, we know that $z_l \geq 0$ and $z_l \geq W_l z_{l-1} + b_l$ holds. Further, we know that $z_l = W_l z_{l-1} + b_l$, if the neuron is activated ($v_l = 1$), and that $z_l = 0$ for $v_l = 0$. By leveraging the activation $v_l \in \{0, 1\}$ and choosing a constant M which exceeds the largest possible absolute value the affine function can take, i.e.:

$$M \geq \underset{z_{l-1} \in [z_\text{lb}, z_\text{ub}]}{\arg\max} \quad |W_l z_{l-1} + b_l|, \tag{6.10}$$

the output behavior of the neuron (6.9) can be encoded as a mixed-integer linear feasibility problem:

$$\underset{z_l, v_l, z_{l-1}}{\text{maximize}} \quad 1 \tag{6.11a}$$

$$\text{subject to} \quad z_\text{lb} \leq z_{l-1} \leq z_\text{ub}, \tag{6.11b}$$

$$z_l \geq W_l z_{l-1} + b_l, \tag{6.11c}$$
$$z_l \leq W_l z_{l-1} + b_l + M(\mathbf{1} - v_l) \tag{6.11d}$$
$$z_l \geq 0, \tag{6.11e}$$
$$z_l \leq M v_l, \tag{6.11f}$$
$$v_l \in \{0, 1\}. \tag{6.11g}$$

Every feasible solution of (6.11a) satisfies (6.9), i.e.:

$$z_l^* = \max\left\{0, W_l z_{l-1}^* + b_l\right\}.$$

By extending (6.11a) such that all neurons of a neural network are considered, the input-output behavior of the complete neural network can be analyzed.

Since the MILP formulation from [83] is an exact encoding, the computed over-approximations of the real sets are *tight* in the sense of the following definition.

Definition 6.1. An intersection of half-spaces $\mathcal{A} := \{p \in \mathbb{R}^m \mid Cp \leq c\}$ with $C \in \mathbb{R}^{n \times m}$ and $c \in \mathbb{R}^n$ is a tight over-approximation of a set $\mathcal{B} \subseteq \mathbb{R}^m$, if:

$$Cb \leq c \;\forall\, b \in \mathcal{B}, \tag{6.12}$$

and if there exists a $b_i \in \mathcal{B}$ for all $i \in \mathbb{N}_{[n]}^+$ such that:

$$C^{(i)} b_i = c^{(i)}. \tag{6.13}$$

This means that each hyperplane of the polytopic set $\mathcal{A}$ intersects at least once with the boundary of $\mathcal{B}$ without traversing the interior of $\mathcal{B}$.

6.3 Verification of safe sets

The main goal of this chapter is to obtain a region in the state space for which the operation of the neural network controlled system deterministically guarantees the satisfaction of the control input and state constraints for all times. The a posteriori verification process is required because the algorithms used to find the optimal parameters for the approximate neural network controller are stochastic and sampling-based, and do therefore not guarantee a sufficient approximation quality - even if the approximation error was modeled in the robust MPC formulation used for generating the training data. The safety analysis is based on the future evolution of the neural network controlled uncertain LTI system:

$$x^+ = Ax + B\mu_{\mathsf{nn}}(x) + Ed, \tag{6.14}$$

where $\mu_{\mathsf{nn}}(\cdot)$ is short for $\mu_{\mathsf{nn}}(\cdot; \theta, n_0, \ldots, n_{L+1})$, because the parameter set θ is assumed to be fixed after the training phase. The closed-loop uncertain system can be used to define the reachable set from a set of initial conditions $\mathcal{X}_{\mathsf{s}}$ for the fixed uncertainty set $\mathcal{D}$ as:

$$\mathcal{R}(\mathcal{X}_{\mathsf{s}}) := \bigcup_{x \in \mathcal{X}_{\mathsf{s}}} \left\{x^+ \in \mathbb{R}^{n_x} \,|\, x^+ = Ax + B\mu_{\mathsf{nn}}(x) + Ed \;\forall\, d \in \mathcal{D}\right\}. \tag{6.15}$$

To view the reachable set as a union of reachable states, in contrast to the set-based formulation (2.3), will be used later in this section to prove the deterministic properties of the proposed verification approach. The reachable set operator $\mathcal{R}(\cdot)$ can also be leveraged to compute r-step reachable sets by applying it recursively:

$$\mathcal{R}_r(\mathcal{X}_{\mathsf{s}}) = \prod_{k=1}^{r} \mathcal{R}(\mathcal{X}_{\mathsf{s}}) \text{ with } \mathcal{R}_0(\mathcal{X}_{\mathsf{s}}) = \mathcal{X}_{\mathsf{s}}. \tag{6.16}$$

The resulting sets contain all states that can be reached after r closed-loop steps.

Based on the r-step reachable set (6.16) and the output set (6.8), it is possible to define requirements that a set in the state space must meet such that the application of the neural network controller guarantees the satisfaction of constraints for all times.

Definition 6.2. A safe set for the closed-loop system (6.14) is defined as an intersection of half-spaces in the state space:

$$\mathcal{X}_{\mathsf{safe}} \coloneqq \{x \in \mathbb{R}^{n_x} \,|\, C_{\mathsf{safe}} x \leq c_{\mathsf{safe}}\}, \tag{6.17}$$

with $C_{\mathsf{safe}} \in \mathbb{R}^{n_{\mathsf{safe}} \times n_x}$ and $c_{\mathsf{safe}} \in \mathbb{R}^{n_{\mathsf{safe}}}$, that satisfies the following conditions for the first r steps:

$$\mu_{\mathsf{nn}}(\mathcal{R}_k(\mathcal{X}_{\mathsf{safe}})) \subseteq \mathcal{U}, \qquad \forall\, k \in \mathbb{N}_{[r-1]}, \tag{6.18a}$$

$$\mathcal{R}_k(\mathcal{X}_{\mathsf{safe}}) \subseteq \mathcal{X}, \qquad \forall\, k \in \mathbb{N}_{[r-1]}, \tag{6.18b}$$

$$\mathcal{R}_r(\mathcal{X}_{\mathsf{safe}}) \subseteq \mathcal{X}_{\mathsf{safe}}. \tag{6.18c}$$

Lemma 6.1. *Each set $\mathcal{X}_{safe}$ satisfying the conditions in Definition 6.2 guarantees closed-loop constraint satisfaction at all times, i.e. $\mathcal{R}_k(\mathcal{X}_{safe}) \subseteq \mathcal{X}$ and $\mu_{nn}(\mathcal{R}_k(\mathcal{X}_{safe})) \subseteq \mathcal{U}$ for all $k \in \mathbb{N}_{[\infty]}$.*

Proof. [Lemma 6.1] The conditions (6.18a) and (6.18b) directly guarantee satisfaction of state and control input constraints for time steps $k \in \mathbb{N}_{[r-1]}$. Because the r-step reachable set is a subset of the initial set $\mathcal{X}_{\mathsf{safe}}$ as stated in (6.18c), it follows with (6.15) that:

$$\mathcal{R}_k(\mathcal{X}_{\mathsf{safe}}) = \mathcal{R}_k\left(\mathcal{R}_r\left(\mathcal{X}_{\mathsf{safe}}\right)\right) \cup \mathcal{R}_k\left(\mathcal{X}_{\mathsf{safe}} \setminus \mathcal{R}_r(\mathcal{X}_{\mathsf{safe}})\right) \supseteq \mathcal{R}_k(\mathcal{R}_r(\mathcal{X}_{\mathsf{safe}})). \tag{6.19}$$

This also means that each reachable set for a multiple of r steps is a subset of $\mathcal{X}_{\mathsf{safe}}$:

$$\mathcal{R}_{j\cdot r}(\mathcal{X}_{\mathsf{safe}}) \subseteq \mathcal{X}_{\mathsf{safe}} \subseteq \mathcal{X} \;\forall\, j \in \mathbb{N}_{[\infty]}. \tag{6.20}$$

From (6.18b), (6.19) and (6.20) follows that

$$\mathcal{R}_k(\mathcal{R}_{j\cdot r}(\mathcal{X}_{\mathsf{safe}})) \subseteq \mathcal{R}_k(\mathcal{X}_{\mathsf{safe}}) \subseteq \mathcal{X} \;\forall\, k \in \mathbb{N}_{[r-1]}, \forall\, j \in \mathbb{N}_{[\infty]}, \tag{6.21}$$

which is equivalent to $\mathcal{R}_k(\mathcal{X}_{\mathsf{safe}}) \subseteq \mathcal{X}$ for all $k \in \mathbb{N}_{[\infty]}$. Further follows from (6.21) and (6.18a) that:

$$\mu_{\mathsf{nn}}\left(\mathcal{R}_k(\mathcal{R}_{j\cdot r}(\mathcal{X}_{\mathsf{safe}}))\right) \subseteq \mu_{\mathsf{nn}}(\mathcal{R}_k(\mathcal{X}_{\mathsf{safe}})) \subseteq \mathcal{U},$$

for all $j \in \mathbb{N}_{[\infty]}$ and $k \in \mathbb{N}_{[r-1]}$ and therefore $\mu_{\text{nn}}(\mathcal{R}_k(\mathcal{X}_{\text{safe}})) \subseteq \mathcal{U}$ for all $k \in \mathbb{N}_{[\infty]}$. $\square$

In the following, we present how output range analysis can be used to compute output sets and reachable sets in order to analyze if the conditions in Definition 6.2 hold.

6.3.1 Approximation of output sets

For the verification of (6.18a), we want to ensure that the real output set of the neural network controller (6.8) satisfies $\mu_{\text{nn}}(\mathcal{X}_{\text{s}}) \subseteq \mathcal{U}$. Because $\mathcal{U}$ is a polytope, it is sufficient to check for each hyperplane individually if the bound can be violated, e.g. by solving:

$$\begin{aligned} \underset{x_{0,i},u_{0,i}}{\text{maximize}} \quad & C_u^{(i)} u_{0,i} && \text{(6.22a)} \\ \text{subject to} \quad & C_{\text{s}} x_0 \le c_{\text{s}}, && \text{(6.22b)} \\ & u_{0,i} = \mu_{\text{nn}}(x_{0,i}). && \text{(6.22c)} \end{aligned}$$

If the solutions satisfy $C_u^{(i)} u_{0,i}^* \le c_u^{(i)}$ for all $i \in \mathbb{N}^+_{[n_\mathcal{U}]}$, where $u_{0,i}^*$is the optimal solution of (6.22) when $C_u^{(i)}$ is considered, then $\mu_{\text{nn}}(\mathcal{X}_{\text{s}}) \subseteq \mathcal{U}$. The nonlinear neural network constraint (6.22c) can be encoded via binary variables based on the parametric description of neural networks with ReLU activations (6.3) and the output range analysis formulation presented in [83], as explained in Section 6.2. This means (6.22) can be equivalently formulated as the mixed-integer linear program:

$$\begin{aligned} \underset{z,v,u_{0,i},x_{0,i}}{\text{maximize}} \quad & C_u^{(i)} u_{0,i} && \text{(6.23a)} \\ \text{subject to} \quad & C_{\text{s}} x_{0,i} \le c_{\text{s}}, && \text{(6.23b)} \\ & z_0 = x_{0,i}, && \text{(6.23c)} \\ & \text{for all } l \in \mathbb{N}^+_{[L]}: && \\ & \quad z_l \ge W_l z_{l-1} + b_l, && \text{(6.23d)} \\ & \quad z_l \le W_l z_{l-1} + b_l + M(\mathbf{1} - v_l), && \text{(6.23e)} \\ & \quad z_l \ge 0, && \text{(6.23f)} \\ & \quad z_l \le M v_l, && \text{(6.23g)} \\ & \quad v_l \in \{0,1\}^{n_l}, && \text{(6.23h)} \\ & u_{0,i} = W_{L+1} z_L + b_{L+1}, && \text{(6.23i)} \end{aligned}$$

where $v = \{v_0, \dots, v_L\}$ and $z = \{z_0, \dots, z_L\}$ are the collection of all activations and outputs of the ReLU layers. The two variables z and v allow modeling the behavior of the hidden ReLU layers of the neural network via the constraints (6.23d)-(6.23h) using the big M method [119, 120]. In a similar fashion as for (6.10), the value of $M \in \mathbb{R}$ needs to be larger than $||W_l z_{l-1} + b_l||_\infty$ for all $l \in \mathbb{N}^+_{[L]}$. A quick estimate for a sufficiently large value can be obtained by propagating the bounding box of $\mathcal{X}_{\text{s}}$ through the layers using the norms $||W_l||_\infty$, $l \in \mathbb{N}^+_{[L]}$. The constraints (6.23b) and (6.23c) ensure that only states in $\mathcal{X}_{\text{s}}$ are considered and (6.23i) models the linear output layer. The MILP (6.23) can be used to approximate the real output set (6.8).

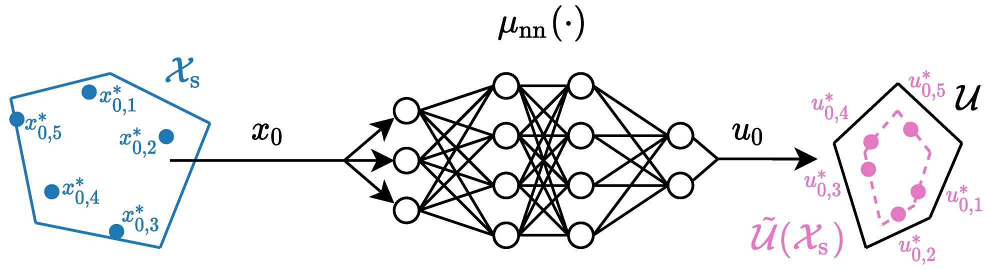


Figure 6.2: Usage of the operator $\tilde{\mathcal{U}}(\cdot)$ to obtain an over-approximation of the true output set $\mu_{\text{nn}}(\mathcal{X}_\text{s})$ for the set of initial conditions $\mathcal{X}_\text{s}$ in order to verify the satisfaction of the control input constraints $\mathcal{U}$. The true output set is not exactly know but lies within $\tilde{\mathcal{U}}(\mathcal{X}_\text{s})$.

Definition 6.3. The set-based operator $\tilde{\mathcal{U}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ returns an intersection of half-spaces for a set of initial conditions which is defined as:

$$\tilde{\mathcal{U}}(\mathcal{X}_\text{s}) := \{u \in \mathbb{R}^{n_u} \,|\, C_u u \leq c_u^*\}\,, \tag{6.24}$$

where $c_u^* = \left[C_u^{(1)} u_{0,1}^*, \ldots, C_u^{(n_\mathcal{U})} u_{0,n_\mathcal{U}}^*\right]$ contains the optimal values for solving (6.23) for each hyperplane $C_u^{(i)}$ with $i \in \mathbb{N}^+_{[n_\mathcal{U}]}$.

Lemma 6.2. *The operator $\tilde{\mathcal{U}}(\cdot)$ provides an over-approximation of the real output set $\mu_{nn}(\cdot)$ for a set of initial conditions $\mathcal{X}_s$, i.e. $\tilde{\mathcal{U}}(\mathcal{X}_s) \supseteq \mu_{nn}(\mathcal{X}_s)$. In addition, this over-approximation is tight according to Definition 6.1.*

Proof. [Lemma 6.2] Because the constraints (6.23c)-(6.23i) guarantee that $u_{0,i} = \mu_\text{nn}(x_{0,i})$ holds and (6.23c) ensures that $x_{0,i} \in \mathcal{X}_\text{s}$, each element $u_{0,i}$ in the real output set $\mu_\text{nn}(\mathcal{X}_\text{s})$ can be considered for computing the bounds c_u^* of $\tilde{\mathcal{U}}(\cdot)$. By solving (6.23) globally, the control input $u_{0,i}^* \in \mu_\text{nn}(\mathcal{X}_\text{s})$ that results in the largest value in the i-th hyperplane normal direction $C_u^{(i)}$ is found. This means that $C_u^{(i)} u_0 \leq C_u^{(i)} u_{0,i}^*$ for all $u_0 \in \mu_\text{nn}(\mathcal{X}_\text{s})$ and hence $u_{0,i}^*$ is on the boundary of $\mu_\text{nn}(\mathcal{X}_\text{s})$. By using $c_u^* \in \mu_\text{nn}(\mathcal{X}_\text{s})$ as the bound in (6.24) to define $\tilde{\mathcal{U}}(\mathcal{X}_\text{s})$ it is guaranteed that the hyperplanes in $\tilde{\mathcal{U}}(\mathcal{X}_\text{s})$ intersect only with the boundary of $\mu_\text{nn}(\mathcal{X}_\text{s})$. By extension to all $n_\mathcal{U}$ hyperplane normal directions follows that $\tilde{\mathcal{U}}(\mathcal{X}_\text{s}) \supseteq \mu_\text{nn}(\mathcal{X}_\text{s})$ and that the over-approximation is tight according to Definition 6.1. □

If the over-approximation of the real output set satisfies $\tilde{\mathcal{U}}(\mathcal{X}_\text{s}) \subseteq \mathcal{U}$, then this is also true for the real output set $\mu_\text{nn}(\mathcal{X}_\text{s}) \subseteq \mathcal{U}$ and satisfaction of input constraints can be easily verified. The full verification process when using $\tilde{\mathcal{U}}(\cdot)$ is visualized in Fig. 6.2 for an input constraint set described by $n_\mathcal{U} = 5$ hyperplanes for which (6.23) accordingly needs to solved five times.

The operator $\tilde{\mathcal{U}}(\cdot)$ can also be used to analyze if (6.18a) holds for $k \in \mathbb{N}^+_{[r_\text{inv}]}$ when the reachable sets $\mathcal{R}_k(\mathcal{X}_\text{safe})$, $k \in [r_\text{inv}]^+$, or approximations are known. In the context of neural network controllers, output range analysis has been previously used to verify reachability of target sets when no control input constraints are present [83] or when they are given as box input constraints [118, 121]. In the following we will show that these formulations can be extended to rigorously verify constraint satisfaction for general polytopic input and state constraints. In the next subsection, the MILP (6.23) will be extended with the closed-loop (6.14) to compute approximations of the reachable sets.

6.3.2 Approximation of reachable sets

In order to analyze if conditions (6.18a) and (6.18b) are satisfied, the computation of reachable sets of the uncertain LTI system is required. For the approximation of reachable sets (6.16), a similar approach can be followed as in the case of output sets by extending the MILP formulation (6.23) with the description of the r-step reachable set (6.16). For the approximation of the reachable sets, the matrix $S \in \mathbb{R}^{n_s \times n_x}$ is considered, in which each row represents a hyperplane normal direction. The following MILP needs to be solved for each of the n_s hyperplanes in S:

$$\begin{aligned}
\underset{\mathbf{z},\mathbf{v},\mathbf{u},\mathbf{d},\mathbf{x}_i}{\text{maximize}} \quad & S^{(i)} x_{r,i} && \text{(6.25a)}\\
\text{subject to} \quad & C_{\mathrm{s}} x_{0,i} \leq c_{\mathrm{s}}, && \text{(6.25b)}\\
& \text{for all } k \in \mathbb{N}_{[r-1]} : && \\
& \quad z_{k,0} = x_{k,i}, && \text{(6.25c)}\\
& \quad \text{for all } l \in \mathbb{N}^{+}_{[L]} : && \\
& \quad\quad v_{k,l} \in \{0,1\}^{n_l}, && \text{(6.25d)}\\
& \quad\quad z_{k,l} \geq W_l z_{k,l-1} + b_l, && \text{(6.25e)}\\
& \quad\quad z_{k,l} \leq W_l z_{k,l-1} + b_l + M(\mathbf{1} - v_{k,l}), && \text{(6.25f)}\\
& \quad\quad z_{k,l} \geq 0, && \text{(6.25g)}\\
& \quad\quad z_{k,l} \leq M v_{k,l}, && \text{(6.25h)}\\
& \quad u_k = W_{L+1} z_{k,L} + b_{L+1}, && \text{(6.25i)}\\
& \quad C_d d_k \leq c_d, && \text{(6.25j)}\\
& \quad x_{k+1,i} = A x_{k,i} + B u_k + d_k, && \text{(6.25k)}
\end{aligned}$$

where $\mathbf{u} = \{u_0, \ldots, u_{r-1}\}$ is the control input trajectory and $\mathbf{d} = \{d_0, \ldots, d_{r-1}\}$ is the sequence of disturbance realizations. The constraint (6.25b) restricts the initial state of the trajectory $\mathbf{x}_i = \{x_{0,i}, \ldots, x_{r,i}\}$ to the set of initial conditions $\mathcal{X}_{\mathrm{s}}$. The constraints (6.25c)-(6.25i) model the values of the binary variables $\mathbf{v} = \{v_{0,1}, \ldots, v_{r-1,L}\}$ and hidden layers $\mathbf{z} = \{z_{0,0}, \ldots, z_{r-1,L}\}$ in each step $k \in \mathbb{N}_{[r-1]}$ according to (6.3). The closed-loop system trajectory is modelled via the constraints (6.25i)-(6.25k). By solving (6.25) the sequence of disturbance realizations and the initial state within $\mathcal{X}_{\mathrm{s}}$ are computed that push the state x_r the furthest into the considered hyperplane normal direction $S^{(i)}$ with respect to the closed-loop dynamics. The MILP (6.25) can be leveraged to define an operator that provides approximations of the r-step reachable set.

Definition 6.4. The set-based operator $\tilde{\mathcal{R}}_r : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ returns an intersection of half-spaces which is defined as:

$$\tilde{\mathcal{R}}_r(\mathcal{S}) = \{x \in \mathbb{R}^{n_x} \mid Sx \leq s^*\}, \tag{6.26}$$

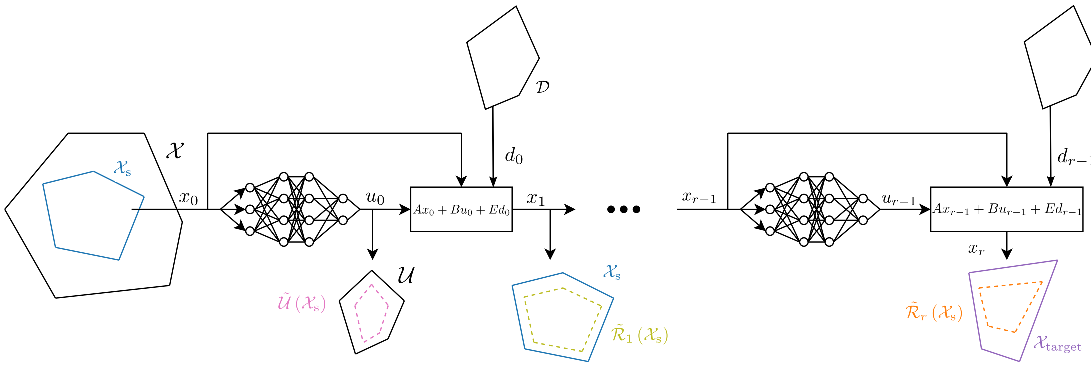

$\mathcal{X}_s$
$\mathcal{X}$
x_0
u_0
$Ax_0 + Bu_0 + Ed_0$
x_1
$\mathcal{D}$
d_0
$\mathcal{U}$
$\tilde{\mathcal{U}}(\mathcal{X}_s)$
$\mathcal{X}_s$
$\tilde{\mathcal{R}}_1(\mathcal{X}_s)$
x_{r-1}
u_{r-1}
$Ax_{r-1} + Bu_{r-1} + Ed_{r-1}$
$\mathcal{D}$
d_{r-1}
x_r
$\tilde{\mathcal{R}}_r(\mathcal{X}_s)$
$\mathcal{X}_{\text{target}}$

Figure 6.3: Schematic overview of the closed-loop (6.14) and visualization of the output sets computed based on (6.23) and reachable sets derived via (6.25). The derived sets enable the verification of safety features like closed-loop constraint satisfaction and asymptotic stability.

where $s^* = \left[S^{(1)}x^*_{r,1}, \dots, S^{(n_s)}x^*_{r,n_s}\right]$ contains the optimal values for solving (6.25) for each hyperplane normal direction $S^{(i)}$ with $i \in \mathbb{N}^+_{[n_s]}$.

Lemma 6.3. *The operator $\tilde{\mathcal{R}}_r(\cdot)$ (6.26) provides an over-approximation of the true r-step reachable set $\mathcal{R}_r(\cdot)$ for a set of initial conditions $\mathcal{X}_\text{s}$, i.e. $\tilde{\mathcal{R}}_r(\mathcal{X}_\text{s}) \supseteq \mathcal{R}_r(\mathcal{X}_\text{s})$. In addition, this over-approximation is tight according to Definition 6.1.*

Proof. [Lemma 6.3] The constraints in (6.25) model every possible true trajectory $\mathbf{x}$ of the closed-loop system starting in $\mathcal{X}_\text{s}$. Because (6.25) is solved globally, the real trajectory $\mathbf{x}^*_i = \left\{x^*_{0,i}, \dots, x^*_{r,i}\right\}$ including the sequence of uncertainty realizations $\mathbf{d}^* \in \mathcal{D}^r$ is found for which x_r reaches the maximum value in the normal direction of the i-th hyperplane, i.e. $S^{(i)}x_{r,i} \leq S^{(i)}x^*_{r,i}$ for any trajectory $\mathbf{x}$ of length r starting in $\mathcal{X}_\text{s}$ and any realization $\mathbf{d} \in \mathcal{D}^r$. Because the bound of the i-th hyperplane is chosen as $s^{*,(i)} = S^{(i)}x^*_r$ (6.26), we obtain $\mathcal{R}_r(\mathcal{X}_\text{s}) \subseteq \tilde{\mathcal{R}}_r(\mathcal{X}_\text{s})$. This also means that the conditions (6.12) and (6.13) defining a tight over-approximation are satisfied. $\square$

Two exemplary usages of the operator $\tilde{\mathcal{R}}_r(\cdot)$ are visualized in Fig. 6.3, where the approximate one step reachable set $\tilde{\mathcal{R}}_1(\mathcal{X}_\text{s})$ is used in combination with $\tilde{\mathcal{U}}(\mathcal{X}_\text{s})$ to establish control-invariance of $\mathcal{X}_\text{s}$. The right part of Fig. 6.3 shows how $\tilde{\mathcal{R}}_r(\cdot)$ can be used to verify that the r-step reachable set is contained in a desired target set $\mathcal{X}_\text{target}$, e.g. the initial set $\mathcal{X}_\text{s}$ in order to verify (6.18c). The introduced operators can also be leveraged to verify that a set of states satisfies all conditions of a safe set given in Defintion 6.2.

6.3.3 Verifying safety

The operators defined for approximating the real output set of the neural network controller and reachable set of the neural network controlled uncertain LTI system can be leveraged to verify, if a candidate set $\mathcal{X}_\text{s}$ is a safe set.

Theorem 6.1. *If a set $\mathcal{X}_\text{s} \subseteq \mathcal{X}$ satisfies the following closed-loop conditions in the first r steps* (6.14)*:*

$$\tilde{\mathcal{U}}(\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s})) \subseteq \mathcal{U}, \qquad \forall\, k \in \mathbb{N}_{[r-1]}, \tag{6.27a}$$

$$\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s}) \subseteq \mathcal{X}, \qquad \forall\, k \in \mathbb{N}_{[r-1]}, \tag{6.27b}$$

$$\tilde{\mathcal{R}}_r(\mathcal{X}_\text{s}) \subseteq \mathcal{X}_\text{s}, \tag{6.27c}$$

then $\mathcal{X}_\text{s}$ is a safe set as given in Definition 6.2.

Proof. [Theorem 6.1] Satisfaction of (6.18b) is guaranteed for $k = 0$ by the choice of the candidate set $\mathcal{X}_\text{s}$. From Lemma 6.3 and the conditions (6.27b) and (6.27c) follows that $\mathcal{R}_k(\mathcal{X}_\text{s}) \subseteq \tilde{\mathcal{R}}_k(\mathcal{X}_\text{s}) \subseteq \mathcal{X}$ for $k \in \mathbb{N}^+_{[r-1]}$ and $\mathcal{R}_r(\mathcal{X}_\text{s}) \subseteq \tilde{\mathcal{R}}_r(\mathcal{X}_\text{s})$, which means that conditions (6.18b) and (6.18c) are satisfied.

In order to show that (6.27a) implies (6.18a), we apply the operator for approximating the output sets to the over-approximation of the reachable set $\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s})$ and true reachable set $\mathcal{R}_k(\mathcal{X}_\text{s})$:

$$\begin{aligned}
&\max_{\substack{x_{0,i}\in\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s}),\\ \text{(6.23c)} - \text{(6.23i)}}} C_u^{(i)} x_{1,i} \\
&= \max\left\{ \max_{\substack{x_{0,i}\in\mathcal{R}_k(\mathcal{X}_\text{s}),\\ \text{(6.23c)} - \text{(6.23i)}}} C_u^{(i)} x_{1,i},\; \max_{\substack{x_{0,i}\in\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s})\setminus\mathcal{R}_k(\mathcal{X}_\text{s}),\\ \text{(6.23c)} - \text{(6.23i)}}} C_u^{(i)} x_{1,i} \right\} \\
&\geq \max_{\substack{x_{0,i}\in\mathcal{R}_k(\mathcal{X}_\text{s}),\\ \text{(6.23c)} - \text{(6.23i)}}} C_u^{(i)} x_{1,i}.
\end{aligned} \tag{6.28}$$

The equation (6.28) shows that $\tilde{\mathcal{U}}(\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s})) \supseteq \tilde{\mathcal{U}}(\mathcal{R}_k(\mathcal{X}_\text{s}))$. Together with Lemma 6.2 and condition (6.27a), it follows that

$$\mu_\text{nn}(\mathcal{R}_k(\mathcal{X}_\text{s})) \subseteq \tilde{\mathcal{U}}(\mathcal{R}_k(\mathcal{X}_\text{s})) \subseteq \tilde{\mathcal{U}}(\tilde{\mathcal{R}}_k(\mathcal{X}_\text{s})) \subseteq \mathcal{U}$$

for all $k \in \mathbb{N}_{[r-1]}$, which means that (6.18a) is satisfied. Therefore, $\mathcal{X}_\text{s}$ satisfies all conditions defining a safe set. □

The procedure to investigate the safety of a candidate set based on Theorem 6.1, summarized in Algorithm 1, can be simplified, if the input constraints are given as box constraints:

$$\mathcal{U} \coloneqq \{u \in \mathbb{R}^{n_u} \mid u_\text{lb} \leq u \leq u_\text{ub}\}, \tag{6.29}$$

where $u_\text{lb} \in \mathbb{R}^{n_u}$ are the lower bounds and $u_\text{ub} \in \mathbb{R}^{n_u}$ are the upper bounds. Satisfaction of control input constraints can then be guaranteed constructively and therefore does not need to be checked explicitly in each step $k \in \mathbb{N}_{[r-1]}$.

Proposition 6.1. *If the control input constraints are box constraints* (6.29)*, each neural network controller $\mu_{nn}(x;\theta,n_0,\dots,n_{L+1})$ with L hidden layers can be adapted such that the modified neural network controller $\mu_{nn,sat}(x;\theta_{sat},n_0,\dots,n_{L_{sat}+1})$ with $L_{sat} = L+2$ hidden layers satisfies* (6.18a) *for all $x \in \mathbb{R}^{n_x}$. In addition, the modified network outputs the same output as the original network, if the output of the original network is input-admissible, i.e. $\mu_{nn,sat}(x;\cdot) = \mu_{nn}(x;\cdot)$ if $u_{lb} \leq \mu_{nn}(x;\cdot) \leq u_{ub}$. The first L layers of the*

Algorithm 1 Verifying the safety of a candidate set.

```
Require: $\mu_{\text{nn}}(\cdot), r_{\text{max}} \geq 1$
 1: Choose admissible candidate set $\mathcal{X}_\text{s} \subseteq \mathcal{X}$ and hyperplane normal directions via $S$
 2: Set $k \leftarrow 1$ and $r \leftarrow 0$
 3: while $k \leq r_{\text{max}}$ do
 4:    $X \leftarrow \tilde{\mathcal{R}}_r(\mathcal{X}_\text{s})$
 5:    $U \leftarrow \tilde{\mathcal{U}}(X)$
 6:    if $U \subseteq \mathcal{U}$ and $X \subseteq \mathcal{X}_\text{s}$ then
 7:       Set $r \leftarrow k$
 8:       break
 9:    else if $U \subseteq \mathcal{U}$ and $X \subseteq \mathcal{X}$ then
10:       $k \leftarrow k + 1$
11:    else
12:       break
13:    end if
14: end while
15: if $r > 0$ then
16:    $\mathcal{X}_\text{s}$ is a safe set for $\mu_{\text{nn}}(\cdot)$
17: end if
```

modified controller are identical to the original controller, i.e. $W_l = W_{l,sat}$ and $b_l = b_{l,sat}$ for $l \in \mathbb{N}^+_{[L]}$. The weights of the layers $L+1$ to $L+3$ are given by:

$$\begin{aligned} W_{L+1,sat} &= -W_{L+1}, & b_{L+1,sat} &= u_{ub} - b_{L+1}, \\ W_{L+2,sat} &= -I, & b_{L+2,sat} &= u_{ub} - u_{lb}, \\ W_{L+3,sat} &= I, & b_{L+3,sat} &= u_{lb}. \end{aligned}$$

Proof. [Proposition 6.1] Consider the output of the L-th layer $z_L = \zeta(x, \Psi(x), L; \cdot)$, which is identical for the original and the modified controller as the weights and biases up to the L-th layer are the same. The corresponding control input signal of the original network is then given by

$$u_{\text{orig}} = W_{L+1} z_L + b_{L+1}.$$

Reformulating the output of the modified network with respect to the control input signal of the original network, the following expression is derived:

$$u_{\text{nn,sat}} = \max(-\max(-u_{\text{orig}} + u_{\text{ub}}, 0) + u_{\text{ub}} - u_{\text{lb}}, 0) + u_{\text{lb}}.$$

The possible outcome of each element

$$u^{(i)}_{\text{nn,sat}} = \begin{cases} u^{(i)}_{\text{ub}}, & \text{if } u^{(i)}_{\text{orig}} \geq u^{(i)}_{\text{ub}}, \\ \tilde{u}^{(i)}, & \text{if } u^{(i)}_{\text{lb}} \leq u^{(i)}_{\text{orig}} \leq u^{(i)}_{\text{ub}}, \\ u^{(i)}_{\text{lb}}, & \text{if } u^{(i)}_{\text{orig}} \leq u^{(i)}_{\text{lb}}, \end{cases}$$

represent either the control input provided for the original, if admissible, or the control input that is saturated according the box constraints. □

The formulation of the mixed-integer problems presented in this section can also be slightly adapted to not only verify given sets, but also search for safe sets. This idea is presented in detail in the next section.

6.4 Computation of safe sets

In this section, we will leverage approximations of preimages to search for safe sets. The preimage $\mathcal{P}(\cdot)$ of a set are all states from which the set of interest can be reached in one closed-loop step:

$$\mathcal{P}(\mathcal{X}_{\mathsf{s}}) \coloneqq \{x \in \mathbb{R}^{n_x} \mid \exists d \in \mathcal{D} \text{ s.t. } Ax + B\mu_{\mathsf{nn}}(x) + Ed \in \mathcal{X}_{\mathsf{s}}\}\,. \tag{6.30}$$

The usage of preimages is especially promising when the closed-loop converges to a region around an optimum. Analogously to the case of r-step reachable sets (6.16), the r-step preimage can be defined by applying the operator $\mathcal{P}(\cdot)$ r times recursively:

$$\mathcal{P}_r(\mathcal{X}_{\mathsf{s}}) = \prod_{k=1}^{r} \mathcal{P}(\mathcal{X}_{\mathsf{s}}) \text{ with } \mathcal{P}_0(\mathcal{X}_{\mathsf{s}}) = \mathcal{X}_{\mathsf{s}}. \tag{6.31}$$

By slightly modifying the formulation in (6.25), it is possible to compute the initial state x_0 with the largest value in the i-th hyperplane normal direction of S, for which an r-step trajectory exists such that the final state x_r belongs to the set of interest:

$$\begin{aligned} &\underset{\mathbf{z},\mathbf{v},\mathbf{u},\mathbf{d},\mathbf{x}_i}{\text{maximize}} && S^{(i)} x_{0,i} && \text{(6.32a)}\\ &\text{subject to} && C_{\mathsf{s}} x_{r,i} \leq c_{\mathsf{s}}, && \text{(6.32b)}\\ & && \text{(6.25c)} - \text{(6.25k)}. && \end{aligned}$$

In a similar manner as for the r-step reachable sets, an operator can be defined that computes an approximation of the real preimages based on (6.32).

Definition 6.5. The set-based operator $\tilde{\mathcal{P}}_r : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ returns an intersection of half-spaces which is defined as:

$$\tilde{\mathcal{P}}_r(\mathcal{X}_{\mathsf{s}}) = \{x_0 \in \mathbb{R}^{n_x} \mid Sx_0 \leq s^*\}\,, \tag{6.33}$$

where $s^* = \left[S^{(1)}x^*_{0,1}, \ldots, S^{(n_s)}x^*_{0,n_s}\right]$ contains the optimal values for solving (6.32) for each hyperplane normal direction $S^{(i)}$ with $i \in \mathbb{N}^+_{[n_s]}$.

Lemma 6.4. *The operator $\tilde{\mathcal{P}}_r(\cdot)$ (6.33) provides an over-approximation of the true r-step preimage $\mathcal{P}_r(\cdot)$ for a set of initial conditions $\mathcal{X}_{\mathsf{s}}$, e.g. $\tilde{\mathcal{P}}_r(\mathcal{X}_{\mathsf{s}}) \supseteq \mathcal{P}_r(\mathcal{X}_{\mathsf{s}})$. In addition, this over-approximation is tight according to Definition 6.1.*

Proof. [Lemma 6.4] The proof can be derived in the same fashion as for Lemma 6.3 and is therefore omitted. □

Algorithm 2 Searching for safe sets.

```
Require: $\mu_{\text{nn}}(\cdot), r_{\text{max}} \geq 1, r_{\text{pre,max}} \geq 1$, admissible state space $\mathcal{X}$
 1: Choose initial set $\mathcal{X}_{\text{s}}$ and set of hyperplanes $S$
 2: Set $r_{\text{pre}} \leftarrow 1$ and $\mathcal{X}_{\text{MRPI}} \leftarrow \emptyset$
 3: while $r_{\text{pre}} \leq r_{\text{pre,max}}$ do
 4:     $X \leftarrow \tilde{\mathcal{P}}_r(\mathcal{X}_{\text{s}})$
 5:     $\mathcal{X}_{\text{cand}} \leftarrow \mathcal{X} \cap X$
 6:     Follow Algorithm 1 with $\mathcal{X}_{\text{s}} \leftarrow \mathcal{X}_{\text{cand}}$ to obtain $r$
 7:     if $r > 0$ then
 8:         Set $\tilde{\mathcal{X}}_{\text{max}} \leftarrow \mathcal{X}_{\text{cand}}$ , $r_{\text{max}} \leftarrow r$, and $r_{\text{pre}} \leftarrow r_{\text{pre}} + 1$
 9:     else
10:         break
11:     end if
12: end while
13: if $\tilde{\mathcal{X}}_{\text{max}} \neq \emptyset$ then
14:     $\tilde{\mathcal{X}}_{\text{max}}$ is a safe set for $\mu_{\text{nn}}(\cdot)$
15: end if
```

The approach to search for an as large as possible safe set is is summarized in Algorithm 2. The main idea is to start from a very simple initial set, for example a region around the origin in case of regularization tasks, and to compute approximations of the preimages using (6.33). Note that any method to approximate preimages could be used alternatively. Each preimage is considered as a candidate safe set and verified via Algorithm 1. From Theorem 6.1, it follows that a preimage is a safe set, when Algorithm 1 terminates succesfully with $r > 0$.

6.5 Stability guarantees

Apart from safety, stability is of major interest in regularization control tasks. In presence of unknown disturbances, the methods presented in the previous chapters enable the approximation of a minimum robust positive invariant (RPI) set, if a minimum RPI exists. However, the goal of this section is to establish asymptotic stability for neural network controllers and hence we consider deterministic linear time-invariant systems:

$$x^+ = Ax + Bu, \tag{6.34}$$

and assume that the pair (A, B) is stabilizable. The classical goal of optimal control is to drive the system to the equilibrium x_{eq}, which is assumed to be at the origin without loss of generality, by minimizing a quadratic objective:

$$J = \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k, \tag{6.35}$$

where $Q \in \mathbb{R}^{n_x \times n_x} \succeq 0$ is a positive semi-definite matrix and $R \in \mathbb{R}^{n_u \times n_u} \succ 0$ is a positive definite matrix. If the optimal control problem is unconstrained, the optimal solution is given by the discrete-time infinite horizon linear-quadratic regulator (LQR), where a state feedback law:

$$u = -K_{\mathsf{lqr}} x, \tag{6.36}$$

is applied. If polytopic state constraints and input constraints are present, the violation-free application of the LQR feedback law is reduced to a region around the origin where no constraints are active. The LQR admissible region is described as a polytopic set [31]:

$$\mathcal{H}_{\mathsf{lqr}} \coloneqq \{x \in \mathbb{R}^{n_x} \mid H_{\mathsf{lqr}} x \leq h_{\mathsf{lqr}}\}, \tag{6.37}$$

where $H_{\mathsf{lqr}} \in \mathbb{R}^{n_{\mathsf{lqr}} \times n_x}$ and $h_{\mathsf{lqr}} \in \mathbb{R}^{n_{\mathsf{lqr}}}$.

6.5.1 Asymptotic stability

To establish asymptotic stability, it is necessary to prove convergence to the equilibrium x_{eq}. To do so for neural network controllers, we first analyze the behavior in the neighborhood of the origin.

Based on the activation pattern of the equilibrium region $\Gamma_{\mathsf{eq}} = \Psi(x_{\mathsf{eq}})$, computed via (6.1), the set of hyperplanes $H_{\mathsf{col}}(\Gamma_{\mathsf{eq}})$ - which define the region around the equilibrium implicitly described by the neural network - is derived through (6.5). By only considering the unique hyperplanes in $H_{\mathsf{col}}(\Gamma_{\mathsf{eq}})$, the minimal half-space representation (6.7) of the polytopic region:

$$\mathcal{H}_{\mathsf{eq}} = \{x \in \mathbb{R}^{n_x} \mid H_{\mathsf{eq}} x \leq h_{\mathsf{eq}}\}, \tag{6.38}$$

in which the equilibrium resides for a given neural network controller, is obtained. Following (6.3) and (6.6), the policy of the neural network controller can be represented for all $x \in \mathcal{H}_{\mathsf{eq}}$ as the affine state feedback:

$$\begin{aligned} \zeta(x, \Gamma_{\mathsf{eq}}; \theta) &= W_{L+1}(W_{\Gamma_{\mathsf{eq}},L} x + b_{\Gamma_{\mathsf{eq}},L}) + b_{L+1} \\ &= W_{L+1} W_{\Gamma_{\mathsf{eq}},L} x + W_{L+1} b_{\Gamma_{\mathsf{eq}},L} + b_{L+1} \end{aligned} \tag{6.39}$$

where $W_{\Gamma_{\mathsf{eq}},L}$ and $b_{\Gamma_{\mathsf{eq}},L}$ are derived from (6.6). This result can be leveraged to formulate conditions for which the feedback law (6.39) is stabilizing in the proximity of the origin.

Lemma 6.5. *If the resulting bias of the neural network is zero, that is:*

$$W_{L+1} b_{\Gamma_{\mathsf{eq}},L} + b_{L+1} = 0, \tag{6.40}$$

and the resulting weight matrix $W_{L+1} W_{\Gamma_{eq}}$ satisfies:

$$\|A + B W_{L+1} W_{\Gamma_{eq},L}\| < 1, \tag{6.41}$$

then there exists a control-invariant set $\mathcal{H}_{as}$ with a non-empty interior, such that the neural network controller is stabilizing for all $x \in \mathcal{H}_{as}$.

Proof. [Lemma 6.5] Consider the linear time-invariant system $x^+ = Ax + Bu$ and the state-feedback $u = -Kx$ satisfying

$$\|A - BK\| < 1. \tag{6.42}$$

The feedback matrix K provides asymptotically stable behavior in $\mathcal{H}_K := \{x \in \mathbb{R}^{n_x} \mid -Kx \in \mathcal{U}\}$. If a neural network controller admits an equal feedback in the equilibrium region via $K = -W_{L+1}W_{\Gamma_{\text{eq}},L}$ and $W_{L+1}b_{\Gamma_{\text{eq}},L} + b_{L+1} = 0$, this would also imply asymptotically stabilizing behavior of the neural network for all $x \in \mathcal{H}_K$. Since the neural network controller only guarantees equilibrium feedback within $\mathcal{H}_{\text{eq}}$, asymptotic stability can only be guaranteed for a control-invariant set:

$$\mathcal{H}_{\text{as}} \subseteq (\mathcal{H}_{\text{eq}} \cap \mathcal{H}_K). \qquad \square$$

Theorem 6.2. *If the neural network controller satisfies* (6.40) *and* (6.41) *and there exists a $r \in \mathbb{N}$ for a set of initial conditions $\mathcal{X}_s$ such that $\tilde{\mathcal{R}}_r(\mathcal{X}_s) \subseteq \mathcal{H}_{as}$, then the closed-loop system $x^+ = Ax + B\mu_{nn}(x)$ is asymptotically stable for all $x \in \mathcal{X}_s$.*

Proof. [Theorem 6.2] Because (6.40) and (6.41) are satisfied, asymptotic stability for all $x \in \mathcal{H}_{\text{as}}$ follows from Lemma 6.5. Since $\mathcal{R}_r(\mathcal{X}_{\text{s}}) \subseteq \tilde{\mathcal{R}}_r(\mathcal{X}_{\text{s}})$, as shown in Lemma 6.3, and $\tilde{\mathcal{R}}_r(\mathcal{X}_{\text{s}}) \subseteq \mathcal{H}_{\text{as}}$, the system reaches $\mathcal{H}_{\text{as}}$ at least after r closed-loop steps for all $x \in \mathcal{X}_{\text{s}}$. Hence, $\mu_{\text{nn}}(\cdot)$ guarantees asymptotic stability for all $x \in \mathcal{X}_{\text{s}}$. $\square$

If the feedback defined by the neural network in the region $\mathcal{H}_{\text{eq}}$ is equal to the LQR controller, then the neural network controller behaves optimally in the neighborhood of the equilibrium point. This is formalized in the following result.

Corollary 6.1. *Consider the feedback of the neural network controller in the neighborhood of the origin* (6.39) *is equal to the LQR feedback:*

$$\zeta(x, \Gamma_{eq}; \theta) = -K_{lqr}x$$

and there exists a $r \in \mathbb{N}$ such that $\tilde{\mathcal{R}}_r(\mathcal{X}_{\text{s}}) \subseteq \mathcal{H}_{as}$. Then the system converges asymptotically to the equilibrium for all $x \in \mathcal{X}_s$. In addition, for all $x \in \mathcal{H}_{as}$, the neural network controller is optimal with respect to (6.35).

Proof. [Corollary 6.1] The LQR state feedback $u = -K_{\text{lqr}}x$ is the optimal solution with respect to (6.35) for all $x \in \mathcal{H}_{\text{lqr}} := \left\{x \in \mathbb{R}^{n_x} \mid -K_{\text{lqr}}x \in \mathcal{U}\right\}$. If $\zeta(x, \Gamma_{\text{eq}}; \theta) = -K_{\text{lqr}}x$, the neural network controller provides the LQR feedback for all $x \in \mathcal{H}_{\text{eq}}$. Hence, the neural network controller returns an LQR optimal control input with respect to (6.35) for all $x \in (\mathcal{H}_{\text{eq}} \cap \mathcal{H}_{\text{lqr}})$. Because the LQR feedback is a special case of an asymptotically stabilizing feedback:

$$K_{\text{lqr}} \in \{K \in \mathbb{R}^{n_u \times n_x} \mid \|A - BK\| < 1\}, \tag{6.43}$$

the proof for asymptotic stability and convergence to the stability set $\mathcal{H}_{\text{as}}$ is analogous to the proof of Lemma 6.5 and Theorem 6.2 by substituting K with K_{lqr} and $\mathcal{H}_K$ with $\mathcal{H}_{\text{lqr}}$. $\square$

In general, a neural network controller does not satisfy the requirements (6.40) and (6.41) for asymptotic stability. The next subsection shows an optimization-based method to ensure that such requirements are satisfied.

6.5.2 LQR-optimized neural network controller

In this subsection, an optimization-based method is presented to modify a neural network such that it provides the same feedback as an LQR controller in a region around the equilibrium without changing the regions implicitly defined by the ReLU layers and without requiring a retraining of the neural network. The goal is that the modified controller satisfies the conditions of Theorem 6.1. The main idea is to only adapt the values of the weight and the bias in the linear output layer $L+1$, because the regions on which the affine functions are defined for a ReLU network are only depending on the L hidden layers, as (6.5) shows.

Lemma 6.6. *For every neural network controller $\mu_{nn}(x;\theta,\cdot)$, it is possible to find values $\hat{W}_{L+1}$ for the weights of the final layer, such that*

$$\hat{W}_{L+1} W_{\Gamma_{eq},L} = -K_{lqr}, \tag{6.44}$$

if the following requirements are satisfied:

$$rank(A_{eq}) = rank\left(\begin{bmatrix} A_{eq} & b_{eq} \end{bmatrix}\right), \tag{6.45a}$$

$$n_u n_L \geq rank\left(A_{eq}\right), \tag{6.45b}$$

where

$$A_{eq} = \begin{bmatrix} W^T_{\Gamma_{eq},L} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & W^T_{\Gamma_{eq},L} \end{bmatrix} \text{ and } b_{eq} = \begin{bmatrix} -K^{(i)^T}_{lqr} \\ \vdots \\ -K^{(n_u)^T}_{lqr} \end{bmatrix}.$$

Proof. [Lemma 6.6] Equation (6.44) can be reformulated as a system of linear equations $A_{\text{eq}} \hat{W}_{\text{vec}} = b_{\text{eq}}$ with $\hat{W}_{\text{vec}} = [\hat{W}^{(1)}_{L+1}, \ldots, \hat{W}^{(n_u)}_{L+1}]^T \in \mathbb{R}^{n_u n_L}$. A system of linear equations admits at least one solution if conditions (6.45) are satisfied. □

Based on this result we propose a systematic adaptation of the weights of the last layer to ensure that LQR performance is achieved in a region around the equilibrium point.

Theorem 6.3. *If conditions (6.45) are satisfied, the convex optimization problem:*

$$\underset{\hat{W}_{L+1},\hat{b}_{L+1}}{\text{minimize}} \quad \sum_{i=1}^{n_u} \sum_{j=1}^{n_L} \left(\hat{W}^{(i,j)}_{L+1} - W^{(i,j)}_{L+1}\right)^2 + \sum_{i=1}^{n_u} \left(\hat{b}^{(i)}_{L+1} - b^{(i)}_{L+1}\right)^2 \tag{6.46a}$$

$$\text{subject to} \quad \hat{W}_{L+1} W_{\Gamma_{eq},L} = -K_{lqr}, \tag{6.46b}$$

$$\hat{W}_{L+1} b_{\Gamma_{eq},L} + \hat{b}_{L+1} = 0, \tag{6.46c}$$

admits a solution which provides the weight $\hat{W}^*_{L+1}$ and bias $\hat{b}^*_{L+1}$ for the last layer, such that $\mu_{nn}(x;\theta_{lqr},\cdot) = -K_{lqr}x$ for all $x \in \mathcal{H}_{eq}$ with $W_{l,lqr} = W_l$ and $b_{l,lqr} = b_l$ for all $l \in \mathbb{N}^+_{[L]}$, and $W_{L+1,lqr} = \hat{W}^*_{L+1}$ and $b_{L+1,lqr} = \hat{b}^*_{L+1}$, while minimizing the change in parameters of the last layer.

Proof. [Theorem 6.3] If the conditions (6.45) are satisfied, Lemma 6.6 guarantees that (6.46b) can always be satisfied. Condition (6.46c) can be satisfied for every $\hat{W}_{L+1}$ by setting $\hat{b}_{L+1} = -\hat{W}_{L+1}b_{\Gamma_{\text{eq}}}$. This means that the feedback of the neural network controller in the equilibrium region is equal to the LQR feedback if the weight and the bias of the last layer are given by $\hat{W}^*_{L+1}$ and bias $\hat{b}^*_{L+1}$. Since the objective function (6.46a) describes the change in the parameters of the last layer and the optimization is convex, the optimal solution guarantees that the change is minimal. □

Now that we have shown that neural network controllers can guarantee closed-loop operation without constraint violations and asymptotically stabilizing behavior, we investigate how performance guarantees can be derived.

6.6 Performance bounds

Another major aspect of controller analysis is the closed-loop performance. Consider the stage and terminal cost as the objective in the mixed-integer formulation (6.25) for a trajectory **x** of length r of the uncertain LTI system starting in $\mathcal{X}_{\text{s}}$ using a neural network controller as described in (6.14). An upper bound for the worst-case cost in closed-loop operation can be computed by solving the following problem:

$$\begin{aligned} &\underset{\mathbf{z},\mathbf{v},\mathbf{u},\mathbf{d},\mathbf{x}}{\text{maximize}} && \sum_{k=0}^{r-1} \ell(x_k,u_k) + \ell_{\text{f}}(x_r) && \text{(6.47a)} \\ &\text{subject to} && \text{(6.25b)} - \text{(6.25k)}. \end{aligned}$$

Usually, we are not only interested in performance guarantees for trajectories of finite length, but in asymptotic performance guarantees like the average stage cost:

$$\ell_r(\mathbf{x}) = \frac{1}{r}\sum_{k=1}^{r} \ell\left(x_k, \mu_{\text{nn}}\left(x_k\right)\right), \tag{6.48}$$

with $r \to \infty$. Since (6.47) becomes intractable for large values of r, the knowledge of invariant sets can help mitigate this issue. If a robust positive invariant set $\mathcal{X}_{\text{RPI}} := \{x \in \mathbb{R}^{n_x} \mid H_{\text{RPI}}x \le h_{\text{RPI}}\}$ exists, it is sufficient to consider a single closed-loop step in:

$$\begin{aligned} &\underset{z,v,\bar{u}_0,d_0,\bar{x}_0}{\text{maximize}} && \ell(\bar{x}_0,\bar{u}_0) && \text{(6.49a)} \\ &\text{subject to} && H_{\text{RPI}}\bar{x}_0 \le h_{\text{RPI}}, && \text{(6.49b)} \\ &&& z_0 = \bar{x}_0, && \text{(6.49c)} \\ &&& \text{for all } l \in \mathbb{N}^+_{[L]}: \\ &&& \quad z_l \ge W_l z_{l-1} + b_l, && \text{(6.49d)} \end{aligned}$$

$$
\begin{aligned}
z_l &\leq W_l z_{l-1} + b_l + M(\mathbf{1} - v_l), && (6.49e)\\
z_l &\geq 0, && (6.49f)\\
z_l &\leq M v_l, && (6.49g)\\
v_l &\in \{0,1\}^{n_l}, && (6.49h)\\
\bar{u}_0 &= W_{L+1} z_L + b_{L+1}, && (6.49i)
\end{aligned}
$$

to bound the worst-case stage cost for all $x \in \mathcal{X}_{\mathsf{RPI}}$ as the system state will reside within $\mathcal{X}_{\mathsf{RPI}}$ for all times. The upper bound $\bar{\ell}$ of the asymptotic average stage cost (6.48) is then obtained via the optimal solution $\bar{x}_0^*$ and $\bar{u}_0^*$ of (6.49) as:

$$\bar{\ell} = \ell\left(\bar{x}_0^*, \bar{u}_0^*\right) \geq \ell\left(x, \mu_{\mathsf{nn}}(x)\right) \quad \forall x \in \mathcal{X}_{\mathsf{RPI}}. \tag{6.50}$$

The bound $\bar{\ell}$ can be leveraged to obtain the asymptotic worst-case performance for a set of initial conditions $\mathcal{X}_{\mathsf{s}}$ from which all trajectories converge to an RPI set.

Theorem 6.4. *If the RPI set $\mathcal{X}_{RPI}$ exists and is an r-step reachable set of $\mathcal{X}_{\mathsf{s}}$, the asymptotic average stage cost will converge to or descend below the solution $\bar{\ell}$* (6.50) *of the mixed-integer problem* (6.49) *for every closed-loop trajectory $\mathbf{x} = \{x_0, \ldots, x_\infty\}$ satisfying* (6.14) *and starting at $x_0 \in \mathcal{X}_{\mathsf{s}}$.*

Proof. [Theorem 6.4] Consider some trajectory $\mathbf{x}$ of length $\bar{r} \geq r$ starting in $\mathcal{X}_{\mathsf{s}}$. Because $\mathcal{X}_{\mathsf{RPI}}$ is robustly positive invariant and an r-step reachable set of $\mathcal{X}_{\mathsf{s}}$, i.e. $\mathcal{R}_r(\mathcal{X}_{\mathsf{s}}) \subseteq \mathcal{X}_{\mathsf{RPI}}$, it follows that $x_k \in \mathcal{X}_{\mathsf{RPI}}$ for all $k \geq r$. Hence, the solution of (6.49) provides an upper bound on the stage cost for all steps $k \geq r$. The average cost for the trajectory is then bounded by:

$$
\begin{aligned}
\ell_{\bar{r}}(\mathbf{x}) &= \frac{1}{\bar{r}} \left(\sum_{k=0}^{r-1} \ell\left(x_k, \mu_{\mathsf{nn}}\left(x_k\right)\right) + \sum_{k=r}^{\bar{r}} \ell\left(x_k, \mu_{\mathsf{nn}}\left(x_k\right)\right) \right)\\
&\leq \frac{1}{\bar{r}} \left(\sum_{j=1}^{r-1} \ell(x_j, \mu_{\mathsf{nn}}(x_j)) + (\bar{r} + 1 - r)\bar{\ell} \right).
\end{aligned}
$$

From $\bar{r} \to \infty$ follows $\ell_\infty(\mathbf{x}) \to \bar{\ell}$. □

Theorem (6.4) is especially effective if a minimum RPI set exists, as will be shown in Section 6.7.

The various methods introduced in the previous sections are summarized in a step-by-step scheme which enables the derivation of neural network controllers with safety, performance and stability guarantees. The scheme consisting of 7 steps is illustrated in Fig. 6.4.

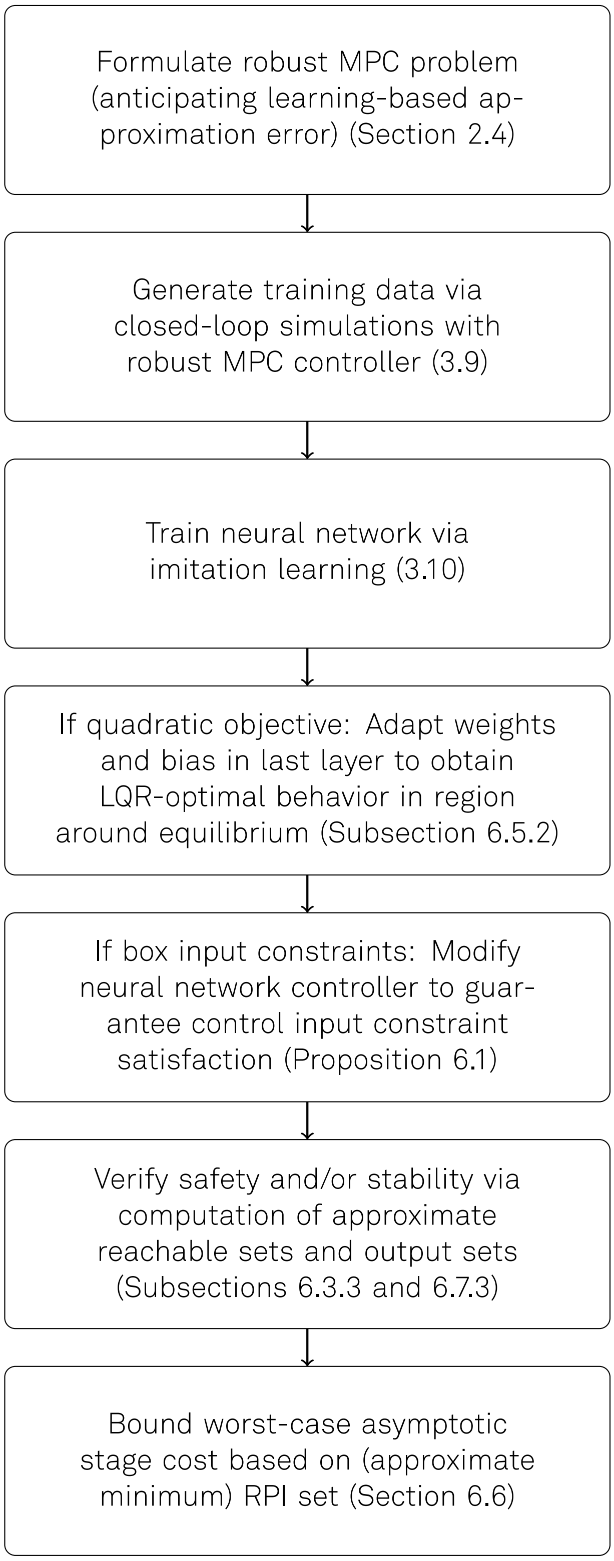

Formulate robust MPC problem (anticipating learning-based approximation error) (Section 2.4)
Generate training data via closed-loop simulations with robust MPC controller (3.9)
Train neural network via imitation learning (3.10)
If quadratic objective: Adapt weights and bias in last layer to obtain LQR-optimal behavior in region around equilibrium (Subsection 6.5.2)
If box input constraints: Modify neural network controller to guarantee control input constraint satisfaction (Proposition 6.1)
Verify safety and/or stability via computation of approximate reachable sets and output sets (Subsections 6.3.3 and 6.7.3)
Bound worst-case asymptotic stage cost based on (approximate minimum) RPI set (Section 6.6)

Figure 6.4: Detailed overview on how to leverage the introduced methods to obtain a neural network controller with safety, performance and stability guarantees.

6.7 Case studies

The proposed methods are illustrated and visualized via a simple two-dimensional case study. We will follow the framework presented in Fig. 6.4 to obtain sets in which the application of the neural network controller is safe with guarantees on the closed-loop performance.

6.7.1 Double integrator

The system matrices (6.14) for the double integrator are given by

$$A = \begin{bmatrix} 1.000 & 0.100 \\ 0.000 & 1.000 \end{bmatrix}, B = \begin{bmatrix} 0.005 \\ 0.100 \end{bmatrix} \text{ and } E = \begin{bmatrix} 0.100 & 0.000 \\ 0.000 & 0.100 \end{bmatrix}.$$

The state constraints and control input constraints are both box constraints with $\mathcal{X} = \{x \in \mathbb{R}^{n_x} \mid -3 \leq x \leq 3\}$ and $\mathcal{U} = \{u \in \mathbb{R}^{n_u} \mid -1 \leq u \leq 1\}$. The weight matrices for the quadratic objective function (2.10a) are given by $Q = 2I$, $R = I$ and $P = 2I$.

6.7.2 Deriving the neural network controller via imitation learning

We chose multi-stage MPC to generate data sets for imitation learning and included the approximation error of the learned controller - estimated as $||\mu_{\text{nn}}(x) - \mu_{\text{ms}}||_2^2 \leq 0.02$ - in the scenario tree formulation. The multi-stage MPC controller was implemented via do-mpc [122], a robust nonlinear MPC toolbox relying on CasADi [123], a symbolic toolbox for automatic differentiation and IPOPT [124], an interior-point solver.

We leveraged the method from [125] to compute the maximum robust control invariant set of the MPC formulation. For generating the learning data $\mathcal{T}$, 500 random initial states within the maximum robust control invariant set were sampled from which 20 closed-loop steps were simulated, resulting in 10000 samples.

We considered a small network with a single hidden layer containing 15 neurons to approximate the MPC law. The optimal network parameters for the learning-based controllers were derived via (3.8) using Keras [89] and TensorFlow [90] with the ADAM algorithm [92]. The resulting neural network controller and the regions that are implicitly defined by the neurons in the hidden layer are portrayed in Fig. 6.5 including the maximum robust control invariant set, the equilibrium region and the equilibrium point x_{eq}, i.e. the origin.

6.7.3 Asymptotic stability

In order to analyze asymptotic stability, we assume that there are no disturbances, i.e. $\mathcal{D} = \{0\}$. Asymptotic stability can be guaranteed if two conditions are met. First, the closed-loop system needs to converge to $\mathcal{H}_{\text{as}} \subseteq (\mathcal{H}_{\text{lqr}} \cap \mathcal{H}_{\text{eq}})$ for all $x \in \mathcal{X}_{\text{s}}$. The stability set $\mathcal{H}_{\text{as}}$ is computed via [125] as the maximum robust control invariant set with state constraints $x \in (\mathcal{H}_{\text{lqr}} \cap \mathcal{H}_{\text{eq}})$. Second, the neural network controller needs to have a stabilizing feedback in $\mathcal{H}_{\text{as}}$. To find the weights and bias of the final layer providing stabilizing feedback, i.e. the LQR feedback for all $x \in \mathcal{H}_{\text{eq}}$, problem (6.46)

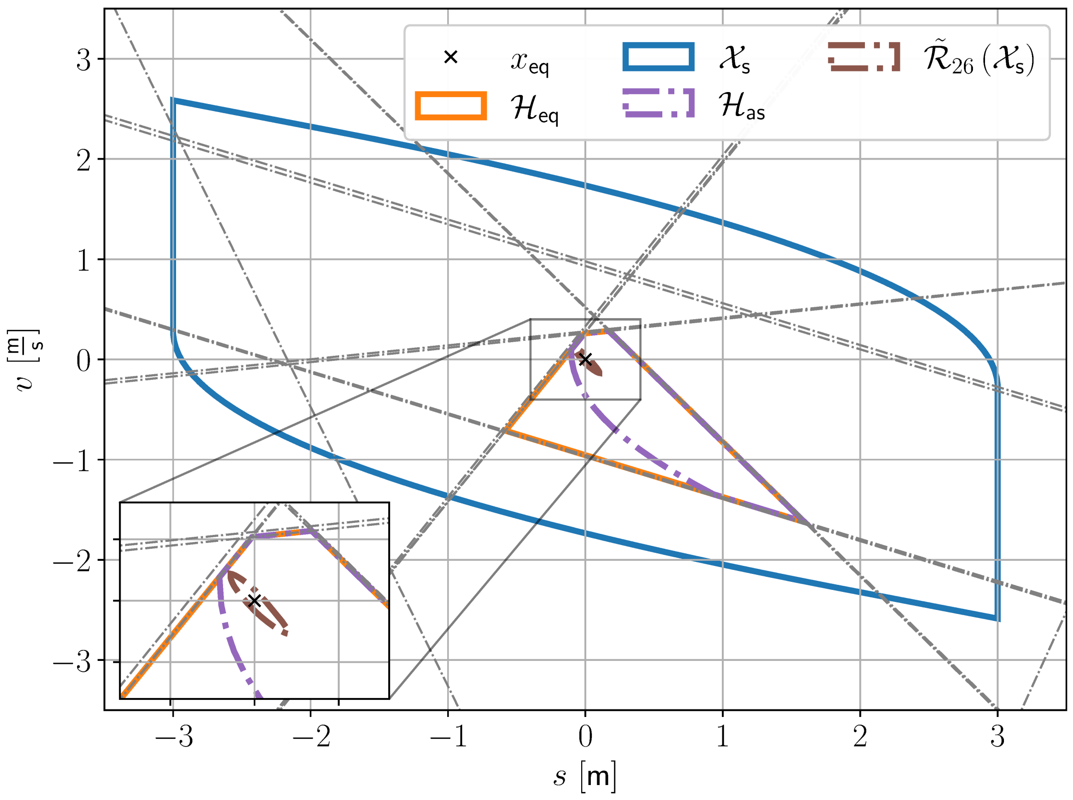


Figure 6.5: The dash-dotted grey lines indicate the hyperplanes that are implicitly defined by the hidden layers of the neural network. The maximum robust control invariant set obtained from the MPC formulation and the equilibrium set are highlighted in in blue and orange.

was solved with $K_{\mathsf{lqr}} = \begin{bmatrix}0.9171 & 1.6356\end{bmatrix}$. The LQR adaptation (6.46) was feasible with an optimal cost of 0.267. The 26-step reachable set of the maximum robust control invariant set is a subset of $\mathcal{H}_{\mathsf{as}}$ which means that $\mu_{\mathsf{lqr}}(\cdot)$ guarantees asymptotic stability for every trajectory starting in the maximum control-invariant set. The 26-step reachable set lies within the boundaries of $\mathcal{H}_{\mathsf{as}}$ and contains the origin as can be seen in the magnified area of Fig. 6.5.

The performance of the original controller $\mu_{\mathsf{nn}}(\cdot)$, whose weights in the last layer have not been optimized, and the LQR-optimized controller $\mu_{\mathsf{lqr}}(\cdot)$ is compared in Fig. 6.6. Note that both controllers were adapted as described in Proposition 6.1 in order to guarantee satisfaction of the control input constraints. The closed-loop trajectories are very similar, as the low cost for the LQR optimal adaptation suggested, but only $\mu_{\mathsf{lqr}}(\cdot)$ drives the system asymptotically to the origin, which can be seen in the magnified area of Fig. 6.6. Our proposal solves an important problem of approximate MPC controllers as there is no set-point tracking error. However, as asymptotic stability does not imply constraint satisfaction, safety concerns have to be considered separately, especially in presence of unknown disturbances.

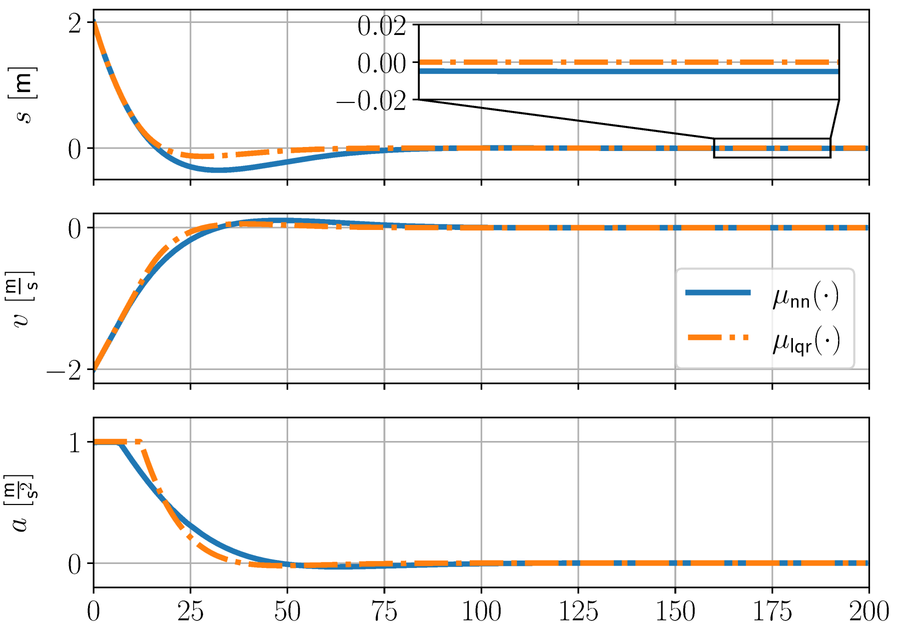


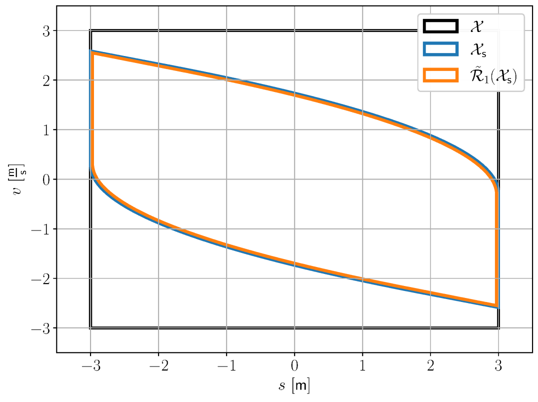
Figure 6.6: Exemplary state trajectories of the closed-loop systems for the LQR optimized controller $\mu_{\text{lqr}}(\cdot)$ and the original controller $\mu_{\text{nn}}(\cdot)$ for the same initial state x_0. The output of both controllers is saturated according to the box input constraints as described in Proposition 6.1.

6.7.4 Safety and performance guarantees

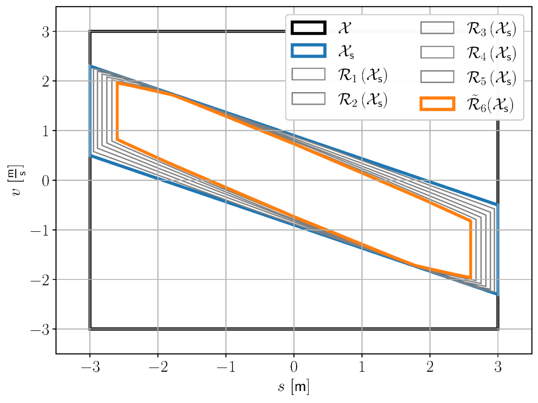
To ensure the robust satisfaction of the state and control input constraints, Algorithm 1 can be followed. The mixed-integer formulations (6.23), (6.25) and (6.32) used in the algorithms were formulated via YALMIP [126] and solved with Gurobi [127]. For the geometric computations, the multi-parametric toolbox [108] was used.

We consider two candidate sets to be verified as safe sets as in Definition (6.2) via Algorithm 1. The first candidate set is a robust control-invariant set computed via the method given in [125] for the robust MPC formulation including the approximation error and was verified as a robust positive invariant set of the closed-loop operated by the neural network controller $\mu_{\text{lqr}}(\cdot)$, as illustrated in Fig. 6.7a. The second candidate set is derived from observing closed-loop simulations with the neural network controller and estimating an area from which exclusively violation-free trajectories originated. The estimated set is verified as an admissible 6-step robust control invariant set of the controller $\mu_{\text{lqr}}(\cdot)$, as shown in Fig. 6.7b, and hence guarantees constraint satisfaction in closed-loop operation. Note that various other sensible candidate sets to be verified might exist.

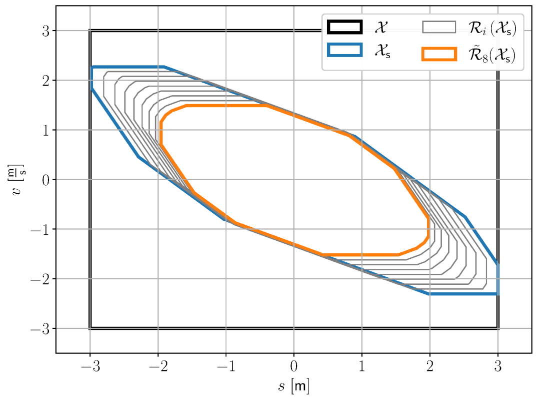
In the next step, the worst-case performance is investigated. The minimum RPI set is approximated as the 40-step reachable set of the candidate set, i.e. $\mathcal{X}_{\text{mRPI}} \approx \tilde{\mathcal{R}}_{40}(\mathcal{X}_{\text{s}})$. The approximate minimum RPI set was used to bound the asymptotic average stage cost by solving (6.49). The evolution of the average stage cost (6.48) along



(a) $\mathcal{X}_s$ as robust control-invariant set of $\mu_{mpc}(\cdot)$



(b) $\mathcal{X}_s$ derived from closed-loop observations



(c) $\mathcal{X}_s$ as $\tilde{\mathcal{X}}_{max}$ from Algorithm 2

Figure 6.7: Safety verification process of candidate sets for the double integrator system.

the trajectory can be seen in Fig. 6.8 for 50 randomly sampled initial states. After at most 190 steps, the average stage costs sinks below the computed bound.

In a last step, Algorithm 2, which uses preimages of the closed-loop system to compute an as large as possible safe set starting from a small set around the equilibrium, is applied. As the initial set, we considered the previously computed approximate minimum RPI set and the algorithm terminated after 20 iterations. The application of the robust neural network controller is therefore guaranteed to be safe for the robust control-invariant set computed based on the MPC formulation, the candidate set obtained as a guess from many observed closed-loop trajectories, and for the safe set found using Algorithm 2 as depicted in Fig. 6.7c.

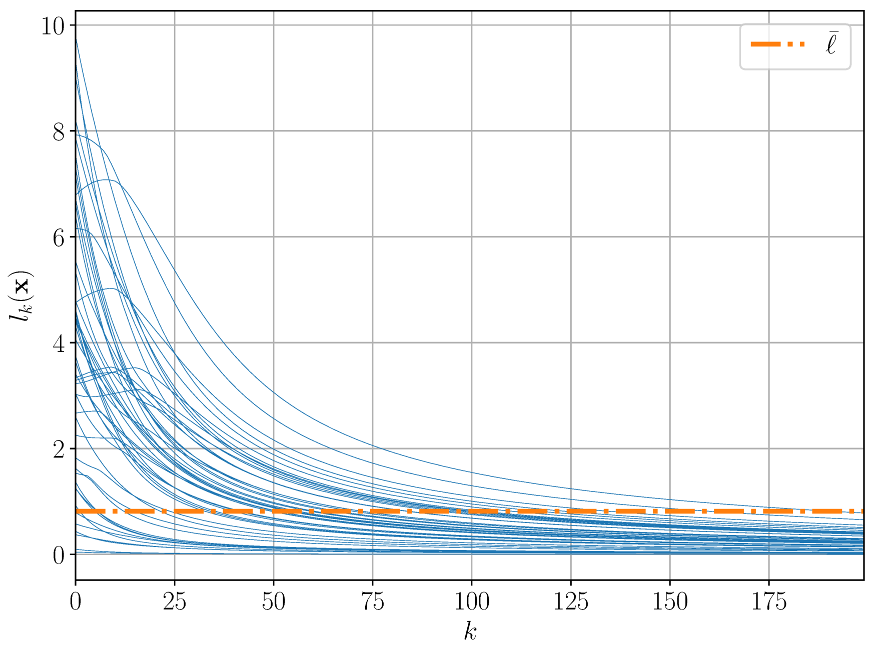


Figure 6.8: Evolution of the average stage cost over 200 closed-loop steps for 50 closed-loop trajectories. After at least $k = 190$ steps the average stage cost sinks below the infinite horizon worst case bound.

Part III

Deep learning-based control for nonlinear systems

Chapter 7

Learning-based output-feedback control

In the previous part, deep neural networks were used for the state-feedback control of linear systems. Despite the technical ability of deep neural networks to exactly represent linear MPC solutions, the more practical approach is to leverage the representation capabilities to closely approximate complex functions such as the robust control laws of uncertain nonlinear systems. In this chapter, it will be shown that deep neural networks have the ability to act as an explicit output-feedback controller for systems described by:

$$\begin{aligned} x^{+} &= f(x, u, d), \\ y &= h(x, u, d) + v_y, \end{aligned} \tag{7.1}$$

where $y \in \mathbb{R}^{n_y}$ are the measurements and $v_y \in \mathbb{R}^{n_y}$ is the measurement noise.

In order to apply MPC-like strategies in the output-feedback case, an accurate estimate of the true state is required. Feed-forward neural networks can provide an estimate of the current state of a system based on past measurements and applied inputs by mirroring the behavior state estimation strategies like moving horizon estimation [128]. We also propose the simultaneous learning of the full estimator-controller algorithm, i.e. the direct mapping from the currently obtained measurements to the optimal control input.

The remainder of the chapter is organized as follows. In Section 7.1, moving horizon estimation, an optimization-based state estimation approach is presented, which is also used to generate the training data set for the learning-based state estimator. The approach to obtain deep learning-based approximations of the robust output-feedback controller and the nonlinear state estimator is illustrated in Section 7.2. Finally, we highlight the advantages of the proposed method by applying the method for an industrial-scale case study which is presented in Section 7.3. The findings in this this chapter were initially presented in [8].

7.1 Moving horizon estimation

The goal of moving horizon estimation (MHE) is to provide an accurate state estimate x_{est} based on N_{est} past measurement and N_{est} past inputs by solving an optimization problem. The optimization problem to be solved shares a similar structure like MPC formulations in the sense that both rely on a system model and use a shifting time-window. But whereas the MPC formulation uses the system model to predict the future behavior of the system to derive optimal control inputs, MHE looks back in order recover the past evolution including an accurate estimate of the current state. The MHE problem is defined in nonlinear least-squares fashion:

$$\begin{aligned}
\underset{\mathbf{x},d_{\text{est}},\mathbf{v}_x,\mathbf{v}_y}{\text{minimize}} \quad & \frac{1}{2}||x_{-N_{\text{est}}} - x_{\text{a}}||^2_{P_a} + \frac{1}{2}||d_{\text{est}} - d_{\text{prev}}||^2_{P_d} \\
& + \frac{1}{2}\sum_{k=-N_{\text{est}}}^{0} ||v_{y,k}||^2_{P_y} + \frac{1}{2}\sum_{k=-N_{\text{est}}}^{-1} ||v_{x,k}||^2_{P_x} && \text{(7.2a)} \\
\text{subject to} \quad & \text{for } k \in \{-N_{\text{est}}, -N_{\text{est}}+1, \dots, -1\}: \\
& \quad x_{k+1} = f\left(x_k, u_k, d_{\text{est}}\right) + v_{x,k}, && \text{(7.2b)} \\
& \quad 0 \geq g_{\text{est}}(x_k, u_k, d_{\text{est}}), && \text{(7.2c)} \\
& \text{for } k \in \{-N_{\text{est}}, -N_{\text{est}}+1, \dots, 0\}: \\
& \quad y_k = h(x_k, u_k, d_{\text{est}}) + v_{y,k}, && \text{(7.2d)} \\
& d_{\text{est}} \in \mathcal{D}, && \text{(7.2e)}
\end{aligned}$$

where $P_a \in \mathbb{R}^{n_x \times n_x}$, $P_d \in \mathbb{R}^{n_d \times n_d}$, $P_y \in \mathbb{R}^{n_y \times n_y}$ and $P_x \in \mathbb{R}^{n_x \times n_x}$ are the weights for the arrival cost, the parameter estimation, the distance between measurements and computed measurements, and the process noise v_x with the notation $||x||^2_P = ||x^T P x||$. The optimization variables are the estimated state trajectory $\mathbf{x} = \{x_{-N_{\text{est}}}, \dots, x_0\}$, sequence of process noise $\mathbf{v}_x = \{v_{x,-N_{\text{est}}}, \dots, v_{x,-1}\}$ with $v_{x,k} \in \mathbb{R}^{n_x}$, sequence of measurement noise $\mathbf{v}_y = \{v_{y,-N_{\text{est}}}, \dots, v_{y,0}\}$ and the current parameter estimate d_{est}. The parameter estimate from the previous step is given by d_{prev} and the parameters are assumed to be constant, as the considered case study, that will be presented in Section 7.3, is a semi-batch reactor with constant parameter values in each batch. If the set to which the uncertainties belong is known, this information can be incorporated via (7.2e). Estimating the values of the uncertain parameters is important in order to obtain accurate state estimates. The constraints (7.2c) can be chosen differently in comparison to their equivalents in the multi-stage MPC formulation because relaxing the constraints for the MHE to enable accurate state estimation - even in the presence of minor constraint violations in the control task - can be beneficial. The values in P_d should be chosen such that only the values of uncertain parameters are penalized and not also the disturbances. To counteract the information loss due to the finite horizon, the arrival cost (the first term in (7.2a)) includes information from previous estimations via x_{a}, which is the $x^*_{-N_{\text{est}}+1}$ from the previous estimation. In the case of output-feedback, the state estimate $x_{\text{est}} \in \mathbb{R}^{n_x}$ computed by the MHE, which is the state at the last time in the considered time-window, i.e. $x_{\text{est}} = x^*_0$, is used as the initial in the MPC formulation (2.8b) via $x_{\text{init}} = x_{\text{est}}$.

The combination of MHE and MPC is a powerful choice to achieve high-performance output-feedback control that can systematically deal with nonlinear systems and constraints. The main drawback of the combination is that, apart from the robust optimal control problem, an additional computationally complex optimization problem to derive a state estimate needs to be solved at each control instant, adding more stress to the potentially limited computational resources. To counteract these drawbacks, we propose to not only approximate the optimal controller, but also the state estimator, via deep neural networks.

7.2 Deep learning-based robust NMPC and MHE

The goal of this chapter is to use deep neural networks to derive an optimization-free approach for the robust output-feedback control of uncertain nonlinear systems, As for the case of approximating MPC controllers, we exploit that the optimization problem in (7.2) is parametric. The parameters that determine the solution of the MHE problem are the past measurement and the input trajectory as well as the previously estimated state and parameters. Because we do not only use deep neural networks as controllers, but also as explicit state estimators, we define the neural networks here in a more general form in comparison to (3.2), as:

$$\mu_{\mathsf{nn}}(x_{\mathsf{in}};\theta,n_0,\ldots,n_{L+1}) = \lambda_{L+1}\circ\prod_{l=1}^{L}\left(\sigma_l\circ\lambda_l(z_{l-1})\right), \tag{7.3}$$

where $z_{l-1} = \sigma_l(W_{l-1}z_{l-2}+b_{l-1})$, $l\in\{2,\ldots,L+1\}$, and $z_0 = x_{\mathsf{in}}$ is either the (estimated) state of the system in case of the controller and the parameters of the MHE problem (7.2) in case of the state estimator. The activation function used in this chapter is the hyperbolic tangent:

$$\sigma_l(\lambda_l(\cdot)) = \tanh(\lambda_l(\cdot)) = \frac{e^{\lambda_l(\cdot)}-e^{-\lambda_l(\cdot)}}{e^{\lambda_l(\cdot)}-e^{-\lambda_l(\cdot)}}, \tag{7.4}$$

which is evaluated element-wise on the input vector.

7.2.1 Learning an approximate explicit representation

We consider three different neural network structures to approximate the functions implicitly described by the output-feedback multi-stage NMPC problem (2.8), which uses the state estimate as the initial state via $x_{\mathsf{init}} = x_{\mathsf{est}}$, and by the MHE problem (7.2). The first two networks approximate directly either the MPC, leading to the neural network controller (NNC), or the MHE, leading to the neural network estimator (NNE). The third network simultaneously learns a controller and an estimator, leading to the neural network estimator and controller (NNEC). The crucial part for learning a good approximation of the MPC and MHE are the data sets, which need to contain enough samples to enable the learned networks to achieve the desired approximation accuracy, and more importantly the desired closed-loop performance, while having a tractable size.

For the data generation, we use closed-loop simulations, as opposed to purely random sampling, quasi-random sampling [129] or gridding, to ensure that only samples that represent situations occurring during operation of the system are included in the data set. Every simulation step out of the n_{tr} considered steps is fully described by a tuple $(s_p, x_{\text{est},p}, x_p, \mu_{\text{ms}}(x_{\text{est},p}), d_p)$, where

$$s_p = \left[x_{\text{a}}^T, y_{-N_{\text{est}}}^T, \dots, y_0^T, u_{-N_{\text{est}}}^T, \dots, u_{-1}^T\right]^T$$

contains the arrival state x_{a}, past measurements $y_{-N_{\text{est}}}, \dots, y_0$ and the applied control inputs $u_{-N_{\text{est}}}, \dots, u_{-1}$ used by the MHE to generate the state estimate $x_{\text{est},p}$, x_p is the current state of the system, $\mu_{\text{ms}}(x_{\text{est},p})$ is the optimal control input computed by the multi-stage MPC based on the state estimate and d_p is the realization of the uncertain parameters. These tuples enable the learning of an NNE and an NNC either separately or simultaneously.

The behavior of the MPC is approximated by the network $\mu_{\text{C}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ and the behavior of the MHE by $\mu_{\text{E}} : \mathbb{R}^{n_s} \to \mathbb{R}^{n_x}$ with $n_s = n_x + N_{\text{est}} \cdot (n_y + n_u)$. That is, the approximate MHE takes as input the trajectories of inputs and measurements of length N_{est} as well as the previously estimated state as in the original MHE formulation. The output of the approximate MHE is the current state estimate.

The third learning-based approach uses the same information as the network for state estimation, but directly gives an optimal input in addition to the state estimate which results in a mapping $\mu_{\text{EC}} : \mathbb{R}^{n_s} \to \mathbb{R}^{n_x+n_u}$. The state estimates of NNE and NNEC will be used after $N_{\text{est}} - 1$ steps as the arrival state x_{a} in closed-loop operation. The optimal weights and biases for each network are determined during training by minimizing the following loss functions:

$$\text{NNC}: \qquad \underset{\theta}{\text{minimize}} \quad \frac{1}{n_{\text{tr}}} \sum_{p=1}^{n_{\text{tr}}} ||\mu_{\text{ms}}(x_{\text{est},p}) - \mu_{\text{C}}(x_{\text{est},p})||_2^2\,, \tag{7.5a}$$

$$\text{NNE}: \qquad \underset{\theta}{\text{minimize}} \quad \frac{1}{n_{\text{tr}}} \sum_{p=1}^{n_{\text{tr}}} ||x_p - \mu_{\text{E}}(s_p)||_2^2\,, \tag{7.5b}$$

$$\text{NNEC}: \qquad \underset{\theta}{\text{minimize}} \quad \frac{1}{n_{\text{tr}}} \sum_{p=1}^{n_{\text{tr}}} \left|\left| \begin{bmatrix} x_p \\ \mu_{\text{ms}}(x_{\text{est},p}) \end{bmatrix} - \mu_{\text{EC}}(s_p) \right|\right|_2^2 . \tag{7.5c}$$

It is interesting to note some differences between the training of the approximations of the control and estimation problems. While in (7.5a) the neural network sees exactly the same data as the MPC in the closed-loop, in (7.5b) and (7.5c) we exploit that the simulation-based data generation grants access to the real state. Because x_p would be the result of a perfect estimation, we use it as the target instead of the state $x_{\text{est},p}$ estimated by the MHE. The possible configurations for the output-feedback control task, combining optimization-based and learning-based methods, are visualized in Fig. 7.1 where also MPC with state-feedback is presented, as it shows the performance in the ideal case, e.g. when the estimator provides exact state estimates and the MPC optimization problem is exactly solved.

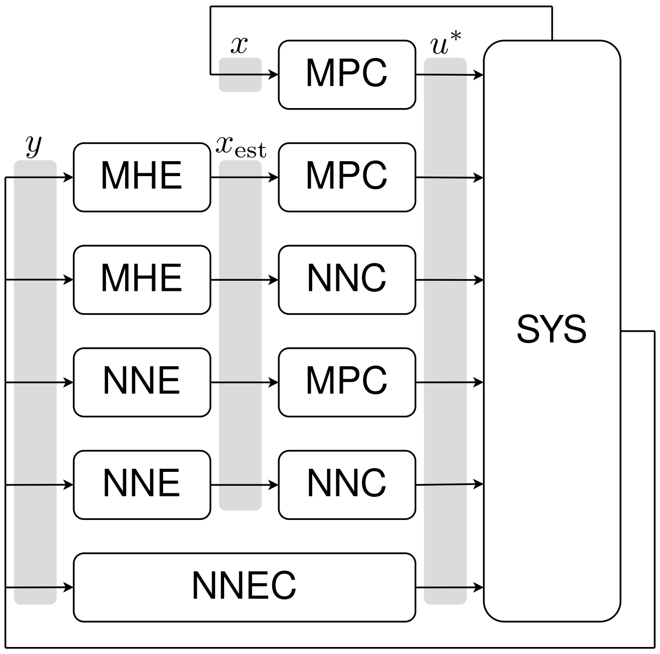


Figure 7.1: Overview of the different configurations which can be built from MHE, MPC and their neural network based approximations NNE and NNC. MPC via state-feedback is also included which would represent the perfect estimator and NNEC is using past measurements and control inputs to provide an optimal control input.

7.2.2 Sensitivity of the closed-loop system

Analyzing an output-feedback controlled system with uncertainty and measurement noise is a difficult task. The complexity is increased when the impact of the learning process on the performance should be investigated because MPC and/or MHE are substituted by their neural network approximations. Because the evolution of the states can only be affected via the control inputs, the influence of the approximation errors of the learning-based methods on the applied control input is crucial. The approximation errors for the neural network controller and the neural network estimator for a sample at time k are defined as:

$$\text{NNC :} \qquad \delta_{\mathsf{k}}^{\mathsf{C}} = ||\mu_{\mathsf{ms}}(x_k) - \mu_{\mathsf{C}}(x_k)||_2^2\,, \tag{7.6}$$

$$\text{NNE :} \qquad \delta_{\mathsf{k}}^{\mathsf{E}} = ||x_k - \mu_{\mathsf{E}}(s_k)||_2^2\,. \tag{7.7}$$

Both errors lead to sub-optimal performance, either directly due to near-optimal control inputs or indirectly through imperfect state estimates on which the control input computation is based. The impact that the approximation error of the approximate controller NNC has on the state trajectory can be analyzed based on the system equation (7.1) by computing the following sensitivity:

$$J_{u,k} = \left.\frac{\partial f(x,u,d)}{\partial u}\right|_{x_k,\mu_{\mathsf{ms}}(x_k),d_k}. \tag{7.8}$$

The resulting changes in the successor state with respect to deviations to the optimal input can then be approximated, using a first-order Taylor approximation, by:

$$x^*_{k+1} - x^{\mathsf{C}}_{k+1} \approx J_{u,k}(\mu_{\mathsf{ms}}(x_k) - \mu_{\mathsf{C}}(x_k)), \tag{7.9}$$

where the successor states are defined based on (7.1) as:

$$x^*_{k+1} = f(x_k, \mu_{\mathsf{ms}}(x_k), d_k), \tag{7.10}$$

when the input is derived from the exact multi-stage MPC formulation and as:

$$x^{\mathsf{C}}_{k+1} = f(x_k, \mu_{\mathsf{C}}(x_k), d_k), \tag{7.11}$$

when the approximate input obtained from the neural network controller is applied.

Quantifying the effects of the estimation quality on deviations from the optimal control input cannot be handled as straightforwardly. Because the state estimate $x_{\mathsf{est},k}$ is used in the robust MPC scheme as the initial state, it would be necessary to compute the sensitivity of the optimal control problem with respect to the estimated state. To circumvent this process, we exploit that the neural network controller is an explicit function which approximates the solution of the optimal control problem. When assuming state-feedback, the sensitivity of the successor state with respect to the current estimated state can be approximated based on (7.11) as:

$$J_{x,k} = \left.\frac{\partial f(x_k, \mu_{\mathsf{C}}(x_k), d_k)}{\partial x}\right|_{x_k, d_k}. \tag{7.12}$$

The local first-order approximation of the deviation from the optimal successor state due to estimation errors can then be written as:

$$x^*_{k+1} - x^{\mathsf{E}}_{k+1} \approx J_{x,k}(x_k - \mu_{\mathsf{E}}(s_k)), \tag{7.13}$$

where the sub-optimal successor state is derived from multi-stage MPC using the estimate obtained from the neural network estimator:

$$\tilde{x}_{k+1} = f(x_k, \mu_{\mathsf{ms}}(\mu_{\mathsf{E}}(s_k)), d_k). \tag{7.14}$$

The deviations ((7.9) and (7.13))from the optimal successor state due to the approximation error in the learned controller (7.6) and learned estimator (7.7) can then be locally upper-bounded based on the maximum singular value, i.e. the spectral norm $\xi_{\mathsf{max}}(\cdot)$, of the sensitivities ((7.8) and (7.12)) as:

$$||x^*_{k+1} - x^{\mathsf{C}}_{k+1}||_2 \leq \xi_{\mathsf{max}}(J_{u,k}) \cdot \delta^{\mathsf{C}}_k, \tag{7.15}$$
$$||x^*_{k+1} - x^{\mathsf{E}}_{k+1}||_2 \leq \xi_{\mathsf{max}}(J_{x,k}) \cdot \delta^{\mathsf{E}}_k. \tag{7.16}$$

The main idea of this analysis is to illustrate that for the same approximation error of the control (δ^{C}_k) and estimation problems (δ^{E}_k), the effect on the closed-loop trajectories can be significantly different depending on the described sensitivities. This effect can be clearly observed in the results presented in Section 7.4.

If the maximum sensitivities and the maximum approximation errors out of all samples are considered, a global estimate for the maximum deviation in the successor states can be computed. Probabilistic validation methods like those presented in [130] can be applied to achieve rigorous probabilistic guarantees of the approximation errors.

7.3 Industrial polymerization reactor

For the analysis of the proposed approach we consider an industrial batch polymerization reactor, which was first presented in [19]. The system can be described by the following differential equations:

$$\begin{aligned}
\dot{m}_{\mathsf{W}} =&\ \dot{m}_{\mathsf{F}}\, \omega_{\mathsf{W,F}}\\
\dot{m}_{\mathsf{A}} =&\ \dot{m}_{\mathsf{F}}\omega_{\mathsf{A,F}} - k_{\mathsf{R1}}\, m_{\mathsf{A,R}} - k_{\mathsf{R2}}\, m_{\mathsf{AWT}}\, m_{\mathsf{A}}/m_{\mathsf{ges}},\\
\dot{m}_{\mathsf{P}} =&\ k_{\mathsf{R1}}\, m_{\mathsf{A,R}} + p_1\, k_{\mathsf{R2}}\, m_{\mathsf{AWT}}\, m_{\mathsf{A}}/m_{\mathsf{ges}},\\
\dot{m}_{\mathsf{A}}^{\mathsf{acc}} =&\ \dot{m}_{\mathsf{F}},\\
\dot{T}_{\mathsf{R}} =&\ 1/(c_{\mathsf{p,R}} m_{\mathsf{ges}})\ [\dot{m}_{\mathsf{F}}\ c_{\mathsf{p,F}}\,(T_{\mathsf{F}} - T_{\mathsf{R}}) + \Delta H_{\mathsf{R}} k_{\mathsf{R1}} m_{\mathsf{A,R}}\\
&\quad - k_{\mathsf{K}} A\,(T_{\mathsf{R}} - T_{\mathsf{S}}) - \dot{m}_{\mathsf{AWT}}\, c_{\mathsf{p,R}}\,(T_{\mathsf{R}} - T_{\mathsf{EK}})],\\
\dot{T}_{S} =&\ 1/(c_{\mathsf{p,S}} m_{\mathsf{S}})\ [k_{\mathsf{K}} A\,(T_{\mathsf{R}} - T_{\mathsf{S}}) - k_{\mathsf{K}} A\,(T_{\mathsf{S}} - T_{\mathsf{M}})],\\
\dot{T}_{\mathsf{M}} =&\ 1/(c_{\mathsf{p,W}} m_{\mathsf{M,KW}})\ [\dot{m}_{\mathsf{M,KW}}\, c_{\mathsf{p,W}} \left(T_{\mathsf{M}}^{\mathsf{IN}} - T_{\mathsf{M}}\right)\\
&\quad + k_{\mathsf{K}} A\,(T_{\mathsf{S}} - T_{\mathsf{M}})],\\
\dot{T}_{\mathsf{EK}} =&\ 1/(c_{\mathsf{p,R}} m_{\mathsf{AWT}})\ [\dot{m}_{\mathsf{AWT}} c_{\mathsf{p,W}}\,(T_{\mathsf{R}} - T_{\mathsf{EK}})\\
&\quad - \alpha\,(T_{\mathsf{EK}} - T_{\mathsf{AWT}}) + k_{\mathsf{R2}}\, m_{\mathsf{A}}\, m_{\mathsf{AWT}} \Delta H_{\mathsf{R}}/m_{\mathsf{ges}}],\\
\dot{T}_{\mathsf{AWT}} =&\ [\dot{m}_{\mathsf{AWT,KW}}\, c_{\mathsf{p,W}}\,(T_{\mathsf{AWT}}^{\mathsf{IN}} - T_{\mathsf{AWT}})\\
&\quad - \alpha\,(T_{\mathsf{AWT}} - T_{\mathsf{EK}})]/(c_{\mathsf{p,W}} m_{\mathsf{AWT,KW}}),\\
\dot{T}_{\mathsf{adiab}} =&\ \frac{\Delta H_{\mathsf{R}}}{m_{\mathsf{ges}} c_{\mathsf{p,R}}}\dot{m}_{\mathsf{A}} - (\dot{m}_{\mathsf{W}} + \dot{m}_{\mathsf{A}} + \dot{m}_{\mathsf{P}}) \left(\frac{m_{\mathsf{A}} \Delta H_{\mathsf{R}}}{m_{\mathsf{ges}}^2 c_{\mathsf{p,R}}}\right) + \dot{T}_{\mathsf{R}}\\
&\text{where:}\\
U =&\ m_{\mathsf{P}}/(m_{\mathsf{A}} + m_{\mathsf{P}}),\\
m_{\mathsf{ges}} =&\ m_{\mathsf{W}} + m_{\mathsf{A}} + m_{\mathsf{P}},\\
k_{\mathsf{R1}} =&\ k_0 e^{\frac{-E_a}{R(T_{\mathsf{R}}+273.15)}} \left(k_{\mathsf{U1}}\,(1-U) + k_{\mathsf{U2}} U\right),\\
k_{\mathsf{R2}} =&\ k_0 e^{\frac{-E_a}{R(T_{\mathsf{EK}}+273.15)}} \left(k_{\mathsf{U1}}\,(1-U) + k_{\mathsf{U2}} U\right),\\
k_{\mathsf{K}} =&\ (m_{\mathsf{W}}\, k_{\mathsf{WS}} + m_{\mathsf{A}}\, k_{\mathsf{AS}} + m_{\mathsf{P}}\, k_{\mathsf{PS}})/m_{\mathsf{ges}},\\
m_{\mathsf{A,R}} =&\ m_{\mathsf{A}} - m_{\mathsf{A}} m_{\mathsf{AWT}}/m_{\mathsf{ges}},
\end{aligned} \tag{7.17}$$

which are discretized via orthogonal collocation on finite elements [131].

The states describing the system are mass balances and product hold-ups for water m_{W}, monomer m_{A} and polymer m_{P} in the reactor and the total mass of monomer that has been fed to the reactor $m_{\mathsf{A}}^{\mathsf{acc}}$. Further states include the temperatures of the reactor T_{R}, the vessel T_{S}, the jacket T_{M}, the fluid mix in the external heat exchanger

T_{EK} and the coolant flowing out of the external heat exchanger T_{AWT}, which are used to describe the energy balances of the system. The details including all the parameter values can be found in [19] and the code is also publicly available[1]. The available control inputs are the feed flow $\dot{m}_{\text{F}}$, the coolant temperature at the inlet of the jacket T_{M}^{IN} and the coolant temperature at the inlet of the external heat exchanger $T_{\text{AWT}}^{\text{IN}}$. An auxiliary variable which plays a major role for the safe control of the reactor is T_{adiab}, which describes the temperature that would be reached if the cooling fails. As an important safety constraint, T_{adiab} always needs to be lower than a certain threshold.

In each batch, a fixed amount of polymer needs to be produced as fast possible, which coincides with minimizing the batch time. At the same time, the constraints for the safe operation of the reactor ($T_{\text{adiab}} \leq T_{\text{adiab,max}}$), a high-quality product ($T_{\text{R,min}} \leq T_{\text{R}} \leq T_{\text{R,max}}$) and the physical limitations of the manipulators need to be satisfied. The two uncertain parameters are k_{U1} and k_{U1} and considered constant throughout one batch.

7.4 Learning-based output-feedback control for a semi-batch polymerization reactor

This section highlights how the learning-based approximations were obtained and presents the resulting performance for each one of the closed-loop configurations depicted in Fig. 7.1. A common and reasonable concern for the use of machine learning in a closed-loop is related to the unexpected performance that can be obtained when applied to situations that have not been *seen* previously during training. To analyze the robustness of both, the optimization-based and the learning-based approaches, we present the results for the case when the intervals of parametric uncertainties and possible initial conditions are larger than those used for training, or assumed in the optimization-based formulations. We also show that the impact of the approximation quality can be analyzed using the sensitivity considerations described in Section 7.2.

7.4.1 Data generation

For the generation of the training data we ran 200 simulations of the batch process with MHE in connection with MPC, which corresponds to the second configuration in Fig. 7.1. For the multi-stage NMPC scheme we used a prediction horizon of $N = 20$ and a robust horizon of $N_{\text{r}} = 1$ with a sampling time of $t_{\text{s}} = 50\,\text{s}$. The MHE implementation used the same sampling time as the controller and the horizon was chosen to $N_{\text{est}} = 10$ which was sufficient for the proper estimation of the uncertain parameters k_{U1} and k_{U2} and subsequently provided good state estimates. For each simulation of a batch, the initial state, the estimated initial state and the value of the uncertain parameters were varied according to the values given in the *Training intervals* column of Table 7.1 following a uniform distribution. We used do-mpc [132] to generate the simulation results, as it supports multi-stage NMPC and MHE natively. The simulation was carried out via the integrators provided by the SUNDIALS toolbox [133] ensuring a high accuracy.

[1] https://www.do-mpc.com/en/latest/example_gallery/industrial_poly.html

Table 7.1: Intervals of initial states and uncertainties as considered in the training process and when the different methods are exposed to extended scenarios. The deviation of the initial estimated state from real state is $e_{\text{init}} = x_{\text{init}} - x_{\text{est}}$ The measurement noise is zero-mean gaussian and fully described by the standard deviation. States without measurement noise cannot be measured.

Param.	**Unit**	**Training intervals**		**Extended intervals**		**Meas. noise**
		x_{init}	e_{init}	x_{init}	e_{init}	
m_{W}	kg	$[9900.0, 10100.0]$	$[10.0, 10.0]$	$[9800.0, 10200.0]$	$[15.0, 15.0]$	1.0
m_{A}	kg	$[851.0, 855.0]$	$[2.0, 2.0]$	$[848.0, 858.0]$	$[4.0, 4.0]$	-
m_{P}	kg	$[26.0, 27.0]$	$[0.5, 0.5]$	$[22.0, 31.0]$	$[0.5, 0.5]$	-
T_{R}	K	$[362.15, 364.15]$	$[0.1, 0.1]$	$[361.65, 364.65]$	$[0.1, 0.1]$	0.1
T_{S}	K	$[362.15, 364.15]$	$[0.1, 0.1]$	$[361.15, 365.15]$	$[0.2, 0.2]$	0.1
T_{M}	K	$[362.15, 364.15]$	$[0.1, 0.1]$	$[361.15, 365.15]$	$[0.2, 0.2]$	0.1
T_{EK}	K	$[306.15, 310.15]$	$[0.1, 0.1]$	$[304.15, 312.15]$	$[0.2, 0.2]$	0.1
T_{AWT}	K	$[306.15, 310.15]$	$[0.1, 0.1]$	$[304.15, 312.15]$	$[0.2, 0.2]$	0.1
m_{monom}	kg	$[290.0, 210.0]$	$[2.0, 2.0]$	$[280.0, 320.0]$	$[4.0, 4.0]$	1.0
k_{U2}	-	$[25.6, 38.4]$		$[22.4, 41.6]$		-
k_{U1}	-	$[3.2, 4.8]$		$[2.8, 5.2]$		-

7.4.2 Training of neural networks

The 200 generated trajectories contain 23299 data tuples of which 90% were used for training the different neural network approaches and 10% were reserved for computing the validation error. While for learning the NNC all tuples can be considered, the number of available samples is reduced to 21299 for NNE and NNEC. Since in every batch the first $N_{\text{est}} = 10$ tuples are used to initialize the input to the networks, there are $200 \cdot 10 = 2000$ less data samples. The data was scaled such that the input and output samples for the networks belonged to the unit hypercube. The training of the neural networks was carried out via Keras [89] relying on the Tensorflow [90] backend using Adam [92] as the optimization algorithm.

The validation error throughout the training progress averaged over 20 training runs, computed via (7.5), is visualized in Fig. 7.2. It can be seen that the validation error decreases steadily for NNC while the curves for NNE and NNEC are characterized by various peaks. The minor peaks are due to the stochastic nature of the optimization algorithm and can also be seen for NNC. The major peaks show that small changes in the network parameters might lead to a big change in the approximation quality, which can be seen as an indicator of the fact that that learning an estimator is more difficult problem than learning a controller. To take into account the complexity of the learning task, the size of the network chosen for the controller was the smallest and

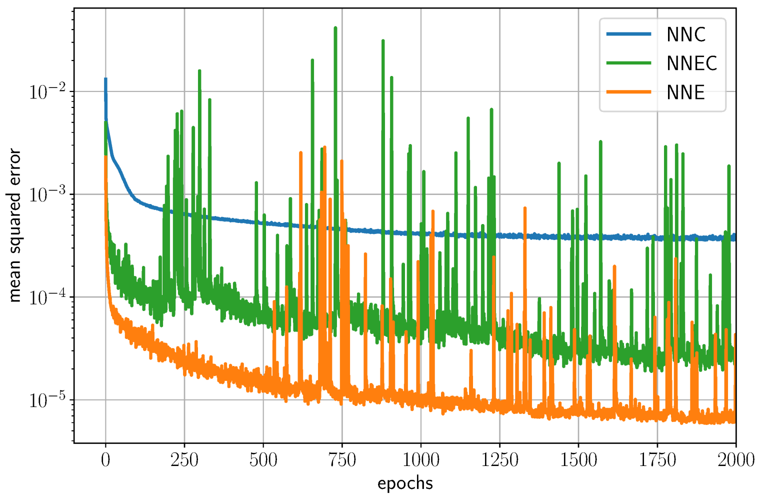


Figure 7.2: Average validation mean squared loss of 20 training runs over 2000 epochs for the learned controller, estimator and the simultaneous approach. The samples were scaled such that the inputs and outputs belong to the unit hypercube with respective dimensionality. Note that the mean squared errors cannot be directly compared as different functions are being approximated.

Table 7.2: Architectural details of the networks including number of inputs n_0, neurons per hidden layer $n_l = n$, $l \in \mathbb{N}^+_{[L]}$, number of hidden layers L and number of outputs n_{L+1}.

	symbol	n_0	n	L	n_{L+1}
NNC	μ_{C}	9	25	6	3
NNE	μ_{E}	109	80	6	9
NNEC	μ_{EC}	109	120	8	12

for the simultaneous approach the largest, because it incorporates the estimator and the controller. The architectural details of the three networks are listed in Table 7.2. For the subsequent investigations the network parameters providing the lowest validation error throughout the 2000 training epochs were selected.

7.4.3 Optimization-based vs. learning-based approaches

The six configurations summarized in Fig. 7.1 were evaluated for 100 randomly generated settings with the training intervals. Each setting is defined by the real initial value of the system x_{init}, the corresponding initial guess x_{est} used for the estimator and the controller, and the realization of the uncertain parameters. Note that the parameters remain uncertain but constant for the whole batch irrespective of the controller used in the closed-loop. The possible variations of all these parameters, the measurement

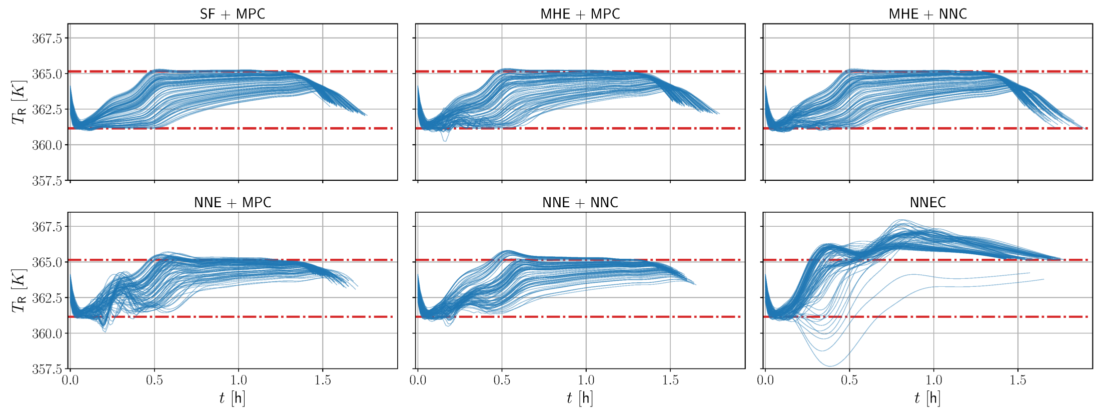


Figure 7.3: Reactor temperature for 100 scenarios randomly generated with the training intervals from Table 7.1 for all six considered configurations (Fig. 7.1).

noise and the initial estimation error $e_{\text{init}} = x_{\text{init}} - x_{\text{est}}$ considered in this subsection are given in the *Training intervals* section of Table 7.1.

The quantitative results for the 100 tests that use the same intervals (but not the same realizations within the intervals) as those used for training are summarized in Table 7.3. Due to the economic objective, it is optimal to operate the reactor at the bounds of T_{R}. This leads to constraint violations even for the baseline, where MPC is used in connection with state-feedback as it can be seen in the first row of Table 7.3. These minor violations are caused by discretization errors. The 100 evaluated trajectories for the considered configurations are presented in Fig. 7.3. All configurations provide a comparable performance with exception of NNEC. The reasons for the inferior performance of NNEC will be explained in detail in the following sections.

7.4.4 Robustness of deep-learning-based approximations

From the previous results it is clear that the learned estimators and controllers provide a solid performance within the domain they were trained for. But during operation, cases might occur where the system leaves the domain which was covered by the training procedure. To analyze the performance for unseen scenarios, we analyzed the behavior of the different configurations, when both uncertainties vary between $\pm 30\%$ instead of $\pm 20\%$. Additionally, the interval in which the initial values of the system may lie and the deviation of the estimated initial state from the real state is enlarged too. For the exact numeral values please consider the columns for the *Extended intervals* in Table 7.1.

In the same manner as for the investigation with the training intervals, we tested each closed-loop configuration for 100 randomly generated scenarios with the extended intervals. For the MHE formulation, we adapted the bounds of the parameters (7.2e) to match the $\pm 30\%$ uncertainty, while the considered uncertainty in the scenario tree of the MPC was kept unchanged. To avoid infeasible optimization problems when solving the exact multi-stage MPC problem (2.8) due to the enlarged uncertainty intervals that are not considered in the scenario tree, the critical constraints were implemented as soft constraints.

Table 7.3: Performance comparison of the five considered estimation and control configurations presented in Fig. 7.1. All configurations were simulated for the same 100 randomly generated uncertainty realizations using the training intervals given in Table 7.1.

Configuration	**Batch time**			**Violation T_R**		**Violation T_{adiab}**	
	Avg.	Min.	Max.	Avg.	Max.	Avg.	Max.
SF + MPC	1.6065	1.5000	1.7639	0.0020	0.1712	0.0000	0.0000
MHE + MPC	1.6157	1.5000	1.7917	0.0032	0.9191	0.0013	0.1751
MHE + NNC	1.6518	1.4861	1.9028	0.0030	0.3013	0.0001	0.1606
NNE + MPC	1.6385	1.4444	1.7083	0.0111	1.0810	0.0071	1.0261
NNE + NNC	1.5243	1.4306	1.6528	0.0185	0.6795	0.0013	0.0001
NNEC	1.5635	1.4167	1.7778	0.6378	3.4836	0.5810	0.0497

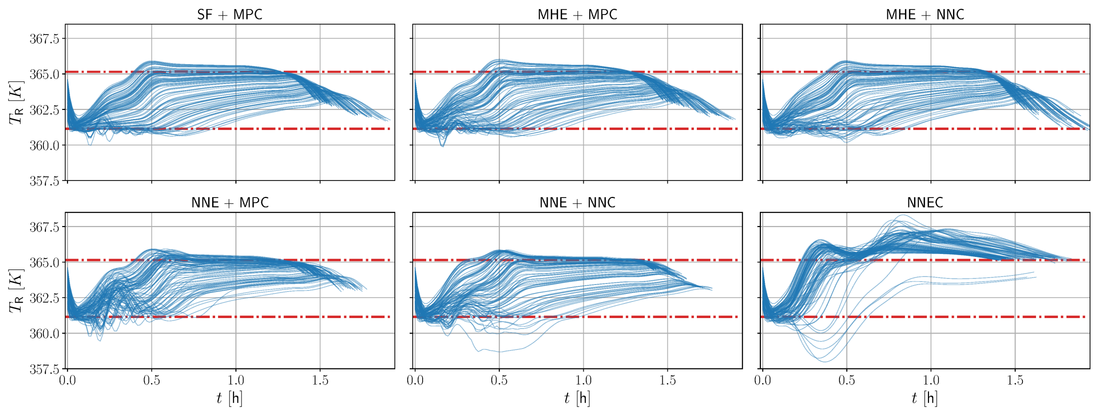


Figure 7.4: Reactor temperature for 100 scenarios randomly generated with intervals larger than in the training process (see extended intervals in Table 7.1) for all six considered configurations (Fig. 7.1).

The results are summarized in Table 7.4 and the 100 trajectories of the reactor temperature for all six configurations are shown in Fig. 7.4. The results confirm that, as expected, the performance for all approaches degrades due to the enlarged uncertainty and initial condition intervals. This can be clearly seen in larger constraint violations. However, the degradation of all six configurations is of the same order and only the NNEC approach has significant average constraint violations for the reactor temperature, as it can be seen in Fig. 7.4. What we would like to point out is that the learning-based approaches, when exposed to unseen scenarios, do not *necessarily* have a worse (average or worst-case) performance in comparison to the optimization-based approaches MHE and MPC when wrong assumptions are made.

Table 7.4: Performance comparison of the five considered estimation and control configurations presented in Fig. 7.1. All configurations were simulated for the same 100 randomly generated uncertainty realizations using the extended intervals given in Table 7.1.

Configuration	**Batch time**			**Violation T$_R$**		**Violation T$_{adiab}$**	
	Avg.	Min.	Max.	Avg.	Max.	Avg.	Max.
SF + MPC	1.6199	1.4583	1.9167	0.0357	1.1692	0.0001	0.0124
MHE + MPC	1.6282	1.4583	1.9028	0.0401	1.2872	0.0015	0.2865
MHE + NNC	1.6722	1.4583	2.0417	0.0418	0.9633	0.0003	0.2078
NNE + MPC	1.5478	1.4028	1.7778	0.0371	1.7051	0.0202	2.8845
NNE + NNC	1.5329	1.4028	1.7639	0.0582	2.4689	0.0003	0.2281
NNEC	1.5565	1.3889	1.8333	0.6279	3.1819	0.0001	0.1138

7.4.5 Performance deterioration due to learning-based approximations

In the previous sections it was shown that the learning-based methods can achieve a performance level comparable to the online optimization-based methods. This section illustrates the fact that the same approximation error of a neural network estimator or a neural network controller can have a very different influence on the closed-loop performance.

To analyze this effect, we make use of the sensitivities described in Subsection 7.2.2 and compute them for $n_{\text{traj}} = 100$ trajectories generated via state-feedback control using the exact multi-stage MPC. For every step $k \in \mathbb{N}^+_{[n_{\text{steps},i}]}$ (except for the first N_{est}) along the i-th trajectory, $i \in \mathbb{N}^+_{[n_{\text{traj}}]}$, the tuple $(s_{i,k}, x_{i,k}, \mu_{\text{ms}}(x_{i,k}), d_{i,k})$ is available, where $x_{i,k}$ is the current state of the system, $d_{i,k}$ is the parameter realization and $s_{i,k}$ carries the information required for the estimator.

The local sensitivities with respect to errors in the computed input and errors in the estimated state can be directly computed via (7.8) and (7.12). As shown in (7.15) and (7.16), the maximum singular values of the sensitivities can be used to obtain an approximate upper bound of the successor state error. The maximum singular value occurring throughout all n_{traj} considered trajectories can be computed via:

$$\xi_{\text{max},x} = \underset{i\in\mathbb{N}^+_{[n_{\text{traj}}]},k\in\mathbb{N}^+_{[n_{\text{steps},i}]}}{\text{maximize}} \quad (\xi_{\text{max}}(J_{x,i,k}))\,, \tag{7.18a}$$

$$\xi_{\text{max},u} = \underset{i\in\mathbb{N}^+_{[n_{\text{traj}}]},k\in\mathbb{N}^+_{[n_{\text{steps},i}]}}{\text{maximize}} \quad (\xi_{\text{max}}(J_{u,i,k}))\,, \tag{7.18b}$$

where, with slight abuse of notation, $J_{x,i,k}$ and $J_{u,i,k}$ denote the sensitivities at the k-th time step of the i-th trajectory. The average singular value along a trajectory can

Table 7.5: The maximum and average approximation errors are computed based on (7.7), (7.6) and (7.20). The maximum and average singular values are computed via (7.18) and (7.19). The sub-optimality of the resulting control inputs for the three considered configurations are given in (7.21).

	NNE + MPC	SF + NNC	NNEC
δ_{max}	1873.09	7054.83	15935.78
δ_{av}	450.29	185.36	7500.17
ξ_{max}	51887.967	100.00	-
ξ_{av}	1075.38	53.75	-
e_u	1300.34	185.36	4116.96

provide further insights on the importance of the approximation errors, and can be computed as:

$$\xi_{\text{av},x} = \frac{1}{n_{\text{steps}}} \sum_{i=1}^{n_{\text{traj}}} \sum_{k=1}^{n_{\text{steps},i}} \xi_{\text{av}}(J_{x,i,k}), \tag{7.19a}$$

$$\xi_{\text{av},u} = \frac{1}{n_{\text{steps}}} \sum_{i=1}^{n_{\text{traj}}} \sum_{k=1}^{n_{\text{steps},i}} \xi_{\text{av}}(J_{u,i,k}), \tag{7.19b}$$

where $n_{\text{steps}} = \sum_{i=1}^{n_{\text{traj}}} n_{\text{steps},i}$. Table 7.5 shows that the average singular values of the sensitivity with respect to the state estimate $\xi_{\text{av},x}$ are higher than for the optimal input $\xi_{\text{av},u}$. This means that a given error in the approximation of NNE leads in general to larger deviations from the optimal trajectory than for NNC.

The approximation errors for NNC and NNE are computed via (7.6) and (7.7). When NNEC is considered, which has the largest network structure since it needs to learn both the estimator and the controller, the approximation error for the k-th sample is given as:

$$\delta_k^{\text{EC}} = \left\| \begin{bmatrix} x_k \\ u_k^* \end{bmatrix} - \mu_{\text{EC}}(s_k) \right\|_2^2. \tag{7.20}$$

The maximum and average approximation errors for each network $\delta_{i,j} \in \{\delta_{i,j}^{\text{C}}, \delta_{i,j}^{\text{E}}, \delta_{i,j}^{\text{EC}}\}$ can be analyzed in an equivalent fashion as the singular values. By replacing the maximum singular values $\xi_{\text{max}}(J_{(\cdot),i,j})$ in (7.18) and average singular values $\xi_{\text{av}}(J_{(\cdot),i,j})$ in (7.19) with $\delta_{i,j}$, the maximum approximation errors ($\delta_{\text{max}} \in \{\delta_{\text{max}}^{\text{C}}, \delta_{\text{max}}^{\text{E}}, \delta_{\text{max}}^{\text{EC}}\}$) and average approximation errors ($\delta_{\text{av}} \in \{\delta_{\text{av}}^{\text{C}}, \delta_{\text{av}}^{\text{E}}, \delta_{\text{av}}^{\text{EC}}\}$) are obtained. The values for the resulting approximated errors are summarized in the upper half of Table 7.5. It can be seen that approximation error of NNEC exceeds both the largest maximum and average approximation error of NNE and NNC. The maximum and the average approximation errors for NNC and NNE are of the same magnitude, but the significantly higher average sensitivity with respect to the state estimate means that replacing MHE with NNE will lead to a larger performance deterioration than replacing MPC with NNC.

To directly quantify the effect of the approximation error in the closed-loop, we can also compare the optimal input $u^*_{i,k} = \mu_{\mathsf{ms}}(x_{i,k})$ to the sub-optimal inputs provided when:

- applying NNC resulting in $u^{\mathsf{C}}_{i,k} = \mu_{\mathsf{C}}(x_{i,k})$;
- the optimal input is computed by solving (2.8) based on the estimate $\mu_{\mathsf{E}}(s_{i,k})$ resulting in $u^{\mathsf{E}}_{i,k} = \mu_{\mathsf{ms}}(\mu_{\mathsf{E}}(s_{i,k}))$;
- applying NNEC extracting the optimal control input $u^{\mathsf{EC}}_{i,k}$ from $\mu_{\mathsf{EC}}(s_{i,k}) = [x^T_{\mathsf{est},i,k}, u^{\mathsf{EC},T}_{i,k}]^T$.

The average root mean squared error of the euclidean distance of the approximate control inputs from the optimal control inputs along all trajectories, denoted as e_u, is then defined by:

$$\text{SF + NNC:} \qquad \frac{1}{n_{\mathsf{steps}}} \sum_{i=1}^{n_{\mathsf{traj}}} \sum_{j=1}^{n_{\mathsf{steps},i}} ||u^{\mathsf{C}}_{i,k} - u^*_{i,k}||_2, \tag{7.21a}$$

$$\text{NNE + MPC:} \qquad \frac{1}{n_{\mathsf{steps}}} \sum_{i=1}^{n_{\mathsf{traj}}} \sum_{j=1}^{n_{\mathsf{steps},i}} ||u^{\mathsf{E}}_{i,k} - u^*_{i,k}||_2, \tag{7.21b}$$

$$\text{NNEC:} \qquad \frac{1}{n_{\mathsf{steps}}} \sum_{i=1}^{n_{\mathsf{traj}}} \sum_{j=1}^{n_{\mathsf{steps},i}} ||u^{\mathsf{EC}}_{i,k} - u^*_{i,k}||_2. \tag{7.21c}$$

The values for the 100 trajectories are summarized in the last row of Table 7.5. The difference between the optimal input and the control input provided by NNEC is the largest which is consistent with its inferior performance observed in Fig. 7.3. The deviation caused when the learning-based estimator is used is significantly larger than the deviation caused by the learning-based controller, which confirms the importance of the larger sensitivities despite similar average and maximum approximation errors (δ_{max}, δ_{av}). This is also consistent with the better performance (smaller constraint violations) observed by the MHE + NNC configuration when compared to the NNE + MPC configuration. Nevertheless, the configuration consisting of two learning-based elements (NNE + NNC) shows comparable performance to the optimization-based configuration (MHE + MPC).

The three main messages of this chapter can be summarized as: i) both the control problem (MPC) as well as the estimation problem (MHE) can be properly approximated using deep neural networks even for non-trivial case studies; ii) the performance degradation obtained when exposed to previously unseen scenarios does not have to be stronger than the degradation observed when solving the MPC and MHE problems exactly under the same wrong assumptions about the possible scenarios; iii) the training error is not a good indicator for the closed-loop performance of an approximate MHE or MPC as it is strongly affected by the sensitivities of the problem as well as by the scaling factors used during training.

In the next chapter, we introduce an approach that further refines the behavior of the purely imitation learning-based approximate controllers. By lending ideas from the field of reinforcement learning, the parameters of a neural network controller can be improved, or adapted when the scenario changes, considering closed-loop trajectories.

Chapter 8

Reinforced approximate robust nonlinear model predictive control

In general, two deep learning paradigms can be distinguished. The first approach, called imitation learning, has been used successfully throughout this thesis, but suffers from two major drawbacks. First, it is difficult to learn behavior that is not well represented in the data, as will be shown in the following. Second, new data sets need to be generated whenever the control task changes, whereas a classical NMPC formulation can be adapted in a straightforward manner.

In the field of reinforcement learning, the second deep learning paradigm, the control policy can be updated during operation as closed-loop information is included in the learning process and adaptations to changes of the control tasks can be made online. However, a very large number of data samples is typically necessary and the consideration of critical constraints is not directly possible. By taking additional measures like projecting onto the feasible space [134–138], constraint satisfaction can be achieved. Another possibility to consider constraints and reduce the required number of data samples for a controller obtained via reinforcement learning is to consider NMPC as the parametric policy within the reinforcement learning framework. The sensitivity of the underlying optimization problems can be used to tune the parameters in the NMPC formulation [139–141], but the computational load is high as each control step requires solving an NMPC problem.

We propose to combine notions from imitation learning and reinforcement learning to counteract these drawbacks. The major advantages of the proposed approach are: i) Controllers computed via imitation learning can be updated to changes in the control task without having to learn a new controller, making it unnecessary to generate new data sets by repeatedly solving complex optimization problems; ii) The resulting learning-based controller can even outperform an exact NMPC approach because the policy parameters are updated based on long simulations with a high fidelity model. Thus, better information about the desired closed-loop performance is introduced that cannot be easily incorporated in an NMPC formulation with a limited prediction horizon. We revisit the industrial-scale polymerization reactor introduced in Chapter 7 to highlight the efficacy of the approach.

The remainder of the chapter is organized as follows. In Section 8.1 the general settings for the proposed method are introduced. The basic principles of reinforcement learning are presented in Section 8.2. The main contribution - combining ideas from imitation learning and reinforcement learning - is presented in Section 8.3. In Section 8.4 the polymerization reactor is recapitulated for which the proposed method is extensively studied in Section 8.5. The results of this chapter are largely based on the findings in [9].

8.1 Problem statement

In contrast to the previous chapter, where an output-feedback control task was analyzed, we consider in this chapter general discrete-time nonlinear systems for the state-feedback case:

$$x^+ = f(x, u, d), \tag{8.1}$$

where the uncertain parameters $d \in \mathbb{R}^{n_d}$ follow the probability distribution χ_d and belong to the set $\mathcal{D} \subseteq \mathbb{R}^{n_d}$. The nonlinear system (8.1) is controlled by a parametric state-feedback policy $\mu : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$, e.g. a deep neural network, defined as:

$$u = \mu(x; \theta), \tag{8.2}$$

where θ are the control policy parameters. Thus, the closed-loop behavior of the system is described by

$$\begin{aligned} x^+ &= f(x, \mu(x;\theta), d), \\ &= f_{\text{cl}}(x, d; \theta). \end{aligned} \tag{8.3}$$

For a given initial state $x_0 \in \mathcal{X}_{\text{init}} \subseteq \mathbb{R}^{n_x}$, that is drawn from the probability distribution χ_{init}, and a sequence of length N_{sim} of realizations of the uncertainties $\mathbf{d} = \{d_0, \ldots, d_{N_{\text{sim}}-1}\}$, $d_i \sim \chi_d \ \forall i \in \mathbb{N}_{[N_{\text{sim}}-1]}$, the state trajectory $\mathbf{x} = \{x_0, \ldots, x_{N_{\text{sim}}}\}$ and control input trajectory $\mathbf{u} = \{u_0, \ldots, u_{N_{\text{sim}}-1}\}$ are uniquely defined by (8.3). We therefore summarize the initial state and realization of uncertain parameters in a single variable:

$$w = \{x_{\text{init}}, d_0, \ldots, d_{N_{\text{sim}}-1}\}, \tag{8.4}$$

that is drawn from the probability distribution:

$$w \sim \chi_w = \chi_{\text{init}} \times \chi_d \times \cdots \times \chi_d,$$

and whose realizations belong to the set:

$$w \in \mathcal{W} = \mathcal{X}_{\text{init}} \times \mathcal{D} \times \cdots \times \mathcal{D}.$$

Further, we define a function that generates the resulting closed-loop trajectory for a realization $w \sim \chi_w$, a given system model $f(\cdot)$ and feedback controller $\mu(\cdot)$, and the number of simulations steps N_{sim} as:

$$\Xi(w; N_{\text{sim}}, f(\cdot), \mu(\cdot)) = \{x_0, \mu(x_0;\theta), d_0, x_1, \mu(x_1;\theta), d_1, \ldots, x_{N_{\text{sim}}}\}. \tag{8.5}$$

The controller design task is to find policy parameters θ such that the control inputs optimize the closed-loop performance index:

$$\phi(w; N_{\text{sim}}, f(\cdot), \mu(\cdot;\theta)) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_d} \times \cdots \times \mathbb{R}^{n_d} \to \mathbb{R}, \tag{8.6}$$

which is defined as an arbitrary function over a finite time window N_{sim} that returns a scalar based on the state and control input trajectories uniquely defined by (8.5).

To specify the optimal behavior of the parametric policy, we formulate an optimization problem that follows a similar structure as for the ideal robust policy (2.5). The optimal control parameters $\hat{\theta}$ are then found by solving:

$$\begin{aligned}
\underset{\theta}{\text{arg min}} \quad & \mathbb{E}\left[\phi(w; N_{\text{sim}}, f(\cdot), \mu(\cdot;\theta))\right] && \text{(8.7a)}\\
\text{subject to} \quad & \mathcal{X}_0 = \mathcal{X}_{\text{init}}, && \text{(8.7b)}\\
& \text{for all } k \in \mathbb{N}_{[N_{\text{sim}}-1]} : && \\
& \qquad \mathcal{X}_{k+1} = \mathcal{R}_{\text{cl}}\left(\mathcal{X}_k, f(\cdot), \mu(\cdot;\theta)\right), && \text{(8.7c)}\\
& \qquad 0 \geq g_{\text{cl}}\left(\mathcal{X}_k, \mu(\cdot;\theta)\right), && \text{(8.7d)}\\
& 0 \geq g_{\text{f}}\left(\mathcal{X}_N\right), && \text{(8.7e)}
\end{aligned}$$

where (8.7a) is the expected value of the performance index for a trajectory of length N_{sim} defined by a randomly drawn sample $w \sim \chi_w$. However, solving (8.7) is usually intractable due to the requirement to propagate reachable sets and arbitrary complex probability distributions for nonlinear systems. The difficulty is even increased when highly nonlinear neural networks are considered as the controllers.

The approach that has been followed throughout this thesis to obtain a parametric controller - which approximately behaves as specified in (8.7) - was to design a multi-stage NMPC with the desired properties which is then closely imitated by the parametric controller. But imitation learning often suffers from shortcomings due to imperfect data and the neglection of the impact of approximation errors on the closed-loop behavior. Further, imitation learning requires generating a new data set for retraining whenever the control task changes. We will use ideas from reinforcement learning to approximately solve (8.7) and to counteract the drawbacks of parametric controllers that were obtained through imitation learning.

8.2 Reinforcement learning

In reinforcement learning, the parametric controller $\mu(\cdot;\theta)$ is usually called agent and interacts with the system by applying a control input $u = \mu(x;\theta)$, called action in the reinforcement learning context, to the system to obtain a stage reward

$$r(x,u) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R} \tag{8.8}$$

after each time-step. Two types of policies are distinguished. Stochastic policies [142] describe a probability distribution in the control input space that depends on the current state. The applied control input is then drawn from this distribution. Deterministic policies, as used throughout this chapter, will always provide the same control input for a given state as the name indicates. In reinforcement learning, the system dynamics are typically described as a Markov Decision Process defined by transition probabilities:

$$T(x^+|x,u) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}_{[0,1]}, \tag{8.9}$$

from the state x to a successor state x^+ under action u. The stochasticity is induced by the probability density function χ_d from which the uncertain parameters d are drawn. In robust control, the possible realizations of the uncertain parameters are directly considered, e.g. through the consideration of reachable sets or approximations. Note that deterministic transition functions, e.g. when no uncertain parameters are present, are a special case of stochastic transition functions which can be described by:

$$T(x^+|x,u) = \begin{cases} 1 & \text{if } x^+ = f(x,u), \\ 0 & \text{else.} \end{cases}$$

Reinforcement learning tries to find the optimal parameters $\hat{\theta}$ for the control policy that maximize the expected accumulated rewards (8.8), also called return, by solving:

$$\hat{\theta} = \arg\max_{\theta} \mathbb{E}\left[\sum_{k=0}^{N_{\text{sim}}} \gamma^k r\left(x_k, \mu(x_k;\theta)\right)\right], \tag{8.10}$$

where $\gamma \in \mathbb{R}_{(0,1]}$ is a discount factor to balance between short-term and long-term rewards and N_{sim} describes the number of future steps to be considered. For continuous tasks, a large value for N_{sim} is chosen ($N_{\text{sim}} \rightarrow \infty$), whereas for finite tasks, usually called episodic tasks, N_{sim} is chosen according to the length of one episode. Solving (8.10) exactly is usually intractable due to the highly nonlinear policy and the probabilistic transition function. Hence, the problem can in general only be solved approximately, e.g. by collecting many data samples of the applied control inputs, transitions and obtained reward as described for various algorithms in [142]. Based on the gathered information, the parameters of the parametric controller are updated via gradient-based methods to improve the performance. One major drawback of reinforcement learning approaches is that the consideration of constraints is not as straight-forward as for NMPC formulations and that many episodes are required to obtain a satisfying result [143].

8.3 Reinforcing a parametric control policy

We propose to combine imitation learning and reinforcement learning to derive explicit high-performance learning-based controllers to mitigate the drawbacks of only applying either of the two learning paradigms. The method is summarized in Fig. 8.1. First, the policy parameters are randomly initialized as θ_{init} before the data set $\mathcal{T}$, generated by approximately solving (2.8) for various initial states, is used to update the policy parameters via imitation learning (3.8) providing the policy parameters θ^*. One drawback of imitation learning is its dependence on MPC schemes that rely on a tractable prediction model, a prediction horizon that possibly provides suboptimal performance and an objective function that only approximates the true performance index due to com-

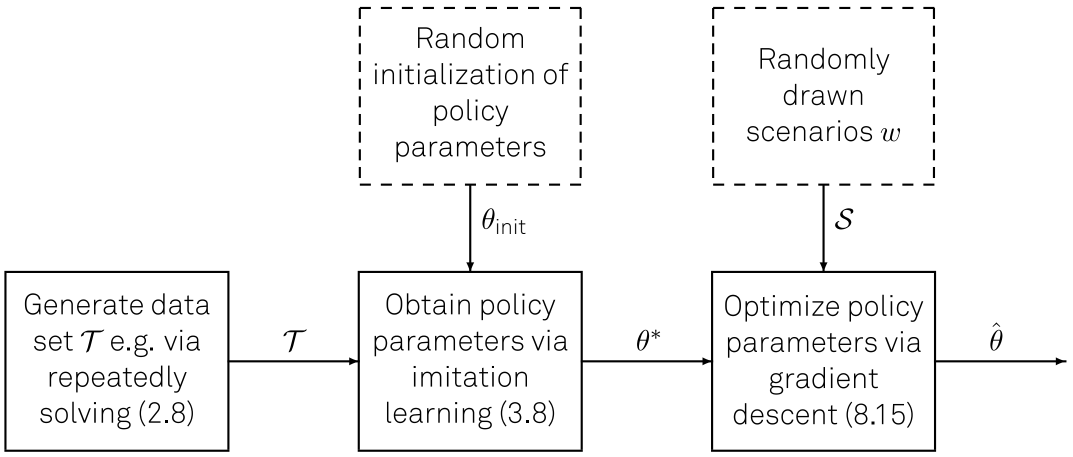

Random initialization of policy parameters
Randomly drawn scenarios w
θ_{init}
$\mathcal{S}$
Generate data set $\mathcal{T}$ e.g. via repeatedly solving (2.8)
$\mathcal{T}$
Obtain policy parameters via imitation learning (3.8)
θ^*
Optimize policy parameters via gradient descent (8.15)
$\hat{\theta}$

Figure 8.1: Proposed method to obtain high-performance learning-based controllers with the three main steps described in each solid block. ©[2021] IEEE.

putational limitations. Additionally, imitation learning does not consider the closed-loop behavior of the system. This means that during training of the imitation learning controller, an approximation error of the MPC problem that has no effect in the closed-loop behavior is considered equal to an approximation error that has a critical influence in the closed-loop performance. By considering whole trajectories of the closed-loop, these drawbacks can be mitigated.

For a parametric control policy (8.2), one can exploit that it possible to explicitly describe the closed-loop behavior:

$$\begin{aligned} x^+ &= f_{\text{hf}}(x, \mu(x;\theta), d_k), \\ &= f_{\text{hf,cl}}(x, d; \theta), \end{aligned} \tag{8.11}$$

using a high fidelity model $f_{\text{hf}}(\cdot)$ that is more accurate than the prediction model used in an MPC formulation. Since the original problem (8.7) that implicitly describes the optimal parameters for a parametric policy is intractable, we formulate a sampling-based approximation. Generating a data set

$$\mathcal{S} = \{w_1, w_2, \dots, w_{N_{\text{sc}}}\} \tag{8.12}$$

by drawing N_{sc} samples $w_i \sim \chi_w, i \in \mathbb{N}^+_{[N_{\text{sc}}]}$, the approximate formulation is given by:

$$\begin{aligned} \underset{\theta}{\text{arg min}} \quad & \sum_{j=1}^{N_{\text{sc}}} \phi(w_j; N_{\text{sim}}, f_{\text{hf}}(\cdot), \mu(\cdot;\theta)) && \text{(8.13a)} \\ \text{subject to} \quad & \text{for all } j \in \mathbb{N}^+_{[N_{\text{sc}}]} : \\ & \quad x_{0,j} = x_{\text{init},j}, && \text{(8.13b)} \\ & \quad \text{for all } k \in \mathbb{N}_{[N_{\text{sim}}-1]} : \\ & \quad\quad x_{k+1,j} = f_{\text{hf,cl}}\left(x_{k,j}, d_{k,j}; \theta\right), && \text{(8.13c)} \\ & \quad\quad 0 \geq g\left(x_{k,j}, \mu(x_{k,j};\theta)\right), && \text{(8.13d)} \end{aligned}$$

$$0 \geq g_{\text{f}}(x_{N,j}). \tag{8.13e}$$

But solving (8.13) may still be intractable due to the often highly nonlinear nature of the parametric policies, especially in case of deep neural networks. Instead, we follow the typical approach used in reinforcement learning of including the constraints (8.13d) and (8.13e) as penalty terms in the cost (reward) function (8.13a), here described for a single realization w:

$$\begin{aligned} \phi_{\text{aug}}(w; N_{\text{sim}}, f_{\text{hf}}(\cdot), \mu(\cdot;\theta)) = {} & \phi(w; N_{\text{sim}}, \theta) \\ & + \sum_{k=0}^{N_{\text{sim}}-1} \rho^T \max\left(0, g_{\text{cl}}(x_k, \mu(\cdot;\theta))\right) \\ & + \rho_{\text{f}}^T \max\left(0, g_{\text{f}}(x_{N_{\text{sim}}})\right), \end{aligned} \tag{8.14}$$

where $\rho \in \mathbb{R}^{n_g}$ and $\rho_{\text{f}} \in \mathbb{R}^{n_{\text{f}}}$ are vectors to weigh the importance of the stage and terminal constraints and x_k denotes the state in the k-th time step of the trajectory resulting from $\Xi(w; N_{\text{sim}}, f_{\text{hf}}(\cdot), \mu(\cdot;\theta))$ (8.5). We can then approximate the expected value of the gradient of the augmented performance index with respect to θ by using the augmented closed-loop performance index (8.14) for all N_{sc} drawn scenarios. The resulting gradient descent step to update the policy parameters is then given by:

$$\theta \leftarrow \theta - \frac{\alpha}{N_{\text{sc}}} \sum_{j=1}^{N_{\text{sc}}} \nabla_\theta \phi_{\text{aug}}(w_j; N_{\text{sim}}, f_{\text{hf}}(\cdot), \mu(\cdot;\theta))), \tag{8.15}$$

where α is the step-length. By repeatedly evaluating (8.15), problem (8.7) is approximately solved. The gradients (8.15) can be easily computed via automatic differentiation. In machine learning, stochastic gradient descent with mini-batches is applied for better convergence results [144]. Instead of considering all N_{sc} samples for the update (8.15) at once, $\mathcal{S}$ is divided into subsets. One epoch of learning is terminated after the gradient step update has been applied for every subset.

8.4 Industrial polymerization reactor

We revisit the industrial batch polymerization reactor which has been introduced in detail in Section 7.3 to demonstrate the efficacy of the proposed approach. We shortly recap the control goals and the most important constraints. For the exact differential equations that govern the behavior of the system, consider (7.17).

The goal of the control is to produce $m_{\text{P,max}} = 20\,600$ kg of polymer m_{P} per batch as fast as possible while not using more than $m_{\text{A,max}}^{\text{acc}} = 30\,000$ kg of monomer $m_{\text{A}}^{\text{acc}}$. Two constraints are of major importance for a successful batch process. The reactor temperature T_{R} is required to stay between $T_{\text{R,min}} = 361.15$ K and $T_{\text{R,max}} = 365.15$ K to ensure a high-quality product and the adiabatic temperature T_{adiab} should not exceed $T_{\text{adiab,max}} = 382.15$ K for a safe operation. The adiabatic temperature is the temperature the reactor would reach due to the exothermic nature of the process if all cooling fails. The state constraints are defined as:

$$\begin{aligned}
&T_{\mathsf{R,min}} \leq T_{\mathsf{R}} \leq T_{\mathsf{R,max}},\\
&T_{\mathsf{adiab}} \leq T_{\mathsf{adiab,max}},\\
&T_{\mathsf{min}} \leq T_{\mathsf{S}}, T_{\mathsf{M}}, T_{\mathsf{EK}}, T_{\mathsf{AWT}} \leq T_{\mathsf{max}},\\
&m_{\mathsf{A}}^{\mathsf{acc}} \leq m_{\mathsf{A,max}}^{\mathsf{acc}},
\end{aligned} \tag{8.16}$$

where $T_{\mathsf{min}} = 273.15\,\mathsf{K}$ and $T_{\mathsf{max}} = 373.15\,\mathsf{K}$. The mass balances and temperatures can be manipulated via the coolant temperature at the inlet of the external heat exchanger $T_{\mathsf{AWT}}^{\mathsf{IN}}$ and at the inlet of the jacket $T_{\mathsf{M}}^{\mathsf{IN}}$, and via the feed flow $\dot{m}_{\mathsf{F}}$. The bounds on the control inputs are given by:

$$\begin{aligned}
&T_{\mathsf{min}}^{\mathsf{IN}} \leq T_{\mathsf{AWT}}^{\mathsf{IN}}, T_{\mathsf{M}}^{\mathsf{IN}} \leq T_{\mathsf{max}}^{\mathsf{IN}},\\
&\dot{m}_{\mathsf{F,min}} \leq \dot{m}_{\mathsf{F}} \quad\ \leq \dot{m}_{\mathsf{F,max}},
\end{aligned} \tag{8.17}$$

where $T_{\mathsf{min}}^{\mathsf{IN}} = 333.15\,\mathsf{K}$, $T_{\mathsf{max}}^{\mathsf{IN}} = 373.15\,\mathsf{K}$, $\dot{m}_{\mathsf{F,min}} = 0\,\mathsf{kg\,h^{-1}}$, and $\dot{m}_{\mathsf{F,max}} = 30\,000\,\mathsf{kg\,h^{-1}}$. The two uncertain parameters k_{U1} and k_{U2}, which describe the influence of the mass proportion U on the reaction rates k_{R1} and k_{R2}, are considered to be constant, but unknown, for each batch. The code for the polymerization reactor is publicly available[1].

8.5 Reinforced deep learning-based MPC for a semi-batch polymerization reactor

For the performance assessment of the proposed approach, we will consider two scenarios. In the first scenario, the uncertain parameters of the simulated *real* system will vary $\pm 20\%$, in the second scenario between $\pm 30\%$. Because the uncertain parameters are constant for each batch, each closed-loop trajectory (8.11) is fully defined by the initial state x_{init} and the realization of the uncertain parameters d. The intervals from which the samples are drawn for the two different degrees of uncertainty are given in Table 8.1. One advantage of the proposed approach is that in general high fidelity models can be used for the optimization of the policy parameters via reinforcing (8.15). But for simplicity of presentation and analysis of the results, the plant model used to simulate the *real* system, the prediction model used in MPC and the high fidelity model used in the reinforcement are considered equal and each batch is aborted after a maximum batch time of 3 hours.

8.5.1 Performance index

For the formulation of the performance index, the general structure of (8.14) is applied. Because the control input constraints (8.17) are given as box constraints, considering them with a penalty term in the performance index can be avoided. The outputs provided by the neural network were saturated in the closed-loop formulation (8.11) via:

$$u = \min(u_{\mathsf{max}}, \max(u_{\mathsf{min}}, \mu(x;\theta))). \tag{8.18}$$

[1] https://www.do-mpc.com/en/latest/example_gallery/industrial_poly.html

Table 8.1: Intervals from which the initial state x_{init} and the realizations of the uncertain parameters k_{U1} and k_{U2} are sampled from with uniform distribution. The adiabatic temperature is explicitly defined by the values of the other states. ©[2021] IEEE.

state/parameter	scenario	lower bound	upper bound	unit
m_{W}		9900.00	10100.00	kg
m_{A}		851.00	855.00	kg
m_{P}		26.00	27.00	kg
$m_{\text{A}}^{\text{acc}}$		362.15	364.15	kg
T_{R}		362.15	364.15	K
T_{S}		362.15	364.15	K
T_{M}		306.15	310.15	K
T_{EK}		306.15	310.15	K
T_{AWT}		290.00	310.00	K
k_{U1}	$\pm 20\%$	3.2	4.8	–
	$\pm 30\%$	2.8	5.2	–
k_{U2}	$\pm 20\%$	25.6	38.4	–
	$\pm 30\%$	22.4	41.6	–

To avoid wasting educts, we include a rule that saturates the feed $\dot{m}_{\text{F},k}$ of the learning-based controller if the necessary amount of monomer $m_{\text{A,max}}^{\text{acc}}$ to produce the desired amount of polymer $m_{\text{P,max}}$ would be exceeded in the next step $k+1$:

$$\dot{m}_{\text{F},k} = \begin{cases} \frac{m_{\text{A,max}}^{\text{acc}} - m_{\text{A},k}^{\text{acc}}}{\Delta t}, & \text{if } m_{\text{A},k}^{\text{acc}} + \Delta t \cdot \dot{m}_{\text{F},k} > m_{\text{A,max}}^{\text{acc}}, \\ \dot{m}_{\text{F},k}, & \text{else.} \end{cases} \tag{8.19}$$

Therefore the constraint $m_{\text{A}}^{\text{acc}} \leq m_{\text{A,max}}^{\text{acc}}$ does not need to be included in the performance index.

The exact number of steps until a batch is finished can not be known beforehand, hence N_{sim} is chosen such that it exceeds the number of steps necessary for the combination of uncertain parameters leading to the longest batch time. To avoid that the performance index and its gradient are affected after the production goal has been reached, an indicator function

$$i(x_k) = \begin{cases} 1, & \text{if } m_{\text{P},k} < m_{\text{P,max}}, \\ 0, & \text{else,} \end{cases} \tag{8.20}$$

is designed, which will be used to set all terms of the performance index to zero when the required amount of polymer has been produced and hence the batch has ended.

Since we want to maximize the profit while meeting the quality and safety requirements, the performance index will consist of three parts. To operate the reactor in the most profitable manner, it is desirable to minimize the batch time which can be achieved by maximizing the amount of product m_{P} that is generated in each step k:

$$\phi_{m_{\mathsf{P}}} = \sum_{k=0}^{N_{\mathsf{sim}}} i(x_k) \cdot (m_{\mathsf{P},\mathsf{max}} - m_{\mathsf{P},k}). \tag{8.21}$$

To guarantee the safe operation of the reactor, we penalize violations on the constraint on T_{adiab}:

$$\phi_{T_{\mathsf{adiab}}} = \sum_{k=0}^{N_{\mathsf{sim}}} i(x_k) \cdot \mathsf{max}(0, T_{\mathsf{adiab},k} - T_{\mathsf{adiab,max}}). \tag{8.22}$$

The performance index for the reactor temperature can be formulated similarly by additionally incorporating the lower bound:

$$\phi_{T_{\mathsf{R}}} = \sum_{k=0}^{N_{\mathsf{sim}}} i(x_k) \cdot \left(\mathsf{max}(0, T_{\mathsf{R},k} - T_{\mathsf{R,max}}) + \mathsf{max}(0, T_{\mathsf{R,min}} - T_{\mathsf{R},k})\right). \tag{8.23}$$

The augmented closed-loop performance indicator (8.14) is then given by:

$$\phi_{\mathsf{aug}}(w; N_{\mathsf{sim}}, f(\cdot), \mu(\cdot;\theta)) = \rho_{m_{\mathsf{P}}}\phi_{m_{\mathsf{P}}} + \rho_{T_{\mathsf{adiab}}}\phi_{T_{\mathsf{adiab}}} + \rho_{T_{\mathsf{R}}}\phi_{T_{\mathsf{R}}}, \tag{8.24}$$

with the weighting factors $\rho_{m_{\mathsf{P}}} = 1e-5$, $\rho_{T_{\mathsf{adiab}}} = 1.0$ and $\rho_{T_{\mathsf{R}}} = 1.0$. The weighting factors were chosen such that satisfaction of the constraints was prioritized over economic efficiency in the gradient step updates (8.15).

8.5.2 Parametric policy optimization

In the multi-stage NMPC formulation, we use as stage cost and terminal cost $\ell(\cdot) = \ell_{\mathsf{f}}(\cdot) = -m_{\mathsf{P}}$, a prediction horizon of $N = 20$, a robust horizon of $N_{\mathsf{robust}} = 1$ and a control step width of 50 s for the polymerization reactor. The multi-stage NMPC controller was used to generate the data sets $\mathcal{T}_{20}$ and $\mathcal{T}_{30}$, each consisting of 350 closed-loop trajectories that were simulated with uncertainty variations of $\pm 20\%$ and $\pm 30\%$, respectively, and that were accordingly represented in the scenario trees. By solving (3.8) for $\mathcal{T}_{20}$ and $\mathcal{T}_{30}$ for deep neural networks containing $L = 6$ layers and $n_l = 30$, $l \in \mathbb{N}_{[L]}^{+}$, neurons per hidden layer, the updated initial policy parameter values θ_{20}^{*} and θ_{30}^{*} are obtained in the imitation learning phase.

For the formulation of the closed-loop performance indicators, the polymerization reactor was discretized using explicit Euler method. The considered trajectories consisted of $N_{\mathsf{sim}} = 4900$ Euler steps with a step width of 1.429 s, corresponding to $1/35$ of the controller step width, enabling to consider batch times of up to 1.94 h. CasADi [123] was used for the symbolic formulation of the closed-loop performance index and the computation of the gradient update (8.15) via automatic differentiation.

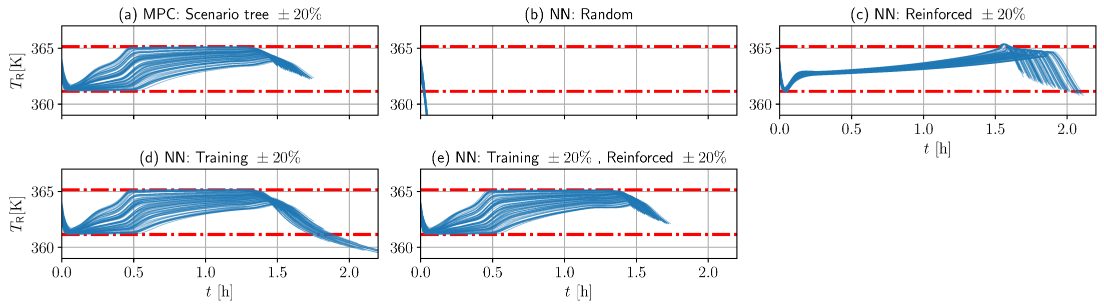


Figure 8.2: 100 trajectories of the reactor temperature T_{R} for the cases considered in Table 8.2. The uncertainties $k_{\mathrm{U}1}$ and $k_{\mathrm{U}2}$ varied $\pm 20\%$ around their nominal values. ©[2021] IEEE.

8.5.3 Performance optimization

In this subsection, we explore how the different learning paradigms affect the resulting performance of the approximate controllers for the case study in the scenario with smaller parameter variations. The data set used for imitation learning is $\mathcal{T}_{20}$ and the scenario set (8.12) that is used to reinforce the corresponding controllers via (8.15) is denoted as $\mathcal{S}_{20}$. Each of the 300 scenarios in $\mathcal{S}_{20}$ is drawn randomly according to Table 8.1 considering vations of $\pm 20\%$ and defines an initial state and a sequence of realizations of the uncertain parameters completely defining the closed-loop trajectory (8.5). Apart from the multi-stage NMPC (denoted as C1), formulated with a scenario tree considering uncertainty variations of $\pm 20\%$, we consider the following four learning-based controllers:

(C2) Neural network $\mu(x;\theta_{\text{init}})$ that was randomly initialized via the glorot uniform distribution [145],

(C3) Neural network C2 that was only reinforced via (8.15) based on $\mathcal{S}_{20}$ without an imitation learning phase to obtain $\mu(x;\hat{\theta}_{\text{init}})$,

(C4) Neural network C2 that was only updated via imitation learning based on $\mathcal{T}_{20}$ to obtain $\mu(x;\theta^*_{20})$,

(C5) Neural network C4 that was additionally reinforced via (8.15) based on $\mathcal{S}_{20}$ to obtain $\mu(x;\hat{\theta}_{20})$.

Each controller was evaluated for a complete batch for the same 100 randomly drawn scenarios which were not used in any learning phase. The results for the 100 Monte Carlo simulations are given in Table 8.2 and visualized in Fig. 8.2. The multi-stage NMPC leads to no constraint violations and often operates close to the bounds to maximize the production of polymer, as can be seen in Fig. 8.2 (a). As expected, due to the use of a naive random neural network, $\mu(x;\theta_{\text{init}})$ does not achieve a satisfactory performance and leads to significant constraint violations (see Fig. 8.2 (b)). The reinforced version $\mu(x,\hat{\theta}_{\text{init}})$ manages to achieve a reasonable performance after only 20 gradient step updates (8.15), but the obtained solution is significantly worse than multi-stage NMPC (see Fig. 8.2 (c)).

Table 8.2: Performance comparison of the multi-stage NMPC and four different neural network controllers. All five controllers were simulated for the same 100 randomly generated uncertainty realizations varying $\pm 20\%$ (see Table 8.1). ©[2021] IEEE.

Controller		**Scenario**	**Reinforced**	**Batch time**			**Violation T_R**		**Violation T_{adiab}**	
Type	Name			Average	Minimum	Maximum	Average	Maximum	Average	Maximum
MPC	C1	$\pm 20\%$	‡	1.60208	1.48611	1.74800	0	0	0	0
NN	C2	‡	‡	3.00000†	3.00000†	3.00000†	21.53376	23.32486	0	0
	C3	‡	$\pm 20\%$	1.86944	1.66667	2.11111	0.00095	0.36908	0.00003	0.10385
	C4	$\pm 20\%$	‡	1.77681	1.48611	2.70833	0.05800	2.73062	0.00007	0.02752
	C5	$\pm 20\%$	$\pm 20\%$	1.58222	1.47222	1.73611	0.00001	0.01265	0.00003	0.03761

† Influenced by the limit of three hours on the maximum batch time.
‡ The controller was not initialized via imitation learning or it was not reinforced.

The purely imitation learning-based controller $\mu(x;\theta^*_{20})$ manages to closely replicate the behavior of the multi-stage NMPC for the first 1.5 h (see Fig. 8.2 (d)), but has difficulties to produce the desired amount of polymer m_{P}, which results in long batches of up to 2.7 h. The reason for the unsatisfactory behavior in the final phase is due to two very different behaviors of system that occur in one batch. In the first phase, the reactor is fed ($\dot{m}_{\mathrm{F}} > 0$), whereas in the second phase at the end of the batch, no feed is added ($\dot{m}_{\mathrm{F}} = 0$). Since the second phase is relatively short in comparison, it is underrepresented in the data set $\mathcal{T}_{20}$ which makes it difficult to learn via imitation learning.

By reinforcing the neural network controller $\mu(x;\theta^*_{20})$ based on the scenarios $\mathcal{S}_{20}$ to derive $\mu(x;\hat{\theta}_{20})$, the performance in the final phase of the batch is significantly improved while avoiding the conservative performance obtained from direct reinforcement learning, as Fig. 8.3 (e) shows. Even multi-stage NMPC is outperformed by $\mu(x;\hat{\theta}_{20})$ in terms of batch times with negligible violations of the bounds on reactor temperature and adiabatic temperature. The advantage of the proposed approach is that the whole trajectory of a batch is considered for reinforcing the policy parameters, while the NMPC only considers a time window of 16.66 min to compute the optimal control input.

The results show that controllers obtained from the proposed approach perform better than purely imitation learning or reinforcement learning-based controllers because approximation errors originating from imitation learning can be corrected and the very good initialization obtained via imitation learning is necessary to find high-performance policy parameters. The performance can be even better than solving the exact NMPC online, because the full batch and a high fidelity model can be considered in the gradient step updates of the policy parameters.

8.5.4 Transfer learning to different control tasks

One major drawback of imitation learning-based controllers is the tedious process of retraining the controller when the control task changes. The retraining process re-

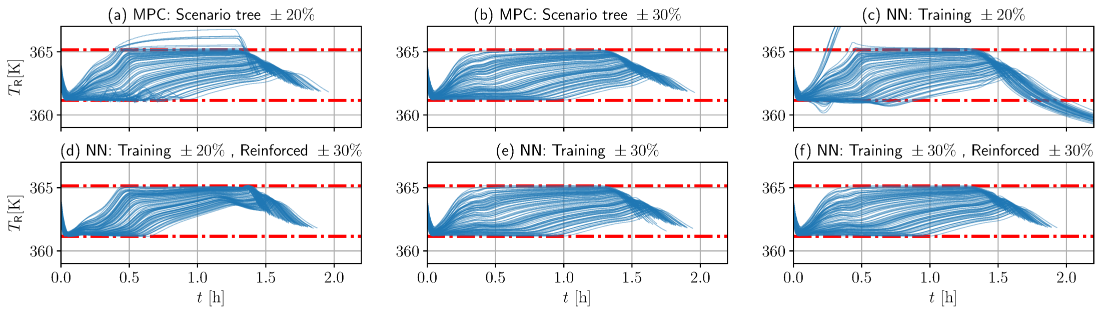


Figure 8.3: Visualization of the 100 trajectories of the reactor temperature T_R for the cases considered in Table 8.3. The uncertainties $k_{\mathrm{U}1}$ and $k_{\mathrm{U}2}$ varied $\pm 30\%$ around their nominal values. ©[2021] IEEE.

quires regenerating a new data set, which implies solving potentially many complex NMPC problems. We show here that reinforcing can make regenerating a new data set unnecessary when the control task changes. It is assumed that the uncertain parameters vary $\pm 30\%$ in the simulations, see Table 8.1, instead of the $\pm 20\%$ considered in the previous case. We investigate the two multi-stage NMPC controllers C1, considering uncertainties of $\pm 20\%$ in its scenario tree, and C6, modeling uncertainties of $\pm 30\%$, and four neural network controllers with respect to the enlarged uncertainty intervals. For deriving and updating the neural network controllers, we use the imitation learning data set $\mathcal{T}_{30}$ generated with C6 and the the 300 scenario realizations contained in $\mathcal{S}_{30}$ which were drawn according the $\pm 30\%$ values in Table 8.1. Apart from C4, which was already evaluated in the previous investigation, the following neural network controllers are considered:

(C7) Neural network C4 that was additionally reinforced via (8.15) based on $\mathcal{S}_{30}$ to obtain $\mu(x; \hat{\theta}_\mathrm{mod})$,

(C8) Neural network C2 that was only updated via imitation learning based on $\mathcal{T}_{30}$ to obtain $\mu(x; \theta^*_{30})$,

(C9) Neural network C8 that was additionally reinforced via (8.15) based on $\mathcal{S}_{30}$ to obtain $\mu(x; \hat{\theta}_{30})$.

The results for 100 batch simulations with randomly sampled uncertainty realizations are given in Table 8.3 and the corresponding 100 trajectories of the reactor temperature are visualized in Fig. 8.3.

By applying multi-stage NMPC with the same settings as in the previous investigation, i.e. representing parameter variations of $\pm 20\%$ in the scenario tree, we obtain violations of the temperature constraints of up to 1.62 K (see Fig. 8.3 (a)). The neural network controller $\mu(x; \theta^*_{20})$, that was derived purely via imitation learning based on $\mathcal{T}_{20}$, performs even worse (see Fig. 8.3 (c)). This shows that making correct assumptions about the uncertainty of the system is essential for optimization-based and learning-based methods.

When the enlarged uncertainty intervals are properly considered, the multi-stage NMPC performs violation-free (see Fig. 8.3 (b)). The network $\mu(x; \theta^*_{30})$ approximating

Table 8.3: Performance comparison of two multi-stage NMPC and four neural network controllers. All controllers were simulated for the same 100 randomly generated uncertainty realizations varying $\pm 30\%$ (see Table 8.1). ©[2021] IEEE.

Controller		**Scenario**	**Reinforced**	**Batch time**			**Violation T$_\mathrm{R}$**		**Violation T$_\mathrm{adiab}$**	
Type	Name			Average	Minimum	Maximum	Average	Maximum	Average	Maximum
MPC	C1	$\pm 20\%$	‡	1.63194	1.37500	1.95833	0.00003	1.61907	0.00004	0.02487
	C6	$\pm 30\%$	‡	1.65931	1.47222	1.95833	0	0	0	0
NN	C4	$\pm 20\%$	‡	2.02972†	1.47222	3.0000†	0.53613	6.31002	0.00104	0.46161
	C7	$\pm 20\%$	$\pm 30\%$	1.61416	1.44444	1.87500	0.00004	0.05500	0.00004	0.05878
	C8	$\pm 30\%$	‡	1.66375	1.47222	1.94444	0.00002	0.01363	0.00058	0.08727
	C9	$\pm 30\%$	$\pm 30\%$	1.64736	1.47222	1.93056	0.00000	0.00165	0	0

† Influenced by the limit of three hours on the maximum batch time.
‡ The controller was not initialized via imitation learning or it was not reinforced.

this multi-stage NMPC has slightly longer batch times than NMPC and exhibits minor violations of the temperature constraints, while the reinforced version of the network, $\mu(x;\hat{\theta}_{30})$, completely avoids violations of the upper bound on T_adiab, has a maximum violation of less than 0.002 K on the reactor temperature and has shorter batch times than the NMPC controller. This highlights that the proposed method, i.e. combining imitation learning and reinforcement learning, leads to high-performance approximate controllers. The controller $\mu(x;\hat{\theta}_\text{mod})$ that was obtained by reinforcing the weights θ^*_{20} learned for the scenario with smaller uncertainties achieves after only 10 reinforcing gradient update steps a performance that is very close to $\mu(x;\hat{\theta}_{30})$, showing that the proposed approach can successfully adapt learned controllers to updated control tasks, even for a challenging case study.

We have shown that combining imitation learning and reinforcement learning can be an effective method to optimize the behavior of a parametric controller by taking into account aspects that cannot be easily incorporated in an NMPC formulation, e.g. because of the use of a short prediction horizon or a less complex prediction model. Additionally, already learned controllers can be modified to a new task without generating a new data set, which avoids the need for solving many, potentially complex, robust NMPC problems in order to generate an updated training data set.

Despite the satisfactory performance of the learning-based controllers, deriving certificates regarding the closed-loop behavior is desirable, especially for safety critical applications. Due to the complexity and nonlinearity of the considered systems and the presence of uncertainties, deriving deterministic guarantees regarding safety and performance, as shown in Chapter 6 for linear systems, is often intractable. Therefore, statistical methods will be leveraged in the next chapter to derive probabilistic guarantees.

Chapter 9

Probabilistic guarantees for closed-loop operation

In the previous chapters, it was shown that neural network controllers can be successfully applied to control complex systems, even in the output-feedback case. However, verfiably assessing closed-loop properties of approximate controllers, or any other controller subject to random disturbances or estimation errors, is particularly challenging in the case of uncertain nonlinear systems. In the case, where the nonlinearities are mild and can be covered by the uncertainty descriptions given in Chapter 6, the corresponding verification methods introduced for linear systems can be applied. But in general, more advanced methods need to be leveraged to obtain safety and performance certificates when nonlinear systems are considered.

The theory of randomized algorithms [146, 147] provides different schemes capable of addressing this issue. For example, statistical learning techniques can be used to design stochastic model predictive controllers with probabilistic guarantees [148], [149, 150]. Also, under a convexity assumption, convex scenario approaches [151] can be used in the context of chance constrained MPC [152–154]. The main limitation of the aforementioned approaches based on statistical learning results [146, 155] and scenario based ones [151] is that the number of random scenarios that have to be generated (sample complexity) grows with the dimension of the problem.

Probabilistic validation [130, 156], allows one to determine if a given controller satisfies, with a prespecified probability of violation and confidence, the constraints. The sample complexity in this case does not depend on the dimension of the problem, but only on the required guaranteed probability of violation and confidence. Examples of probabilistic verification approaches in the context of control of nonlinear uncertain systems can be found, for example, in [130, 147, 157]. These techniques have also been used for the probabilistic certification of the behavior of off-line approximations of nonlinear control laws [72, 79].

The main contribution of this chapter is a novel validation scheme which is based on the formulation of general closed-loop performance indicators that are not restricted to binary functions as in [72] and can be computed simulating the closed-loop system with any given controller. We also provide sample complexity bounds that do not grow with the size of the problem for the case of a finite family of design parameters and general performance indicators. Our approach allows to discard a finite number of worst-case closed-loop simulations, improving significantly the applicability of the probabilistic validation scheme compared to existing works. The potential of the presented approach is illustrated for a highly nonlinear towing kite system including a real-time capable embedded implementation of an approximate, but probabilistically safe, robust nonlinear model predictive controller on a low-cost microcontroller.

The chapter is organized as follows. The closed-loop performance indicators are introduced in Section 9.1 and are used in a novel probabilistic validation methodology for arbitrary controllers in Section 9.2. In Section 9.3, the multi-stage NMPC formulation is modified to anticipate the approximation error that results from learning via constraint tightening. The case study is detailed in Section 9.4 for which the probabilistic validation scheme is applied in Section 9.5.

9.1 Closed-loop performance indicators

As in Chapter 7, we are interested in optimally controlling a class of nonlinear discrete-time systems for which not all states can be measured and which is described by:

$$\begin{aligned} x^+ &= f(x,u,d), \\ y &= h(x,u,d). \end{aligned} \tag{9.1}$$

The main difference in comparison to (7.1) is that the measurement noise (v_y) is incorporated in $d \in \mathbb{R}^{n_d}$, which contains all uncertain parameters and disturbances, to ease notation. Since not all states can be measured, a state estimate $x_{\text{est}} \in \mathbb{R}^{n_x}$ should be computed based on past measurements and past inputs. It is assumed that the disturbances d take values, with high probability, from a known set $\mathcal{D}$. We further assume the following.

Assumption 9.1. *The nonlinear discrete-time system* (9.1) *is observable and controllable.*

To ensure safety, the behavior of the closed-loop system should be such that general nonlinear constraints:

$$g(x,u,d) \leq 0, \tag{9.2}$$

with $g(\cdot) \in \mathbb{R}^{n_g}$, are satisfied at all times for a trajectory of the closed-loop system:

$$x^+ = f(x,\mu(x_{\text{est}}),d), \tag{9.3}$$

where $\mu : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ is an arbitrary state-feedback controller and d is random realization of the disturbance.

9.1.1 Closed-loop behavior

In this chapter, we will focus on the assessment of controllers and will assume that a state estimate x_{est} is provided. The goal of a controller $\mu(\cdot)$ is that the closed-loop trajectory of the uncertain nonlinear system obtains a desired performance level, i.e. does not violate the predefined constraints, despite the presence of uncertainty, for any initial state x_{init} in the set $\mathcal{X}_{\text{init}}$ of feasible initial conditions, for any admissible initial estimation error $x_{\text{init}} - x_{\text{est}}$ and for any sequence of uncertainty realizations $\{d_0, d_1, \ldots, d_\infty\}$.

Determining if a given controller provides admissible closed-loop trajectories, under the presence of nonlinearity and uncertainty, is in general an intractable problem [158]. Instead, we focus on the use of finite-time closed-loop performance indicators that can be obtained by simulating the closed-loop system. The underlying assumption is that accurate models which can be run a large number of times are available so that statistical guarantees can be obtained. A closed-loop performance indicator is defined as follows.

Definition 9.1 (Closed-loop finite-time performance indicator)**.** Let the set of variables

$$w = \{x_{\text{init}}, x_{\text{est}}, d_0, \ldots, d_{N_{\text{sim}}-1}\} \tag{9.4}$$

that contains an initial condition x_{init}, an initial state estimate x_{est}, and a sequence of uncertainty realizations $\{d_0, \ldots, d_{N_{\text{sim}}-1}\}$ that also include the measurement noise, uniquely define the closed-loop trajectory

$$\Xi(w; N_{\text{sim}}, \mu) = \{x_0, x_{\text{est},0}, \mu(x_{\text{est},0}), d_0, x_1, \mu(x_{\text{est},1}), d_1, \ldots, x_{N_{\text{sim}}}\} \tag{9.5}$$

obeying (9.3) given a controller $\mu(\cdot)$ and a finite number of closed-loop simulation steps N_{sim} and where $x_{\text{est}} = x_{\text{est},0}$ and $x_{\text{init}} = x_0$. A closed-loop finite-time performance indicator is a measurable function

$$\phi(w; N_{\text{sim}}, \mu) : \mathcal{W} = \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_d} \times \cdots \times \mathbb{R}^{n_d} \to \mathbb{R} \tag{9.6}$$

that takes as input all variables defining the closed-loop trajectories for a controller $\mu(\cdot)$ until time N_{sim} and returns a scalar as a measure of the closed-loop performance.

The main difference between the set of variables (9.4), the definition of the closed-loop trajectory (9.5) and the performance indicator (9.6) in comparison to their counterparts (see (8.4), (8.5) and (8.6)) defined for the state-feedback case in Chapter 8 is that the state estimates need to be considered. Additionally, the formulation of the the performance indicator (9.6) does not explicitly specify the used system model (as was done in (8.6)) as only a single model will be considered throughout this chapter. In the following, further assumptions are made to enable the probabilistic analysis of any feedback controller in closed-loop operation.

Assumption 9.2. *There exists a simulator that is able to compute closed-loop trajectories defined by* (9.3)*. In addition, there exists a known operator $\Phi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_x}$ that provides the state estimation $x_{\text{est},k}$ based on the previous estimate $x_{\text{est},k-1}$, the current measurement y_k and the currently applied control input u_k. That is,*

$$x_{\text{est},k} = \Phi(x_{\text{est},k-1}, y_k, u_k). \tag{9.7}$$

Assumption 9.2 implies that given N_{sim} and the controller $\mu(\cdot)$, the closed-loop trajectories (9.5) are completely determined by any set of variables w that follow the probability distribution χ_w. Probabilistic validation normally relies on a binary performance indicator that determines if the closed-loop is admissible or not. That is,

$$\phi(w; N_{\text{sim}}, \mu) = \begin{cases} 0 & \text{if the closed-loop trajectory is admissible for } w, \\ 1 & \text{otherwise.} \end{cases}$$

For this particular setting, one can resort to well-known results to obtain probabilistic guarantees about the probability $\text{Pr}_{\chi_w}(\phi(w; N_{\text{sim}}, \mu))$ of obtaining an admissible trajectory or not when a realization w is drawn randomly from χ_w. For a review on these results, see [130]. See also [72], where Hoeffding's inequality [159] is used to derive probabilistic guarantees in the context of learning an approximate model predictive controller.

In this thesis, we address a more general setting in which we do not circumscribe the performance indicator to the class of binary functions. For example, we consider the closed-loop finite-time performance indicator given by the largest value of any component $g(\cdot)^{(i)}$, $i \in \mathbb{N}^+_{[n_g]}$, of (9.2) along the closed-loop trajectory (9.5) as:

$$\phi(w; N_{\text{sim}}, \mu) = \max_{\substack{k \in \mathbb{N}_{[N_{\text{sim}}-1]}, \\ i \in \mathbb{N}^+_{[n_g]}}} g(x_k, \mu(x_{\text{est},k}), d_k)^{(i)}. \tag{9.8}$$

Another possibility is to consider the average constraint violation as a performance indicator. That is,

$$\phi(w; N_{\text{sim}}, \mu) = \frac{1}{N_{\text{sim}} n_{\text{g}}} \sum_{k=0}^{N_{\text{sim}}-1} \sum_{i=1}^{n_g} \max\{0, g(x_k, \mu(x_{\text{est},k}), d_k)^{(i)}\}. \tag{9.9}$$

Moreover, in many applications it is relevant to consider indicators related to the closed-loop cost, such as an average stage cost:

$$\phi(w; N_{\text{sim}}, \mu) = \frac{1}{N_{\text{sim}}} \sum_{k=0}^{N_{\text{sim-1}}} \ell(x_k, \mu(x_{\text{est},k})), \tag{9.10}$$

or any other combination. In the following section we address how to obtain probabilistic guarantees on the random variable $\phi(w; N_{\text{sim}}, \mu)$. The strength of the proposed approach is that it can be used to basically verify any closed-loop performance measure probabilistically.

9.2 Probabilistic validation

The derived closed-loop performance indicators can be used in the framework of probabilistic validation [130, 147] to obtain probabilistic guarantees regarding the satisfaction of a given set of control specifications. In this section we present a novel result that allows us to address the probabilistic validation of arbitrary control schemes where the performance is influenced by hyper-parameters such as backoff parameters or the control sampling time.

9.2.1 Probabilistic performance indicator levels

We consider a finite family of controllers

$$\mu_i(x_{\text{est}}),\ i \in \mathbb{N}^+_{[F]},$$

corresponding to F different combinations of control hyper-parameter values. The objective of this section is to provide a probabilistic validation scheme that allows us to choose, from the F possible controllers, the one with the best probabilistic certification for any given closed-loop finite-time performance indicator $\phi(w; N_{\text{sim}}, \mu_i)$. For simplicity in the notation, we denote the closed-loop finite-time performance indicator obtained with the controller μ_i with N_{sim} simulation steps as $\phi_i(w)$.

The stochastic variable w that defines the closed-loop trajectories follows the probability distribution χ_w from which it is possible to obtain independent identically distributed (i.i.d.) samples that can be used to define probabilistic performance indicator levels.

Definition 9.2 (Probabilistic performance indicator level)**.** We say that $\gamma \in \mathbb{R}$ is a probabilistic performance indicator level with violation probability $\epsilon \in (0,1)$ for a sample w drawn from χ_w for the measurable function $\phi : \mathcal{W} \to \mathbb{R}$ if the probability of violation $\text{Pr}_{\chi_w}\{\cdot\}$ satisfies

$$\text{Pr}_{\chi_w}\{\phi(w) > \gamma\} \le \epsilon.$$

To obtain probabilistic performance indicator levels for the considered controllers μ_i, $i \in \mathbb{N}^+_{[F]}$, we generate N_{sc} i.i.d. scenarios

$$w_j = \{x_{0,j}, x_{\text{est},0,j}, d_{0,j}, \dots, d_{N_{\text{sim}}-1,j}\},\ j \in \mathbb{N}^+_{[N_{\text{sc}}]}.$$

For a given controller μ_i, $i \in \mathbb{N}^+_{[F]}$, and the uncertain realizations w_j, $j \in \mathbb{N}^+_{[N_{\text{sc}}]}$, one could simulate the closed-loop dynamics and obtain the performance indicator corresponding to each uncertain realization:

$$\mathbf{v}_i = [\phi_i(w_1), \phi_i(w_2), \dots, \phi_i(w_{N_{\text{sc}}})]^T \in \mathbb{R}^{N_{\text{sc}}}.$$

The largest value of the components of $\mathbf{v}_i$ could serve as an empirical performance level for the controller μ_i provided that the number of considered scenarios N_{sc} is large enough [160]. Another possibility is to discard the $q-1$ largest components of $\mathbf{v}_i$ and consider the largest of the remaining components as a (less conservative) empirical performance indicator level (q equal to one corresponds to not discarding any component) [161]. In the following we show how to choose N_{sc} such that the obtained empirical performance indicator levels are, with high confidence $1-\delta$, probabilistic performance indicator levels with probability of violation ϵ.

9.2.2 Sample complexity

We first present a generalization of the notion of the maximum of a collection of scalars. This generalization is borrowed from the field of order statistics [162, 163], and will allow us to reduce the conservativeness that follows from the use of the standard notion of max function. See also Section 3 of [161].

Definition 9.3 (Generalized max function)**.** Given the vector

$$\mathbf{v} = [v_1, v_2, \ldots, v_{N_{\text{sc}}}]^T \in \mathbb{R}^{N_{\text{sc}}},$$

and the integer $q \in \mathbb{N}^+_{[N_{\text{sc}}]}$ we denote the q-th largest value of the obtained performance indicators as

$$\psi(\mathbf{v}, q) = v_+^{(q)},$$

where the vector $\mathbf{v}_+ = [v_+^{(1)}, v_+^{(2)}, \ldots, v_+^{(N)}]^T \in \mathbb{R}^{N_{\text{sc}}}$ is obtained by rearranging the values of the components of $\mathbf{v}$ in a non-increasing order. That is,

$$v_+^{(1)} \geq v_+^{(2)} \geq \ldots \geq v_+^{(N_{\text{sc}}-1)} \geq v_+^{(N_{\text{sc}})}.$$

From $\mathbf{v} = [v_1, \ldots, v_{N_{\text{sc}}}]^T \in \mathbb{R}^{N_{\text{sc}}}$ the minimum and maximum value of the performance indicator of the different scenario realizations is obtained via

$$\psi(\mathbf{v}, 1) = \underset{i \in \mathbb{N}^+_{[N_{\text{sc}}]}}{\text{maximize}} \quad v^{(i)} \text{ and } \psi(\mathbf{v}, N_{\text{sc}}) = \underset{i \in \mathbb{N}^+_{[N_{\text{sc}}]}}{\text{minimize}} \quad v^{(i)},$$

respectively. Furthermore, $\psi(\mathbf{v}, 2)$ denotes the second largest value in $\mathbf{v}$, $\psi(\mathbf{v}, 3)$ the third largest one, and so forth. Please note that the notation $\psi(\mathbf{v}, q)$ does not need to make explicit N_{sc}, i.e. the number of components of $\mathbf{v}$.

The next theorem provides a way to compute probabilistic performance levels for a family of F controllers. The theorem constitutes a generalization of a similar result, presented in [161] for the particular case $F = 1$. See also the seminal paper [160] for the particularization of the result to the case $q = 1, F = 1$.

Theorem 9.1. *Given the controllers μ_i, $i \in \mathbb{N}^+_{[F]}$, and the integer $q \geq 1$, suppose that N_{sc} i.i.d. scenarios*

$$w_j = \{x_{0,j}, x_{\text{est},0,j}, d_{0,j}, \ldots, d_{N_{sim}-1,j}\},\ j \in \mathbb{N}^+_{[N_{sc}]},$$

are generated. We denote with $\mathbf{v}_i$, $i \in \mathbb{N}^+_{[F]}$, the vector of performance indicators corresponding to the controller μ_i. That is,

$$\mathbf{v}_i = [\phi_i(w_1), \phi_i(w_2), \ldots, \phi_i(w_{N_{\text{sc}}})]^T \in \mathbb{R}^{N_{\text{sc}}},\ i \in \mathbb{N}^+_{[F]}.$$

Then, with probability no smaller than $1 - \delta$, we have a probability of violation

$$Pr_{\chi_w}\{\phi_i(w) > \psi(\mathbf{v}_i, q)\} \leq \epsilon,\ i \in \mathbb{N}^+_{[F]},$$

provided that $q \in \mathbb{N}^+_{[N_{\text{sc}}]}$ and

$$\sum_{\zeta=0}^{q-1} \begin{pmatrix} N_{\text{sc}} \\ \zeta \end{pmatrix} \epsilon^{\zeta}(1-\epsilon)^{N_{\text{sc}}-\zeta} \leq \frac{\delta}{F}. \tag{9.11}$$

In addition, (9.11) *is satisfied if:*

$$N_{\text{sc}} \geq \frac{1}{\epsilon}\left(q - 1 + \ln\frac{F}{\delta} + \sqrt{2(q-1)\ln\frac{F}{\delta}}\right). \tag{9.12}$$

Proof. [Theorem 9.1] Given the controller μ_i and $\gamma \in \mathbb{R}$, we denote $E_i(\gamma)$ the probability of the event $\phi_i(w) > \gamma$. That is,

$$E_i(\gamma) := \mathsf{Pr}_{\chi_w}\{\phi_i(w) > \gamma\}.$$

The probability of generating N_{sc} i.i.d scenarios and obtaining an empirical probabilistic performance level that does not meet the probabilistic specification on probability of violation is denoted as the probability of asymptotic failure. We now make use of Property 3 in [161], which states that, with probability no smaller than

$$1 - \sum_{\zeta=0}^{q-1} \begin{pmatrix} N_{\mathsf{sc}} \\ \zeta \end{pmatrix} \epsilon^{\zeta}(1-\epsilon)^{N_{\mathsf{sc}}-\zeta},$$

we have

$$E_i(\psi(\mathbf{v}_i, q)) = \mathsf{Pr}_{\chi_w}\{\phi_i(w) > \psi(\mathbf{v}_i, q)\} \leq \epsilon.$$

This means that the probability of asymptotic failure $\mathsf{Pr}_{\chi_w^{N_{\mathsf{sc}}}}\{\cdot\}$ for N_{sc} samples $w_1, \ldots,$ $w_{N_{\mathsf{sc}}}$ drawn from χ_w satisfies

$$\mathsf{Pr}_{\chi_w^{N_{\mathsf{sc}}}}\{E_i(\psi(\mathbf{v}_i, q)) > \epsilon\} \leq \sum_{\zeta=0}^{r-q} \begin{pmatrix} N_{\mathsf{sc}} \\ \zeta \end{pmatrix} \epsilon^{\zeta}(1-\epsilon)^{N_{\mathsf{sc}}-\zeta} := B(N_{\mathsf{sc}}, \epsilon, q-1).$$

Consider now the probability δ_{fail} that, after drawing N_{sc} i.i.d. samples w_j, $j \in \mathbb{N}^+_{[N_{\mathsf{sc}}]}$, one or more of the empirical performance indicator levels

$$\gamma_i = \psi(\mathbf{v}_i, q),\ i \in \mathbb{N}^+_{[F]},$$

are not probabilistic performance indicator levels with violation probability ϵ. We have

$$\begin{aligned} \delta_{\mathsf{fail}} &= \mathsf{Pr}_{\chi_w^{N_{\mathsf{sc}}}}\{\max_{i\in\mathbb{N}^+_{[F]}} E_i(\psi(\mathbf{v}_i, q)) > \epsilon\} \\ &\leq \sum_{i=1}^{F} \mathsf{Pr}_{\chi_w^{N_{\mathsf{sc}}}}\{E_i(\psi(\mathbf{v}_i, q)) > \epsilon\} \\ &\leq F \sum_{\zeta=0}^{q-1} \begin{pmatrix} N_{\mathsf{sc}} \\ \zeta \end{pmatrix} \epsilon^{\zeta}(1-\epsilon)^{N_{\mathsf{sc}}-\zeta} \leq \delta. \end{aligned}$$

That is, δ_{fail} is smaller or equal than δ provided that (9.11) holds. This proves the first claim of the property. The second one follows directly from Corollary 1 of [130], which provides an explicit number N_{sc} of samples that guarantees that a binomial expression $B(N_{\mathsf{sc}}, \epsilon, q-1)$ is smaller than a given constant. $\square$

The major advantage of Theorem 9.1 is that a family of controllers can be evaluated for the same N_{sc} samples. This is beneficial when the family of controllers can be evaluated in parallel or when drawing samples is expensive, e.g. in experimental setups. The number of required samples for the same probabilistic statement is significantly smaller as when all controllers would be evaluated in a sequential approach as in [130, 164, 165].

Another advantage of Theorem 9.1 is if one is given a family of controllers μ_i, $i \in \mathbb{N}^+_{[F]}$, one does not need to compute all the empirical performance indicator levels $\psi(\mathbf{v}_i, q), i \in \mathbb{N}^+_{[F]}$. It is sufficient to find a single one that meets the desired performance indicator levels. For example, if the performance indicator $\phi_i(w)$ is defined as the average constraint violation along the trajectory (9.9), then the controller μ_i provides an admissible closed-loop trajectory for w if and only if $\phi_i(w) = 0$. In this case, the empirical performance indicator $\psi(\mathbf{v}_i, q)$ corresponding to N_{sc} i.i.d. scenarios is equal to 0 if no more than $q-1$ trajectories are non-admissible when applying the controller μ_i to the scenarios. If N_{sc} is chosen according to (9.11) then Theorem 9.1 implies that with probability no smaller than $1-\delta$, all the controllers μ_i, $i \in \mathbb{N}^+_{[F]}$, providing $\psi(\mathbf{v}_i, q) = 0$ are such that

$$\text{Pr}_{\chi_w}(\phi_i(w) > 0) \leq \epsilon.$$

It is also important to remark that the cardinality F of the family of proposed controllers has little effect on the sample complexity N_{sc} because it appears into a logarithm. See also Subsection 4.2 in [130] for other randomized approaches based on a design space of finite cardinality.

Further, Theorem 9.1 can also be applied in the case when the performance indicators only take binary values. This has been presented in a similar form in [166] and was used for control design problems. See, for example, [157, 167].

9.3 Constraint tightening

The goal of this chapter is to derive a probabilistically safe robust output-feedback controller in form of a neural network. By leveraging the definition of reachable sets of the closed-loop system (2.3) and of output sets of feedback controllers (2.4), we can adapt the ideal formulation of the robust control problem (2.5) to the robust output-feedback NMPC problem:

$$\begin{aligned}
\underset{\mu_{\text{ideal}}(\cdot)}{\text{minimize}} \quad & J(\mathcal{X}_0, \mathcal{D}, \mu_{\text{ideal}}, N) && (9.13a)\\
\text{subject to} \quad & \mathcal{X}_0 = \{x_{\text{est}}\} \oplus \mathcal{E}_{\text{est}}, && (9.13b)\\
& \text{for all } k \in \mathbb{N}_{[N-1]}: &&\\
& \quad \mathcal{X}_{k+1} = \mathcal{R}(\mathcal{X}_k, \mathcal{D}, \mu_{\text{ideal}}) && (9.13c)\\
& \quad \mathcal{U}_k = \mathcal{Z}(\mathcal{X}_k, \mu_{\text{ideal}}) && (9.13d)\\
& \quad g_{\text{cl}}(\mathcal{X}_k, \mathcal{D}, \mu_{\text{ideal}}) \leq 0, && (9.13e)\\
& g_{\text{f}}(\mathcal{X}_N) \leq 0, && (9.13f)
\end{aligned}$$

where $\mathcal{E}_{\text{est}} \subseteq \mathbb{R}^{n_x}$ describes the set which contains the state estimation error (with high probability).

Due to the intractability of the exact reachable and output sets, we use the multi-stage NMPC formulation (2.8) to approximate (9.13) by explicitly taking into account a finite set of realizations $\tilde{\mathcal{D}} = \{d_1, d_2, \ldots, d_s\}$, that contain the most important uncertainties that affect the system. Further, we neglect the estimation error, and obtain the following multi-stage NMPC formulation:

$$\begin{aligned}
\underset{\mu_{\mathsf{ms}}(\cdot)}{\text{minimize}} \quad & \ell_{\mathsf{f}}(\mathcal{X}_N) + \sum_{k=0}^{N-1} \ell_{\mathsf{cl}}(\mathcal{X}_k, \mu_{\mathsf{ms}}), && \text{(9.14a)}\\
\text{subject to} \quad & \mathcal{X}_0 = \{x_{\mathsf{est}}\}, && \text{(9.14b)}\\
& \text{for all } k \in \mathbb{N}_{[N-1]} : \\
& \quad \mathcal{X}_{k+1} = \mathcal{R}(\mathcal{X}_k, \tilde{\mathcal{D}}, \mu_{\mathsf{ms}}) && \text{(9.14c)}\\
& \quad \mathcal{U}_k = \mathcal{Z}(\mathcal{X}_k, \mu_{\mathsf{ms}}) && \text{(9.14d)}\\
& \quad g_{\mathsf{cl}}(\mathcal{X}_k, \tilde{\mathcal{D}}, \mu_{\mathsf{ms}}) \leq 0, && \text{(9.14e)}\\
& g_{\mathsf{f}}(\mathcal{X}_N) \leq 0, && \text{(9.14f)}
\end{aligned}$$

where (9.14a) approximates the original objective (9.13a) based on explicitly considered scenarios in the set-based closed-loop stage cost:

$$\ell_{\mathsf{cl}}(\mathcal{X}_k, \mu_{\mathsf{ms}}) = \frac{1}{|\mathcal{X}_k|} \sum_{x_k \in \mathcal{X}_k} \ell(x_k, \mu_{\mathsf{ms}}(x_k)), \tag{9.15}$$

which is based on the standard stage cost $\ell(x, u)$, and the set-based terminal cost:

$$\ell_{\mathsf{f}}(\mathcal{X}_N) = \frac{1}{|\mathcal{X}_N|} \sum_{x_N \in \mathcal{X}_N} \ell_{\mathsf{f}}(x). \tag{9.16}$$

Problem (9.14) can be also formulated such that a robust horizon is considered, see Section 2.4 for more details.

Since accounting for all possible uncertainties and obtaining exact state estimates and reachable and output sets is virtually impossible, the multi-stage policy (9.14) is affected by two sources of error in comparison to the ideal policy (9.13):

$$||\mu_{\mathsf{ideal}}(x_k) - \mu_{\mathsf{ms}}(x_{\mathsf{est},k})|| \leq \epsilon_{\mathsf{est}} + \epsilon_{\mathsf{ms}}, \tag{9.17}$$

where ϵ_{est} is the estimation and measurement error and ϵ_{ms} is the error caused by the approximation of the reachable set by a set of discrete scenarios. Because solving the multi-stage NMPC problem (9.14) online is challenging, our goal is to approximate the solution of the multi-stage NMPC problem (9.14) with a deep neural network (3.2). This means that the resulting closed-loop will be controlled using the learning-based feedback law $\mu_{\mathsf{nn}}(\cdot)$ which approximates the behavior of $\mu_{\mathsf{ms}}(\cdot)$. This introduces another error $\epsilon_{\mathsf{approx}}$ on top of those described in (9.17):

$$\begin{aligned}
||\mu_{\mathsf{ideal}}(x_k) - \mu_{\mathsf{nn}}(x_{\mathsf{est},k})|| &= ||\mu_{\mathsf{ideal}}(x_k) - \mu_{\mathsf{ms}}(x_{\mathsf{est},k}) + \mu_{\mathsf{ms}}(x_{\mathsf{est},k}) - \mu_{\mathsf{nn}}(x_{\mathsf{est},k})|| \\
&\leq ||\mu_{\mathsf{ideal}}(x_k) - \mu_{\mathsf{ms}}(x_{\mathsf{est},k})|| + ||\mu_{\mathsf{ms}}(x_{\mathsf{est},k}) - \mu_{\mathsf{nn}}(x_{\mathsf{est},k})|| \\
&\leq \epsilon_{\mathsf{est}} + \epsilon_{\mathsf{ms}} + \epsilon_{\mathsf{approx}}.
\end{aligned} \tag{9.18}$$

Finding upper-bounds for each one of the errors to apply traditional robust NMPC schemes is not possible for the general nonlinear case.

To counteract the possible errors ϵ_{est}, ϵ_{ms}, and ϵ_{approx}, an additional backoff $\eta \in \mathbb{R}^{n_g}$ is used, following ideas from tube-based MPC, to tighten the original constraints of the robust NMPC problem. The resulting multi-stage NMPC problem whose solution is to be approximated via the neural network controllers is then described by:

$$
\begin{aligned}
\underset{\mu_{\text{ms},\eta}(\cdot)}{\text{minimize}} \quad & \ell_{\text{f}}(\mathcal{X}_N) + \sum_{k=0}^{N-1} \ell_{\text{cl}}(\mathcal{X}_k, \mu_{\text{ms},\eta}), && \text{(9.19a)}\\
\text{subject to} \quad & \mathcal{X}_0 = \{x_{\text{est}}\}. && \text{(9.19b)}\\
& \text{for all } k \in \mathbb{N}_{[N-1]} : \\
& \quad \mathcal{X}_{k+1} = \mathcal{R}(\mathcal{X}_k, \tilde{\mathcal{D}}, \mu_{\text{ms},\eta}) && \text{(9.19c)}\\
& \quad \mathcal{U}_k = \mathcal{Z}(\mathcal{X}_k, \mu_{\text{ms},\eta}) && \text{(9.19d)}\\
& \quad g_{\text{cl}}(\mathcal{X}_k, \tilde{\mathcal{D}}, \mu_{\text{ms},\eta}) \leq -\eta, && \text{(9.19e)}\\
& g_{\text{f}}(\mathcal{X}_N) \leq 0, && \text{(9.19f)}\\
& \eta \geq 0, && \text{(9.19g)}
\end{aligned}
$$

where the constraints (9.19e) are relaxed to account for the various error sources. Solving (9.19) online in each sampling instant would lead to the feedback controller $\mu_{\text{ms},\eta}(\cdot)$. We are however interested in the proposed approximate robust NMPC $\mu_{\text{nn},\eta}(\cdot)$ that is obtained by training a deep neural network based on input-output data generated by solving (9.19) for many different initial conditions. Introducing a backoff η does not guarantee in general that the closed-loop satisfies the constraints. For this reason, closed-loop constraint satisfaction is also not ensured a priori with any terminal set defined via (9.19f). The probabilistic verification scheme presented in the previous sections is applied to select the backoff parameter η such that the criteria specified by the designer are satisfied. The proposed methodology provides means to obtain probabilistic guarantees on the performance indicators of interest of the closed-loop uncertain system, as will be shown via the following example.

9.4 Towing kite

We investigate the optimal control of a kite which is used to tow a boat. The stable and safe operation of the kite is challenging due to the highly nonlinear system dynamics, uncertain parameters, strong influence from disturbances like wind speed and noisy measurements. To develop optimal control schemes of a kite system, typically models with moderate complexity such as [152, 168] are considered because of the required short sampling times. Although for our proposed strategy, also a high-fidelity models could be considered since the majority of the computational load is shifted offline, we consider a popular three-state model as presented in [169] to facilitate the comparison of the results with previous works. We derive an approximate deep learning-based controller from a robust NMPC formulation, which enables a very fast and easy evaluation of the controller even on computationally limited hardware. The idea of learning a controller for a kite has already been exploited in [170], where polynomial basis functions were used to approximate the behavior of a human pilot based on measurements.

9.4.1 Kite model

In the context of NMPC, we focus on the model presented in [169] which consists of three states, one control input and two uncertain parameters. The state evolution is given by the ordinary differential equations of the three angles θ_{kite}, ϕ_{kite} and ψ_{kite} of the spherical coordinate system describing the position of the kite:

$$\dot{\theta}_{\text{kite}} = \frac{v_{\text{a}}}{L_{\text{T}}}\left(\cos\psi_{\text{kite}} - \frac{\tan\theta_{\text{kite}}}{E}\right), \tag{9.20a}$$

$$\dot{\phi}_{\text{kite}} = -\frac{v_{\text{a}}}{L_{\text{T}}\sin\theta_{\text{kite}}}\sin\psi_{\text{kite}}, \tag{9.20b}$$

$$\dot{\psi}_{\text{kite}} = \frac{v_{\text{a}}}{L_{\text{T}}}\tilde{u} + \dot{\phi}_{\text{kite}}\cos\theta_{\text{kite}}, \tag{9.20c}$$

where

$$v_{\text{a}} = v_0 E\cos\theta_{\text{kite}}, \tag{9.20d}$$

$$E = E_0 - \tilde{c}\tilde{u}^2. \tag{9.20e}$$

The angle between wind and tether (zenith angle) is described by θ_{kite}, the angle between the vertical and the plane is denoted by $\phi_{\text{kite}}$and ψ_{kite}represents the orientation of the kite. The three states can be manipulated via the steering deflection $\tilde{u}$. The area of the kite is denoted as A, and L_{T} is the length of the tether. The effect of the wind is denoted as v_{a}, which is strongly influenced by the wind speed v_0, the first uncertain parameter. The glide ratio E is dependent on the base glide ratio E_0, the second uncertain parameter, and the magnitude of the steering deflection $\tilde{u}$ [171]. The parameters of the kite model are shown in the upper part of Table 9.1.

9.4.2 Wind model

The wind speed v_0 is considered as a single uncertainty in (9.19), but the realizations of the values are computed based on a simulation model. The underlying wind model was presented in [172] and is described by:

$$v_0 = v_{\text{m}} + \bar{v}_{\text{N}} + \sigma_v c_v p_v, \tag{9.21a}$$

where

$$\sigma_v = k_{\sigma_v} v_{\text{m}}, \tag{9.21b}$$

$$\bar{v}_{\text{N}} = -\sigma_v/(2v_{\text{m}}), \tag{9.21c}$$

$$\tau_{\text{F}} = L_v/v_{\text{m}}, \tag{9.21d}$$

$$K_{\text{F}} = \sqrt{1.49\tau_{\text{F}}/T_v}, \tag{9.21e}$$

$$c_v = K_F/\tau_{\text{F}}, \tag{9.21f}$$

$$\dot{p}_v = -p_v/\tau_{\text{F}} + w_{\text{tb}}, \tag{9.21g}$$

Table 9.1: Overview of the model states and parameters and as which variable they are considered in (9.19).

	Symbol	Type	Values / Constraints	Units	Variable
kite model	θ_{kite}	State	$[0, \frac{\pi}{2}]$	rad	x
	ϕ_{kite}	State	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	rad	x
	ψ_{kite}	State	$[0, 2\pi]$	rad	x
	$\tilde{u}$	Control input	$[-10, 10]$	N	u
	$\tilde{c}$	Known parameter	0.028	-	-
	β	Known parameter	0	rad	-
	ρ	Known parameter	1	kg m^{-3}	-
	h_{min}	Known parameter	100	m	-
	E_0	Uncertain parameter	unif$(4, 6)$	-	d
wind model	p_v	State	-	s	d via v_0
	k_{σ_v}	Known parameter	0.14	-	
	L_v	Known parameter	100	m	
	T_v	Known parameter	0.15	s	
	v_{m}	Uncertain parameter	unif$(7, 9)$	m s^{-1}	
	w_{tb}	Uncertain parameter	normal$(0, 0.25)$	-	

when the wind shear is neglected. The term w_{m} gives the current average wind speed, w_{tb} is introduced as a white noise generator to model the short term turbulence and $p_{v,0} =$ normal$(0, 0.25)$ is the initial state of the turbulence, where

$$x_{\text{normal}} = \text{normal}(\kappa_{\text{normal}}, \sigma_{\text{normal}})$$

denotes that the variable x_{normal} follows a normal distribution with mean κ_{normal} and standard deviation σ_{normal}. In a similar manner, $x_{\text{unif}} = \text{unif}(a_{\text{unif}}, b_{\text{unif}})$ means that the variable x_{unif} follows a uniform distribution between a_{unif} and b_{unif}. An overview of the parameters for the wind model is given in the lower part of Table 9.1. For further details on modeling assumptions and the choice of parameters the reader is referred to [172].

9.4.3 Extended Kalman Filter

We assume that we can measure the two angles θ_{kite} and ϕ_{kite} and the wind speed v_0. The operator $\Phi_k(\cdot)$ mentioned in Assumption 9.2 is realized as an Extended Kalman

Filter (EKF). The EKF restores an augmented state $x_{\text{aug}} = [\theta_{\text{kite}}, \phi_{\text{kite}}, \psi_{\text{kite}}, E_0, v_0]^T$ in each control instance from the measurements:

$$y_{\text{aug}} = [\theta_{\text{kite}} + w_{\theta_{\text{kite}}}, \phi_{\text{kite}} + w_{\phi_{\text{kite}}}, v_0 + w_{v_0}]^T, \tag{9.22}$$

with the zero-mean gaussian noises $w_{\theta_{\text{kite}}} = \text{normal}(0, 0.01)$, $w_{\phi_{\text{kite}}} = \text{normal}(0, 0.01)$ and $w_{v_0} = \text{normal}(0, 0.05)$. The augmented state is initialized for all simulations as

$$x_{\text{aug},0} = [\theta_{\text{kite},0} \cdot \delta_{\theta_{\text{kite}}}, \phi_{\text{kite},0} \cdot \delta_{\phi_{\text{kite}}}, \psi_{\text{kite},0} \cdot \delta_{\psi_{\text{kite}}}, E_0 \cdot \delta_{E_0}, v_0(0) \cdot \delta_{v_0}]^T,$$

where all noises $\delta_{(\cdot)}$ are drawn from normal$(1, 0.05)$. Neither the estimates of the uncertain parameters nor the measurement of the wind speed are used in the computations of the controller, because various possible realizations are considered in the scenario tree of the robust NMPC approach. The initial covariance matrix, the estimate of the process noise and the measurement noise matrix are given by:

$$\begin{aligned}
P_{\text{EKF}} &= \text{diag}([1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-2}, 1.0, 2 \times 10^{-1}), \\
Q_{\text{EKF}} &= \text{diag}([1 \times 10^{-5}, 1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-5}, 3 \times 10^{-3}]), \\
R_{\text{EKF}} &= \text{diag}([1 \times 10^{-2}, 1 \times 10^{-2}, 5 \times 10^{-2}]),
\end{aligned}$$

respectively, and the observer has a sampling time of $t_{\text{EKF}} = 0.05\,\text{s}$.

9.4.4 Objective, constraints and control settings

The goal of the control is to maximize the thrust of the tether defined by:

$$T_{\text{F}} = \frac{1}{2}\rho v_0^2 A \cos^2\theta_{\text{kite}}(E+1)\sqrt{E^2+1} \cdot (\cos\theta_{\text{kite}}\cos\beta + \sin\theta_{\text{kite}}\sin\beta\sin\phi_{\text{kite}}), \tag{9.23}$$

while maintaining a smooth control performance and satisfying the constraints. The desired behavior is enforced in the stage cost:

$$\ell(x, u) = -w_{\text{F}} T_{\text{F}} + w_u(\tilde{u} - \tilde{u}_{\text{prev}})^2, \tag{9.24}$$

where $w_{\text{F}} = 1e - 4$ and $w_u = 0.5$ are weights and $\tilde{u}_{\text{prev}}$ is the control input applied in the previous control instant. The sampling time of the controller is $t_{\text{c}} = 0.15\,\text{s}$ and a prediction horizon of $N = 40$ steps was used in the formulation of the multi-stage NMPC problem.

Throughout the operation of the kite it has to be ensured that the height of the kite:

$$h_{\text{kite}}(x) = L_{\text{T}} \sin\theta_{\text{kite}} \cos\phi_{\text{kite}}, \tag{9.25}$$

never falls below $h_{\text{min}} = 100\,\text{m}$. The height constraint is a critical constraint of the control task since the best performance is obtained when the kite is operated close to h_{min}. Because of the error ϵ_{ms} caused by the approximation of the reachable sets in the multi-stage NMPC formulation, the errors due to a deep learning-based approximation ϵ_{approx} as well as the errors related to estimation and measurement errors ϵ_{est}, constraint satisfaction can not be guaranteed. To cover the effect of the errors, the backoff parameter $\eta > 0$ m is introduced and the height constraint:

$$h_{\mathsf{kite}}(x) > h_{\mathsf{min}} + \eta, \tag{9.26}$$

is formulated as a soft constraint to avoid numerical problems.

To build a multi-stage NMPC controller, we consider the combinations of the extreme values of the base glide ratio $E_0 \in [4, 6]$ and the wind speed $v_0 \in [6\,\mathrm{m\,s^{-1}}, 10\,\mathrm{m\,s^{-1}}]$ and a one-step robust horizon resulting in a total of four scenarios. The interval for the wind speed is obtained by summarizing the possible effects of the uncertain wind model parameters v_{m}, $p_v(0)$ and w_{tb} into the single uncertain variable v_0.

9.4.5 Simulation

For the simulation of the system, it is assumed that the uncertain parameters E_0 and w_{m} are constant over a given closed-loop simulation and that w_{tb} changes every $t_{\mathsf{c}} = 0.15\,\mathrm{s}$. The values of the uncertain parameters are drawn from the probability distribution described in Table 9.1.

9.5 Probabilistic verification of a towing kite

The proposed method for the probabilistic verification of controllers is analyzed for the towing kite case study. The baseline controller for our investigations, which is also used for the training data generation for the corresponding approximate neural network controller $\mu_{\mathsf{nn},\eta}$, is the exact multi-stage NMPC controller $\mu_{\mathsf{ms}}(x_{\mathsf{est}}, \eta)$ (9.19) that derives its initial state estimate x_{est} from the EKF based on the current measurement (9.22). This means that the baseline controller is affected by the estimation error ϵ_{est} and the error ϵ_{ms} caused by the discrete representation of the uncertainties in the scenario tree and hence no formal guarantees on constraint satisfaction can be given. To avoid numerical problems for the solver in case of violations, the critical height constraint (9.26) is implemented as a soft constraint.

9.5.1 Learning an approximate output-feedback robust NMPC controller

The training process of a neural network is determined by the quality of the data and the chosen hyperparameters like activation function of the hidden layers and network size. In the following, we discuss how the training data can be generated in a way that reduces the number of samples that are needed to achieve a satisfactory approximation in comparison to a random sampling. All considered networks use hyperbolic tangent (*tanh*) (7.4) as activation function in the hidden layers and a linear output layer.

We consider two training data sets $\mathcal{T}_{\mathsf{feas}}$ and $\mathcal{T}_{\mathsf{opt}}$, and two validation data sets $\mathcal{V}_{\mathsf{feas}}$ and $\mathcal{V}_{\mathsf{opt}}$. Each data set contains samples $(x_i, \mu_{\mathsf{ms}}(x_i, \eta))$ corresponding to the numerical solution of the multi-stage problem (9.19) at state x_i. The subscript *opt* indicates that the data was derived from optimal closed-loop trajectories, e.g.

$$\mathcal{T}_{\mathsf{opt}} = \{(x_i, \mu_{\mathsf{ms}}(x_i)), \ldots, (x_{N_{\mathsf{sim}} \cdot N_{\mathsf{traj}}}, \mu_{\mathsf{ms}}(x_{N_{\mathsf{sim}} \cdot N_{\mathsf{traj}}}))\}$$

Table 9.2: Overview of the the parameter sampling via uniform distribution, normal distribution, beta(2,5) and pareto(5) distribution and results of evaluating the approximate controller μ_{nn} with $\eta = 4\,\text{m}$ for 1388 randomly drawn scenarios w. The measurement noise $w_{\text{meas}} = [w_{\theta_{\text{kite}}}, w_{\phi_{\text{kite}}}, w_{\psi_{\text{kite}}}]^T$, the initial state of the turbulence $p_{v,0} = \text{normal}(0, 0.25)$, the white noise modelling the short term turbulences $w_{\text{tb}} = \text{normal}(0, 0.25)$ and the initialization of the estimation vector $x_{\text{aug},0}$ is for all scenario spaces identical.

distribution	$\theta_{\text{kite},0}$ [°]	$\phi_{\text{kite},0}$ [°]	$\psi_{\text{kite},0}$ [°]	E_0 [-]	v_{m} [m s^{-1}]	feasible traj.	$\psi(\mathbf{v}, 4)$ [m]
Uniform	(28.0,30.0)	(-10.0,10.0)	(-2.0,2.0)	(4.0,6.0)	(7.0,9.0)	1385/1388	-0.316
Normal	(29.0,0.35)	(0.0,3.5)	(0.0,0.7)	(5.0,0.35)	(8.0,0.35)	1387/1388	-0.739
Beta	(2.0,28.0)	(20.0,-10.0)	(4.0,-2.0)	(2.0,4.0)	(2.0,7.0)	1387/1388	-0.556
Pareto	(5.0,28.0)	(5.0,-10.0)	(5.0,2.0)	(5.0,4.5)	(5.0,7.5)	1385/1388	-0.037

is composed of N_{traj} state-feedback closed-loop simulations of length N_{sim} using the exact multi-stage NMPC (9.19) under the dynamics presented in (9.20), where the uncertain parameters of the model and the initial conditions are drawn according to the distributions given in Table 9.1 and first row of Table 9.2 respectively. The subscript *feas* means that the data was obtained at randomly sampled states, e.g.

$$\mathcal{T}_{\text{feas}} = \{(x_i, \mu_{\text{ms}}(x_i)), \ldots, (x_{N_{\text{s}}}, \mu_{\text{ms}}(x_{n_{\text{tr}}}))\}$$

is obtained by sampling x_i uniformly from the feasible state space and solving (9.19).

Since the training data is generated based on simulations, the application of output-feedback via EKF is not necessary and not used for the data generation. Each trajectory consists of $N_{\text{sim}} = 400$ simulation steps which results in a total simulation time of $t_{\text{sim}} = N_{\text{sim}} \cdot t_{\text{c}} = 60\,\text{s}$. For $\mathcal{T}_{\text{opt}}$, $N_{\text{traj}} = 200$ closed-loop runs were simulated leading to $N_{\text{traj}} \cdot N_{\text{sim}} = 80000$ data pairs and for the validation $N_{\text{traj}} = 50$ simulations were rolled out, resulting in $N_{\text{traj}} \cdot N_{\text{sim}} = 20000$ samples in $\mathcal{V}_{\text{opt}}$. For the data sets $\mathcal{T}_{\text{feas}}$ and $\mathcal{V}_{\text{feas}}$, 80000 and 20000 random samples were drawn, respectively.

For the following investigations, we trained five deep networks with $L = 6$ layers and $n_l = 30, l \in \mathbb{N}^+_{[L]}$, neurons per layer on each training set and evaluated all five obtained networks on each validation set. By averaging the results over five networks the impact of the stochastic learning is reduced. Training a deep neural network with the data pairs $\mathcal{T}_{\text{opt}}$ leads to a significantly smaller average mean squared error (MSE) when compared to the training using the training data $\mathcal{T}_{\text{feas}}$, as Figure 9.1 shows, because the sampled space of optimal trajectories is smaller in comparison to the feasible space.

To investigate the impact of the training data set on the actual performance, the networks are tested on the validation sets $\mathcal{V}_{\text{feas}}$ and $\mathcal{V}_{\text{opt}}$. The networks trained on $\mathcal{T}_{\text{feas}}$ perform better when evaluated on whole feasible space with an average MSE of 0.0048 in comparison to the networks trained on $\mathcal{T}_{\text{opt}}$ with an average MSE of 0.2105. But when the networks are evaluated on the space of optimal closed-loop trajectories via $\mathcal{V}_{\text{opt}}$, the networks trained on $\mathcal{T}_{\text{opt}}$ have a significantly smaller average MSE of 0.0087 than networks trained on $\mathcal{T}_{\text{feas}}$ with an average MSE of 0.1642. The fact that

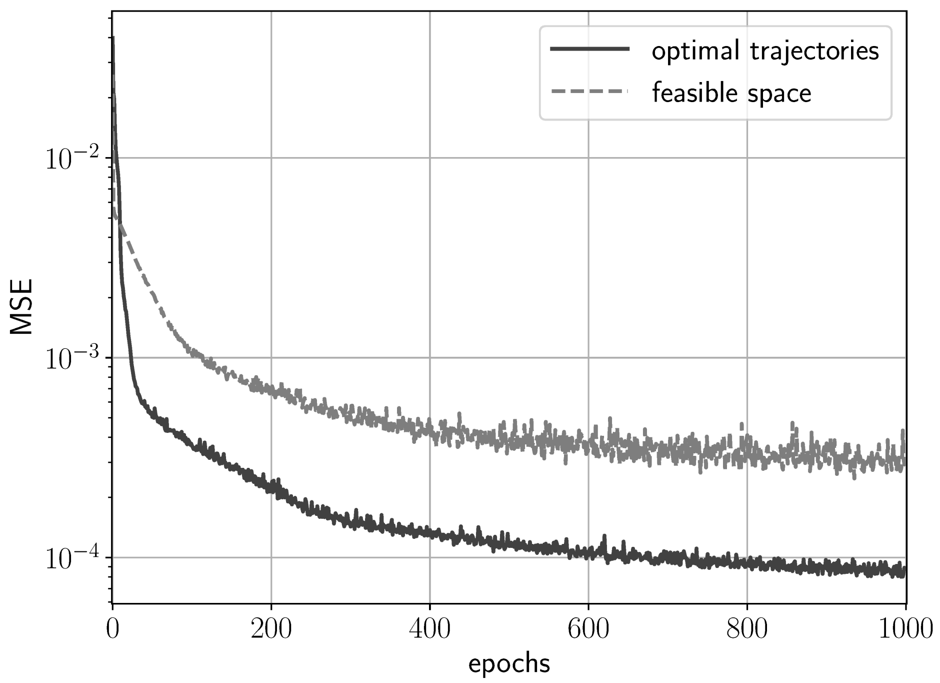


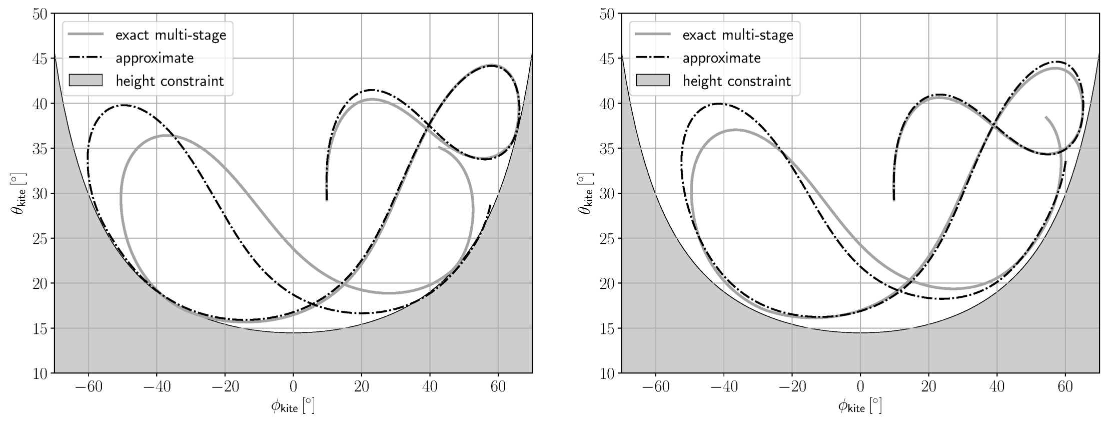
Figure 9.1: Mean squared error obtained when training a deep neural network using the space of optimal trajectories $\mathcal{T}_{\text{opt}}$ or the full feasible space $\mathcal{T}_{\text{feas}}$ as training data.

controllers trained on $\mathcal{T}_{\text{opt}}$ do not cover the whole feasible space is not critical since the learning-based controller will only operate in the neighborhood of optimal trajectories where a close approximation of the exact multi-stage NMPC is achieved. Additionally, the controllers will be probabilistically validated, and this validation is completely independent of the data used for training. Our experience shows that extracting training data from closed-loop trajectories can significantly reduce the necessary number of training samples n_{tr} and the dimensions of the neural network to obtain a desired approximation error of the deep learning-based controller in the critical domain.

For all the results presented in the remainder of the paper, we use deep neural networks with $L = 6$ and $n_l = 30$, $l \in \mathbb{N}^+_{[L]}$, which were trained on the space of optimal trajectories $\mathcal{T}_{\text{opt}}$ due to the observed superior approximation quality in the crucial regions of the state space.

9.5.2 Verification of a deep learning-based output-feedback robust NMPC

Because of the approximation errors, measurement and estimation errors as well as the errors derived from the multi-stage formulation, we refrain from a worst-case deterministic analysis and resort to the probabilistic verification scheme based on closed-loop trajectories presented in Section 9.2. We consider four possible values for the backoff hyper-parameter η, i.e. $\eta \in \{0\,\text{m}, 2\,\text{m}, 4\,\text{m}, 6\,\text{m}\}$. This leads to a family of $F = 4$ deep learning-based approximate controllers $\mu_{\text{nn},\eta}$. Each one of these controllers was obtained training on data sets $\mathcal{T}_{\text{opt},\eta}$ containing 80000 data pairs each. The resulting controllers were analyzed for N_{sc} i.i.d. scenarios w_j, $j \in \mathbb{N}^+_{[N_{\text{sc}}]}$, corresponding to N_{sc} closed-loop simulations under the dynamics presented in (9.20), where the



(a) $\eta = 0\,\text{m}$ (b) $\eta = 4\,\text{m}$

Figure 9.2: Comparison of the exact multi-stage NMPC and its deep learning-based approximation for one sample w and two different choices of the back-off parameter η. By choosing no backoff ($\eta = 0\,\text{m}$) the kite is often operated at the bound which leads to frequent constraint violations due to the estimation and uncertainty discretization errors ϵ_{est} and ϵ_{ms}. The constraint violations by the neural network controller $\mu_{\text{nn},0}$ are more significant as it is additionally affected by the approximation error ϵ_{approx} (a). By introducing a backoff of $\eta = 4\,\text{m}$ the impact of the three error sources is mitigated which enables the probabilistically safe operation of the kite with both controllers (b).

uncertain parameters of the model and the initial conditions are drawn according to the distributions given in Table 9.1 and first row of Table 9.2 respectively.

Since the height constraint (9.26) is the most critical constraint, we define the performance indicator:

$$\phi(w; N_{\text{sim}}, \mu_{\text{nn},\eta}) = \max_{k \in \mathbb{N}^+_{[N_{\text{sim}}]}} (h_{\text{min}} - h_{\text{kite}}(x_k)), \tag{9.27}$$

where x_k is the real state at sampling time k of the trajectory $\Xi(w; N_{\text{sim}}, \mu_{\text{nn},\eta})$. The performance indicator (9.27) extracts the largest violation of the minimum height h_{min}, if a violation occurs, or the closest value to h_{min} throughout one scenario. Each scenario has a duration of $60\,\text{s}$ which means $N_{\text{sim}} = 400$. To consider a controller probabilistically safe, we require that the probabilistic performance indicator satisfies:

$$\text{Pr}_{\chi_w}(\phi(w; N_{\text{sim}}, \mu_{\text{nn},\eta}) > 0) \leq \epsilon, \tag{9.28}$$

with confidence $1 - \delta$ for a randomly sampled scenario w. Following the notation of Theorem 9.1, the performance indicators corresponding to backoff parameters

$$\{0\,\text{m}, 2\,\text{m}, 4\,\text{m}, 6\,\text{m}\}$$

are collected into vectors $\mathbf{v_1}$ $\mathbf{v_2}$, $\mathbf{v_3}$, and $\mathbf{v_4}$. We consider a value of the discarding parameter $q = 4$. That is, a controller is probabilistically validated if no more than 3 simulations violate the height constraint. For these specifications ($\epsilon = 0.02$, $\delta = 1 \times 10^{-6}$, and $q = 4$), $N = 1388$ samples are required by computing the sample complexity according to (9.12). Hence, the family of controllers was evaluated for 1388 i.i.d. scenarios w_j, $j \in \mathbb{N}^+_{[1388]}$, and the results are summarized in Table 9.3.

If no backoff is considered ($\eta = 0\,\text{m}$) the exact multi-stage NMPC operates often at the constraint bound which leads to small violations of the height constraint as ϵ_{ms} and ϵ_{est} are ignored. The corresponding approximate controller $\mu_{\text{nn},0}$ is additionally affected by ϵ_{approx} (9.18) which leads to violations of the height constraint in more than half of the scenarios when applied. Exemplary trajectories for the exact multi-stage NMPC and the approximate controller for one scenario are visualized in Figure 9.2a.

By considering $\eta = 2\,\text{m}$, the amount of violations can be significantly reduced to 8 scenarios, which shows the importance of the backoff parameter. However, the performance of $\mu_{\text{nn},2}$ is not considered probabilistically safe because after discarding the allowed number of worst-case simulation runs, we get $\psi(\mathbf{v_2}, 4) = 0.273\,\text{m} > 0\,\text{m}$.

With larger backoffs $\eta = 4\,\text{m}$ and $\eta = 6\,\text{m}$, we obtain two probabilistically safe controllers with performance indicator levels $\psi(\mathbf{v_3}, 4) = -0.316\,\text{m}$ and $\psi(\mathbf{v_4}, 6) = -1.818\,\text{m}$, respectively. For the same scenario w as in Figure 9.2a, the trajectories of exact multi-stage NMPC with $\eta = 4\,\text{m}$ and $\mu_{\text{nn},4}$ are depicted in Figure 9.2b. Due to the backoff, the kite is keeping a safety distance to the constraint bound and the impact of ϵ_{ms} and ϵ_{est} does not directly lead to constraint violations. Also the trajectory of the approximate controller does not violate the trajectories despite being affected by the additional approximation error ϵ_{approx}. The preferred deep learning-based controller is $\mu_{\text{nn},4}$ due to the higher average tether thrust T_{F} provided.

By introducing a performance indicator level for the average thrust per simulation run:

$$\phi_{T_{\text{F}}}(w; N_{\text{sim}}, \mu) = \frac{1}{N_{\text{sim}}} \sum_{k=0}^{N_{\text{sim-1}}} -T_{\text{F},k}, \tag{9.29}$$

where $T_{\text{F},k}$ is the trust computed via (9.23) based on the k-th state of the trajectory defined by w, it is possible to obtain probabilistic statements about the performance in the same fashion as for violation of the height constraint. Using the parameters $\delta = 1 \times 10^{-6}$, $\epsilon = 0.02$, $F = 4$ and $q = 4$ we obtain, for the controller $\mu_{\text{nn},4}$, that with confidence $1 - \delta$, the probability that the average thrust for a simulation run of $60\,\text{s}$ duration is lower than $111.346\,\text{kN}$ is not larger than $\epsilon = 0.02$.

A smaller number of samples is required if the discarding parameter q is set equal to 1. However, this leads to more conservative results because violations of the height constraint occur throughout the closed-loop simulations used for verification. This is even worse when the performance index is a binary function determining if the trajectories are admissible or not. In this case, the obtained results are often not informative because in a binary setting with $q = 1$, a single violated trajectory out of N_{sc} determines that the controller does not meet the probabilistic constraints. Larger values for q, along with the consideration of non-binary violation performance indexes, provide more informative results. One more advantage of the proposed probabilistic method is that a family of controllers can be evaluated in parallel in the closed-loop for the same set of sampled scenarios. This can reduce the verification effort significantly, if drawing samples w from χ_w is costly or the closed-loop experiments have a long duration.

Table 9.3: Comparison of the members of a deep learning-based based family of controllers defined by $F = 4$ different choices of the backoff parameter $\eta = \{0\,\text{m}, 2\,\text{m}, 4\,\text{m}, 6\,\text{m}\}$. The parameters for the probabilistic safety certificate were chosen to $\epsilon = 0.02$ and $\delta = 1 \times 10^{-6}$. The necessary number of samples for 3 discarded worst-case runs ($q = 4$) is $N_{\text{sc}} = 1388$ and computed via (9.12).

controller	$\mu_{\text{nn},0}$	$\mu_{\text{nn},2}$	$\mu_{\text{nn},4}$	$\mu_{\text{nn},6}$
feasible trajectories	660/1388	1380/1388	1385/1388	1387/1388
$\psi(\mathbf{v}, 4)$ [m] (9.27)	1.682	0.273	-0.316	-1.818
T_{F} (avg.) [kN] (9.29)	227.516	225.997	224.185	222.179
probabilistically safe	No	No	Yes	Yes

9.5.3 Robustness of the probabilistic validation scheme

All obtained probabilistic guarantees are only valid if the assumptions about the probability density functions (PDFs) of χ_w from which the scenarios w are drawn are correct. For the verification, the N_{sc} closed-loop simulations were generated using the dynamics presented in (9.20) and the different controllers $\mu_{\text{nn},\eta}$, $\eta \in \{0\,\text{m}, 2\,\text{m}, 4\,\text{m}, 6\,\text{m}\}$. The uncertain parameters of the model and the initial conditions were drawn according to the distributions given in Table 9.1 and first row of Table 9.2 respectively.

To test the robustness of the probabilistic statements with respect to wrong assumptions about the PDFs, the performance of the approximate controllers $\mu_{\text{nn},\eta}$ is evaluated using not the distribution of the first row of Table 9.2, but the second (normal distribution), the third (beta distribution) and the fourth one (pareto distribution). The first parameter in the description of the beta distribution is the scaling and the second parameter is the offset, e.g.

$$\theta_{\text{kite},0} = 2.0 \cdot \text{beta}(2, 5) + 28.0.$$

The long-tailed pareto distribution is also described with two parameters, where the first one is the tail index and the second one is the scaling, e.g.

$$\theta_{\text{kite},0} = \text{pareto}(5.0) + 28.0.$$

The possible extreme values of samples from the space of beta distributions $\chi_{w,\text{beta}}$ are identical with those when sampling from the space of uniform distributions χ_w, see Figure 9.3. In case of sampling from $\chi_{w,\text{normal}}$ and $\chi_{w,\text{pareto}}$, which have infinite support, the occurrence of values in w which exceed the bounds of the scenarios considered in the robust MPC formulation and the verification scenarios is likely, which highlights the importance of including the discarding parameter q. The four different considered PDFs including the bounds applied in the NMPC formulation are shown in Figure 9.3 for the example base glade ratio E_0.

The results corresponding to drawing 1388 scenarios from each of the distributions $\chi_{w,\text{normal}}$, $\chi_{w,\text{beta}}$ and $\chi_{w,\text{pareto}}$, and evaluating the approximate controller $\mu_{\text{nn},4}$

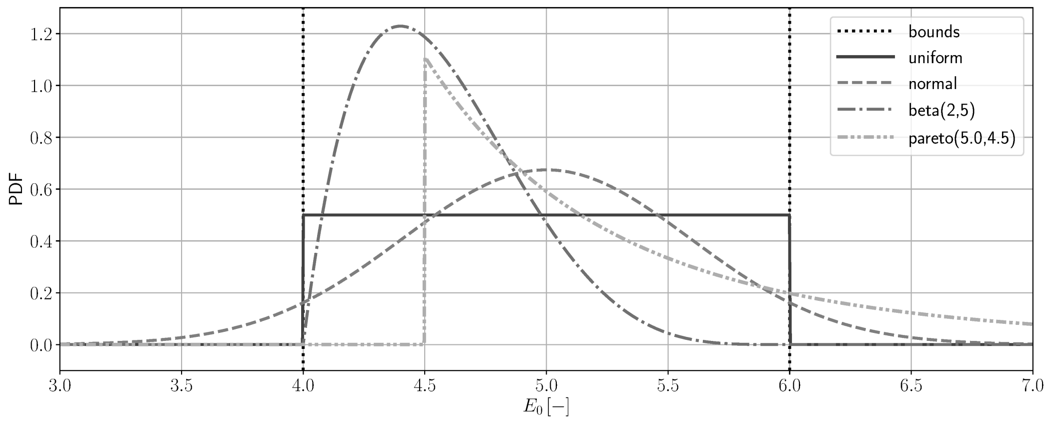


Figure 9.3: Four different considerations of the uncertain parameter base glide ratio E_0 and the considered bounds in the NMPC formulation. The normal and the pareto distribution exceed the considered bounds.

are given in Table 9.2. For the case of $\chi_{w,\text{normal}}$ and $\chi_{w,\text{beta}}$ one simulation run violates the height constraint, while three simulation runs violate the constraints for $\chi_{w,\text{pareto}}$. This means that the probabilistic requirements for the safety certificate ($\epsilon = 0.02$, $\delta = 1 \times 10^{-6}$, $F = 4$, $q = 4$) hold for all alternative choices of distributions. This shows that neither the training of the network nor the verification approach fails catastrophically when the statistical assumptions are not exactly fulfilled.

9.5.4 Embedded implementation

One of the major advantages of learning the complex optimal control law via deep neural networks is the reduction of computational load and the fast evaluation. The computation of the control input is reduced from solving an optimization problem to one matrix-vector multiplication per layer and the evaluation of the *tanh*-function. This enables the implementation of a probabilistically validated, approximate robust nonlinear model predictive control on limited hardware such as microcontrollers or field programmable gate arrays. We deployed the approximate controller on a low-cost microcontroller (ARM Cortex-M3 32-bit) running with a frequency of 89 MHz with 96 kB RAM. The memory footprint of both the EKF and the neural network that describes the approximate robust NMPC is only 67.0 kB of the 512 kB flash memory. The average time needed to evaluate the neural network was 32.1 ms (max. evaluation time: 33.0 ms) and the average evaluation time for one EKF step was 28.3 ms (max. evaluation time: 30.0 ms), which shows that the proposed controller is real-time capable with a worst-case evaluation time of 63.0 ms.

We analyzed the impact of the evaluation time on the safety by simulating the kite for the same 1388 scenarios considered in Table 9.3 drawn from the uniform distribution, but by applying the computed control inputs with a delay of $t_{\text{delay}} = 65.0$ ms, emulating a hardware-in-the-loop setting. We rounded the time delay up to 65.0 ms to account for possible time measurement errors. To deal with the additional error ϵ_{delay} caused by t_{delay}, we chose $\eta = 6$ m. Out of the 1388 simulated scenarios, 1374 were free of violations. This means the controller violates the height constraint in 0.86% of the cases, which is less than the probabilistically guarantee $\epsilon = 0.02$ chosen in Subsection 9.5.2, despite of the additional errors induced through the delay. If the performance needs to be further improved for the hardware-in-the-loop setting, training data for the controller could be generated where the deterministic t_{delay} is incorporated in the NMPC formulation. This is an additional advantage of the proposed approach, because the evaluation time of a given neural network is deterministic and can be known in advance. Additional measures to counteract the impact of delayed application of the control inputs such as advanced-step NMPC [173] or the real-time iteration scheme [174] could be also incorporated in the original robust NMPC formulation.

Part IV

Conclusions and future work

Chapter 10

Conclusions and Future work

10.1 Conclusions

The main goal of this thesis was to develop a framework that allows the deployment of robust nonlinear model predictive controllers while avoiding the challenging online optimization. This goal has been achieved by leveraging the ability of deep neural networks to act as potent function approximators. By learning the solution of a robust NMPC problem, the online application of the advanced control approach is reduced from solving a complex optimization problem in every control instance to evaluating the simple arithmetic functions which constitute the deep neural network. The three parts of the thesis explored the theoretical background for designing deep neural networks that approximate a robust nonlinear model predictive control law and methods to obtain deterministic and probabilistic guarantees on the closed-loop operation of neural network controlled systems.

The first part introduced the basic concepts, i.e. robust nonlinear model predictive control in the form of multi-stage NMPC and artificial neural networks, that allow to develop the framework in which learning-based explicit robust controllers can be obtained for complex control tasks. Further, we cast light on the origins of neural networks and recent developments - both in the theoretical field as well as for applications - that motivate the usage of neural networks as explicit robust controllers and present the neural network structure that was used throughout the thesis. In a final step, the most commonly used approach to obtain neural network controllers, namely imitation learning, is introduced as a general framework.

The second part investigated the properties of deep neural networks when applied as controllers for linear systems. It was shown that deep neural networks are theoretically capable of exactly representing the solution of model predictive control problems. However, the more significant use-case lies in the field of approximate explicit MPC. Motivated by the exponentially growing representative capabilities with respect to the depth of the network, deep neural networks were investigated as an approximate MPC approach which turned out be be a highly efficient method both in terms of computational requirements for the online evaluation as well as memory footprint. We could show that deep neural networks can be used as explicit controllers where other methods such as exact explicit MPC fail due to intractable computations in the offline phase or cumbersome evaluation in the online phase. Deep neural networks significantly outperform other approximate explicit MPC methods with respect to the closed-loop performance and the required memory footprint to store the controller on a device, which can be critical for embedded applications. Furthermore, we presented an approach that combines ideas from output range analysis of neural networks and control theoretic properties that enables the deterministic analysis of the behavior of a system controlled by a neural network controller. This approach can be used to deterministically guarantee satisfaction of constraints at all times, compute a bound on the worst-case performance and guarantee asymptotically stabilizing behavior in closed-loop operation.

The third part dealt with the more challenging case of nonlinear systems with potentially incomplete and noisy feedback information. First, we showed that deep neural networks can not only be used as an explicit controller for uncertain nonlinear systems, but also as an explicit state estimator by imitating state estimation strategies like moving horizon estimation. The analysis of the impact of approximation errors, in case of a learned controller and in case of a learned estimator, on the closed-loop behavior based on sensitivities showed that the approximation error itself is not a good indicator regarding the resulting closed-loop performance and additional steps need to be considered. One way to improve the performance of the approximate controllers is to leverage ideas from reinforcement learning because the performance of MPC approaches, from which the data for imitation learning is derived, is limited by the complexity of the models that can be considered in the optimization problem and the length of the prediction horizon. Directly optimizing the closed-loop performance in a reinforcement learning-based manner allows to consider data from detailed simulators and significantly longer prediction horizons resulting in an improved performance of the deep neural network controllers. The reinforcement learning-based approach has also been successfully applied when the control task changed, which avoids fully redesigning the approximate controller. However, the careful design of the approximate controllers does not ensure the satisfaction of the operational requirements of the closed-loop system. Due to the complexity of analyzing uncertain nonlinear systems in presence of estimation errors and noise that are controlled by highly nonlinear neural networks, we resorted to probabilistic approaches to obtain guarantees on safety and performance. The developed method is based on closed-loop performance indicators and allows to discard outliers, which results in a more informative and less conservative verification procedure in comparison to the often used binary verification methods. Further, the design of the verification approach allows to choose control parameters like the control sampling time or backoff parameters such that probabilistic specifications are met.

In summary, it could be shown that the cheap-to-evaluate and memory-efficient deep neural networks are an effective means of obtaining explicit controllers that provide on par performance with computationally challenging optimization-based NMPC controllers for systems of significant complexity and size. This enables the application of advanced control algorithms on machines with limited hardware resources such as embedded devices, and for applications that require a very high control sampling rate. Further, tools for deterministically and probabilistically verifying the safety and performance of the learning-based controllers were presented.

However, two major disadvantages reside. The first drawback is that it is unclear how many data samples have to be generated and which structure a neural network must have in order to guarantee a priori the satisfaction of the desired control performance. The second drawback concerns the verification procedures. Even though the verification is carried out offline, the number and complexity of the optimization problems to be solved in the deterministic case and the number of simulations required with potentially highly accurate models to obtain probabilistic guarantees can be challenging.

10.2 Future work

This thesis has shown that deep neural network are an effective tool to enable the deployment of complex control strategies where classic optimization-based approaches would fail. However, there are several points that can be addressed to further facilitate and improve the implementation and verification of the learning-based controllers.

When we consider imitation learning, the goal of the process is that the trained neural network should approximate any feedback control law close enough such that a specified behavior in closed-loop operation is guaranteed. Therefore it would be very beneficial to know in advance which size and structure the neural network should have in order to enable this close approximation.

Interlinked with the imitation learning problem is the generation of sufficiently rich data sets. It is difficult to know a priori how many samples should be generated such that a deep neural network can learn a close approximation. This issue could be tackled by leveraging tools from active learning that enable to locate areas where new samples provide the largest impact for learning a better closed-loop behavior.

When imitation learning is used to derive a neural network controller, the resulting controller usually behaves in a similar fashion as the original NMPC. However, when reinforcement learning is used obtain a neural network control and decision-making algorithm, the resulting behavior is more difficult to predict because the control goals need to be encoded in the loss function. One approach to shed light onto the inner *reasoning* of the neural network is analysis via explainable artificial intelligence (XAI) methods. XAI methods help to describe correlations between inputs of the network and its outputs, which can be especially beneficial in a domain that is too complex for human-designed logical controllers or optimization-based controllers. First steps towards that direction have been published in [175] for the case study of a building energy management system.

Bibliography

(1) Schwenzer, M.; Ay, M.; Bergs, T.; Abel, D. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology* **2021**, *117*, 1327–1349.

(2) Lazic, N.; Boutilier, C.; Lu, T.; Wong, E.; Roy, B.; Ryu, M.; Imwalle, G. Data center cooling using model-predictive control. *Advances in Neural Information Processing Systems* **2018**, *31*, 3818–3827.

(3) Di Carlo, J.; Wensing, P. M.; Katz, B.; Bledt, G.; Kim, S. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. *Proceedings of the International Conference on Intelligent Robots and Systems* **2018**, 1–9.

(4) Karg, B.; Alamo, T.; Lucia, S. Probabilistic performance validation of deep learning-based robust NMPC controllers. *International Journal of Robust and Nonlinear Control* **2021**, *31*, 8855–8876.

(5) Karg, B.; Lucia, S. Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics* **2020**, *50*, 3866–3878.

(6) Karg, B.; Lucia, S. Stability and feasibility of neural network-based controllers via output range analysis. *Proceedings of the Conference on Decision and Control* **2020**, 4947–4954.

(7) Karg, B.; Lucia, S. Guaranteed safe control of systems with parametric uncertainties via neural network controllers. *Proceedings of the Conference on Decision and Control* **2022**, 7302–7308.

(8) Karg, B.; Lucia, S. Approximate moving horizon estimation and robust nonlinear model predictive control via deep learning. *Computers & Chemical Engineering* **2021**, *148*, 107266.

(9) Karg, B.; Lucia, S. Reinforced approximate robust nonlinear model predictive control. *Proceedings of the International Conference on Process Control* **2021**, 149–156.

(10) Althoff, M.; Krogh, B. H. Reachability analysis of nonlinear differential-algebraic systems. *IEEE Transactions on Automatic Control* **2013**, *59*, 371–383.

(11) Sahlodin, A. M.; Chachuat, B. Convex/concave relaxations of parametric ODEs using Taylor models. *Computers & Chemical Engineering* **2011**, *35*, 844–857.

(12) Hewing, L.; Zeilinger, M. N. Stochastic model predictive control for linear systems using probabilistic reachable sets. *Proceedings of the Conference on Decision and Control* **2018**, 5182–5188.

(13) Hewing, L.; Zeilinger, M. N. Scenario-based probabilistic reachable sets for recursively feasible stochastic model predictive control. *IEEE Control Systems Letters* **2019**, *4*, 450–455.

(14) Witsenhausen, H. A minimax control problem for sampled linear systems. *IEEE Transactions on Automatic Control* **1968**, *13*, 5–21.

(15) Campo, P. J.; Morari, M. Robust model predictive control. *Proceedings of the American Control Conference* **1987**, 1021–1026.

(16) Lee, J. H.; Yu, Z. Worst-case formulations of model predictive control for systems with bounded parameters. *Automatica* **1997**, *33*, 763–781.

(17) Scokaert, P. O.; Mayne, D. Q. Min-max feedback model predictive control for constrained linear systems. *IEEE Transactions on Automatic control* **1998**, *43*, 1136–1142.

(18) De la Pena, D. M.; Bemporad, A.; Alamo, T. Stochastic programming applied to model predictive control. *Proceedings of the Conference on Decision and Control* **2005**, 1361–1366.

(19) Lucia, S.; Andersson, J. A.; Brandt, H.; Diehl, M.; Engell, S. Handling uncertainty in economic nonlinear model predictive control: A comparative case study. *Journal of Process Control* **2014**, *24*, 1247–1259.

(20) Mesbah, A.; Kolmanovsky, I. V.; Di Cairano, S. In *Handbook of model predictive control*; Springer: 2019, pp 75–97.

(21) Mayne, D. Q.; Seron, M. M.; Rakovic, S. V. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica* **2005**, *41*, 219–224.

(22) Rakovic, S. V.; Kouvaritakis, B.; Cannon, M.; Panos, C.; Findeisen, R. Fully parameterized tube MPC. *IFAC Proceedings Volumes* **2011**, *44*, 197–202.

(23) Fleming, J.; Kouvaritakis, B.; Cannon, M. Robust tube MPC for linear systems with multiplicative uncertainty. *IEEE Transactions on Automatic Control* **2014**, *60*, 1087–1092.

(24) Subramanian, S.; Abdelsalam, Y.; Lucia, S.; Engell, S. Robust tube-enhanced multi-stage nmpc with stability guarantees. *IEEE Control Systems Letters* **2021**, *6*, 1112–1117.

(25) Goulart, P. J.; Kerrigan, E. C.; Maciejowski, J. M. Optimization over state feedback policies for robust control with constraints. *Automatica* **2006**, *42*, 523–533.

(26) Lucia, S.; Finkler, T.; Engell, S. Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty. *Journal of Process Control* **2013**, *23*, 1306–1319.

(27) Bernardini, D.; Bemporad, A. Scenario-based model predictive control of stochastic constrained linear systems. *Proceedings of the Conference on Decision and Control and the Chinese Control Conference* **2009**, 6333–6338.

(28) Lucia, S., *Robust multi-stage nonlinear model predictive control*; Dissertation TU Dortmund University, Shaker, Aachen: 2015.

(29) Lucia, S.; Subramanian, S.; Limon, D.; Engell, S. Stability properties of multi-stage nonlinear model predictive control. *Systems & Control Letters* **2020**, *143*, 104743.

(30) Mayne, D. Q.; Rawlings, J. B.; Rao, C. V.; Scokaert, P. O. Constrained model predictive control: Stability and optimality. *Automatica* **2000**, *36*, 789–814.

(31) Bemporad, A.; Morari, M.; Dua, V.; Pistikopoulos, E. N. The explicit linear quadratic regulator for constrained systems. *Automatica* **2002**, *38*, 3–20.

(32) Borrelli, F.; Bemporad, A.; Morari, M. Geometric algorithm for multiparametric linear programming. *Journal of Optimization Theory and Applications* **2003**, *118*, 515–540.

(33) Geyer, T.; Torrisi, F. D.; Morari, M. Optimal complexity reduction of piecewise affine models based on hyperplane arrangements. *Proceedings of the American Control Conference* **2004**, *2*, 1190–1195.

(34) Ingole, D.; Kvasnica, M.; De Silva, H.; Gustafson, J. Reducing memory footprints in explicit model predictive control using universal numbers. *IFAC - PapersOnLine* **2017**, *50*, 11595–11600.

(35) Tøndel, P.; Johansen, T. A.; Bemporad, A. Evaluation of piecewise affine control via binary search tree. *Automatica* **2003**, *39*, 945–950.

(36) Bemporad, A.; Filippi, C. An algorithm for approximate multiparametric convex programming. *Computational Optimization and Applications* **2006**, *35*, 87–108.

(37) Johansen, T. A.; Grancharova, A. Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Transactions on Automatic Control* **2003**, *48*, 810–815.

(38) Johansen, T. A. Approximate explicit receding horizon control of constrained nonlinear systems. *Automatica* **2004**, *40*, 293–300.

(39) Fuchs, A. N.; Jones, C.; Morari, M. Optimized decision trees for point location in polytopic data sets-application to explicit MPC. *Proceedings of the American Control Conference* **2010**, 5507–5512.

(40) Bayat, F.; Johansen, T. A.; Jalali, A. A. Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC. *IEEE Transactions on Control Systems Technology* **2011**, *20*, 632–640.

(41) Krizhevsky, A.; Sutskever, I.; Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM* **2017**, *60*, 84–90.

(42) Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint: 1312. 5602* **2013**.

(43) Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M., et al. Mastering the game of Go with deep neural networks and tree search. *nature* **2016**, *529*, 484–489.

(44) Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A., et al. Mastering the game of go without human knowledge. *nature* **2017**, *550*, 354–359.

(45) McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* **1943**, *5*, 115–133.

(46) Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* **1958**, *65*, 386.

(47) Hebb, D. O., *The organization of behavior: A neuropsychological theory*; Psychology Press: 2005.

(48) Little, W. A. The existence of persistent states in the brain. *Mathematical Biosciences* **1974**, *19*, 101–120.

(49) Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* **1982**, *79*, 2554–2558.

(50) Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, G.; Cai, J., et al. Recent advances in convolutional neural networks. *Pattern Recognition* **2018**, *77*, 354–377.

(51) O'Shea, K.; Nash, R. An introduction to convolutional neural networks. *arXiv preprint: 1511. 08458* **2015**.

(52) Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **1989**, *2*, 359–366.

(53) Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **1989**, *2*, 303–314.

(54) Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural networks* **1991**, *4*, 251–257.

(55) Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory* **1993**, *39*, 930–945.

(56) Yarotsky, D. Error bounds for approximations with deep ReLU networks. *Neural Networks* **2017**, *94*, 103–114.

(57) Lu, Z.; Pu, H.; Wang, F.; Hu, Z.; Wang, L. The expressive power of neural networks: A view from the width. *Advances in Neural Information Processing Systems* **2017**, *30*, 6232–6240.

(58) Hanin, B.; Sellke, M. Approximating continuous functions by relu nets of minimal width. *arXiv preprint: 1710. 11278* **2017**.

(59) Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; Xing, E. P. Neural architecture search with bayesian optimisation and optimal transport. *Advances in Neural Information Processing Systems* **2018**, *31*, 2020–2029.

(60) Kandasamy, K.; Vysyaraju, K. R.; Neiswanger, W.; Paria, B.; Collins, C. R.; Schneider, J.; Poczos, B.; Xing, E. P. Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly. *Journal of Machine Learning Research* **2020**, *21*, 1–27.

(61) Kelley, H. J. Gradient theory of optimal flight paths. *Ars Journal* **1960**, *30*, 947–954.

(62) Werbos, P., *Beyond regression: new tools for prediction and analysis in the behavioral sciences*; Dissertation Harvard University: 1974.

(63) Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. Learning representations by back-propagating errors. *nature* **1986**, *323*, 533–536.

(64) Mesbah, A.; Wabersich, K. P.; Schoellig, A. P.; Zeilinger, M. N.; Lucia, S.; Badgwell, T. A.; Paulson, J. A. Fusion of Machine Learning and MPC under Uncertainty: What Advances Are on the Horizon? *Proceedings of the American Control Conference* **2022**, 342–357.

(65) Ljung, L. In *Signal analysis and prediction*; Springer: 1998, pp 163–173.

(66) Cui, J.; Chai, T.; Liu, X. Deep-neural-network-based economic model predictive control for ultrasupercritical power plant. *IEEE Transactions on Industrial Informatics* **2020**, *16*, 5905–5913.

(67) Nodozi, I.; O'Leary, J.; Mesbah, A.; Halder, A. A Physics-informed Deep Learning Approach for Minimum Effort Stochastic Control of Colloidal Self-Assembly. *arXiv preprint: 2208. 09182* **2022**.

(68) Williams, C. K.; Rasmussen, C. E., *Gaussian processes for machine learning*; 3; MIT press Cambridge, MA: 2006; Vol. 2.

(69) Soloperto, R.; Müller, M. A.; Trimpe, S.; Allgöwer, F. Learning-based robust model predictive control with state-dependent uncertainty. *IFAC - PapersOnLine* **2018**, *51*, 442–447.

(70) Hewing, L.; Kabzan, J.; Zeilinger, M. N. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology* **2019**, *28*, 2736–2743.

(71) Chen, S.; Saulnier, K.; Atanasov, N.; Lee, D. D.; Kumar, V.; Pappas, G. J.; Morari, M. Approximating explicit model predictive control using constrained neural networks. *Proceedings of the American Control Conference* **2018**, 1520–1527.

(72) Hertneck, M.; Köhler, J.; Trimpe, S.; Allgöwer, F. Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters* **2018**, *2*, 543–548.

(73) Zhu, L.; Takami, G.; Kawahara, M.; Kanokogi, H.; Matsubara, T. Alleviating parameter-tuning burden in reinforcement learning for large-scale process control. *Computers & Chemical Engineering* **2022**, *158*, 107658.

(74) Spielberg, S.; Tulsyan, A.; Lawrence, N. P.; Loewen, P. D.; Bhushan Gopaluni, R. Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE journal* **2019**, *65*, 16689.

(75) Spielberg, S.; Gopaluni, R.; Loewen, P. Deep reinforcement learning approaches for process control. *Proceedings of the International Symposium on Advanced Control of Industrial Processes* **2017**, 201–206.

(76) Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint: 1509. 02971* **2015**.

(77) Yoo, H.; Kim, B.; Kim, J. W.; Lee, J. H. Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation. *Computers & Chemical Engineering* **2021**, *144*, 107133.

(78) Chen, S. W.; Wang, T.; Atanasov, N.; Kumar, V.; Morari, M. Large scale model predictive control with neural networks and primal active sets. *Automatica* **2022**, *135*, 109947.

(79) Zhang, X.; Bujarbaruah, M.; Borrelli, F. Safe and near-optimal policy learning for model predictive control using primal-dual neural networks. *Proceedings of the American Control Conference* **2019**, 354–359.

(80) Bonzanini, A. D.; Paulson, J. A.; Mesbah, A. Safe learning-based model predictive control under state-and input-dependent uncertainty using scenario trees. *Proceedings of the Conference on Decision and Control* **2020**, 2448–2454.

(81) Wabersich, K. P.; Zeilinger, M. N. Linear model predictive safety certification for learning-based control. *Proceedings of the Conference on Decision and Control* **2018**, 7130–7135.

(82) Liu, C.; Arnon, T.; Lazarus, C.; Strong, C.; Barrett, C.; Kochenderfer, M. J., et al. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization* **2021**, *4*, 244–404.

(83) Dutta, S.; Jha, S.; Sankaranarayanan, S.; Tiwari, A. Output range analysis for deep feedforward neural networks. *Proceedings of the NASA Formal Methods Symposium* **2018**, 121–138.

(84) Fazlyab, M.; Morari, M.; Pappas, G. J. Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control* **2020**, *67*, 1–15.

(85) Dutta, S.; Jha, S.; Sankaranarayanan, S.; Tiwari, A. Learning and verification of feedback control systems using feedforward neural networks. *IFAC - PapersOnLine* **2018**, *51*, 151–156.

(86) Fabiani, F.; Goulart, P. J. Reliably-stabilizing piecewise-affine neural network controllers. *arXiv preprint: 2111. 07183* **2021**.

(87) Parisini, T.; Zoppoli, R. Neural networks for feedback feedforward nonlinear control systems. *IEEE Transactions on Neural Networks* **1994**, *5*, 436–449.

(88) Parisini, T.; Zoppoli, R. A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica* **1995**, *31*, 1443–1451.

(89) Chollet, F. et al. Keras, `https://keras.io`, visited on: 2023-03-27, 2015.

(90) Martin Abadi et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, visited on: 2023-03-27, 2015.

(91) Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* **2019**, *32*, 8024–8035.

(92) Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint: 1412. 6980* **2014**.

(93) Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* **2011**, *12*, 2121–2159.

(94) Lucia, S.; Navarro, D.; Karg, B.; Sarnago, H.; Lucia, O. Deep learning-based model predictive control for resonant power converters. *IEEE Transactions on Industrial Informatics* **2020**, *17*, 409–420.

(95) Bonzanini, A. D.; Paulson, J. A.; Graves, D. B.; Mesbah, A. Toward safe dose delivery in plasma medicine using projected neural network-based fast approximate NMPC. *IFAC - PapersOnLine* **2020**, *53*, 5279–5285.

(96) Karg, B.; Lucia, S. Deep learning-based embedded mixed-integer model predictive control. *Proceedings of the European Control Conference* **2018**, 2075–2080.

(97) Chen, J.; Chen, Y.; Tong, L.; Peng, L.; Kang, Y. A backpropagation neural network-based explicit model predictive control for DC–DC converters with high switching frequency. *IEEE Journal of Emerging and Selected Topics in Power Electronics* **2020**, *8*, 2124–2142.

(98) Chan, K. J.; Paulson, J. A.; Mesbah, A. Deep learning-based approximate nonlinear model predictive control with offset-free tracking for embedded applications. *Proceedings of the American Control Conference* **2021**, 3475–3481.

(99) Montufar, G. F.; Pascanu, R.; Cho, K.; Bengio, Y. On the number of linear regions of deep neural networks. *Advances in Neural Information Processing Systems* **2014**, *27*, 2924–2932.

(100) Serra, T.; Tjandraatmadja, C.; Ramalingam, S. Bounding and counting linear regions of deep neural networks. *Proceedings of the International Conference on Machine Learning* **2018**, 4558–4566.

(101) Hanin, B.; Rolnick, D. Deep relu networks have surprisingly few activation patterns. *Advances in Neural Information Processing Systems* **2019**, *32*, 361–370.

(102) Hanin, B.; Rolnick, D. Complexity of linear regions in deep networks. *Proceedings of the International Conference on Machine Learning* **2019**, 2596–2604.

(103) Ba, J.; Caruana, R. Do deep nets really need to be deep? *Advances in Neural Information Processing Systems* **2014**, *27*, 2654–2662.

(104) Hu, X.; Chu, L.; Pei, J.; Liu, W.; Bian, J. Model complexity of deep learning: A survey. *Knowledge and Information Systems* **2021**, *63*, 2585–2619.

(105) Kripfganz, A.; Schulze, R. Piecewise affine functions as a difference of two convex functions. *Optimization* **1987**, *18*, 23–29.

(106) Hempel, A. B.; Goulart, P. J.; Lygeros, J. Every continuous piecewise affine function can be obtained by solving a parametric linear program. *Proceedings of the European Control Conference* **2013**, 2657–2662.

(107) Hanin, B. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics* **2019**, *7*, 992.

(108) Herceg, M.; Kvasnica, M.; Jones, C.; Morari, M. Multi-Parametric Toolbox 3.0. *Proceedings of the European Control Conference* **2013**, 502–510.

(109) Alessio, A.; Bemporad, A. In *Nonlinear model predictive control*; Springer: 2009, pp 345–369.

(110) Bemporad, A.; Oliveri, A.; Poggi, T.; Storace, M. Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations. *IEEE Transactions on Automatic Control* **2011**, *56*, 2883–2897.

(111) Scibilia, F.; Olaru, S.; Hovd, M. Approximate explicit linear MPC via delaunay tessellation. *Proceedings of the European Control Conference* **2009**, 2833–2838.

(112) Jones, C. N.; Morari, M. Approximate explicit MPC using bilevel optimization. *Proceedings of the European Control Conference* **2009**, 2396–2401.

(113) Wen, C.; Ma, X.; Ydstie, B. E. Analytical expression of explicit MPC solution via lattice piecewise-affine function. *Automatica* **2009**, *45*, 910–917.

(114) Holaza, J.; Takács, B.; Kvasnica, M. Synthesis of simple explicit MPC optimizers by function approximation. *Proceedings of the International Conference on Process Control* **2013**, 377–382.

(115) Kvasnica, M.; Löfberg, J.; Fikar, M. Stabilizing polynomial approximation of explicit MPC. *Automatica* **2011**, *47*, 2292–2297.

(116) Cseko, L. H.; Kvasnica, M.; Lantos, B. Explicit MPC-based RBF neural network controller design with discrete-time actual Kalman filter for semiactive suspension. *IEEE Transactions on Control Systems Technology* **2015**, *23*, 1736–1753.

(117) Wang, Y.; Boyd, S. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology* **2009**, *18*, 267–278.

(118) Hu, H.; Fazlyab, M.; Morari, M.; Pappas, G. J. Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. *Proceedings of the Conference on Decision and Control* **2020**, 5929–5934.

(119) Griva, I.; Nash, S. G.; Sofer, A., *Linear and nonlinear optimization*; Siam: 2009; Vol. 108.

(120) Williams, H. P., *Model building in mathematical programming*; John Wiley & Sons: 2013.

(121) Fazlyab, M.; Morari, M.; Pappas, G. J. An introduction to neural network analysis via semidefinite programming. *Proceedings of the Conference on Decision and Control* **2021**, 6341–6350.

(122) Lucia, S.; Tatulea-Codrean, A.; Schoppmeyer, C.; Engell, S. Rapid development of modular and sustainable nonlinear model predictive control solutions. *Control Engineering Practice* **2017**, *60*, 51–62.

(123) Andersson, J. A.; Gillis, J.; Horn, G.; Rawlings, J. B.; Diehl, M. CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* **2019**, *11*, 1–36.

(124) Wächter, A.; Biegler, L. T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* **2006**, *106*, 25–57.

(125) Rungger, M.; Tabuada, P. Computing robust controlled invariant sets of linear systems. *IEEE Transactions on Automatic Control* **2017**, *62*, 3665–3670.

(126) Lofberg, J. YALMIP: A toolbox for modeling and optimization in MATLAB. *Proceedings of the Conference on Robotics and Automation* **2004**, 284–289.

(127) Gurobi Optimization, LLC Gurobi Optimizer Reference Manual, `https://www.gurobi.com`, visited on:2023-03-27, 2023.

(128) Rao, C. V.; Rawlings, J. B.; Mayne, D. Q. Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations. *IEEE Transactions on Automatic Control* **2003**, *48*, 246–258.

(129) James, F.; Hoogland, J.; Kleiss, R. Multidimensional sampling for simulation and integration: measures, discrepancies, and quasi-random numbers. *Computer Physics Communications* **1997**, *99*, 180–220.

(130) Alamo, T.; Tempo, R.; Luque, A.; Ramirez, D. R. Randomized methods for design of uncertain systems: Sample complexity and sequential algorithms. *Automatica* **2015**, *52*, 160–172.

(131) Biegler, L. T., *Nonlinear programming: concepts, algorithms, and applications to chemical processes*; SIAM: 2010.

(132) Lucia, S.; Navarro, D.; Lucia, O.; Zometa, P.; Findeisen, R. Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool. *IEEE Transactions on Industrial Informatics* **2017**, *14*, 137–145.

(133) Hindmarsh, A. C.; Brown, P. N.; Grant, K. E.; Lee, S. L.; Serban, R.; Shumaker, D. E.; Woodward, C. S. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software* **2005**, *31*, 363–396.

(134) Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; Topcu, U. Safe reinforcement learning via shielding. *Proceedings of the Conference on Artificial Intelligence* **2018**, *32*, 2669–2678.

(135) Cheng, R.; Orosz, G.; Murray, R. M.; Burdick, J. W. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. *Proceedings of the Conference on Artificial Intelligence* **2019**, *33*, 3387–3395.

(136) Chow, Y.; Nachum, O.; Duenez-Guzman, E.; Ghavamzadeh, M. A lyapunov-based approach to safe reinforcement learning. *Advances in Neural Information Processing Systems* **2018**, *31*, 8103–8112.

(137) Fulton, N.; Platzer, A. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. *Proceedings of the Conference on Artificial Intelligence* **2018**, *32*, 6485–6492.

(138) Wabersich, K. P.; Zeilinger, M. N. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. *arXiv preprint: 1812.05506* **2018**.

(139) Gros, S.; Zanon, M. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control* **2019**, *65*, 636–648.

(140) Gros, S.; Zanon, M.; Bemporad, A. Safe reinforcement learning via projection on a safe set: How to achieve optimality? *IFAC - PapersOnLine* **2020**, *53*, 8076–8081.

(141) Zanon, M.; Kungurtsev, V.; Gros, S. Reinforcement learning based on real-time iteration NMPC. *IFAC - PapersOnLine* **2020**, *53*, 5213–5218.

(142) Sutton, R. S.; Barto, A. G., *Reinforcement learning: An introduction*; MIT press: 2018.

(143) Sun, W.; Venkatraman, A.; Gordon, G. J.; Boots, B.; Bagnell, J. A. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *Proceedings of the International Conference on Machine Learning* **2017**, 3309–3318.

(144) Li, M.; Zhang, T.; Chen, Y.; Smola, A. J. Efficient mini-batch training for stochastic optimization. *Proceedings of International Conference on Knowledge Discovery and Data Mining* **2014**, 661–670.

(145) Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the International Conference on Artificial Intelligence and Statistics* **2010**, 249–256.

(146) Vidyasagar, M., *A theory of learning and generalization: with applications to neural networks and control systems*; Springer: 1997.

(147) Tempo, R.; Calafiore, G.; Dabbene, F., *Randomized algorithms for analysis and control of uncertain systems: with applications*; Springer: 2013.

(148) Grammatico, S.; Zhang, X.; Margellos, K.; Goulart, P.; Lygeros, J. A scenario approach for non-convex control design. *IEEE Transactions on Automatic Control* **2015**, *61*, 334–345.

(149) Lorenzen, M.; Dabbene, F.; Tempo, R.; Allgöwer, F. Stochastic MPC with offline uncertainty sampling. *Automatica* **2017**, *81*, 176–183.

(150) Mammarella, M.; Lorenzen, M.; Capello, E.; Park, H.; Dabbene, F.; Guglieri, G.; Romano, M.; Allgöwer, F. An offline-sampling SMPC framework with application to autonomous space maneuvers. *IEEE Transactions on Control Systems Technology* **2018**, *28*, 388–402.

(151) Calafiore, G.; Campi, M. The scenario approach to robust control design. *IEEE Transactions on Automatic Control* **2006**, *51*, 742–753.

(152) Fagiano, L.; Milanese, M.; Razza, V.; Gerlero, I. Control of Power Kites for Naval Propulsion. *Proceedings of the American Control Conference* **2010**, 4325–4330.

(153) Deori, L.; Garatti, S.; Prandini, M. Trading performance for state constraint feasibility in stochastic constrained control: A randomized approach. *Journal of the Franklin Institute* **2017**, *354*, 501–529.

(154) Margellos, K.; Goulart, P.; Lygeros, J. On the road between robust optimization and the scenario approach for chance constrained optimization problems. *IEEE Transactions on Automatic Control* **2014**, *59*, 2258–2263.

(155) Alamo, T.; Tempo, R.; Camacho, E. F. Randomized strategies for probabilistic solutions of uncertain feasibility and optimization problems. *IEEE Transactions on Automatic Control* **2009**, *54*, 2545–2559.

(156) Tempo, R.; Bai, E.; Dabbene, F. Probabilistic robustness analysis: explicit bounds for the minimum number of samples. *Systems & Control Letters* **1997**, *30*, 237–242.

(157) Alamir, M.; Fiacchini, M.; Queinnec, I.; Tarbouriech, S.; Mazerolles, M. Feedback law with probabilistic certification for Propofol-based control of BIS during anesthesia. *International Journal of Robust and Nonlinear Control* **2018**, *28*, 6254–6266.

(158) Blondel, V. D.; Tsitsiklis, J. N. A survey of computational complexity results in systems and control. *Automatica* **2000**, *36*, 1249–1274.

(159) Hoeffding, W. In *The collected works of Wassily Hoeffding*; Springer: 1994, pp 409–426.

(160) Tempo, R.; Bai, E.-W.; Dabbene, F. Probabilistic robustness analysis: Explicit bounds for the minimum number of samples. *Proceedings of the Conference on Decision and Control* **1996**, *3*, 3424–3428.

(161) Alamo, T.; Manzano, J.; Camacho, E. In *Uncertainty in Complex Networked Systems. In Honor of Roberto Tempo*, Basar, T., Ed.; Birkhäuser: 2018, pp 247–274.

(162) Ahsanullah, M.; Nevzorov, V. B.; Shakil, M., *An introduction to order statistics*; Springer: 2013; Vol. 8.

(163) Arnold, B. C.; Balakrishnan, N.; Nagaraja, H. N., *A first course in order statistics*; SIAM: 2008.

(164) Oishi, Y. Polynomial-time algorithms for probabilistic solutions of parameter-dependent linear matrix inequalities. *Automatica* **2007**, *43*, 538–545.

(165) Calafiore, G. C.; Dabbene, F.; Tempo, R. Research on probabilistic methods for control system design. *Automatica* **2011**, *47*, 1279–1293.

(166) Alamo, T.; Tempo, R.; Luque, A. On the sample complexity of randomized approaches to the analysis and design under uncertainty. *Proceedings of the American Control Conference* **2010**, 4671–4676.

(167) Alamir, M. On probabilistic certification of combined cancer therapies using strongly uncertain models. *Journal of Theoretical Biology* **2015**, *384*, 59–69.

(168) Houska, B.; Diehl, M. Optimal control of towing kites. *Proceedings of the Conference on Decision and Control* **2006**, 2693–2697.

(169) Erhard, M.; Strauch, H. Control of towing kites for seagoing vessels. *IEEE Transactions on Control Systems Technology* **2012**, *21*, 1629–1640.

(170) Fagiano, L.; Novara, C. Automatic crosswind flight of tethered wings for airborne wind energy: a direct data-driven approach. *IFAC Proceedings Volumes* **2014**, *47*, 4927–4932.

(171) Costello, S.; François, G.; Bonvin, D. Real-time optimization for kites. *IFAC Proceedings Volumes* **2013**, *46*, 64–69.

(172) Costello, S.; François, G.; Bonvin, D. Crosswind kite control–a benchmark problem for advanced control and dynamic optimization. *European Journal of Control* **2017**, *35*, 1–10.

(173) Zavala, V. M.; Biegler, L. T. The advanced-step NMPC controller: Optimality, stability and robustness. *Automatica* **2009**, *45*, 86–93.

(174) Diehl, M.; Bock, H. G.; Schlöder, J. P.; Findeisen, R.; Nagy, Z.; Allgöwer, F. Real-time optimization and nonlinear model predictive control of processes governed by differential- algebraic equations. *Journal of Process Control* **2002**, *12*, 577–585.

(175) Utama, C.; Karg, B.; Meske, C.; Lucia, S. Explainable artificial intelligence for deep learning-based model predictive controllers. *Proceedings of the International Conference on System Theory, Control and Computing* **2022**, 464–471.