KTH ROYAL INSTITUTE
OF TECHNOLOGY

Licentiate Thesis in Electrical Engineering

Learning in the Loop: On Neural Network-based Model Predictive Control and Cooperative System Identification

REBECKA WINQVIST

Stockholm, Sweden 2023

Learning in the Loop: On Neural Network-based Model Predictive Control and Cooperative System Identification

REBECKA WINQVIST

Academic Dissertation which, with due permission of the KTH Royal Institute of Technology, is submitted for public defence for the Degree of Licentiate of Engineering on Wednesday the 22nd November 2023, at 1:00 p.m. in Q2, Malvinas väg 10, Stockholm.

Licentiate Thesis in Electrical Engineering
KTH Royal Institute of Technology
Stockholm, Sweden 2023

TRITA-EECS-AVL-2023:63
ISBN 978-91-8040-710-6

Printed by: Universitetsservice US-AB, Sweden 2023

Till mormor

Abstract

In the context of control systems, the integration of machine learning mechanisms has emerged as a key approach for improving performance and adaptability. Notable progress has been made across several aspects of the control loop, including learning-based techniques for system identification and estimation, filtering and denoising, and controller design. This thesis delves into the rapidly expanding domain of learning in control, with a particular focus placed on learning-based controllers and learning-based identification methods.

The first part of this thesis is devoted to the investigation of *Neural Network* approximations of *Model Predictive Control (MPC)*. Model-agnostic neural network structures are compared to networks employing MPC-specific information, and evaluated in terms of two performance metrics. The main novel aspect lies in the incorporation of gradient data in the training process, which is shown to enhance the accuracy of the network generated control inputs. Furthermore, experimental results reveal that MPC-informed networks outperform the agnostic counterparts in scenarios when training data is limited.

In acknowledgement of the crucial role accurate system models play in in the control loop, the second part of this thesis lends its focus to learning-based identification methods. This line of work addresses the important task of characterizing and modeling dynamical systems, by introducing cooperative system identification techniques to enhance estimation performance. Specifically, it presents a novel and generalized formulation of the *Correctional Learning* framework, leveraging tools from *Optimal Transport*. The correctional learning framework centers around a *teacher-student model*, where an expert agent (teacher) modifies the sampled data used by the learner agent (student), to improve the student's estimation process. By formulating correctional learning as an optimal transport problem, a more adaptable framework is achieved, better suited for estimating complex system characteristics and accommodating alternative intervention strategies.

Sammanfattning

Inom reglerteknik har integrationen av maskininlärningsmetoder framträtt som en central strategi för att förbättra prestanda och adaptivitet hos styrsystem. Betydande framsteg har gjorts inom flera viktiga aspekter av reglerkretsen, såsom inlärningsbaserade metoder för systemidentifiering och parameterskattning, filtrering och brusreducering samt reglersyntes. Denna avhandling fördjupar sig i området inlärning för reglerteknik med särskild betoning på inlärningsbaserade regulatorer och identifieringsmetoder.

Avhandlingens första del behandlar undersökningen av neuronnätsbaserad *Modellprediktiv Reglering (MPC)*. Olika nätstrukturer studeras, både generella black box-nät och nät som väver in MPC-specifik information i sin struktur. Dessa nät jämförs och utvärderas med avseende på två prestandamått genom experiment på realistiska två- och fyrdimensionella system. Den huvudsakliga nyskapande aspekten är inkluderingen av gradientdata i träningsprocessen, vilket visar sig förbättra noggrannheten av de genererade styrsignalerna. Vidare påvisar de experimentella resultaten att en MPC-informerad nätstruktur leder till förbättrad prestanda när mängden träningsdata är begränsad.

Med insikt om vikten av noggranna matematiska modeller av styrsystemet, riktar den andra delen av avhandlingen sitt fokus mot inlärningsbaserade identifieringsmetoder. Denna forskningsgren behandlar karakterisering och modellering av dynamiska system med hjälp av maskininlärning. Avhandlingen bidrar till området genom att introducera kooperativa systemidentifieringsmetoder för att förbättra parameterskattningen. Specifikt utnyttjas verktyg från *Optimal Transport* för att introducera en ny och mer generell formulering av ramverket *Correctional Learning*. Detta ramverk är baserat på en *mästare-lärlingsmodell*, där en expertagent (mästare) observerar och modifierar den insamlade data som används av en lärande agent (lärling), med syftet att förbättra lärlingens skattningsprocess. Genom att formulera correctional learning som ett optimal transport-problem erhålls ett mer flexibelt ramverk, bättre lämpat för skattning av komplexa systemegenskaper samt anpassning till alternativa handlingsstrategier.

Acknowledgements

This thesis is the result of half a life's worth of choices and adventures, many of which would never have been possible if not for the people below. I owe a lot to many, to you I owe the most.

I would like to start by expressing my sincerest gratitude to my supervisor, Bo Wahlberg, whose commitment to creating a strong and supportive research atmosphere has made my academic journey a truly exceptional one. I am especially thankful for your invaluable input and guidance throughout these years, and for giving me the trust and freedom to explore different research directions. A deep thank you also to my co-supervisor, Cristian Rojas, for your willingness to help, discuss and inspire by sharing from your bottomless pool of knowledge. Your guidance has been central for developing the second part of this thesis.

Embarking on a PhD journey is a challenge in and of itself. Doing so in the midst of a burning pandemic indeed adds another level to it. Therefore, I extend a special thank you to Christine Korssell, for keeping me sane with our long, well-needed walks when working from home, and for being a great friend in general. A massive thanks also to Robert Bereza, both for the above, but also for all the help and support at work during this time. My first year would have been absolutely dreadful without you, and I doubt I would have made it out on the other side, had it not been for you. Thank you for never failing to initiate... let's call them "interesting"... discussions during lunches, and for sharing your exciting life (train) stories with us oldies. Keep keeping us young, wild and crazy.

Another special thanks goes out to Inês Lourenço, for being a great mentor, collaborator and friend. Thank you for your kindness, for your contagious joy and laughter, and for everything you have taught me. I wish you all the best for your future booking endeavors and adventures. Dženan Lapandić, Rijad Alisic and Elis Stefansson, thank you for all the great times brightening up those not-so-great working days. Dženan especially for always making me laugh, Rijad for reminding me of home, and Elis for the joy and music. Alessio Russo, Adam Miksits, Elisa Bin and Jacob Lindbäck for being great office mates and contributing to the friendly and fun atmosphere at work. May you all live long and linear. Mayank "Majs" Sewlia for conversations on just about everything, and for being easily startled – drive safe! Javad Parsa, for all the fun we have had during our trips; a moment is never dull with you. Miguel Aguiar, Adrian Wiltz, Braghadeesh Lakshminarayanan, Viktor Molnö and Erik Berglund, for friendship. Joana Fonseca and Pedro Roque, for your unwavering kindness, encouragement, help and advice when it was needed the most.

Caroline Halfvarsson, my longest standing friend – thank you! After all these years of long-distance friendship, I'm very much looking forward to appropriately timed (and discounted) in-person dinners, second-hand shopping and inspiring conversations. May we happily discover that every street has a store with a Thorne.

Wera Maurtiz, nearing one decade of friendship, I can see why you might feel the need to travel halfway across the world to get some time away from me. Trust

me, if anything, this work has taught me patience and perseverance – so, I'll still be here when you get back. Until you do, I hope your journey will be everything you ever hoped for, and more. Enjoy the spiders.

David Ekvall, for being many things, but a great friend in particular. I will be eternally grateful for your help in moving all my cows from the equator. For that, I owe you countless hours of wall-painting and garage restoring (applies only to properties on Gotland).

Carl-Daniel Ahlberg, for your endless love, patience and support, even after I almost accidentally killed you with pesto. Thank you for teaching me things (even when I get frustrated), for inspiring me (especially when prehab exercises suck), for your internal GPS when we are (well, I am) lost, and for always having (and scratching) my chlorine dried-out back.

Samuel, I'd better mention you here, just in case you become more successful than me (which, let's face it, you probably will). Never forget who eagerly tried to teach you how to read, who proofread your school assignments, and who still makes sure that all birthday and Christmas presents are bought on time.

Finally, my parents, whom without none of this would ever have been possible. Thank you for everything that you have done, and will continue to do. For you, I am eternally grateful.

Rebecka Winqvist
October 2023.

Abbrevations & Acronyms

BBNN	Black Box Neural Network
CLQR	Constrained Linear Quadratic Regulator
cpmf	Conditional Probability Mass Function
DPP	Disciplined Parametrized Programming
eMPC	Explicit Model Predictive Control
FFNN	Feed-Forward Neural Network
HAR	Hit-And-Run
LQ	Linear-Quadratic
LQR	Linear-Quadratic Regulator
LQR-PNN	LQR Projection Neural Network
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
MSE	Mean Squared Error
NMSE	Normalized Mean Squared Error
PID	Proportional-Integral-Derivative (control)
PNN	Projection Neural Network
PWA	Piece-Wise Affine
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

Symbols & Notation

General

$\mathbb{N}_0$	The set of natural numbers including zero
$\mathbb{N}_+$	The set of non-negative natural numbers
$\mathbb{R}$, $(\mathbb{R}_+)$	The set of (non-negative) real numbers
$\mathbb{Z}$, $(\mathbb{Z}_+)$	The set of (non-negative) integers
$A \subseteq B$	Set A is a subset of set B
$A \subset B$	Set A is a proper subset of set B
$\text{card}(\mathcal{S})$	The cardinality of a set $\mathcal{S}$
$\bigcap$	Set intersection
$\bigcup$	Set union
$\mathbb{1}$	Vector of ones
$\mathbb{I}(\cdot)$	The indicator function
I	The identity matrix
v^T	Transpose of vector v
$\ell_p = \|v\|_p$	The p-norm of a vector v
$A \succ 0$	Matrix A is positive definite
$A \succeq 0$	Matrix A is positive semi-definite
$\mathbb{U}[a,b]$, $\mathbb{D}[a,b)$, $\mathbb{U}(a,b)$	Continuous uniform distribution with closed, half-open, open intervals
$\mathbf{U}(\{a,\ldots,b\})$	Discrete unfirom distribution
$\Pr[\cdot]$	Probability of event $\cdot$

$\sim$	Distributed according to
$\mathbb{E}[X]$	Expected value of X
$\mathrm{var}(X)$	Variance of X
$\nabla f(\cdot)$	Gradient of $f(\cdot)$
$\partial f(\cdot)$	Partial derivative of $f(\cdot)$
k	Discrete time

Part I:

x	State
u	Control signal (input)
y	Output
$\mathcal{X}$	State space (state constraints)
$\mathcal{U}$	Input space (input constraints)
$\mathcal{Y}$	Output space (output constraints)
n	State dimension (dimension of x)
m	Input dimension (dimension of u)
p	Output dimension (dimension of y)
A	Dynamics matrix
B	Control matrix
C	Sensor matrix
D	Direct term
λ	Eigenvalue
x_0	Initial state (of trajectory)
$\mathcal{C}$	Control invariant set
$\mathcal{C}_\infty$	Maximal control invariant set
$\mathcal{X}_\mathrm{f}$	Terminal state constraints
N	MPC horizon length
$Q,\ Q_N,\ R$	Cost matrices

L	The control matrix of the LQR
q	The number of eMPC regions
J, J_{n}	Control cost, Normalized control cost
θ	Neural network parameters
d	Dimension of network parameters θ
$\varphi(\cdot)$	Neural network activation function
$f(\cdot\,;\theta)$	Learned neural network mapping
$\mathcal{L}(\cdot)$	Neural network loss function
$\hat{u}$	Neural network predicted control signal
u'	Gradient of u w.r.t. x
γ	Gradient weight coefficient
$\mathcal{D}_x$, $\mathcal{D}_u$, $\mathcal{D}_{u'}$	Training datasets: states, control inputs, control input gradients
Part II:	
M	Number of outcomes in multinomial distribution
$[v]_i$	The ith element of a vector v
$(\mathcal{Y}, \mathcal{B})$	A measurable space: $\mathcal{Y}$ is a set, $\mathcal{B}$ is a σ-algebra of subsets of $\mathcal{Y}$
$\mathbb{M}_+(\mathcal{Y})$	The set of positive measures on $(\mathcal{Y}, \mathcal{B})$
$\mathcal{M}$	Model class
θ	Parameter vector
θ_0	True value of parameter vector
d	Dimension of θ
$\mathcal{O}$, $(\tilde{\mathcal{O}})$	Set of (modified) observations
N	Number of observed samples
n	The number of states
$L(\cdot)$	Likelihood function
b	The teacher's intervention budget
P	State transition matrix
μ^*	Optimal online policy for the teacher

Contents

Chapter 1

Introduction

"The expert in anything was once a beginner."
— Helen Hayes

During my time as an undergraduate in an engineering program, there was a running joke among students suggesting that our education was really all about "learning to learn". It carried the underlying message that the most valuable quality in an engineer was the ability to quickly learn, process and apply new information. In one way or another, this outlook on engineering seemed to manifest itself in every course we took, emphasizing the importance of the skill to learn. As we progressed through our degree, we began to realize that perhaps this joke held more truth than we initially thought. By moving from recognizing the importance of learning to explaining its essence, we raise some interesting questions: what exactly is learning, and why have we become so intent on understanding *how* we do it?

Learning is an innate and fundamental process that lies at the very heart of human progress. It defines our growth and development as individuals, and is what enables societal and technological advancements. In some sense, learning can be viewed as a lifelong process of acquiring knowledge, skills and experience. Whether it is through formal education, personal exploration, or through interactions with our environment, learning is what shapes our understanding of ourselves as well as the complexities of the world around us.

Throughout life, we experience learning in different ways. When we think of learning, we might think of repeated experiences and training, such as revising a topic for an exam, or practicing a skill. However, we also encounter instantaneous learning from e.g. the pain of touching a hot stove, or slipping on an icy surface. In psychology, learning is referred to as a *hypothetical construct*, meaning it cannot be directly observed, but only inferred from observable behaviour [1]. While this work will not delve deeper into the topic of hypothetical constructs, it is still an interesting remark to consider for the discussions in the upcoming chapters.

The field of psychology offers many different and sophisticated definitions of learning, see [1] for an introduction. In this work, we will consider a more conventional (and more relaxed) definition found in the Oxford dictionary [2] that better aligns with an engineering context.

Definition 1.1 (Learning)**.** *The process of acquiring knowledge or a skill as a result of study, experience, or teaching.*

In a world that thrives on knowledge and innovation, the importance of learning is becoming increasingly evident. With the rapid expansion of technologies such as the *Internet of Things* and the evolution of smart cities, the available data has become too vast and complex for humans to analyze. As a result, *Machine Learning (ML)* has emerged as a tool to pick up where human capabilities leave off. At its core, ML tries to mimic the human learning process by leveraging large volumes of data to train mathematical models to perform various tasks. In simpler terms, machine learning is essentially about function estimation, i.e., uncovering mathematical relationships from data. In some sense, we can say that human learning and machine learning form a symbiotic relationship: as humans, we provide the foundational knowledge, intuition and creativity that shape the learning models, whereas machine learning augments our capabilities by detecting complex patterns and revealing insights that would otherwise be too intricate or too time-consuming for us to discover.

Over the past few decades, machine learning has gained immense popularity and achieved great success in a wide range of applications. It has found its way into our everyday lives through the integration of smart home devices, recommender systems for streaming services, and through voice assistants such as Google Assistant, Amazon Alexa and iPhone's Siri. In the service sector, ML is extensively used in finance for algorithmic trading strategies, portfolio management, and fraud detection [3, 4]. ML technology has also been adopted in healthcare through health monitoring devices such as smart watches and other wellness trackers [5, 6]. In addition, ML-based approaches has become an integral part of speech and image processing applications, where it enables technologies like facial recognition, noise cancellation, and medical imaging and diagnosis. Other highly relevant applications include autonomous vehicles [7, 8], natural language processing models [9], including the recently developed chatGPT, and smart solutions in the industrial sector [10–12].

In this thesis, we will explore machine learning in the context of control systems engineering. Specifically, we will focus on employing machine learning techniques for controller design and parameter estimation. Throughout the remainder of this work, we will use the term *learning* to refer to some form of artificial or machine learning (non-human learning).

1.1 Learning in the Control Loop

Combining learning and control theory is not a novel concept. In fact, the fields of control, information and neural science were once considered an integral part of and studied under the interdisciplinary banner of cybernetics [13]. Since then, the individual disciplines gradually diverged into the fields as we know them today, which caused an unfortunate breakdown in communication between them. This separation has resulted in the co-development of different approaches to the same problems, largely due to differences in terminology and notations [13, 14].

However, with the explosive progression of machine learning, we are observing a notable shift in this trend, with researchers actively working to re-bridge the gap between the communities, see e.g. [14]. This newfound companionship is also reflected in the growing body of published works and literature on machine learning for control [15–18] and control for machine learning [19–22]. In this thesis, our primary focus will be on the former.

In essence, the overall goal of any control problem is to construct a controller (or regulator) for generating control inputs that will drive the system to a desired state [23]. In real-life applications, additional performance specifications must often be considered. Typically, we want to minimize the control error, that is, the discrepancy between the performed and desired system behaviour. Moreover, robustness against external external disturbances or system variations, as well as ensuring safety and stability, are often required. To meet these specifications, feedback control is commonly employed, where measurements of the system's state or output are fed back to the controller. This scheme of "closing the loop" holds many nice properties that can be exploited when designing control systems, see e.g. [23, 24].

A typical example of a feedback loop is shown in Figure 1.2. As we can see, the controller is just one of many essential components that form the closed-loop system. For instance, in a classical control problem, estimation techniques are typically employed to estimate the system's internal state or to identify a mathematical model describing the dynamics of the system. This process is known as system identification and plays an important part in the design process [25]. Furthermore, measurements signals are generally corrupted by internal and/or external noise. Feedback control, for example, has the potential disadvantage of introducing unwanted sensor noise into the system. To improve the quality of the signals, filtering and smoothing techniques are often applied. To solve these tasks, several approaches utilizing tools from machine learning have emerged alongside more traditional methods. For example, notable progress has been made in the development of learning-based techniques for system identification and estimation [26, 27], filtering and denoising [28], and controller design [29, 30].

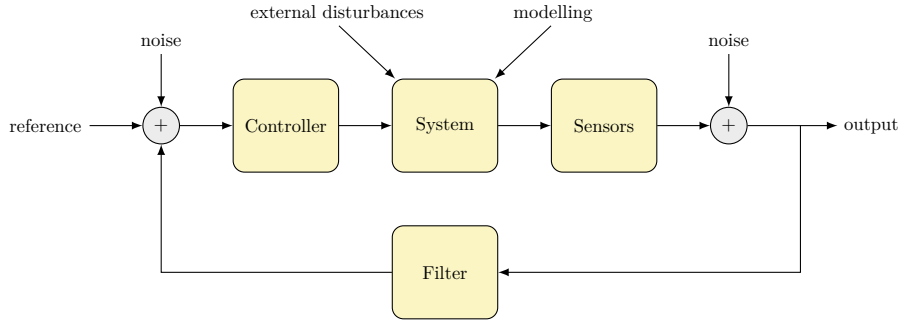


Figure 1.1: A traditional control loop with output feedback.

1.2 Learning for Controller Design

When it comes to designing and implementing controllers, there are many control paradigms and techniques to choose from, each offering its own approach to achieve desired system behaviour and performance. Among the most widely used ones we find *Proportional Derivative Integral (PID)* control; Optimal Control, including the *Linear Quadratic Regulator (LQR)* and the receding horizon approach *Model Predictive Control (MPC)*; *Adaptive Control*; and *Robust Control*. The choice of control strategy depends on factors such as performance requirements, system dynamics, as well as robustness and safety needs.

Over the last decades, various machine learning techniques have been applied in controller design, either in combination with or as replacements for conventional controllers. We call these strategies *hybrid analytics control systems* and *Machine Learning Control (MLC)*, respectively. ML-based controllers offers the potential for more efficient control strategies, improved system performance, and the ability to handle complex systems where traditional control methods fall short. For example, the inherent properties of *Neural Networks (NNs)* make them a natural fit for nonlinear and multivariable control problems [13, 31, 32].

ML-based controllers have found applications in various domains, including process control, robotics and automation, as well as aerial and under-water vehicles [32]. The controllers are typically based on supervised learning or *Reinforcement Learning (RL)* parardigms. In a supervised learning setting, the controller is trained using labeled data, where the correct control actions are known for given input states, see e.g. [32] and [33] for an overview. Reinforcement learning bears many similarities to adaptive control [34]. In RL, a controller is learned from trial and error, where it receives feedback from the environment in the form of rewards based on its actions. RL-based controllers have been successfully applied in a wide range of settings, including quadrotor control [35], power systems [36], and autonomous vehicles [8, 37].

Despite their advantages, ML-based controllers also face many challenges. Learn-

ing algorithms are in general very data-hungry, and require extensive datasets to be able to fully explore and capture the system's behaviour. Collecting such data can be both expensive and time-consuming, and in some cases even impractical or impossible. A major challenge then lies in generating sufficient training data for the algorithms to explore the entire state space.

In addition, ML-based controllers generally fail to provide stability and safety guarantees, as well as mathematical analysis and explanation, largely due to their probabilistic nature. This limitation is a significant disadvantage, especially when dealing with highly safety-critical systems. Privacy concerns also arise as the data-driven nature of ML introduce the potential risk of exposing sensitive or private information during both the data collection and the learning process.

These concerns are not exclusive to the control community. To address them, much effort has been put into the development of *interpretable* and *explainable* learning models, see e.g. [38] for an overview. The topic of differential privacy has emerged as a means to address privacy issues [39].

With this work, we contribute to the field through an investigation of the performance of different neural network-based model predictive controllers and how safety measures can be included in their learnable structure.

1.2.1 Neural Network-based Model Predictive Control

Most real-life systems, and safety-critical systems in particular, impose several constraints on the regulator design. These constraints are generally characterized by safety and performance specifications on the system's states, or by physical limitations on the actuators. Effectively handling such constraints is thus an important aspect of the control synthesis.

Model predictive control offers a significant advantage in constraint handling, which has contributed to its widespread use in e.g. automotive applications, chemical processes, power systems, robotics and aerospace [40]. The MPC is an optimization-based control strategy that relies on a mathematical model of the underlying system's dynamics. This model is used by the MPC to predict the future behaviour of the system based on its current state and the applied control inputs. The MPC paradigm involves computing an optimal control sequence at each time step by minimizing a cost function that reflects the control objective. The first input from this sequence is then applied to the system, after which the procedure is repeated over a receding horizon, enabling the MPC to continuously adapt the control input. As the number of states and constraints increases, it is easy to see how this procedure may quickly become computationally intractable.

A strategy for circumventing this issue is through offline pre-computation of the control law. This transforms the problem into that of specifying a mapping or lookup table in the form of a *Piece-Wise Affine (PWA)* function defined over partitions of the state space. This approach is known as the *Explicit MPC (eMPC)*, and has further inspired the viewing of the MPC as a general learning problem.

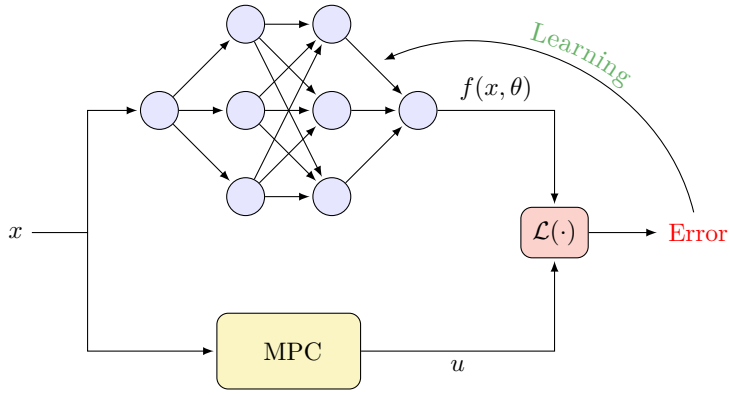


Figure 1.2: Visualization of learning the MPC, where x is a sampled state, u^* the corresponding optimal control action, $\mu(x, \theta)$ the control action generated by the network, and $\mathcal{L}(\cdot)$ the loss function comparing the two control actions.

As a result, a number of works involving the use of learning-based approaches have been proposed recently, primarily in the form of neural networks [41–43].

The popularity of neural networks in approximating the MPC can be attributed to their advanced features such as the *Rectified Linear Unit (ReLU)* activation function, the *Long-Short Term Memory (LSTM)* structure, and recently developed differentiable convex optimization layers [44, 45]. To some extent, these features each hold some properties akin to those of the MPC and the eMPC, making them an attractive choice for approximating MPC. Furthermore, recent work [46] shows how ReLU-based networks can be extended with a projection block using Dykstra's algorithm to achieve feasibility guarantees on the generated control inputs.

While the idea of approximating the MPC by neural networks is by no means novel, there are still many challenges with such approaches, both in terms of modelling, training and evaluation. In particular, it is not obvious how to efficiently generate training data for learning the control law as a mapping. Earlier work employ either grid-based methods or random smapling, neither which scales well to high dimensions. Moreover, it still remains unclear as to how different network structures can be evaluated consistently. Thus, we recognize a need for a standardized framework for treatment of such approaches from an experimental design point of view. In light of these challenges, we consider the following problems in this work.

Problem 1.1. *We examine how to train a neural network for implementing the MPC, using state as input and the control law as output. Our aim is to investigate how MPC-aware network structures compare against their model agnostic likes. For data generation, we study the use of an efficient Hit-and-Run sampler that scales well to high dimensions.*

Problem 1.2. *As an alternative to an online projection strategy, we propose an offline projection approach by incorporating a state of the art convex optimization layer into the network's learnable structure. In this way, we introduce MPC problem specific information into the network that will influence the network's weight update during the training process.*

Problem 1.3. *As an extension to our framework, we propose an approach for learning the MPC where we explicitly use structural information in the form of gradient data in the training process. This is motivated by the observation that recently proposed tools in differentiable convex optimization can provide both the MPC feedback law as well as its corresponding gradient.*

1.3 Learning for System Identification

The success of model-based control strategies similar to the MPC heavily relies on having an accurate and reliable model of the system being controlled. However, in many cases, obtaining such models directly from analysis of the system's physical characteristics is difficult, and we must resort to data-driven methods.

System identification is a classical topic in automatic control concerned with constructing and validating models of dynamical systems from observed data. It is a well-established field with several textbooks, see e.g. [25], software packages, and a wide range of applications, spanning from process industries to biomedicine [47]. Over the years, researchers have developed an extensive collection of identification methods for both parametric and non-parametric models, as well as for black-box and grey-box models.

The underlying problem of system identification and machine learning is mutual: inferring models from or determining mathematical relationships between input and output data. There are therefore several obvious links connecting the two fields, as discussed in [48] and [49]. In recent years, deep learning models have emerged as the new standard for modeling highly complex systems in many disciplines, see [9]. This is mainly attributed to their inherent ability to identify and capture patterns as a black-box model, which makes them particularly appealing when dealing with non-linearities, uncertainties and large-scale systems.

A typical system identification problem consists of four steps:

(1) *Experiment design*: for data collection.

(2) *Model selection*: to decide upon a set of model structures.

(3) *Parameter estimation*: to determine the values of the model structure's parameters that best align with the collected data.

(4) *Model validation*: to evaluate the model's accuracy and reliability.

Similar to the control problem, learning can be applied to different tasks within a system identification problem. In this thesis, we restrict our focus to parameter estimation, with an emphasis on cooperative estimation methods. In these methods, we leverage the power of collaborative learning to improve the estimation process when data is limited and/or unrepresentative of the underlying system characteristics.

1.3.1 Correctional Learning

In a parametric system identification scheme, the selected model set or structure can be defined by a set of finite-dimensional parameters that represent the *unknown* characteristics or properties of the system. These parameters can include means and variances, or other properties that define the underlying data-generating mechanism. The goal of parameter estimation is to determine the values of these parameters that best align with the observed data.

Popular estimation techniques include traditional methods such as maximum likelihood, maximum a posteriori (MAP), and Bayesian inference methods, as well as more recent learning-based approaches. Common to most of these estimators is their data-driven nature. However, in many real-world applications, the available data often does not accurately represent the underlying distribution or behavior of the system under study. This can be the result of limited sample sizes, biased sampling methods, measurement errors, or the presence of outliers [50]. Relying on such data when using data-driven estimators can result in inaccurate parameter estimation and poor model performance. For example, a recent study found that commercially available facial analysis algorithms showed higher error rates for darker-skinned individuals and women [51], which could be linked to unrepresentative training data. In addition, some existing methods rely on prior knowledge in the estimation process. This comes with its own set of limitations, as determining appropriate priors can be very difficult in practice.

To address these issues, *Correctional Learning* was recently developed to provide an alternative approach for incorporating external or prior knowledge into the estimation process [52]. It arises from the idea of cooperative learning problems, i.e., settings in which two or more agents work together towards a common goal. The framework is structured around a teacher-student model, where an expert (teacher) agent seeks to assist a learner (student) agent in its estimation process. More specifically, the teacher's goal is to modify (or correct) the observed data, based on which the student forms its estimation. See Figure 1.3 for an illustration of this process.

Correctional learning has shown promising results in an offline (batch) setting [52], in which the teacher modifies entire batches of observations at once. However, in many applications, observations arrive sequentially, which requires immediate (online) action, due to the need for a learning process that adapts and rapidly changes. Online algorithms often make learning faster and computationally cheaper. In light of this, we consider the following problem.

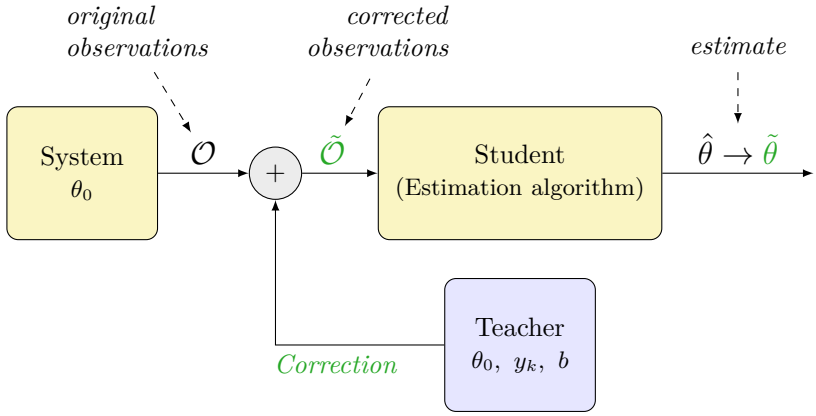


Figure 1.3: A schematic view of the correctional learning framework. The teacher knows the true parameter value θ_0 and the original samples x_i. The teacher modifies the original sequence of observations $\mathcal{O}$ into $\tilde{\mathcal{O}}$ by changing at most B samples.

Problem 1.4. *Consider the batch correctional learning framework. Our goal is to extend the idea to an online setting, where the teacher has to determine at each time instant how to modify the observed data. We formulate the online problem as a* Markov Decision Process (MDP) *and solve for the optimal strategy using dynamical programming.*

The correctional framework is further limited in its application in that the theoretical performance guarantees only hold for for simple systems. To describe real-world phenomena, we typically require more complex distributions. For example, a Gaussian distribution can be used to describe biological data such as the heights of people. To model the probability of failure of an appliance, we can use the Weibull distribution. Another disadvantage is that the teacher's policy follows explicitly from the solution, leaving no room for alternative intervention strategies to be considered. To apply correctional learning in more advanced settings, these limitations must be tackled. We therefore consider the following problem.

Problem 1.5. *In the context of correctional learning, we note that the optimal corrections can be viewed as a transportation of probability mass from an initial distribution into a target distribution. This interpretation has clear links to optimal transport, a mathematical framework concerned with determining the most efficient way of transporting mass from one location to another, according to some cost function. Using tools from optimal transport, our goal is to generalize the framework by expressing the problem in terms of probability measures, thereby enabling the estimation of more complex parameters and allowing for multiple intervention policies for the teacher.*

1.4 Thesis Outline

This section provides an overview of the content of this thesis. The work is divided into two main parts: learning-based MPC and cooperative system identification. Each chapter is summarized and accompanied by their respective contributions.

Chapter 2 (Preliminaries). In this chapter, we will introduce the notation that will be used throughout this thesis, as well as provide essential background material. We will briefly cover a range of concepts within the fields of control systems and machine learning, including constrained linear systems, optimal control strategies, system identification, neural networks, and Markov decision processes.

Chapter 3 (Neural Network-Based MPC). This chapter is devoted to our work on neural network-based MPC, where we investigate controller identification given data from a constrained MPC. We propose an approach for learning the MPC that explicitly incorporates gradient information during the training process. We leverage recent advances in differential convex optimization MPC solvers, which can provide both the optimal feedback laws as well as the corresponding gradients. As a proof of concept, we apply the approach to eMPC using neural networks with rectified linear units (ReLU), chosen for their structural similarities to eMPC. The motivation behind this work is to replace online solvers with neural networks to implement the MPC and to simplify the computation in larger input dimensions. In addition, we also study experimental design, model evaluation, and propose propose a hit-and-run sampling algorithm for input design. The proposed framework is illustrated and numerically evaluated on a second order MPC problem.

The chapter is based on the contribution:

- R. Winqvist, A. Venkitaraman, and B. Wahlberg, "Learning Models of Model Predictive Controllers using Gradient Data", 19th IFAC Symposium on System Identification (SYSID), 2021.

Chapter 4 (Online Correctional Learning). This chapter covers our work on extending the off-line correctional learning framework to an on-line setting. We consider the same underlying cooperative system identification problem: an expert (teacher) agent aims to assist a learner (student) agent in its estimation process by modifying the student's observed data. In the on-line setting, data samples are received sequentially, requiring the teacher to decide at each time instant whether to modify a sample or not, while staying within a predefined intervention budget. We formulate the problem as a Markov decision process and use dynamic programming to solve for the optimal policy. Through our approach, we show how the variance of the student's estimate can be reduced with the help of the teacher. The framework is validated through numerical experiments, and we also compare the on-line optimal policy to the established off-line optimal policy.

The chapter is based on the publication listed below. As a co-author, my contributions to the work included (partial) formulation of the Markov decision process, the numerical experimental design and validation, analysis, literature review, and manuscript writing.

- I. Lourenço, R. Winqvist, C. R. Rojas, B. Wahlberg, "A Teacher-Student Markov Decision Process-based Framework for Online Correctional Learning", IEEE 61st Conference on Decision and Control (CDC), pp. 3456-3461, 2022.

Chapter 5 (Optimal Transport for Correctional Learning). In this chapter, we present a generalized formulation of the off-line correctional learning framework by leveraging tools from optimal transport. Optimal transport is a mathematical framework concerned with finding the most efficient way to transport mass from one location to the other, according to some cost function. Motivated by the view of correctional learning as a means of transforming an initial probability distribution into a target distribution, we pose the correctional learning problem as an optimization program in terms of distribution functions – i.e, as an optimal transport problem. Compared to existing formulations of correctional learning, this novel approach provides several benefits. For example, it allows for the estimation of more complex characteristics as well as the consideration of multiple intervention policies for the teacher. Our approach is evaluated on theoretical examples and a human-robot interaction application in an inverse reinforcement learning setting. The results highlight the potential of the framework across different domains.

The chapter is based on the following contribution:

- R. Winqvist, I. Lourenço, F. Quinzan, C. R. Rojas, B. Wahlberg, "Optimal Transport for Correctional Learning" *Accepted to be presented at the IEEE 62nd Conference on Decision and Control (CDC) 2023.*

Chapter 6 (Conclusions and Future Work). In the final chapter, we summarize the main contributions of this thesis and discuss possible directions for future work.

Chapter 2

Preliminaries

This chapter serves the purpose of laying the necessary groundwork for understanding the concepts and ideas presented in the main part of this thesis. We begin with a brief introduction of constrained linear systems, which lie at the base of this research. We then delve into optimal control strategies, exploring their significance in addressing the challenges posed by constrained system. Furthermore, we provide an overview of system identification and parameter estimation, with a particular emphasis on recently developed cooperative learning-based techniques. Lastly, we present the concepts of Markov decision processes and optimal transport as vital tools for implementing the main ideas of this research.

2.1 Notation

The topics covered in this thesis span a wide range of areas. In order to stay consistent with the standard frameworks, the notation will shift accordingly throughout the two parts. To aid the reader in keeping track of the notation, a comprehensive nomenclature is provided in "Symbols & Notation" in the front matter. Whenever a symbol takes on a new meaning that is not immediately clear from the context, it will be explicitly redefined. Below we list some of the most frequently used notations.

The set of natural numbers (including zero) is denoted by $\mathbb{N}_0$, the set of non-negative natural numbers by $\mathbb{N}_+$, the set of (non-negative) real numbers by $\mathbb{R}$ ($\mathbb{R}_+$), the set of (non-negative) integers by $\mathbb{Z}$ ($\mathbb{Z}_+$). All vectors are column vectors, unless transposed, and inequalities between them are considered element-wise. For a vector v, we use v^T to denote its transpose, and $[v]_i$ to denote its ith element. The ℓ_1 and ℓ_2 norms are denoted by $\|v\|_1$ and $\|v\|_2$, respectively. The indicator function $\mathbb{I}(\texttt{expr})$ returns a 1 if the expression `expr` is fulfilled, and 0 otherwise. We let $\mathbb{1}$ denote the vector of ones, and I the identity matrix. The expected value of a random variable X is denoted by $\mathbb{E}[X]$, and its variance is defined as $\text{var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^T(X - \mathbb{E}[X])]$. We use $(\mathcal{Y}, \mathcal{B})$ to denote a measurable space,

in which $\mathcal{Y}$ is a set and $\mathcal{B}$ is a σ-algebra of subsets of $\mathcal{Y}$. We denote the set of positive measures on $(\mathcal{Y}, \mathcal{B})$ by $\mathbb{M}_+(\mathcal{Y})$. For a set $\mathcal{S}$, we use $\text{card}(\mathcal{S})$ to denote its cardinality.

2.2 Constrained Linear Systems

This section serves as a gentle introduction to constrained linear discrete-time systems and their structural properties. It also includes a brief section on set invariance, which formally introduces the concept of control invariant sets.

2.2.1 Modeling of Discrete-Time Systems

A linear, time-invariant discrete-time system can be described by the difference equations

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned} \tag{2.1}$$

where $x_k \in \mathbb{R}^n$ is the state vector of the system at time $k \in \mathbb{Z}_+$, $u_k \in \mathbb{R}^m$ the input vector, $y_k \in \mathbb{R}^p$ the output vector, $A \in \mathbb{R}^{n\times n}$ the dynamics matrix, $B \in \mathbb{R}^{n\times m}$ the control matrix, $C \in \mathbb{R}^{p\times n}$ the sensor matrix, and $D \in \mathbb{R}^{p\times m}$ the direct term (often set to 0) [23].

While such a model provides a good understanding of the underlying dynamics of a system, it is generally unable to capture the full complexity of most physical processes. For this reason, there will always be some deviation between the mathematical model in (2.1) and the real system it is representing. Exploiting feedback in the system control design is an important means for dealing with this uncertainty.

Most real-life systems are in addition subject to constraints on both the state and the input, defined on their most general form by

$$x_k \in \mathcal{X} \quad \forall k \in \mathbb{Z}_+ \tag{2.2a}$$

$$u_k \in \mathcal{U} \quad \forall k \in \mathbb{Z}_+, \tag{2.2b}$$

where the sets $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$ are polyhedra. The state constraints are generally imposed by safety and performance specifications or other behavioural requirements, while the input constraints, on the other hand, are characterized by actual physical limitations — meaning they will be enforced whether the synthesized controller manages to satisfy them or not. Handling constraints thus becomes an important aspect of regulator design; not only for safety reasons, but also because it generally holds that operating near constraint boundaries increases performance [53–55].

2.2.2 Structural Properties

The structural properties of a system are key concepts for understanding the synthesis of controllers with observers and state feedback. They describe how the

applied input affects the state vector as well as how the latter shows in the output, thereby providing means of quantifying both the ability to induce certain behaviour in the system as well as the ability to reconstruct the state from output measurements [23,56]. Below follow the definitions[1] of the structural properties of a system of the form (2.1).

Definition 2.1 (Controllability)**.** A system is said to be *controllable*, if for any pair of states (x, z) there exists an input sequence $\{u_0, u_1, \ldots, u_{T-1}\}$ such that z can be reached from x in a finite time T.

Definition 2.2 (Observability)**.** A system is said to be *observable*, if there exists a finite T such that the state x_T can be uniquely determined through the measurements of the input and output sequences $\{u_0, u_1, \ldots, u_{T-1}\}$ and $\{y_0, y_1, \ldots, y_{T-1}\}$.

A powerful tool for testing a system for controllability and observability is the Hautus lemma for discrete time systems, see e.g. [58].

Lemma 2.1. *A system is controllable, if and only if*

$$\operatorname{rank} \begin{bmatrix} \lambda I - A & B \end{bmatrix} = n, \quad \forall \lambda \in \operatorname{eig}(A).$$

Lemma 2.2. *A system is observable, if and only if*

$$\operatorname{rank} \begin{bmatrix} \lambda I - A \\ C \end{bmatrix} = n, \quad \forall \lambda \in \operatorname{eig}(A).$$

For a controllable system it holds that all eigenvalues (modes) of A can be modified by applying state feedback. Similarly, for an observable system it holds that all modes of A can be modified by applying output feedback. For uncontrollable (unobservable) systems, there are modes which cannot be modified using feedback. These are referred to as uncontrollable (unobservable) modes, and it is of interest to study their stability. The following weaker notions are therefore useful [54, 56, 59].

Definition 2.3 (Stabilizability, Detectability)**.** A system is *stabilizable*, if its uncontrollable modes lie within the stability region. It is said to be *detectable*, if its unobservable modes lie within the stability region.

The Hautus conditions for stabilizability and detectability are as follows.

Lemma 2.3. *A discrete-time system is stabilizable, if and only if*

$$\operatorname{rank} \begin{bmatrix} \lambda I - A & B \end{bmatrix} = n, \quad \forall \lambda : |\lambda| \geq 1. \tag{2.3}$$

Lemma 2.4. *A discrete-time system is detectable, if and only if*

$$\operatorname{rank} \begin{bmatrix} \lambda I - A \\ C \end{bmatrix} = n, \quad \forall \lambda : |\lambda| \geq 1. \tag{2.4}$$

It is easy to see that observability implies stabilizability, just as observability implies detectability. The reverse implications, however, do not hold.

[1] Definitions adapted from [23, 54, 57].

2.2.3 Set Invariance

The concept of set invariance is closely related with safety and feasibility of control systems. In fact, the existence of control invariant sets is often a fundamental step for solving the control synthesis problem in the presence of constraints [55, 60, 61].

In words, a control invariant set is defined as a set of admissible initial states in the state space for which there exists a control law such that the generated trajectory is kept within the set. More specifically, the set contains all initial states whose trajectories does not violate the system constraints. Below follow the more rigorous definitions from [61].

Definition 2.4 (Control invariant set)**.** *A set $\mathcal{C} \subset \mathcal{X}$ is said to be a control invariant set for the system* (2.1) *subject to the constraints in* (2.2) *if*

$$x_k \in \mathcal{X} \implies \exists u_k \text{ such that } x_{k+1} \in \mathcal{C}.$$

Definition 2.5 (Maximal control invariant set)**.** *The set $\mathcal{C}_\infty \subset \mathcal{X}$ is said to be the maximal control invariant set for the system* (2.1) *subject to the constraints in* (2.2)*, if it is control invariant and contains all control invariant sets contained in $\mathcal{X}$.*

From the definition, it follows that the maximal control invariant set is the largest set for which one can expect *any* controller to work [61]. An algorithm for computing $\mathcal{C}_\infty$ can be found in [61]. The procedure is also provided in Algorithm 1 for the reader's convenience[2].

Algorithm 1 Computation of $\mathcal{C}_\infty$

1: $\Omega_0 \leftarrow \mathcal{X}$, $k \leftarrow -1$
2: **repeat**
3: $\quad k \leftarrow k + 1$
4: $\quad \Omega_{k+1} \leftarrow \text{Pre}(\Omega_k) \cap \Omega_k$
5: **until** $\Omega_{k+1} = \Omega_k$
6: $\mathcal{C}_\infty \leftarrow \Omega_{k+1}$
7: **return** $\mathcal{C}_\infty$

Example 2.2.1. *Consider the second order system*

$$x_{k+1} = Ax_k + Bu_k = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} x_k + \begin{bmatrix} 2 \\ 1 \end{bmatrix} u_k, \tag{2.5}$$

subject to the constraints

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad -1 \leq u_k \leq 1, \quad \forall k. \tag{2.6}$$

[2]The precursor set of a set $\mathcal{S}$ is defined as: $\text{Pre}(\mathcal{S}) = \{x \in \mathbb{R}^n : \exists u \in \mathcal{U} \text{ s.t.} Ax + Bu \in \mathcal{S}\}$.

Using Algorithm 1, the related maximal control invariant set is computed as

$$\begin{bmatrix} -0.71 & -0.71 \\ -0.89 & -0.45 \\ 0.71 & 0.71 \\ 0.89 & 0.45 \\ -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} x \leq \begin{bmatrix} 4.24 \\ 4.02 \\ 2.24 \\ 4.02 \\ 5 \\ 5 \\ 5 \\ 5 \end{bmatrix}. \tag{2.7}$$

The set is also depicted in Figure 2.1.

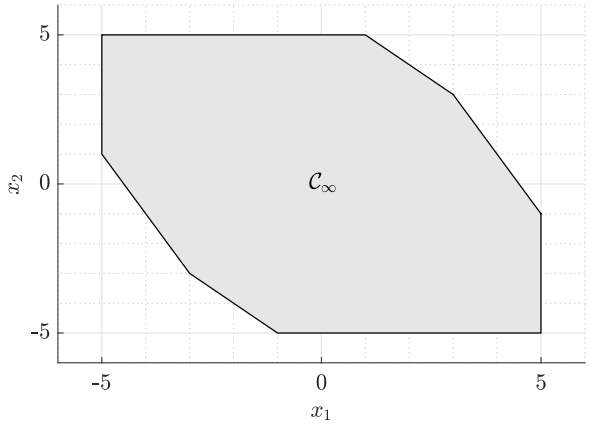


Figure 2.1: The maximal control invariant set, $\mathcal{C}_\infty$, for the system in (2.5) subject to the constraints in (2.6).

Remark 2.1. *It is important to note that Algorithm 1 will not necessarily terminate in finite time, and is thus not guaranteed to converge to the maximal control invariant set. In that case, we can define another safety set.*

2.3 Optimal Control Strategies

This section provides an introduction to the control principles considered in this thesis. It begins with a summary of the unconstrained and constrained *Linear-Quadratic Regulator (LQR)*, followed by a discussion on the principles of *Model Predictive Control (MPC)* and how it relates to the LQR. The chapter is concluded with a section on *Explicit Model Predictive Control (eMPC)* and an example of the same.

2.3.1 Linear Quadratic Control

Consider a discrete-time linear time-invariant system evolving in times as

$$x_{k+1} = Ax_k + Bu_k, \tag{2.8}$$

where, as before, $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ denote the state and input vectors at time k, respectively. Assume that the pair (A, B) is stabilizable.

The infinite-horizon *Linear Quadratic (LQ)* problem is then defined as the problem of finding a control input sequence $\{u_0, u_1, \ldots, u_\infty\}$ that will steer the system in (2.8) from some initial state, $x_0 = x(0)$, to the origin while minimizing the control cost function defined as

$$J_\infty = \sum_{k=0}^{\infty} (x_k^\top Q x_k + u_k^\top R u_k). \tag{2.9}$$

Here, $R \succ 0$ is a symmetric cost matrix penalizing the input, and $Q \succeq 0$ a symmetric cost matrix penalizing the state error, such that $(Q^{1/2}, A)$ is detectable. The solution to this problem results in the optimal control law known as the LQR given by[3]

$$u_k^* = -(B^\top P_\infty B + R)^{-1} B P_\infty A x_k = -L x_k, \tag{2.10}$$

where P_∞ solves the *Discrete Time Algebraic Riccati Equation* [56, 61]

$$P_\infty = A^\top P_\infty A - A^\top P_\infty B (R + B^\top P_\infty B)^{-1} B^\top P_\infty A + Q. \tag{2.11}$$

The cost function in (2.9) represents a trade-off between the control action and the control error (distance to the origin). A controller for which Q is set large relative to R will prioritize reaching the origin as quickly as possible over reducing the control action, and vice versa. A central aspect of LQR design is choosing appropriate values of Q and R, also known as tuning, which requires knowledge of the system under study. A common, practical choice is to set the matrices to be diagonal

$$Q = \begin{bmatrix} q_1 & & 0 \\ & \ddots & \\ 0 & & q_n \end{bmatrix}, \quad R = \begin{bmatrix} r_1 & & 0 \\ & \ddots & \\ 0 & & r_m \end{bmatrix}.$$

In this way, the diagonal weights will reflect the influence of each individual (squared) state and input on the total cost [23, 54].

The LQR formulation above can be extended to also consider systems subject to state and input constraints of the form

$$x_k \in \mathcal{X} \tag{2.12a}$$
$$u_k \in \mathcal{U}, \tag{2.12b}$$

[3] Reader is referred to [56, 61] for derivation.

where the sets $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$ are polyhedra. The problem is then reformulated as

$$\begin{aligned} \underset{u_{0:\infty}}{\arg\min} \quad & J_\infty = \sum_{k=0}^{\infty} (x_k^\top Q x_k + u_k^\top R u_k) \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, && \forall k \in \mathbb{Z}_+ \\ & x_k \in \mathcal{X}, && \forall k \in \mathbb{Z}_+ \\ & u_k \in \mathcal{U}, && \forall k \in \mathbb{Z}_+ \\ & x_0 = x(0), \end{aligned} \tag{2.13}$$

and is referred to as the infinite-horizon *Constrained Linear Quadratic Regulation (CLQR)* problem.

Due to the infinite number of decision variables and constraints included in the optimization, the CLQR is in general very difficult to solve. There exists some work on how to address the CLQR directly, including e.g. [62], [63] and [64], but one often resorts to approximate methods instead. Model predictive control is the most prevalent approximation scheme to date and is presented in the section below.

2.3.2 Model Predictive Control

Model predictive control is an online strategy for designing an infinite-horizon suboptimal controller in a receding horizon fashion described in the following. At each time instant k, an optimal control input sequence $\{u_k, u_{k+1}, \ldots, u_{k+N}\}$ is computed by solving a constrained optimization problem over a finite-time horizon N. The first input in the sequence, u_k, is then applied to the system and the same procedure is repeated at the next time instant, $k+1$, based on the evolved state to generate the next control sequence $\{u_{k+1}, u_{k+1}, \ldots, u_{k+N+1}\}$. Thus, the horizon recedes over time.

The general MPC problem is formulated as

$$\begin{aligned} \underset{u_{0:N-1}}{\arg\min} \quad & J(x_0) = x_N^T Q_N x_N + \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, && \forall k \in [0, N-1] \\ & x_k \in \mathcal{X}, && \forall k \in [0, N-1] \\ & u_k \in \mathcal{U}, && \forall k \in [0, N-1] \\ & x_N \in \mathcal{X}_\text{f}, \end{aligned} \tag{2.14}$$

where x_0 is the current state, N the prediction horizon, $Q_N = Q_N^\top \succeq 0$ the terminal cost matrix, and $\mathcal{X}_\text{f}$ the terminal state constraints. As with the cost matrices, it is not obvious how the prediction horizon should be chosen. A short horizon is preferred for computational simplicity. However, a longer horizon will in general yield more accurate trajectory predictions.

2.3.2.1 Explicit Model Predictive Control

One considerable disadvantage of MPC is the computational effort required for solving (2.14) online, which makes MPC challenging for very fast processes [65]. Explicit MPC is a strategy for circumventing this issue by pre-computing the control law offline using multiparametric programming techniques.

Given a polytopic set $\mathcal{X}$, for each $x \in \mathcal{X}$ the eMPC computes a piecewise affine mapping from x to u defined over q regions of $\mathcal{X}$. In this way, the eMPC performs a characterization of the control law akin to a look-up table: the only computation required online is then to determine what region the current state is in [65, 66].

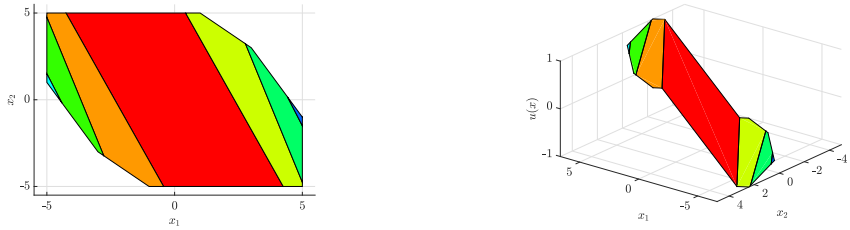
Unlike the control law of the MPC, the control law of the eMPC becomes explicitly dependent on x according to

$$u(x) = \begin{cases} F_1 x + g_1, & \text{if } H_1 x \leq b_1 \\ F_2 x + g_2, & \text{if } H_2 x \leq b_2 \\ \vdots & \\ F_q x + g_q, & \text{if } H_q x \leq b_q. \end{cases} \tag{2.15}$$

Here, F_i and g_i are the parameters of the local functions, and $H_i x \leq b_i$ denote the expressions that characterize the different regions of $\mathcal{X}$.

While the eMPC on the one hand reduces the computational burden in terms of solving for different states, it typically does not scale well with the dimension of the state due to nature of the function (2.15). The complexity of the solution further increases with the number of regions q, as this is what dictates the amount of memory required for storing the functions in (2.14) [65, 67].

Example 2.3.1. *Consider again the double integrator in* (2.5) *subject to the constraints in* (2.6)*. Let the cost parameters in* (2.14) *be defined as* $Q = I$, $R = 10$, *and* $Q_N = I$, *the horizon as* $N = 3$, *and the terminal constraints as* $\mathcal{X}_f = \mathbb{R}^2$. *Using the MPT3 [68] toolbox in Matlab to construct the eMPC, one obtains the controller that consists of the seven regions plotted in Figure 2.2a. The corresponding feedback law* (2.15) *is plotted in Figure 2.2b.*



(a) The seven regions of the eMPC.

(b) The PWA feedback law.

Figure 2.2: Visualization of the computed eMPC in Example 2.3.1.

2.4 System Identification and Estimation

In Section 2.2, we introduced the use of a set of difference equations[4] to model the behaviour of systems. Such mathematical models may be constructed using one of two (or the combination of both) approaches: *modeling* or *system identification*. Modeling involves dividing the whole system into subsystems whose properties and physical characteristics are well understood from earlier empirical work. These subsystems are then mathematically combined to form a model of the complete system. Thus, the procedure of modeling does not necessarily require experimentation on the actual system, but instead relies heavily on identifying appropriate subsystems and utilizing existing knowledge about them.

System identification, on the other hand, is directly based on experimentation. It involves collecting and analyzing input and output data from the physical system to infer a model. As mentioned in Chapter 1, a system identification procedure typically involves the following four steps:

(1) *Experiment design*: designing an experiment to collect data (observations), $\mathcal{O}$, from the system.

(2) *Model selection*: choosing a model structure, $\mathcal{M}$, to characterize the system.

(3) *Parameter estimation*: determining the values of the model structure's parameters that best align with the collected data, using the identification method $\mathcal{I}$.

(4) *Model validation*: design a validation procedure to evaluate the accuracy and reliability of the model.

The process of identifying a model is often iterative, and may involve revising or adjusting some of the choices during the course of the procedure.

2.4.1 Parameter Estimation

An important part of the identification process is to select a good model structure. Once a model structure has been selected, it can be defined by a parameter vector $\theta \in D_{\mathcal{M}} \subset \mathbb{R}^d$ that represents the *unknown* quantities of the system. The collection of these parameterized candidate models is called the *model set* and is defined as

$$\mathcal{M}^* = \{\mathcal{M}(\theta) \mid \theta \in D_{\mathcal{M}}\}, \tag{2.16}$$

where $D_{\mathcal{M}}$ is the set of values over which θ ranges in a model structure $\mathcal{M}$. The identification problem of finding the best model within this set then becomes a problem of determining or estimating the value of θ that best align with the experimental data.

[4] We use differential equations to model continuous systems.

There are many established approaches for parameter estimation in literature. For a comprehensive understanding and a guide on how to apply them in practice, the book [25] is an excellent resource on the topic that covers a wide range of methods. In this work, we will not consider any of the traditional estimation techniques, and interested readers are encouraged to explore external literature such as [25, 47, 69] for more details.

2.5 Cooperative System Identification

A common trait of traditional parameter estimators is their data-driven nature. However, in many real-world applications, the available data often does not accurately represent the underlying distribution or behavior of the system under study. This can be due to limited sample sizes, biased sampling methods, measurement errors, or the presence of outliers [50]. Relying on such data can lead to inaccurate parameter estimation and poor model performance. On the other hand, existing methods that incorporate prior knowledge into the estimation process also come with a set of limitations, as determining appropriate priors can be very difficult in practice.

Correctional Learning is a recently developed off-line framework designed to address these issues, by providing an alternative approach for incorporating external or prior knowledge into the estimation process [52]. The framework is structured around a teacher-student model, where an expert (teacher) agent seeks to assist a learner (student) agent in its estimation process by transferring its knowledge in the space of induced probability distributions.

Consider a standard system identification problem. The student collects a sequence of N *Independent and Identically Distributed (i.i.d.)* observations $\mathcal{O} = \{y_k\}_{k=1}^{N}$ from the system, where each observation y_k belongs to the observation space $\mathcal{Y}$. Assume that the system can be described by some parameterized model $m_0(\theta_0) \in \mathcal{M}$, where θ_0 represents the true model parameters. Each observation is sampled according to the distribution

$$y_k \sim p(y \mid y_{k-1}, \dots, y_1; \theta_0), \tag{2.17}$$

for $k = 1, \dots, N$. The student estimates θ_0 (the model) by solving the optimization problem

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} F(m(\theta), \mathcal{O}), \tag{2.18}$$

Here, $F(m, \mathcal{O})$ is a measure of fit of the student's model m, typically the likelihood-function of the observed data.

The teacher agent knows the true parameter θ_0. However, it might be infeasible or undesirable for the teacher to directly transfer this knowledge to the student. For example, the teacher's information might be too abstract or too complex, or it could be due to inconsistencies between the two agent's models or parameterizations, or

privacy concerns. To overcome this limitation, the two agents operate in the space of induced probability distributions instead.

Based on the observations $\mathcal{O}$, the student computes an empirical estimate of p as $\hat{p}(y)$, and finds the optimal parameter estimate by solving

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} G(\hat{p}, p_\theta). \tag{2.19}$$

Here, where G is a distance measure between two probability density functions, and p_θ the distribution induced by the model. The teacher, who knows the true distribution $p_0(y)$ of the data, aims to improve the students estimate $\hat{p}$, and, consequently, also $\hat{\theta}$, by intercepting and altering (i.e., *correcting*) the observations collected by the student. This results in the modified dataset $\tilde{\mathcal{O}} = \{\tilde{y}_k\}_{k=1}^N$ that better reflects the true distribution, along with the corresponding probability estimate $\tilde{p}$. The goal is to provide the student with information that leads to an altered estimate $\tilde{\theta}$ that either lies closer to the true estimate θ_0

$$\|\tilde{\theta}_N - \theta_0\| \leq \|\hat{\theta}_N - \theta_0\|, \tag{2.20}$$

or converges to the true parameter faster, i.e.,

$$\operatorname{var}(\tilde{\theta}) < \operatorname{var}(\hat{\theta}). \tag{2.21}$$

To model the communication restrictions, the teacher is assigned an intervention budget, denoted by b, which imposes a constraint on the teacher's actions. This budget constraint is expressed as follows:

$$B(\mathcal{O}, \tilde{\mathcal{O}}) \leq b. \tag{2.22}$$

Here, $B(\cdot, \cdot)$ represents a distance measure between two sets. In the case of discrete observations, B can be defined as the ℓ_1-norm

$$\frac{1}{N} \sum_{k=1}^{N} |y_k - \tilde{y}_k| \leq b. \tag{2.23}$$

This formulation implies that the budget restricts the number of samples that the teacher is allowed to modify.

The goal of any identification problem is to minimize the discrepancy between the constructed model and the true system. The correctional learning problem can therefore be formulated as the optimization problem

$$\begin{aligned} \min_{\tilde{\mathcal{O}}} \quad & V(p_0, \tilde{p}) \\ \text{s.t.} \quad & \tilde{y}_k \in \mathcal{Y}, \quad \forall \tilde{y}_k \in \tilde{\mathcal{O}} \\ & B(\mathcal{O}, \tilde{\mathcal{O}}) \leq b. \end{aligned} \tag{2.24}$$

Here, V is a statistical distance measure between two probability density functions, such as the KL-divergence. The first constraint ensures that the modified samples $\tilde{y}_k$ remain within the observation space $\mathcal{Y}$, preventing the teacher from arbitrarily modifying the samples. The second constraint guarantees that the distance between the original and modified datasets satisfies the specified budget constraint.

2.5.1 Influence of the Budget

For finite distributions with the ℓ_1-norm as the distance measure in (2.24), the following results regarding the errors have been established [52].

The smallest attainable error, as a function of the number of observations N and the true parameter θ_0, is given by

$$e_{\min}(N, \theta_0) = \left\| \theta_0 - \frac{[\theta_0 N]}{N} \right\|_1 . \tag{2.25}$$

Here, the notation $[\cdot]$ represents rounding to the closest integer value while adhering to the constraint $\mathbb{1}^T \theta_{\min} = 1$, where $\theta_{\min} = \frac{[\theta_0 N]}{N}$.

The estimation error is defined as the difference between the corrected parameter $\tilde{\theta}$ and the true value θ_0. It can be expressed as a function of the number of observations N, the true parameter θ_0, the budget b, and the original estimate as

$$e(N, \theta_0, b, \hat{\theta}) = \max \left\{ \|\theta_0 - \hat{\theta}\|_1 - \frac{2b}{N}, e_{\min} \right\}, \tag{2.26}$$

where $e_{\min}$ is defined as in (2.25).

2.6 Neural Networks

This section introduces the reader to the basic concepts of neural networks necessary for understanding this work. It will start off by describing the neurons, the computing elements of the network, and then move on to discuss different network structures and the supervised learning paradigm. The chapter ends with an introduction to convex optimization layers, a state-of-the-art network architecture for integrating differentiable optimization programs in the network structure.

2.6.1 The Neuron

The fundamental building blocks of a neural network are the processing units known as neurons. A neuron represents a mathematical function that generates an output signal based on inputs received from other neurons in the network. The neurons communicate by transmitting signals across their connecting links, each assigned a weight coefficient, w. In mathematical terms, this implies that the signal going from a neuron i to a neuron j will be multiplied by the weight w_{ij} of the link connecting the two.

The operation realized by the neuron is given by

$$y_j = \varphi\left(\sum_{i=1}^{l} w_{ij}x_i + b_j\right), \tag{2.27}$$

where x_i is the input signal coming from the ith neuron, l the number of incoming signals, b_j the bias, $\varphi(\cdot)$ the activation function, and y_j the generated neuron output. Together, the weights and biases form the learnable parameters that are adjusted by the network during training. A visual representation of a neuron is provided in Figure 2.3.

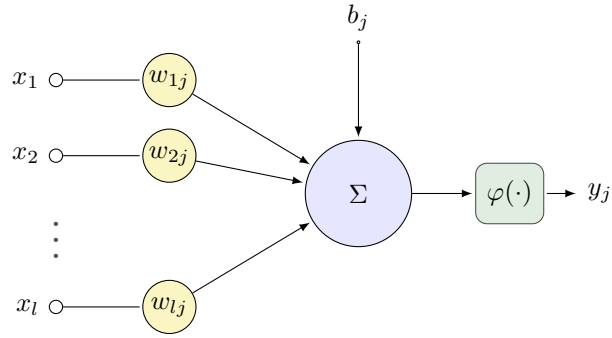


Figure 2.3: Visual representation of a neuron. The link weights are labeled as w_{ij}, while the bias is denoted as b_j. The activation function is denoted by $\varphi(\cdot)$. Figure adapted from [70].

2.6.2 Activation Functions

The activation function determines the output of a neuron and plays a fundamental role in the neural network. The non-linearity in the form of activation functions appearing in multiple is key to the remarkable performance of neural networks in many real-world tasks. From (2.27), it is easy to see that by omitting the activation function, the neural operation is reduced to that of a linear transformation. The same happens when the activation function is chosen to be linear. The common practice is therefore to employ non-linear activation functions, as these enable the integrated network to learn and process more complex data patterns [70–72].

In addition to being non-linear, an activation function should also be differentiable and monotonic. This is because training of neural networks typically proceeds in the form of backpropagation, which relies on gradients of the network parameters. Popular choices include the logistic sigmoid function, tanh and the ReLU function shown together in Figure 2.4.

The sigmoid is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.28}$$

and is typically used in probabilistic classifiers, as it restricts its outputs to lie in the range $[0, 1]$. The tanh relates to the sigmoid by $\tanh(x) = 2\sigma(2x) - 1$ and has the mathematical form

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.29}$$

Both the sigmoid and the tanh are saturating functions. This means that their gradients will approach zero for very small and very large values of x. In gradient-based learning, where the parameter update is proportional to the gradient of some error or loss function, this phenomenon may induce an issue known as the vanishing gradient problem. Should it occur, it may prevent the weights from updating and could thereby stop the network from learning altogether [73–75].

The *Rectified Linear Unit (ReLU)* function defined as

$$\mathrm{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0, \end{cases} \tag{2.30}$$

does not suffer from this issue. Its gradient will, however, evaluate to zero for negative values of x, causing the ReLU to "die" and output zero (0) for all future inputs[5]. Nevertheless, because the ReLU function is unbounded, the set of inputs for which the gradient is nonzero is still larger compared to those of the tanh and sigmoid. Furthermore, whenever the gradient is nonzero for the ReLU, it is always one, whereas for the sigmoid and tanh, the gradient lies in the range $[0, 1]$. This adds to the computational efficiency of the ReLU.

Another advantage of the ReLU activation is its computational simplicity. As opposed to the sigmoid and tanh, the ReLU does not involve any exponential operations in its computation, which makes it significantly more efficient in comparison. Because of this, the ReLU has become the most popular choice of activation function in most network-based applications [75–77].

2.6.3 Feed-forward Neural Networks

In its most general form, a neural network is a weighted, directed graph comprised of neurons ordered into different layers. Depending on the interconnection pattern of these layers, one distinguishes between the two main network structures (architectures): *Feed-Forward Neural Networks (FFNN)* and *Recurrent Neural Networks (RNN)*.

In a feed-forward neural network, the layers are sequentially connected without any feedback loops. Each neuron of a particular layer is unidirectionally connected to the neurons of the subsequent layer, thereby restricting the information to flow only in the forward direction. The simplest form of a feed-forward network is the single-layer FFNN consisting only of an input layer and an output layer. The

[5] This is known as the dying ReLU problem and can be avoided by implementing a leaky ReLU.

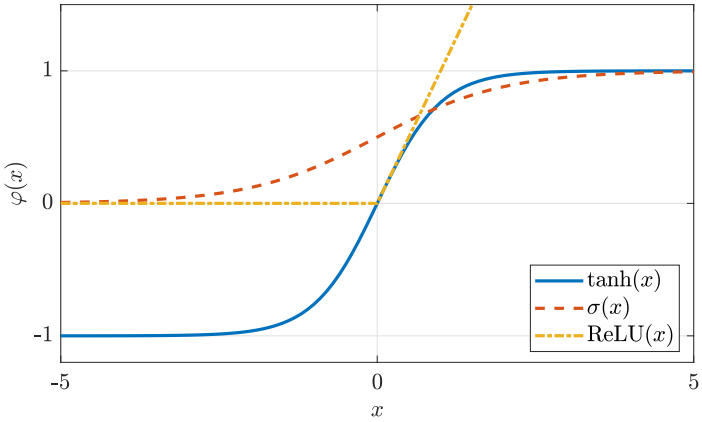


Figure 2.4: The plot shows three common activation functions: the tanh, sigmoid and ReLU.

designation "single-layer" stems from the input layer not being considered as a layer, as it merely feeds the inputs to the output layer without performing any computation. All processing is thus limited to the output layer.

A more complex feed-forward network is the multi-layer FFNN, which incorporates one or more hidden layers in its structure. The hidden layers are inserted between the input and output layers to induce nonlinearities in the network, which in turn enables the network to extract higher order features from data [70–72]. It is known that with a low number of layers the performance typically saturates after a point, and that increasing the number of layers often adds to the performance. The general structure of an FFNN is shown in Figure 2.5.

2.6.4 Supervised Learning

Supervised learning is the most common form of learning and refers to paradigms in which the network is provided a set of labeled input-output pairs to train on. For a given input from the training set, the network generates an output to be compared with the known desired output value, using an error metric. This error metric, or loss function, averaged over all the training samples is then used to train the network model. That is, the network parameters are chosen to minimize the loss function.

2.6.5 Backpropagation Algorithm

The backpropagation algorithm is a technique used in supervised learning of feed-forward networks that utilizes a gradient descent method to minimize a loss function

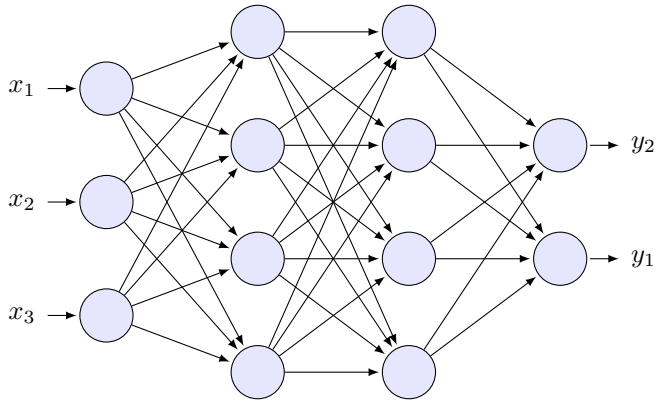


Figure 2.5: An example of a fully connected FFNN with two hidden layers. The weights and biases have been omitted for simplicity.

with respect to the network's parameters. For brevity, this section presents a simplified version in Algorithm 2, while the full mathematical derivation can be found in [71, 72].

Algorithm 2 Backpropagation algorithm (simplified)

repeat
  for (x_i, y_i) in the training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ **do**
    Apply x_i to the network;
    Propagate x_i through the network to obtain the prediction $\hat{y}_i = f(x_i)$;
    Compute the loss; $\mathcal{L}_i$, by comparing $\hat{y}_i$ to y_i,
    Backpropagate $\mathcal{L}_i$ by computing the gradients, $\nabla\mathcal{L}_i$, w.r.t. the weights and biases,
    Update weights and biases to minimize $\mathcal{L}_i$.
  end for
until error is below a specified threshold.

2.6.6 Differentiable Convex Optimization Layers

In certain scenarios, convex objectives are integrated into the neural network architecture. In these cases, the process of updating the neural network parameters requires the computation of gradients of the convex problem. As a result, there arises a need to transform the convex problems into differentiable ones, so that backpropagation can be applied for training.

Recent work on neural network architectures has introduced a framework for embedding differentiable optimization problems as individual layers within a deep, feed-forward network as a means for introducing domain-specific knowledge to its

structure. This subsection will give a brief introduction to the cvxpylayers [45], a Python library for implementing differentiable convex opitmization layers using CVXPY [78], which is a Python-embedded modeling language for convex optimization.

The cvxpylayers requires the optimization program to be constructed by the grammar *Disciplined Parametrized Programming (DPP)* introduced in [45]. DPP can be viewed as a subset of the modeling methodology disciplined convex optimization, which is used for constructing convex optimization models. The DPP consists of a collection of functions with known curvature (affine, convex or concave) and per-argument monotonicities, and a composition ruleset that will guarantee the convexity of the constructed program [45, 79].

Using DPP to construct the expressions, a disciplined parametrized program can then be formulated as an optimization problem of the form

$$\begin{aligned} \arg\min_{x} \quad & f(x,\theta) \\ \text{s.t.} \quad & g_i(x,\theta) \leq \tilde{g}_i(x,\theta), \qquad \forall i \in [1, m_1] \\ & h_j(x,\theta) = \tilde{h}_j(x,\theta), \qquad \forall j \in [1, m_2]. \end{aligned} \tag{2.31}$$

Here, $x \in \mathbb{R}^n$ is a variable, $\theta \in \mathbb{R}^p$ is a learnable parameter vector, the g_i are convex, $\tilde{g}_j$ are concave, and h_i and $\tilde{h}_i$ are affine functions. The parameter θ is learned to minimize some scalar function of the solution x^* of (2.31).

The cvxpylayer solves the problem in (2.31) by first canonicalizing it into a cone program of the form

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{s.t.} \quad & b - Ax \in \mathcal{K}, \end{aligned} \tag{2.32}$$

where $x \in \mathbb{R}^n$ is the variable, $\mathcal{K}$ a non-empty, closed and convex cone, and A, b and c are the matrices and vectors of appropriate sizes denoting the cone problem data. Next, it calls a cone solver $s_{\mathcal{K}}$, to obtain the solution

$$\tilde{x}^* = s_{\mathcal{K}}(A, b, c), \tag{2.33}$$

which is mapped to a solution x^* of the original problem. The solution map $\mathcal{S} : \mathbb{R}^p \to \mathbb{R}^n$ can thus be expressed as

$$\mathcal{S} = R \circ s_{\mathcal{K}} \circ C, \tag{2.34}$$

where R is the retriever that maps (2.33) to x^*, and C the canonicalizer. The derivation of the derivative of (2.34) required for the backpropagation algorithm has been left out for simplicity, but can be found in [45].

2.7 Markov Decision Processes

In many real-life scenarios, the evolution of a system cannot be predicted with certainty. This is often observed in phenomena such as stock market prices, the

spread of a disease, and traffic flow. To describe the behaviour of such systems, we use *stochastic processes*.

A discrete-time stochastic process is a collection $\{X(t)\}_{t\in T}$ of random variables defined on a common probability space. That is, for every time index[6] $t \in T$, $X(t)$ is a random variable that takes on values in the sample space $\mathcal{X}$. The sample space, also known as the state space, is the set of all possible configurations of the system. In this thesis, we limit ourselves to finite or countable state spaces, which we denote as $\mathcal{X} = \{1, \ldots, X\}$. However, it is worth noting that infinite or continuous state spaces have also been studied in literature.

A stochastic process is said to satisfy the *Markov property* if the current state of the system depends only on its immediate previous state. In other words, the probability of moving into the new state x_{t+1} is determined solely by the current state x_t. These transition probabilities are typically stored in the transition matrix $P \in \mathbb{R}^{X\times X}$, whose elements represent the probabilities of moving between different states in the system as

$$\begin{aligned}[P_t]_{ij} &= \Pr[x_{t+1} = j \mid x_t = i, x_{t-1} = l, \ldots, x_0 = z] \\ &= \Pr[x_{t+1} = j \mid x_t = i],\end{aligned} \tag{2.35}$$

where $i, j, l, z \in \mathcal{X}$, $0 \leq [P_t]_{ij} \leq 1$, and $P\mathbb{1} = \mathbb{1}$.

A Markov chain is fully autonomous, meaning its evolution is independent of any external inputs. However, real-life scenarios often involve applying different actions to a system in an attempt to influence its behaviour. A *Markov Decision Process (MDP)* is a mathematical framework for modeling such decision-making by including a decision-maker's actions in the transition probabilities. By considering external input, the MDP extends the concept of a Markov chain, transforming it into a *controlled* stochastic process.

An MDP is typically characterized by a four-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where

- $\mathcal{S}$ denotes the state space;
- $\mathcal{A}$ denotes the action space containing the set of actions the decision-maker can apply to the system;
- P denotes the transition probability matrix, and
- R denotes the reward function that captures the desired behaviour of the system.

At each time step t, the system resides in a certain state $s_t \in \mathcal{S}$. The decision-maker may choose to perform any action $a_t \in \mathcal{A}$ available in s_t. In the next time step, the system then responds by evolving into the new state s_{t+1} and providing the decision-maker with the corresponding reward r_t. The new state is determined

[6] This is a slight abuse of notation. Usually, k is used to denote discrete time in systems and control. However, to stay consistent with the standard frameworks, we use t instead.

by the transition probability matrix, P, which follows a similar structure as in (2.35) but now incorporates the chosen action:

$$[P_t(a)]_{ij} = \Pr[s_{t+1} = j \mid s_t = i, a_t = a], \tag{2.36}$$

where $i, j \in \mathcal{S}$, and $a_t \in \mathcal{A}$. Note how the Markov property remains satisfied, since the transition probabilities depends solely on the current state and action.

The goal in an MDP is to find an optimal mapping from the state space to the action space such that the expected cumulative reward is maximized. That is, we want find the optimal *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [1, 0]$ that specifies which action the decision-maker should apply in state s. The corresponding optimization problem is formulated as

$$\pi^* \in \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{N} r_t(s_t, a_t) \;\middle|\; s_0 = s\right], \tag{2.37}$$

where N is the horizon length.

MDP's are often solved using dynamic programming techniques, such as *value iteration* or *policy iteration*, which rely on the assumption that the underlying four-tuple is known.

Remark 2.2 (Notation)**.** *To stay coherent with the standard MDP framework, note that we here made the notational changes: $x \to s$, $u \to a$ and $k \to t$.*

2.8 Optimal Transport

Suppose that we are given a probability measure $\mu \in \mathbb{M}_+(\mathcal{X})$, which can be interpreted as, say, a distribution of sand in $\mathcal{X}$ of total mass 1. Assume further that we wish to transform μ into another probability measure $\nu \in \mathbb{M}_+(\tilde{\mathcal{X}})$, corresponding to a different distribution of sand, by "moving" the grains of sand with minimal transportation cost. The cost of transporting one unit of mass from location x to location $\tilde{x}$ is quantified by a metric $c(x, \tilde{x})$ on $\mathcal{X}$. To compute the total transportation cost, we must also define a transportation map, which is modelled by a probability measure $\tau \in \mathbb{M}_+(\mathcal{X} \times \tilde{\mathcal{X}})$, with $d\tau(x, \tilde{x})$ denoting the amount of mass transferred from x to $\tilde{x}$. Since we cannot move more mass than what we originally have, it must hold that the mass moved from one point in μ must be received by ν and vice versa. In mathematical terms, we express those conditions by

$$\int_{\tilde{\mathcal{X}}} d\tau(x, \tilde{x}) = d\mu(x) \quad \text{and} \quad \int_{\mathcal{X}} d\tau(x, \tilde{x}) = d\nu(\tilde{x}), \tag{2.38}$$

respectively.

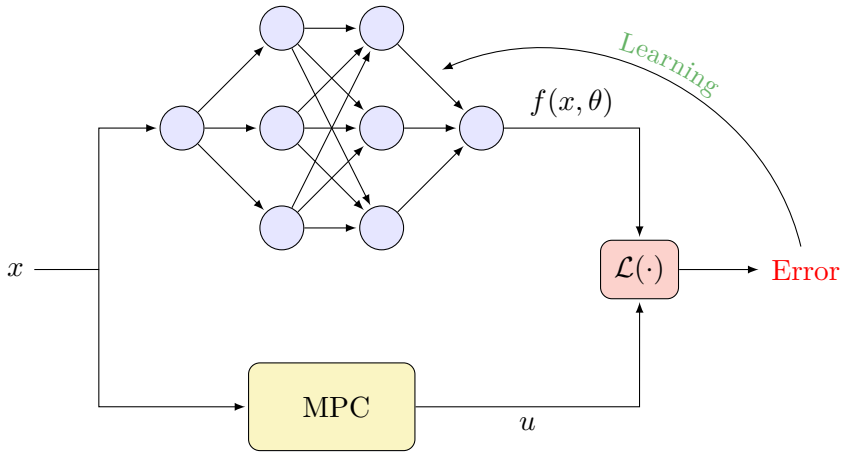
The problem can now be posed as the following optimization program, known as Kantorovich's optimal transport problem:

$$\begin{aligned} \min_{\tau \in \mathbb{M}_+(\mathcal{X} \times \tilde{\mathcal{X}})} \quad & \int_{\mathcal{X} \times \tilde{\mathcal{X}}} c(x, \tilde{x}) d\tau(x, \tilde{x}) \\ \text{s.t.} \quad & \int_{\tilde{x} \in \tilde{\mathcal{X}}} d\tau(x, \tilde{x}) = d\mu(x) \\ & \int_{x \in \mathcal{X}} d\tau(x, \tilde{x}) = d\nu(\tilde{x}). \end{aligned} \tag{2.39}$$

Here, the double integral in the objective corresponds to the total transportation cost under the transport plan τ. This problem provides a relaxation of the original formulation by Monge and has various applications in optimal transport theory. For more details, the interested reader is referred to [80–82].

Part I

Learning Model Predictive Control



Chapter 3

Neural Network Approaches for Model Predictive Control

One of the challenges in the design of feedback control of safety critical systems is to ensure that the closed loop system always satisfies a set of given specifications. This is well understood when using classical control concepts as *Proportional Integral Derivative (PID)* and *Linear Qudratic Regulator (LQR)* control. It is less understood when it comes to constrained *Model Predicitve Contol (MPC)*, which is usually implemented using online optimization. While MPC has gained significant popularity in various industries, the lack of rigorous methods for verification of MPC often becomes a deterring factor against its widespread adoption, despite its great potential. This in turn has prompted extensive research efforts in certified real-time optimization for MPC over the recent years, see e.g. [83] for an overview.

Moreover, the control community has experienced a renewed interest in learning-based MPC controller design, thanks to the increasing availability of computational power and advancements in sensing and communication capabilities. Various machine learning techniques are being readily applied in data-driven adaptions, with the aim to achieve improved performance, simplified deployment, and a reduced need for manual controller tuning [17].

In this chapter, our main focus lies in exploring the use of neural networks for approximating MPC. By leveraging these techniques, we aim to exploit the growing body of resources in machine learning to address some of the challenges associated with traditional MPC approaches.

3.1 Introduction

The process of controller identification is an extensively studied research area focused on estimating a model of a feedback controller from observed input and output data. Traditionally, this field has primarily dealt with cases where the system and the controller can be described by linear dynamical models, see e.g. [25]. However,

in recent years, there has been a growing interest in using neural network-based learning models for identifying constrained MPCs [41–43, 60, 84–86]. Indeed, the idea of using neural network-based MPC is not new, with early contributions dating back to [87] and [88].

As detailed in Subsection 2.3.2, the MPC is commonly formulated as a quadratic cost function subject to linear constraints, which encompass both system dynamics along with state and input constraints. Finding the optimal solution to such a constrained quadratic program typically involves identifying the active constraints, which can be achieved through various methods. See e.g. [89] for results on fast online MPC implementations. Nevertheless, computing the MPC control law entails solving an optimization problem at each time step, which could quickly introduce computational bottlenecks when applied to systems repeatedly.

To overcome this challenge, an alternative approach is to pre-compute the control law offline using multiparametric programming techniques. This transforms the problem into specifying a mapping or lookup table in the form of a PWA function, which, in turn, generates the optimal control law based on the input state. This approach is known as the eMPC [66, 90], and was previously introduced in Subsection 2.3.2.1. The characteristic structure of the eMPC has further inspired the viewing of the MPC as a general learning problem. As a result, a number of works, mainly employing neural networks, have been proposed recently [41–43, 60].

A driving factor behind these approaches is the recent discovery that neural networks employing *Rectified Linear Units (ReLUs)* can effectively represent *Piece-Wise Affine (PWA)* functions or functions with linear regions [91]. This property makes such networks well-suited for the MPC learning problem [41], especially in the context of *Explicit Model Predictive Control (eMPC)*, where the optimal control law is an affine function of the state vector [65]. Recent work on the topic demonstrates how to construct a set of ReLU networks to exactly represent an eMPC control law [84], while another approach explores linking a ReLU network to an eMPC via multiparametric programming techniques [85].

However, learning-based approaches, particularly those based on neural networks, face many challenges both in terms of modeling, training and evaluation. Therefore, there is a growing need for a standardized framework for treatment of such approaches from an experimental design point of view. As a response to this, the first part of this chapter focuses on the following key problems:

Problem 3.1 (Structure of the neural network)**.** *We study the training of neural networks for learning linear MPC. We explore different network structures, ranging from model-agnostic black box neural networks to architectures that are specifically designed to include structural information about the MPC problem. This allows us to compare the performance of structurally-aware networks with completely black box networks.*

Problem 3.2 (Nature of the data generation)**.** *Efficiently generating training data for learning the control law is not trivial, especially when dealing with high-dimensional systems. Traditional grid-based approaches for sampling the input*

space would become cumbersome in such cases. Keeping this in mind, we study the use of an efficient and statistically motivated Hit-and-Run (HAR) *sampler that extends well to higher state dimensions.*

Problem 3.3 (Evaluation of performance)**.** *Various constrained neural networks approaches have recently been proposed in the literature for solving the MPC problem [41, 42, 60]. However, it generally remains unclear as to how these different structures can be evaluated consistently. Here, we study how different neural network approaches compare in terms of different metrics.*

The primary objective of any learning-based approach for MPC is to effectively learn a mapping from the state to the control input that describes the MPC control law. In the context of neural network-based methods, this mapping corresponds to the function learned by the network. Most existing neural network-based learning approaches typically rely on supervised learning, where the mapping is learned using only observations of the system's state and control input. However, like any learning approach, an MPC mapping learnt in such a manner could perform poorly when the training data is scarce. In such scenarios, additional structural information can often be of merit in aiding the training of the network. This is particularly relevant for learning the eMPC, where the control law is a PWA function of the state. The gradient of the optimal control law with respect to the state is then piece-wise constant and contains important structural information for effective learning. Motivated by this observation, we formulate the central problem addressed in this chapter:

Problem 3.4. *We propose a neural network-based learning approach for MPC that explicitly leverages structural information in the form of gradient data during the training process.*

Although training data in the form of gradient information is typically unavailable or difficult to obtain, the unique structure of the MPC problem, coupled with recently proposed tools in differentiable convex optimization [44, 78], enable us to overcome this limitation.

3.2 Related Work

In this section, we will discuss the state-of-the-art in the field at the time this work was initiated. By understanding the prior state of the field, we can emphasize the contributions and novelty of our research. At the end of the chapter, we will provide a brief review of more recent work to highlight the directions in which this research area has progressed since then.

In the past two decades, several strategies have been proposed for approximating optimal control laws using learning-based methods. Earlier works, such as [87] and [88], introduced neural network implementations for non-linear receding-horizon control problems. These works explored different approaches, with [87] restricting

their attention to a two-layered sigmoidal feed-forward network for learning optimal control policies, while [88] proposed a method to directly minimize the control cost, thereby escaping the need for offline computation of the optimal MPC control signals.

Following the advances in various machine learning areas, neural network designs have since then evolved to incorporate more complex features. Examples of such include the ReLU activation function, the *Long Short-Term Memory (LSTM)* structure and the two optimization layers OptNet [44] and cxvpylayers [45]. To some extent, these features each hold some properties akin to those of the MPC and eMPC, and have as such been explored in more recent work on learning-based optimal control.

One approach, introduced in [41], makes use of a reinforcement learning technique to train a ReLU-based deep neural network for approximating the control law. The authors highlight that by choosing the ReLU as activation, the integrated network will represent a PWA function that overlaps with the structure of the eMPC, and thus makes for an attractive choice for synthesizing the controller. Another key aspect of their approach is ensuring feasibility and constraint satisfaction of the generated control inputs. They achieve this by incorporating an orthogonal projection operation into their network, using Dykstra's projection algorithm [92] to project all network outputs onto a safe region defined by the intersection between the maximal control invariant set and the set of input constraints.

In [43], a similar question is addressed, but with a different approach. Unlike in [41], the authors do not rely on the system model in their method. Instead, they incorporate an implicit parametric quadratic program layer[1] in their network architecture and prove that, in combination with two linear layers, it is able to capture the structure of any *linear* MPC problem. They also show that the resulting closed-loop system can be certified for stability a posteriori.

Another interesting factor to consider when learning the MPC is the temporal dependency between the control actions. The ability to model sequential data is an inherent trait of a recurrent neural network and is the reason why such models have already been applied in a variety of MPC settings. Two examples include the works [93] and [94], which both employ recurrent network models with an LSTM structure to synthesize MPC controllers.

3.3 Preliminaries

In this section, we provide a brief review of the fundamental MPC problem, eMPC, and neural networks. A more detailed coverage of these topics can be found in Chapter 2.

[1] An OptNet [44] instance.

3.3.1 Model Predictive Control

Consider a discrete-time linear time-invariant system that evolves in time as

$$x_{k+1} = Ax_k + Bu_k. \tag{3.1}$$

Here, $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ denote the state vector and the control action at time k, respectively. The matrices A and B correspond to the system dynamics. In this context, we will study the simplified MPC problem of steering the state of the system (3.1) from an initial state x_0 to the origin by minimizing a control objective. This objective is subject to state and input constraints, defined as

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}. \tag{3.2}$$

Here, $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$ are polyhedra, characterizing the sets of constraints for the state and input, respectively. These sets define the boundaries within which the system must operate.

The MPC problem for an infinite time horizon can be formulated as the optimization problem

$$\begin{aligned} \min_{u} \quad & \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \\ & x_0 = \bar{x}, \end{aligned} \tag{3.3}$$

where Q and R positive semi-definite cost matrices. In an unconstrained setting of the state and input vectors, the solution to (3.3) yields the optimal feedback control law of the LQR, given by

$$u_k^* = -Lx_k, \quad L = (B^T P_\infty B + R)^{-1} B P_\infty A, \tag{3.4}$$

where P_∞ solves the *Algebraic Riccati Equation* [61].

However, solving the infinite time horizon MPC problem (3.3) is generally very challenging. Instead, it is common to resort to a finite-horizon MPC approach, where the following problem is solved at each step

$$\begin{aligned} \min_{\{u_0,\dots,u_{N-1}\}} \quad & J = x_N^T Q_N x_N + \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \\ & x_0 = \bar{x}. \end{aligned} \tag{3.5}$$

Here, N denotes the finite time horizon length, and Q_N the terminal cost matrix.

3.3.1.1 Explicit MPC

The eMPC is a strategy for circumventing the computational efforts required for solving the MPC online, by pre-computing the optimal control law offline. Given a polytopic set $\mathcal{X}$, for each $x \in \mathcal{X}$ the eMPC computes a PWA mapping from x to u defined over q regions of $\mathcal{X}$. See Subsection 2.3.2.1 for an example of this.

3.3.2 Satisfaction of Feasibility Constraints

The concept of set invariance plays an important role when it comes to characterizing the MPC constraints. In fact, set invariance is closely linked with feasibility, as discussed in [55,60,65]. According to the definition in [61], a set $\mathcal{C} \subseteq \mathcal{X}$ is considered a control invariant set for the system (3.1), subject to the constraints (3.2), if

$$x_k \in \mathcal{C} \implies \exists u_k \text{ s.t. } x_{k+1} \in \mathcal{C}, \quad \forall k \in \mathbb{Z}_+. \tag{3.6}$$

In other words, for any initial state in $\mathcal{C}$, there exists a controller that ensures that all future states remain within $\mathcal{C}$. The *maximal control invariant* set is then defined as the control invariant set containing all control invariant sets contained in $\mathcal{X}$, and is denoted by $\mathcal{C}_\infty$. For more details, see Subsection 2.2.3. Given that $\mathcal{C}_\infty$ is a polytope, it can be expressed as an intersection of halfspaces in $\mathcal{H}$-representation as

$$\mathcal{C}_\infty = \{x \in \mathbb{R}^n \mid C_x x \leq d_x\}. \tag{3.7}$$

Later in this chapter, we will explore how this set can be used as part of a projection strategy within neural networks to obtain feasibility guarantees for the control law. To compute $\mathcal{C}_\infty$, we use Algorithm 10.2 *"Computation of $\mathcal{C}_\infty$"* provided in [61], along with the accompanying software. It is worth noting, as discussed in [41], that this algorithm does not come with any termination guarantees. However, it was found to converge in our experiments.

In Figure 3.1 we plot $\mathcal{C}_\infty$ for system (3.1) with

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{3.8}$$

subject to the constraints

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad -2 \leq u_k \leq 2. \tag{3.9}$$

3.3.3 Neural Networks

Neural networks and deep learning approaches have become ubiquitous and can be found at the core of many modern learning-based techniques [95]. Neural networks learn a mapping from the input to the output from known training examples. This

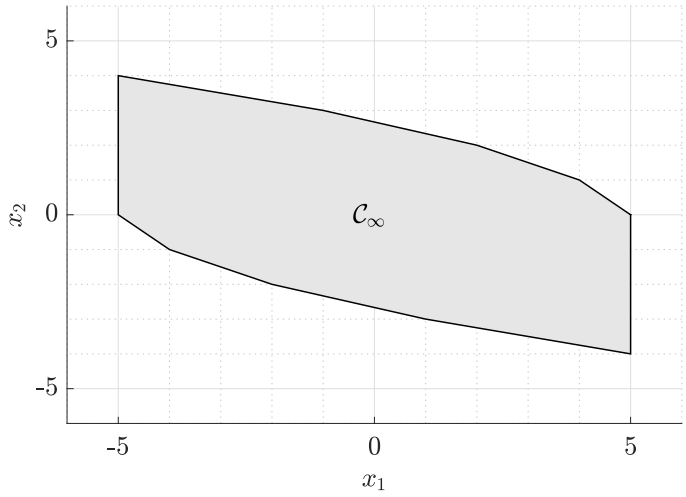


Figure 3.1: The maximal control invariant set $\mathcal{C}_\infty$ computed for system (3.1), with system and control matrices (3.8), and subject to the constraints (3.9).

is particularly useful when the problem at hand lacks a clear closed-form input-output relationship or when such a relationship is intractable to work with [72].

Neural networks consists of interconnected processing units known as neurons, which perform a combination of linear and non-linear transformations. In mathematical terms, a neural network $f(x)$ learns a mapping from the input x to the predicted output $\hat{y}$ of the form

$$\hat{y} = f(x) = \varphi(W_{N_\mathrm{L}}\varphi(W_{N_\mathrm{L}-1}\varphi(\cdots\varphi(W_1 x + b_1)\cdots) + b_{N_\mathrm{L}}).$$

Here, $\varphi(\cdot)$ represents the activation function, and $\theta = \{W_i, b_i\}_{i=1}^{N_\mathrm{L}}$ the parameters (weights and biases) that the network learns during training, with N_L denoting the number of neuron layers. The learning is typically achieved through backpropogation, a procedure that relies on gradients of an error or loss function $\mathcal{L}(\cdot)$ with respect to the network parameters θ. For reasons of computational complexity and stability, the ReLU is the most commonly employed activation function [72, 95]. It is important to note here that the use of ReLU as the activation function is well-motivated in the MPC setting due to its piece-wise linear nature [41]. A schematic of a two-layer neural network is shown in Figure 3.2.

3.4 Designing Neural Network Structures

While the concept of learning-based methods for MPC is not novel, a comprehensive framework for characterizing such approaches is lacking in literature. This chapter

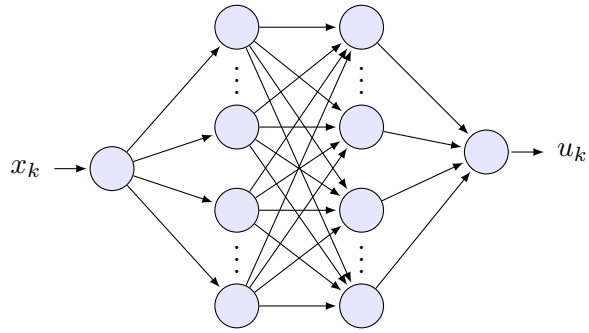


Figure 3.2: Schematic of a two-layer neural network.

aims to contribute to the development of such a framework by addressing these aspects, with a specific focus placed on training and evaluation through numerical experiments. In this section, we explore Problems 3.1, 3.2 and 3.3 outlined in Section 3.1, using the framework proposed in our preliminary study [96]. More specifically, we design and study three different neural networks that incorporate varying degrees of MPC-specific structure. We then evaluate and compare these networks in terms of two performance metrics. Based on the obtained results, we identify general performance trends that may serve as a guide in the process of choosing between the different modes of operation of learning the MPC. In addition, we also study the use of a well-motivated HAR sampler for efficient data generation.

3.4.1 Characterization of Learning Approaches

As mentioned in Section 3.1, the primary objective of any learning approach for solving MPC problems is to obtain a meaningful mapping $f(x)$ that, given an initial state x, produces the control law

$$\hat{u} = f(x),$$

where $\hat{u}$ is the predicted solution to the MPC. To make a consistent characterization of any learning approach, two important aspects have to be analyzed:

- the nature of the mapping, and
- the nature of the data, particularly in the context of training and evaluation.

We touch upon these two aspects in relation to the designed network structures and the data generation in the upcoming sections.

3.4.2 Designing Neural Networks for MPC

In this section, we motivate and detail the construction of different neural network structures for learning the MPC.

3.4.2.1 Understanding the Nature of the Mapping

Depending on the horizon, the state and input constraints, and the strategy employed, we have the following common variants of the MPC problem in general:

- LQR, which corresponds to the unconstrained version of the infinite horizon problem in (3.3);
- Finite or infinite horizon control with constraints on the input and state vectors;
- eMPC, which is a reformulation of the constrained MPC in terms of control invariant sets, giving a pre-computed offline lookup table-like characterization of the control law in terms of the state vector [65];
- Learning-based MPC approaches, typically in the form of neural networks, which use supervised learning from seen data examples.

Correspondingly, each of these variants produce their own mapping $u(x)$. In the case of quadratic program-based approaches for constrained MPC, $u(x)$ is obtained by solving quadratic programs through convex optimization solvers such as the OSQP [97]. In the next section, it will become clear how these insights influence our network designs to address the specific characterstics of the MPC problem.

3.4.2.2 Considered Network Structures

As mentioned in Section 3.1, recent work has demonstrated how ReLU-based networks can effectively represent PWA functions, such as the eMPC. We therefore let this structure serve as the basis for our network designs.

However, while ReLU-based networks do exhibit structural similarities with the eMPC, relying entirely on such a black box approach does not guarantee the generation of feasible control signals. This is due to the fact that such a strategy requires the training data to inherently abstract the structure of the underlying MPC problem without using any explicit domain knowledge. In contrast, approaches like the one presented in [41] enforce feasibility guarantees by employing a projection algorithm that utilizes the knowledge of the MPC structure to project any infeasible neural network outputs onto a feasible region. This ensures the feasibility of the subsequent state and control trajectories. Following the same line of thought, we explore a similar projection strategy by incorporating a state-of-the-art differentiable convex optimization layer as a final layer to two of our networks. More specifically, we investigate the following network structures:

1. *Black Box Neural Network (BBNN)*: This refers to a black box neural network that learns solely from input-output samples without MPC-specific information. An illustration of this network is provided in Figure 3.2.

2. *Projection Neural Network (PNN)*: This network incorporates a projection layer that enforces the feasibility constraints of the MPC [41]. The projection block is included as a neural network layer, which causes it to directly influence the network's learnable parameters. As previously explained, the training of the neural network relies on backpropagation, which requires the gradient of the entire network with respect to its parameters. It is therefore necessary that the projection block is differentiable and admits a gradient operation. To achieve this, we use an instance of cvxpy-layers [45], a Python library that offers a CVXPY-based [78] framework for obtaining differentiable convex layers in PyTorch. A schematic of the PNN is shown in Figure 3.3.

3. *LQR-PNN*: This network combines the projection layer with an LQR structure. The LQR block is introduced to enforce stability of the network and serves as a safeguard against the network producing unstable solutions. Figure 3.4 illustrates how the outputs from the LQR block and the neural network are fed into the feasibility projection layer. This structure thus offers a trade-off between the infinite horizon LQR solution and the learned neural network solution.

Each of the three networks takes the state vector x as input and generates the corresponding output mapping $\hat{u} = f(x)$. The cvxpy-layer realizes the projection by solving a constrained optimization problem. Following the approach in [41], we use the $\mathcal{H}$-representations of $\mathcal{C}_\infty$ and $\mathcal{U}$ to describe the constraints as follows

$$\mathcal{C}_\infty = \{x \in \mathbb{R}^n \mid C_x x \leq d_x\} \tag{3.10a}$$
$$\mathcal{U} = \{u \in \mathbb{R}^m \mid C_u u \leq d_u\}. \tag{3.10b}$$

The optimization problem for the projection can then be expressed as

$$\begin{aligned} \underset{\hat{u}_k}{\arg\min} \quad & \|\tilde{u}_k - \hat{u}_k\|_2^2 \\ \text{s.t.} \quad & C_x B \hat{u}_k \leq d_x - C_x A x_k \\ & C_u \hat{u}_k \leq d_u. \end{aligned} \tag{3.11}$$

Here, $\tilde{u}_k$ represents the input to the cvxpy-layer, and $\hat{u}_k$ the resulting predicted control signal for input x_k.

All three networks share the same structure concerning the trainable neuron layers: one input layer of width n, two hidden layers with a width of 8, and a fourth layer of width m. An illustration is provided in Fig. 3.2. A ReLU activation function follows the input layer and both hidden layers, while the last conventional layer is directly connected to the projection layer[2] to allow for any negative values of $\hat{u}$.

[2] Or, directly interpreted as the predicted output, in case of the BBNN.

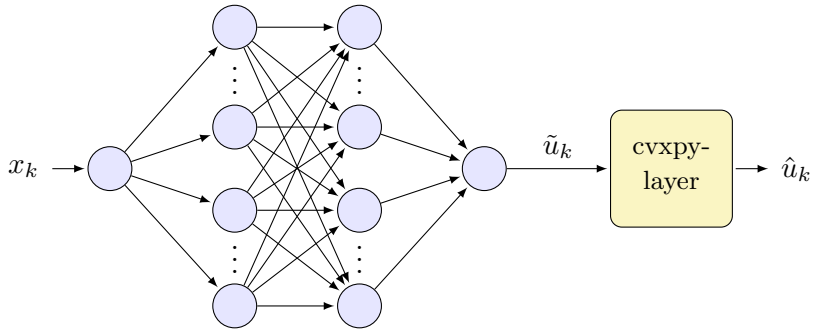


Figure 3.3: The structure of the PNN.

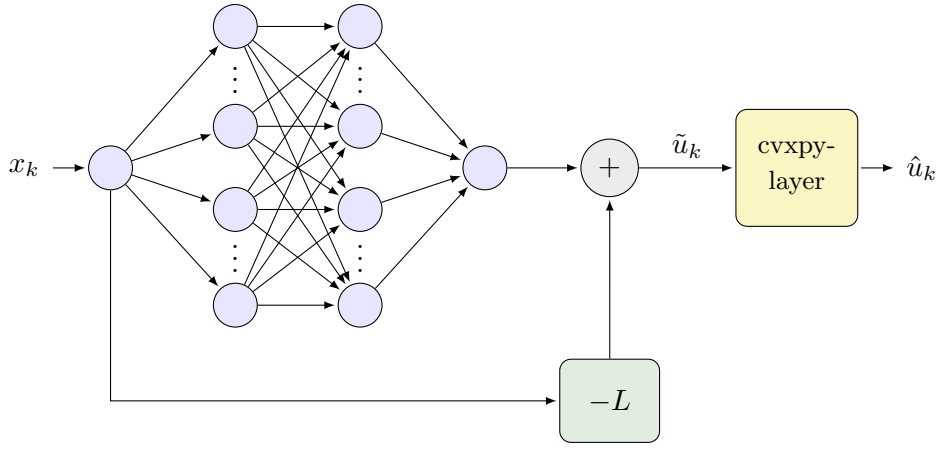


Figure 3.4: The structure of the LQR-PNN.

3.4.3 Data Generation and Training

Data generation and input design pose a significant challenge for most learning-based approaches, especially in the context of safety-critical control applications. To take a closer look at this, we start this section with a discussion on the nature of data and data generation in the context of learning, particularly focusing on the MPC application. Following that, we present our systematic strategy for effeciently generating training and test datasets for learning MPC problems.

3.4.3.1 The Nature of Data and Data Generation

One of the aforementioned key aspects in characterizing a learned mapping is its relation with the available dataset. In the context of training, this dictates how the sampling of the input space should be performed to ensure that the learning is meaningful and generalizes well. This aspect is particularly important for many

real-life applications, such as vehicle control, where access to an extensive and diverse training dataset is often limited.

In the case of the simple LQR,

$$u = -Lx,$$

this poses no real difficulty. Finding the control matrix L from training data is straightforward, since it requires only a single set of n linearly independent samples of x and their corresponding values of u.

However, in the case of eMPC, the complexity increases as one needs to sample a set of n linearly independent points within each region where the control law is constant. For q such regions, at least $n \times q$ samples of x and corresponding u-values are needed.

Naturally, this issue also applies to neural network-based mappings, raising the question of how to span the training space. One intuitively reasonable approach is to uniformly sample the feasible set. To this end, we propose the use of a statistically motivated Markov Chain Monte Carlo sampling technique that will be introduced in the following section.

The idea of learning-based MPC is also closely connected to the classical problem of function estimation from experimental data, dating back to the work of Box and co-workers on response surfaces [98]. For example, Gaussian processes and Bayesian optimization offer solid statistical frameworks for function estimation and input design [99]. In this context, samples are typically chosen by optimizing a so-called acquisition function, which measures the current uncertainty of the function estimate. A simple method is uncertainty sampling, where the next sample is taken where the uncertainty is the highest.

Unfortunately, optimal input design for training neural networks remains less explored. To gain insight into the approximation properties and generalization abilities, we need tools that can analyze the error and at the same provide information on where to sample new data. Provided that the network is flexible enough, the maximal error on the training data should be small after training. The challenge then lies in estimating the error for x outside the training set and sampling additional x where the error is large.

With this in mind, it is worth noting that we can compute the gradients[3] with respect to x, i.e., $\nabla f(x)$ and $\nabla u(x)$. These gradient computations can be carried out using CVXPY, as described in [45]. This approach allows us to gather information about the approximation error in a neighborhood of interesting sample points by computing the first order approximation

$$[f(x) - u(x)] \approx [f(x_i) - u(x_i)] + [\nabla f(x_i) - \nabla u(x_i)]^T (x - x_i),$$

within a neighbourhood where $\|x - x_i\| \leq \epsilon$, and where this region does not overlap with existing samples of x. To generate new training data, one can then identify

[3] Subgradients can be used where the mappings are non-differentiable.

regions where the approximation error is large. One way is to solve the QP problem

$$\begin{aligned} \max_{x} \quad & \|[\nabla f(x_i) - \nabla u(x_i)]^T (x - x_i)\| \\ \text{s.t.} \quad & \|x - x_i\| \leq \epsilon, \end{aligned}$$

for selected x_i, to obtain a measure where to take new samples. This demonstrates that it is possible to develop a well-motivated and systematic framework for the analysis and evaluation of learning-based MPC.

3.4.3.2 Sampling

In our data generation process, we construct training and test datasets comprising of state and control input pairs (x, u). We follow the same procedure for generating both sets. We start by sampling a set of states $\mathcal{S} = \{x_1 \ldots, x_{N_\text{s}}\}$, followed by solving for the corresponding optimal control input u_i for each $x_i \in \mathcal{S}$ using OSQP [97]. To sample from $\mathcal{C}_\infty$, we employ a *Hit-and-Run (HAR)* sampler, which is a Markov chain Monte Carlo method for sampling uniformly from convex shapes [100]. Essentially, starting from any point in the convex set, the sampler generates a set of points (states) $\mathcal{S}$ by walking random distances Δ in randomly generated (unit) directions. The steps involved in the procedure are detailed in Algorithm 3. We have chosen this approach because it guarantees that the generated data cover the feasibility region in a reasonably uniform manner [100, 101]. This, in turn, ensures that the network has observed training samples that span the entire feasible set on an average, thereby aiding its ability to generalize.

In Figure 3.5, we present a collection of 1000 points sampled using the HAR sampler from the set $\mathcal{C}_\infty$ displayed in Figure 3.1.

3.4.3.3 Training Procedure

Once the training dataset $\mathcal{D} = \{(x_i, u_i)\}_{i=1}^{N_\text{s}}$ has been generated, we proceed the network training using a supervised learning approach. During the training, we optimize the network parameters θ by minimizing a loss function $\mathcal{L}(\theta)$. In this work, we employ the *Mean Squared Error* (MSE):

$$\mathcal{L}(\theta) = \frac{1}{N_\text{s}} \sum_{i=1}^{N_\text{s}} (f(x_i, \theta) - u_i)^2. \tag{3.12}$$

In order to increase the training speed, we split the data into smaller subsets (mini-batches) and compute the MSE loss (3.12) for each batch. We then use the gradient descent-based *Adam* optimizer [102] to backpropagate the loss and update the parameters θ following each batch. Once all the mini-batches have been iterated over, one training epoch is completed. We train the networks for as many epochs required to reach convergence.

Algorithm 3 Hit-and-Run Sampler

procedure HIT-AND-RUN $(\mathcal{C}_\infty, N_\mathrm{s})$
  Pick random point $x \in \mathcal{C}_\infty = \{x \in \mathbb{R}^n \mid C_x x \leq d_x\}$
  $\mathcal{S} \leftarrow \{x\}$
  for $i = 1, \ldots, N_\mathrm{s} - 1$ **do**
    $\Delta_i \leftarrow \infty$
    Generate random unit direction δ_i
    for (c, d) in (C_x, d_x) **do**
      $\Delta \leftarrow \dfrac{d - c \cdot x}{c \cdot \delta_i}$
      if $\Delta > 0$ **then** ▷ *To ensure right direction*
        $\Delta_i \leftarrow \min(\Delta_i, \Delta)$
      end if
    end for
    $\Delta_i \leftarrow$ drawn from $\mathbb{U}[0, \Delta_i)$
    $x \leftarrow x + \Delta_i \delta_i$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{x\}$
  end for
  return $\mathcal{S}$
end procedure

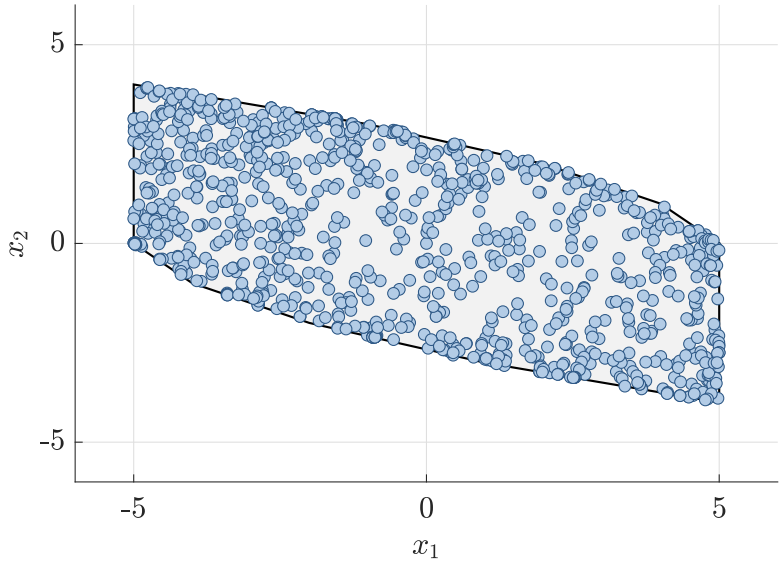


Figure 3.5: HAR sampling of 1000 points from the set $\mathcal{C}_\infty$ displayed in Figure 3.1.

3.4.4 Evaluation and Comparison of the Networks

We now consider the application of the proposed networks on three MPC examples. We evaluate the different network architectures in terms of two performance metrics:

1. The *Normalized Mean Square Error (NMSE)* which is defined as

$$\text{NMSE} = \frac{\mathbb{E}\left[\|\hat{u} - u\|_2^2\right]}{\mathbb{E}\left[\|u\|_2^2\right]}; \tag{3.13}$$

2. The normalized control cost J_n, defined as the control cost J in (3.5) normalized by $x_0^T x_0$:

$$J_\text{n} = \frac{x_N^T Q_N x_N + \sum_{k=0}^{N-1} \left[x_k^T Q x_k + u_k^T R u_k\right]}{x_0^T x_0}, \tag{3.14}$$

   where x_0 is the initial state of the trajectory.

Both metrics are evaluated on test data, previously unseen by the networks during training. The NMSE helps evaluate the control law predicted by the network with respect to the ground truth, whereas the control cost measures how well the control law is in terms of minimizing the control objective: the smaller the J, the better the control achieved.

3.4.4.1 Double integrator

We first consider the example of a two-dimensional state vector with a scalar input under constraints. Despite being relatively low-dimensional, such a scenario occurs regularly in many real-life control applications. For example, in the case of a simplified bicycle model employed in autonomous vehicles at Scania [103], the state model is given by

$$x_{k+1} = \begin{bmatrix} \cos(\nu_k \kappa_k) & \sin(\nu_k \kappa_k) \\ -\kappa \cos(\nu_k \kappa_k) & \sin(\nu_k \kappa_k) \end{bmatrix} x_k + \begin{bmatrix} (1 - \cos(\nu_k \kappa_k)/\kappa_k^2 \\ \sin(\nu_k \kappa_k)/\kappa_k \end{bmatrix} u_k.$$

Here, the state vector x_k consists of the direction coordinate and yaw, and ν and κ are parameters proportional to the velocity and curvature of the vehicle, respectively. The control objective matrices are usually of the form $Q = I$ and a scalar non-negative R. Though the most general models are time-varying (as seen from the state matrices), it specializes to a time-invariant MPC problem when the velocity and curvature of the vehicle are kept constant (that is, when ν_k and κ_k are constant over time) [103]. This motivates us to consider first the case of a two-dimensional MPC problem.

We consider a two-dimensional double integrator system specified as follows [61]:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{3.15}$$

subject to the constraints

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad -2 \leq u_k \leq 2, \tag{3.16}$$

and with cost parameters

$$Q_N = Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = 10, \quad N = 3. \tag{3.17}$$

We generate training and testing data by sampling states from $\mathcal{C}_\infty$ for the system (3.15) subject to (3.16) using Algorithm 3, followed by solving for the optimal controls using OSQP with cost parameters (3.17).

In Figure 3.6, we plot the NMSE in dB as a function of the size of the training dataset N_s. In Figure 3.7, we show the normalized control cost J_n computed for 100 trajectories for the different network-generated control laws. For the NMSE evaluation, we use a test dataset of 500 samples. For the control cost evaluation, we use a dataset of 1000 samples to train the networks. The initial states $\{x_0^{(1)}, \ldots, x_0^{(100)}\}$ of the trajectories are sampled from $\mathcal{C}_\infty$ using Algorithm 3.

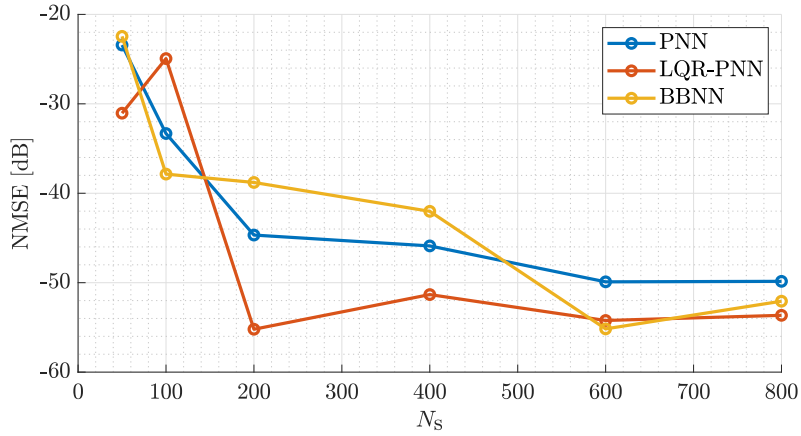


Figure 3.6: NMSE-comparison of the network-generated control laws in the 2D-example.

3.4.4.2 4-Dimensional system

In our second example we consider a 4-dimensional system from [41]

$$A = \begin{bmatrix} 0.7 & -0.1 & 0 & 0 \\ 0.2 & -0.5 & 0.1 & 0 \\ 0 & 0.1 & 0.1 & 0 \\ 0.5 & 0 & 0.5 & 0.5 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0.1 \\ 0.1 & 1 \\ 0.1 & 0 \\ 0 & 0 \end{bmatrix}, \tag{3.18}$$

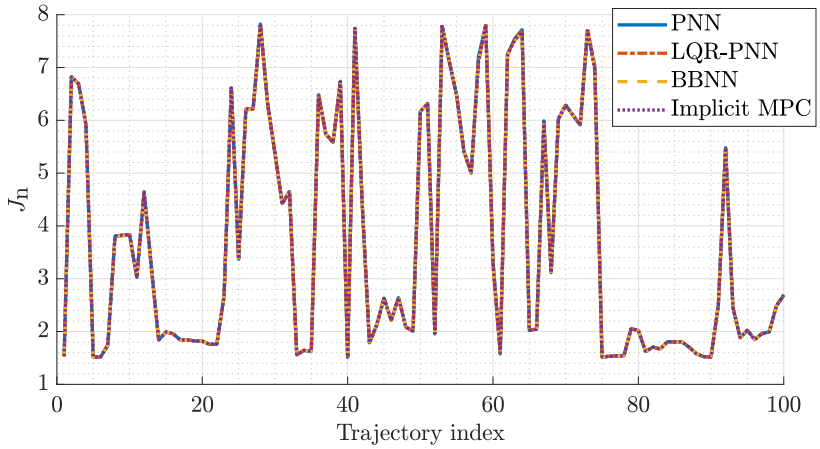


Figure 3.7: Comparison of the control costs computed for the three network-generated control laws in the 2D example.

subject to the constraints

$$\begin{bmatrix} -6 \\ -6 \\ -1 \\ -0.5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 6 \\ 6 \\ 1 \\ 0.5 \end{bmatrix}, \quad \begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq u_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \tag{3.19}$$

and with cost parameters

$$Q_N = Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad N = 10. \tag{3.20}$$

We generate training and testing data by sampling states from $\mathcal{C}_\infty$ for the system (3.18) subject to (3.19) using Algorithm 3, followed by solving for the optimal controls using OSQP with cost parameters (3.20).

3.4.5 Section Summary

In this section, we have presented a framework for offline training and evaluation of neural network approaches for MPC. The core concept involves approximating the MPC mapping from state to control input using constrained ReLU-based neural networks that include a projection layer. This structure is well-motivated by recent works, including [60] and [41], both which have successfully demonstrated the capabilities of deep ReLU-networks in representing continuous PWA functions

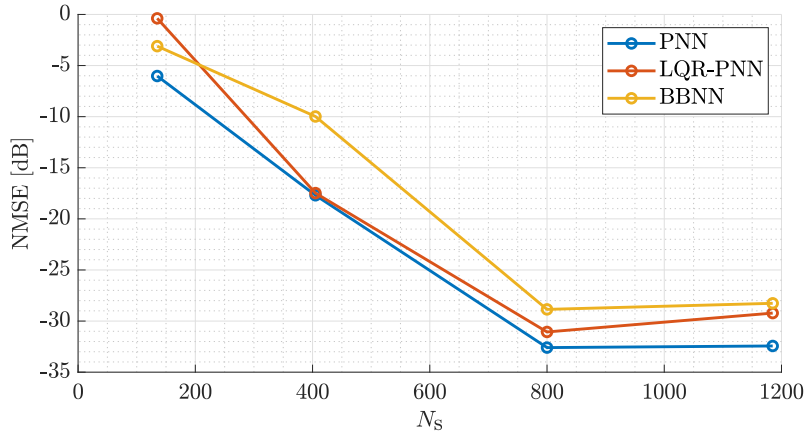


Figure 3.8: NMSE-comparison of the network-generated control laws in the 4D example.

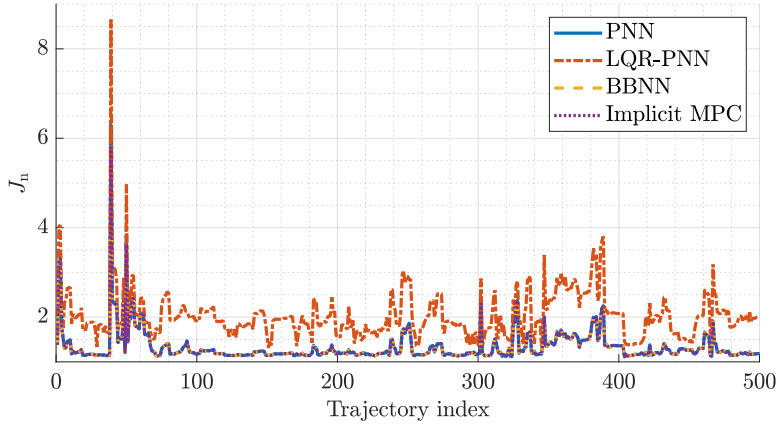


Figure 3.9: Comparison of the control costs computed for the three network-generated control laws in the 4D example.

on polyhedra, such as the eMPC. The role of the projection layer is to guarantee recursive feasibility and asymptotic stability.

We studied and compared three different network structures: the MPC-agnostic BBNN, the MPC-aware PNN, and the LQR-PNN, which employs an LQR-block as a safety measure. Based on our experimental result, we draw the conclusion that incorporating structural MPC information into the network significantly increases its performance.

In the upcoming section, we shift our focus towards the main contribution of this chapter. Here, we explore leveraging gradient information as an alternative strategy to gain structural information about the MPC without having to explicitly incorporate it into the network's neuronal structure. We demonstrate how the use of gradient data can enhance the training process, especially in scenarios where training data is limited, leading to improved control laws.

3.5 Learning MPC from Gradient Data

In recent years, there has been a growing interest in using gradient information for solving MPC problems. This includes the work of [86] on infinite-horizon differentiable MPC and its connections to imitation learning[4]. By incorporating gradient information, learning algorithms can gain additional structural information about the underlying MPC problem, which may be useful during the training process.

In this section, we consider the problem of learning a linear MPC using gradient data. More specifically, we consider the following problem:

Problem 3.5 (Learning models from gradient data)**.** *We design and evaluate algorithms to train ReLU-based neural networks for learning the MPC. The networks are trained on input and output data* (x_i, u_i) *together with the corresponding gradient data*

$$u_i' = \left.\frac{\partial u(x)}{\partial x}\right|_{x=x_i}.$$

We aim to demonstrate that incorporating gradient information in the training process can significantly reduce the amount of training data required to achieve high accuracy.

To illustrate the potential of using gradient data, we provide a motivating example of a general identification problem.

Example 3.5.1 (Identification using Gradient Data)**.** *Consider the following scalar linear feedback example with measurements*

$$u_k = l_1 x_k + l_2 + e_k, \quad \forall k \in [1, N_s].$$

[4]Imitation learning is a subfield of machine learning where an agent learns to perform a task by observing demonstrations provided by an expert.

with control signal u_k, scalar state signal x_k and additive white zero mean Gaussian noise e_k with variance σ_e^2. Assume that it is possible to measure the derivative of u with respect to x,

$$u'_k = l_1 + v_k, \quad \forall k \in [1, N_s],$$

where v_k is white zero mean Gaussian distributed noise with variance σ_v^2. The maximum likelihood estimate of l_1 and l_2 given the data $\{x_k, u_k, u'_k\}$, $k \in [1, N_s]$, is found by solving the least squares problem

$$V(l_1, l_2) = \sum_{k=1}^{N_s} \frac{[u_k - l_1 x_k - l_2]^2}{\sigma_e^2} + \frac{[u'_k - l_1]^2}{\sigma_v^2}.$$

The error covariance matrix of the least squares estimate $(\hat{l}_1,\ \hat{l}_2)^T$ equals (see [25]),

$$\frac{\sigma_e^2}{N_s} \left[\frac{1}{N_s} \sum_{k=1}^{N_s} \begin{bmatrix} (x_k^2 + \sigma_e^2/\sigma_v^2) & x_k \\ x_k & 1 \end{bmatrix} \right]^{-1}$$

For the choice $x_k = 1$ it is not possible to estimate l_1 and l_2 individually without the extra gradient data. In this case, the estimation error covariance matrix equals

$$\frac{1}{N_s} \begin{bmatrix} \sigma_v^2 & -\sigma_v^2 \\ -\sigma_v^2 & (\sigma_e^2 + \sigma_v^2) \end{bmatrix}$$

This result can also be found by analyzing

$$u_k - u'_k = b_2 + e_k - v_k.$$

The least squares estimate of l_2 is just the average of this difference signal. Notice that the variance of $\hat{l}_1$ is lower than the variance of $\hat{l}_2$, which is expected given the extra information on l_1 and the added noises when estimating l_2. Hence, gradient information can be crucial in terms of estimation quality for low input excitation.

3.5.1 Gradient-based Learning

As before, let $x_0 \in \mathbb{R}^n$ denote the initial state vector $x(0)$, and $u_0 \in \mathbb{R}^m$ the corresponding optimal control signal $u(0)$. Further, let $u' \in \mathbb{R}^{m \times n}$ denote the true gradient of the optimal control with respect to x. Consider that we are given a training set of N_s samples of triplets as

$$\{(x_i, u_i, u'_i)\}_{i=1}^{N_s}, \tag{3.21}$$

where the x_i's denote the initial states, u_i the corresponding optimal control law, and u'_i the corresponding gradient. Note that the subscript i denotes the ith sample

and not the time index. We also define the three sets

$$\mathcal{D}_x = \{x_1, \cdots, x_{N_\mathrm{s}}\} \tag{3.22a}$$
$$\mathcal{D}_u = \{u_1, \cdots, u_{N_\mathrm{s}}\} \tag{3.22b}$$
$$\mathcal{D}_{u'} = \{u'_1, \cdots, u'_{N_\mathrm{s}}\}. \tag{3.22c}$$

Keeping the same underlying network structure as before, our goal is now to train a black box ReLU-based network to predict the optimal control law using gradient information. The network learns the mapping

$$f(x, \theta) : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^m,$$

where, again, $\theta \in \mathbb{R}^d$ denotes all the learnable weights and biases of the neural network. Note that for this approach we omit the projection layer, relying solely on the structural information provided by the added gradient data.

The training of a neural network typically involves minimizing only the model error

$$\sum_{i=1}^{N_\mathrm{s}} \|u_i - f(x_i, \theta)\|_2^2,$$

which is the same loss function we employed in (3.12). However, it is well known that such a training procedure makes the network data-hungry, and that its performance often suffers when the number of training observations is limited. This is often the case in many control problems, given the generally very large state spaces the controllers operate in. As seen from our discussions in Section 3.4, explicitly incorporating structural information can significantly improve the network training in such scenarios. Assuming that training samples are randomly drawn from the feasible state space, with ideally one sample per region of the space, a uniform distribution of samples would give the network information about a large portion of the feasible set. In such cases, we expect that that including gradient data in the training dataset would improve the learning process, even with small sample sizes, thanks to the incorporation of the additional structural information the gradient data provides.

In light of this, we propose to train the neural network on the adjusted loss function $\mathcal{L}(\theta) : \mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_{u'} \to \mathbb{R}$, which has been extended to explicitly use gradient information. Specifically, we aim to learn $f(x, \theta)$ by minimizing

$$\mathcal{L}(\theta) = \sum_{i=1}^{N_\mathrm{s}} \left(\|u_i - f(x_i, \theta)\|_2^2 + \gamma \left\| u'_i - \left. \frac{\partial f(x, \theta)}{\partial x} \right|_{x=x_i} \right\|_2^2 \right). \tag{3.23}$$

Here, the first term represents the model error, the second term represents the goodness of fit for the gradients, and $\gamma > 0$ denotes the weight coefficient penalizing

the gradient error[5]. Our approach thus represents a trade-off between a completely data-driven and structurally aware MPC solver. An illustration of this network is provided in Figure 3.10. In contrast, a regular neural network-based MPC solver that relies solely on the model error during training is agnostic to this information and must abstract such structure from the training samples.

3.5.1.1 Computing Gradients of the MPC and eMPC

We note that our approach requires the value of the true gradients of the MPC problem evaluated at a given x. As we later detail in Subsection 3.5.2, we rely on OptNet [44] and cvxpylayers [45], both which offer frameworks for modelling convex problems amenable to mathematical operations.

Regarding the eMPC, we note that, given a polytopic set $\mathcal{X}$, the eMPC computes for each $x \in \mathcal{X}$ PWA mapping $u = u(x)$ defined over q regions of $\mathcal{X}$. Obtaining the true gradients in this context is thus straightforward since the gradients are also PWA constant. We use the MPT3 toolbox [68] in Matlab to perform these computations.

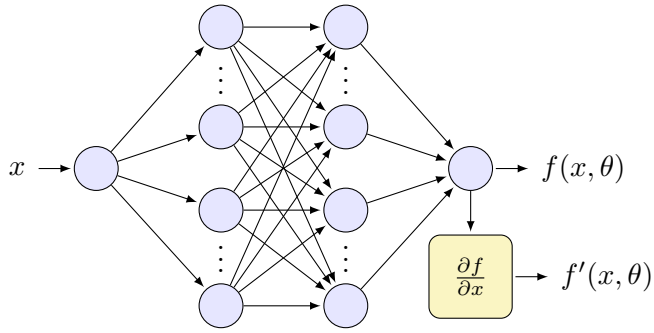


Figure 3.10: The structure of the proposed neural network.

3.5.2 Data Generation and Training

We follow a similar approach for generating the training data as described in Subsection 3.4.3. That is, we start by sampling the feasible space $\mathcal{C}_\infty$ using the HAR-sampler in Algorithm 3. The corresponding optimal control inputs and gradient sets, $\mathcal{D}_u$ and $\mathcal{D}_{u'}$, are then obtained from the eMPC formulation computed using

[5]We note that the ReLU is strictly not a globally differentiable function, as it has an analytically computable gradient at all values except at origin. The gradient of the ReLU is equal to zero or one, corresponding to negative and positive values of the argument. Consequently, the gradient $\partial f(x,\theta)/\partial x$ is not defined globally. Nevertheless, given that an exact zero is rarely encountered in practice, depending on whether the value of the argument is close to zero from the left side or the right side, appropriate ReLU gradient values are employed in computing the gradient of the neural network.

the MPT3 [68] toolbox in Matlab. We use the same strategy to generate test data with the difference that it does not contain gradient data. For higher-dimensional problems, for which the eMPC might be difficult to compute, the data must be generated from the implicit MPC. In this case, the gradient information may be obtained by using either a stand-alone instance of OptNet [44] or cvxpylayers [45].

Once the training and test datasets are generated, we again employ a supervised learning scheme to train the network on a dataset $\mathcal{D} = \{(x_i, u_i, u_i')\}$ of input-output pairs. During the training, we learn for the network parameters by minimizing the loss function $\mathcal{L}(\theta)$ defined in (3.23) with respect to θ. We again perform the training over mini-batches of five training samples and compute the training loss function (3.23) for each mini-batch. We then use the gradient descent-based *Adam* optimizer [102] to backpropagate the loss and update the parameters θ following each batch. Once all the mini-batches have been iterated over, one training epoch is completed. We train the networks until $\mathcal{L}(\theta)$ is reduced to 0.01 or for a maximum of 50000 epochs.

3.5.3 Performance Evaluation

We now consider the application of the proposed concepts on a set of networks with different weight coefficients γ, trained on MPC problems for a two-dimensional system. All the networks in the set are comprised of an input layer of size n, two hidden layers of eight neurons each, and an output layer of size m. We evaluate the networks in terms of the NMSE defined in (3.13) and the normalized control cost defined (3.14). While we illustrate our approach on an eMPC, the algorithm also extends to more general MPC problems.

Remark 3.1. *Note that the gradient information is* not *required during the test phase, and is used only in the training of the neural network. The networks are therefore not evaluated in terms of the gradient prediction.*

3.5.4 Two-Dimensional system

We consider the same example as before, featuring a two-dimensional state vector with a scalar input under constraints. We repeat the definition for the reader's convenience:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{3.24}$$

subject to the constraints

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad -1 \leq u_k \leq 1, \tag{3.25}$$

and with cost parameters

$$Q_N = Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = 10, \quad N = 3. \tag{3.26}$$

Table 3.1: Evaluation of the NMSE and control cost.

Set	NMSE [dB] $\gamma = 0$	NMSE [dB] $\gamma = 1$	$\tilde{J}_{\mathrm{n}}$ $\gamma = 0$	$\tilde{J}_{\mathrm{n}}$ $\gamma = 1$
$S_{(25)}$	-23.4	-26.2	6.0764	6.0700
$S_{(50)}$	-25.7	-38.5	6.0603	6.0450
$S_{(100)}$	-27.3	-43.6	6.0474	6.0347
MPT3	-		6.0345	

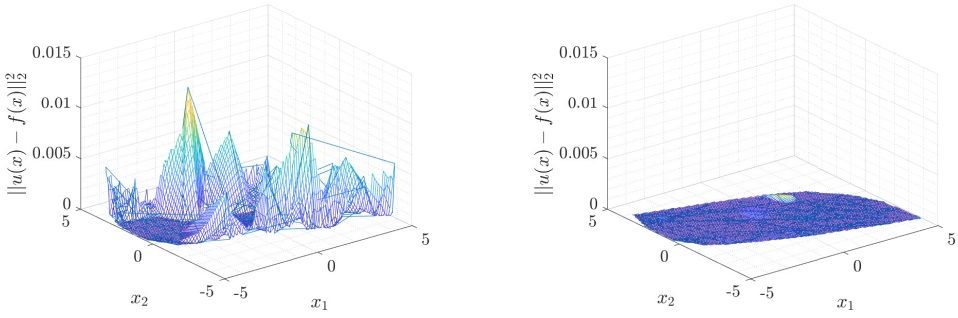
We generate training and test data by first sampling states from $\mathcal{C}_\infty$ for the system (3.24) subject to (3.25) using Algorithm 3. We then solve for the optimal controls and corresponding gradients using MPT3 with cost parameters given by (3.26).

In our experimental setup we consider three sets of ten neural networks for two values of the weight coefficient $\gamma \in \{0, 1\}$. For each set of networks we (pseudo) randomly generate 10 sets of training data, one for each network in the set. The first set of networks is trained using 25 samples, the second set using 50 samples, and the third set using 100 samples. We use $S_{(25)}$, $S_{(50)}$ and $S_{(100)}$ to denote the three network sets.

For the NMSE evaluation, we use a test dataset of 100 samples. For the control cost evaluation, we sample a set of 100 initial states from $\mathcal{C}_\infty$ using Algorithm 3. Starting from these, we then simulate the networks in closed loop for $N = 3$ time steps to generate trajectories.

The results from the NMSE and control cost evaluation are presented in Table 3.1. The NMSE is shown in dB averaged over the ten trained networks in each network set, and the control cost is averaged over 100 trajectories and over the 10 networks in each set. We denote the average as $\tilde{J}_{\mathrm{n}}$. For reference, the control cost obtained from simulating the true eMPC is also included. We see that, on average, $\gamma = 1$ results in the lowest NMSE as well as the lowest control cost for each network set. Figure 3.11 shows the absolute control error for a single network trained on 1000 samples and tested on 300 samples. Similar to what we observed in Table 3.1, we obtain the lowest test error for $\gamma = 1$. These results suggest that including gradient information during the training phase improves the performance of the network. From a theoretical point of view, the choice of γ should not be so important since we do not have measurement noise. Notice that the we only evaluate the NMSE in the control mapping and not in its gradient.

Remark 3.2. *As we pointed out before, we do not employ a projection strategy for ensuring constraint satisfaction.*



(a) Control error surface for $\gamma = 0$.

(b) Control error surface for $\gamma = 1$.

Figure 3.11: The control error surfaces for $\gamma \in \{0, 1\}$ when a single network has been trained on 1000 samples and tested on 300 samples.

3.6 Section Summary

In this section, we have explored the effectiveness of using gradient data to incorporate structural information into the process of learning the MPC. We have used MPT3 [68] and PyTorch [104] to implement this framework. The numerical experiments suggest that the gradients carries important information that improves the neural network training, resulting in better predicted control laws. It should be noted that we do not assume any specific model information, other than from (x, u, u'), which means our approach is not restricted to eMPC or even time-invariant MPC problems.

3.7 Chapter Conclusion

We conclude this chapter by exploring the latest research and advancements in the field that have emerged following the publications of the studies upon which this thesis is based. By doing so, we aim to provide valuable insights into the recent progress made in the field and to identify new research directions that have surfaced since the completion of this study.

3.8 Recent Advances

Learning-based MPC is an actively researched field that has witnessed significant recent advancements. One particular area of focus has been on finding alternative methods to ensure feasibility without relying on projection-based techniques. This is mainly motivated by the potential computational slowdown introduced by the additional optimization step involved in the projection, as well as by the challenges

in computing control invariant sets for general non-linear systems. Another key focus is on incorporating physical priors into the learning algorithms.

In the recent paper [105], a novel strategy for learning MPC with linear dynamics and constraints is proposed. The authors draw inspiration from two-phase interior point methods commonly used in convex optimization problems. Specifically, in phase one, the authors employ a simple function to obtain a feasible starting point. In phase two, the original problem is solved using a neural network architecture capable of encoding arbitrary polytopic constraints to enforce safety, thereby providing a projection-free feasibility guarantee.

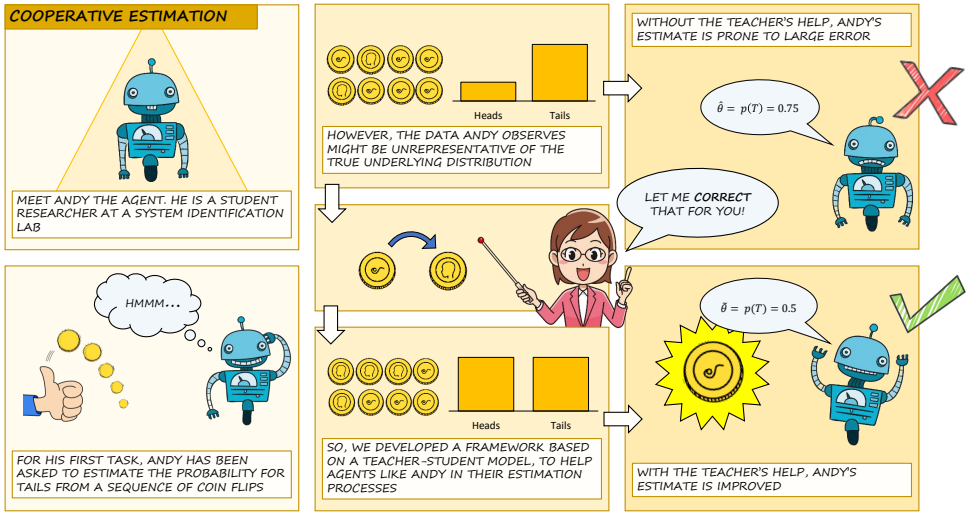
Another explored direction in safety filters involves the use of control Lyapunov functions (CLFs) and control barrier functions (CLBs) to enforce stability and feasibility. In [106], the authors present a method that simultaneously searches for both the optimal control policy and a CLF certificate to support its soundness. A similar approach is taken in [107], where the authors propose training a neural Lyapunov function in conjunction with a neural control policy. Other works, such as [108] and [109], develop learning frameworks integrated with CBFs. See [110] for an introductory survey.

Physics-informed machine learning (PIML) is another technique that has been applied for learning MPC. Recent advances have resulted in various approaches for integrating PIML and exploiting physical priors within MPC. These methods demonstrate how learnt PIML models can effectively replace the dynamics within the MPC optimization formulation to implicitly leverage physical priors through an efficient surrogate model. For an overview of the topic, see [111].

In the context of non-linear MPC, [112] propose a learning strategy based on safety-augmented neural networks. Their approach utilize a neural network to compute the complete input sequence of the MPC and verifies its feasibility online. As a safety measure, they compare the network-generated sequence to a candidate sequence computed using standard MPC techniques. If the network-predicted inputs are deemed unsafe or infeasible, or if they result in a higher control cost compared to the candidate, the candidate is applied instead.

Part II

Cooperative System Identification



This comic strip has been designed using assets from Freepik.com

Chapter 4

Online Correctional Learning

The majority of control strategies rely heavily on accurate models of the underlying system they aim to control. The quality of these models are crucial for drawing conclusions about the controller's performance on the real system. However, obtaining a good model can often be time-consuming, costly, and in some cases even impractical. For example, building and estimating a model for an industrial plant has been reported as one of the most expensive aspects of the control synthesis process [113].

In this chapter, we introduce online *Correctional Learning* as an extension of the batch correctional learning framework presented in Section 2.5. Correctional learning provides a cooperative strategy for improving parameter estimation in system identification problems by incorporating expert knowledge into the estimation process. In the online setting, the helping expert determines the optimal presentation of the observed data to the estimating agent at each time step. This online approach enables immediate action and on-demand decision-making, which is particularly attractive in real-life settings where data is typically acquired sequentially.

4.1 Introduction

Parameter estimation is the process of determining a model's parameter values from observed data. The values of these parameters hold significant importance as they have a direct impact on the distribution of the data generated by the modeled system. Therefore, estimation theory stands as a well-researched topic with several established methods, see e.g. [25]. The interest in the subject is widespread with applications to be found in various domains, including the process industries, control applications, as well as in the research of biological functions and systems [47]. Popular estimation methods range from the conventional maximum likelihood and Bayesian inference methods, to more recent learning-based methods.

As alluded to above, the system identification process often proves very time-consuming and expensive. In addition, the estimation process is prone to large error

when the observed data fail to accurately represent the underlying distribution, as discussed in Section 2.5. For instance, a recent study found that commercially available facial analysis algorithms showed higher error rates for darker-skinned individuals and women [51], which could be linked to unrepresentative training data.

Correctional learning is a recently developed framework that may be used to address these issues [52]. The framework arises from the idea of cooperative (learning) problems, i.e., settings in which two or more agents work together towards a common goal. The framework is structured around a teacher-student model, where an expert (teacher) agent seeks to assist a learner (student) agent in its estimation process. This strategy draws inspiration from cooperative teacher-student paradigms in the machine learning-literature, such as *learning from demonstration* [114] and *imitation learning* [115], where an expert agent (teacher) helps a learning agent (student) by demonstrating optimal policies to improve convergence.

In correctional learning, the teacher's goal is to modify (correct) the collected observations, based on which the student forms its estimation. See Figure 4.1 for an illustration of this. Correctional learning can thus be viewed as a means for finding an optimal mapping from the original observations to a modified sequence that minimizes the student's estimation error.

Beyond its applications in control and estimation, the correctional learning framework extends its utility to many traditional learning settings, such as aiding policy learning in reinforcement learning, manual output-error correction of machine learning models, cooperative learning for task performance, estimating user preferences and ratings, and more. In addition, the correctional learning framework may serve as a tool for diversifying information presented to users, thereby combating issues like echo chambers, confirmation bias, and the spread of misinformation in social media and search engines. In the finance sector, our framework might find itself useful for improving an investor's market state predictions for stock portfolio allocation.

For many of these domains, however, the need for immediate (online) action becomes imperative as observations arrive sequentially. This underscores the necessity for a learning process that can adapt and update quickly. Therefore, this chapter presents the concept of an *online* correctional learning framework. In this setting, the teacher must decide at each time step whether to modify the current observation or not, while adhering to a budget constraint. To account for the the stochastic nature of incoming observations, we use a *Markov Decision Process (MDP)* to model the decision-making process.

The chapter's main contributions can be summarized as follows:

- Firstly, it establishes a theoretical bound on the improvement induced by the teacher.
- Secondly, it formulates an MDP for the online correctional learning framework.

- Thirdly, it demonstrates the framework through two numerical experiments.
- Lastly, it compares the proposed online framework with the original batch framework.

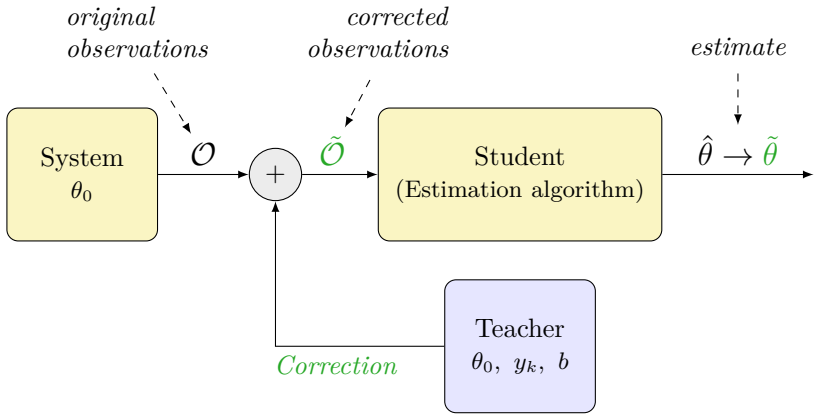


Figure 4.1: Schematic representation of the correctional learning framework.

4.1.1 Related work

Learning from experts is widely studied problem in machine learning. *Learning from demonstrations* [114] and *imitation learning* [115] are two closely related paradigms where a robot learns from observing the behavior of an expert. In *corrective feedback*, on the other hand, the expert provides corrections to the robot's actions to improve its learning process. This is contrast to correctional learning, where the corrections are made to the data that the robot learns from.

By viewing our framework as a means for customizing a dataset to better suit a specific learning task, we find similarities with other techniques and statistics. *Feature selection* [116] is one such example, where the aim is to find the most informative and relevant features to improve learning. Similar to our correctional learning framework, this family of methods has seen a shift from batch to online techniques [117], offering efficient and scalable machine learning algorithms for large-scale applications.

Our work also aligns with the field of *input design for system identification* [118, 119], where input signals are designed to ensure a certain model accuracy. *Active learning* is another related topic, in which the learner queries the teacher for desired labels to guide the learning process [120].

Another area of relevance is *counterfactual explanations* [121], a branch within explainable artificial intelligence that employs feature importance to explain how small perturbations in input data affect the output of machine learning models.

The notion of *learning with side information* [122] also bears relevance, as it entails providing additional information to the learner in its learning process. Aside from machine learning applications, this topic is also studied within the realm of information theory, particularly in connection to communication problems [123].

4.2 Batch Correctional Learning

For the reader's convenience, this section will revisit the concept of batch correctional learning by providing a short summary of the key points of Section 2.5.

Correctional learning is a framework designed to address cooperative system identification problems. The framework is based on a teacher-student model, where an expert (teacher) agent assists a learner (student) agent in the process of estimating unknown system parameters.

The student collects information in the form of N i.i.d. observations $\mathcal{O} = \{y_k\}_{k=1}^N$, where each observation y_k belong to the observation space $\mathcal{Y}$. Using this observed data, the student estimates the underlying data-generating distribution p as $\hat{p}$. Based on this estimated distribution, the student further estimates the unknown system parameters θ as $\hat{\theta}$.

Due to various constraints, the teacher is restricted from directly communicating the true parameter value θ_0 to the student. Instead, the agents operate within the space of induced probability distributions. The teacher, who also knows the true distribution p_0 of the data, aims to improve the students estimate $\hat{p}$, and, consequently, $\hat{\theta}$, by correcting the observations collected by the student. This leads to a modified dataset $\tilde{\mathcal{O}} = \{\tilde{y}_k\}_{k=1}^N$ that better reflects the true distribution, along with the corresponding probability estimate $\tilde{p}$. The goal is to provide the student with information that brings the student's altered estimate closer to the true value or enables faster convergence.

The communication restrictions between the two agents are modeled by a budget constraint, characterized by the teacher's intervention budget b and some distance (or cost) measure c between two sets as

$$c(\mathcal{O}, \tilde{\mathcal{O}}) \leq b. \tag{4.1}$$

Essentially, the budget constraint limits the number of samples that the teacher can modify.

The goal of the teacher is to minimize the discrepancy between the student's estimated model and the true system. The offline (batch) correctional learning problem can therefore be formulated as the optimization problem

$$\begin{aligned} \min_{\tilde{\mathcal{O}}} \quad & V(p_0, \tilde{p}) \\ \text{s.t.} \quad & \tilde{y}_k \in \mathcal{Y}, \ \forall \ \tilde{y}_k \in \tilde{\mathcal{O}}, \\ & c(\mathcal{O}, \tilde{\mathcal{O}}) \leq b. \end{aligned} \tag{4.2}$$

Here, V is a statistical distance measure between two probability density functions, such as the KL-divergence. The objective is to find the modified dataset $\tilde{\mathcal{O}}$ that reduces the discrepancy between the true distribution p_0 and the corrected distribution $\tilde{p}$, while satisfying the constraints.

In [52], it was demonstrated that the resulting set $\tilde{O}$ of corrected observations was optimal in the case of binomial data, and the reduction in variance of the corrected estimate compared to the original estimate was quantified. In this chapter, we extend the framework by formulating an MDP to address the problem in an online setting across various applications.

4.3 Correctional Learning Bounds for Discrete Systems

In this section, we analyse the effectiveness of the teacher in helping the student when the observations are discrete. We will consider the mean values of two sequences of observations: the original sequence and the modified sequence. The measure of the teacher's effectiveness will be based on how much the variance of the corrected estimate decreases, which reflects a reduction in the estimation error. The following theorem establishes a relationship between the estimates of the mean values and quantifies the decrease in variance for the altered estimate.

Theorem 4.1 (Variance decrease of the altered estimate)**.** *Let $X_1, \dots, X_N$ be i.i.d. random variables taking values in the set $\{0, 1, \dots, M-1\}$, with a mean of μ. We denote the sum of these variables by $Y = X_1 + \cdots + X_N$. Let $b \in \{0, 1, \dots, N\}$, and*

$$\tilde{Y} = \underset{Z \in \{0,\dots,N\}: |Y-Z| \le b}{\arg\min} |Z - N\mu|. \tag{4.3}$$

Then,

$$\operatorname{var}(\tilde{Y}/N) \le M^2 \exp\left(-\frac{2b^2}{NM^2}\right). \tag{4.4}$$

Let us further assume that $X_i \sim \mathbb{U}(\{0, \dots, M-1\})$. This assumption is not crucial, but provides a special case that is easier to understand. Then,

$$\frac{\operatorname{var}(\tilde{Y}/N])}{\operatorname{var}(Y/N)} \le \frac{6M}{5M+1} \exp\left(-\frac{2b^2}{NM^2}\right). \tag{4.5}$$

The proof of Theorem 4.1 can be found in Appendix 4.A. Essentially, the theorem provides an upper bound on the decrease in variance of the student's estimate achieved by the help of the teacher, given a budget b. It implies the following:

i) The teacher's ability to improve the student's learning increases with a larger budget;

ii) For a fixed budget b, the improvement becomes less significant as the number of observations N grows. This is reasonable, since the average deviation of Y/N around μ is of order $O(1/\sqrt{N})$, while the improvement achieved with the help of the teacher can be at most b/N;

iii) For a fixed budget b and sample size N, the improvement degrades as the number of possible outcomes M increases. This is because the variance of Y/N increases with M, making it increasingly challenging for a teacher to compensate for "bad" samples.

Having established the potential improvement of the student's estimation process in a discrete setting, we will now propose a framework for the teacher to achieve this by correcting the observations in real-time.

4.4 Online Correctional Learning

We now present our main contribution of this chapter: a framework for computing the optimal online policy for the teacher agent. Unlike the batch case, where all observations are available upfront, the online setting considers more realistic scenarios in which observations are obtained sequentially. In this setting, the teacher has to decide, at each time step, whether or not to change the current sample and, if so, what to change it into.

4.4.1 Formulation of the Markov Decision Process

To address the online problem, we develop an MDP framework that represents the teacher's optimal policy. In this setting, the student samples discrete observations $y_k \in \{0, 1, \ldots, M-1\} = \mathcal{Y}$ from a system, with the goal of estimating its true parameters θ_0.

As detailed in Section 2.7, an MDP is characterized by a four-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, P the transition probability matrix, and R the reward function. For the correctional learning problem, we define the MDP as follows:

States: In the online setting, the teacher requires specific information at each time step to be able to make a decision effectively, namely:

i) the observed frequency of each outcome $y \in \mathcal{Y}$, which will provide insight into the current empirical distribution;

ii) the remaining intervention budget, which will guide the teacher's assessment of whether changing an observation is worth it or not;

iii) the current observation.

In light of this, we represent the state at time k by the three-tuple $s = (x_k, b_k, y_k)$. Here, x_k is an $M \times 1$ vector containing the observed frequencies of each outcome until time k, $b_k \in \mathbb{N}_0$ represents the remaining budget at time k, and y_k is the observation received at time k.

The number of states, denoted by $\text{card}(\mathcal{S})$, is finite and upper bounded by $N^{M+1}b$. However, due to the constraint

$$\sum_{m=0}^{M-1} [x]_m \leq N, \tag{4.6}$$

which ensures that the sum of all unique observations is less than or equal to N, we can derive a tighter upper bound for the number of states as

$$\text{card}(\mathcal{S}) \leq \text{card}(x)bN. \tag{4.7}$$

Here, $\text{card}(x)$ represents the number of possible observed frequency vectors and can be computed by using the concept of *multisets coefficients*. It is given by

$$\text{card}(x) = \sum_{n=1}^{N} \left(\binom{M}{n} \right) = \sum_{n=1}^{N} \frac{M(M+1)\dots(M+n-1)}{n!}, \tag{4.8}$$

which counts the numbers of ways to choose N elements with repetition from a set of M elements.

Terminal states: The states where all N observations have been received, i.e., where

$$\sum_{m=0}^{M-1} [x]_m = N. \tag{4.9}$$

Actions: The teacher can choose to either keep the current observation, y_k, or change it into another outcome $\tilde{y}_k \in \mathcal{Y}$. The number of actions is thus $\text{card}(\mathcal{A}) = M$.

Reward function: Non-terminal states are assigned a zero reward. In the terminal states, the reward is set to be inversely proportional to the estimation error resulting from the teacher's corrections.

Transition probabilities: The state evolution depends on the action taken by the teacher at a given state s. However, regardless of the action, the first element of the next state depends solely on the probability of the upcoming observation, denoted by $p(y_{k+1})$, where $\sum_{y \in \mathcal{Y}} p(y) = 1$. Depending on the chosen action, the state evolution varies as follows:

- If the teacher keeps the current observation y_k, the remaining budget is not affected, and so the state evolves as

$$s = (x_k, b_k, y_k) \xrightarrow{\text{keep } y_k} s' = (x_{k+1}, b_k, y_{k+1}), \tag{4.10}$$

with probability $p(y_{k+1})$. Here, $[x_{k+1}]_{y_{k+1}}$ is updated as $[x_{k+1}]_{y_{k+1}} = [x_k]_{y_{k+1}} + 1$. This reflects adding 1 to the entry in the vector that corresponds to the observed frequency of the outcome y_{k+1}. We express this evolution in terms of a conditional probability as

$$\Pr[(x_{k+1}, b, y_{k+1}) \mid (x_k, b, y_k),\ \text{“keep } y_k\text{”}] = p(y_{k+1}). \tag{4.11}$$

- If the teacher instead chooses to change the observation into $\tilde{y}_k$, the state will instead evolve as

$$(x_k, b_k, y_k) \xrightarrow{\text{change into } \tilde{y}_k} (x_{k+1}, b_k - 1, y_{k+1}), \tag{4.12}$$

with probability $p(y_{k+1})$. Here, the elements of the state are updated as

  - $[x_{k+1}]_{\tilde{y}_k} = [x_k]_{\tilde{y}_k} + 1$, reflecting the addition of the modified outcome at time k.
  - $[x_{k+1}]_{y_k} = [x_k]_{y_k} - 1$, reflecting the removal of the original outcome at time k.
  - $[x_{k+1}]_{y_{k+1}} = [x_k]_{y_k} + 1$, reflecting the addition of the new sample observed at time $k+1$.

We express this evolution in terms of a conditional probability as

$$\Pr[(x_{k+1}, b-1, y_{k+1}) \mid (x_k, b, y_k),\ \text{“change into } \tilde{y}_k\text{”}] = p(y_{k+1}). \tag{4.13}$$

Note how the budget and the total count of observed outcomes is adjusted to account for the correcting action.

Budget constraint: The budget constraint is enforced by assigning an infinitely negative reward to transitions into states where the remaining budget would result in $b_{k+1} < 0$.

To summarize, the MDP is defined as follows:

$$\begin{aligned}
&\textbf{States:} \quad s = (x_k, b_k, y_k) \\
&\textbf{Actions:} \quad a = \{\text{keep } y_k,\ \text{change to } \tilde{y}_k\} \\
&\textbf{Time-horizon:} \quad N \\
&\textbf{Reward function:} \quad -\|\tilde{\theta}_N - \theta_0\|_1 \\
&\textbf{Constraint:} \quad \text{number of corrections} \le b \\
&\textbf{Transition probabilities:} \quad \text{see (4.11) and (4.13).}
\end{aligned} \tag{4.14}$$

Remark 4.1. *Note that the chosen formulation of the states and actions satisfies the Markov property.*

The optimal policy for the online correctional learning problem, as described the MDP in (4.14), can be obtained using dynamic programming [124]. It is worth noting that this framework can be adapted to different scenarios by adjusting the reward function to align with the student's specific goals for the task at hand. The framework can also be extended to handle continuous observations by discretizing the observation space and changing the constraint to the total amount of correction $\sum_{k=1}^{N} |y_k - \tilde{y}_k| \leq b$.

To further clarify the defined framework, let us consider a simple example that illustrates the transition probabilities of the MDP.

Example 4.4.1 (Transition probabilities)**.** *Consider the correctional learning setup with observation space $\mathcal{Y} = \{A, B, C\}$, where each outcome is equally likely to be observed. Assume that up until now the individual outcomes have been observed 4,5 and 6 times, respectively, and that the teacher has a remaining budget of $b = 4$. Assume that the current observation is an A. Then, we are currently residing in state*

$$s = \left(\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}^T,\ 4,\ A \right). \tag{4.15}$$

Now, depending on the choice of action, we consider a total of 9 transition probabilities, three for the "keep" action and six for the "correct" action. If the teacher decides to keep the current observation, the state will evolve into one of the three following states with equal probability ($p = 1/3$):

$$s = \left(\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}^T,\ 4,\ A \right) \xrightarrow{\text{keep } A} \begin{cases} s' = \left(\begin{bmatrix} 5 & 5 & 6 \end{bmatrix}^T,\ 4,\ A \right) \\ s' = \left(\begin{bmatrix} 4 & 6 & 6 \end{bmatrix}^T,\ 4,\ B \right) \\ s' = \left(\begin{bmatrix} 4 & 5 & 7 \end{bmatrix}^T,\ 4,\ C \right) \end{cases}. \tag{4.16}$$

If the teacher instead decides to change the observed A into a B, the state will evolve into one of the three following state with equal probability:

$$s = \left(\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}^T,\ 4,\ A \right) \xrightarrow{A \to B} \begin{cases} s' = \left(\begin{bmatrix} 4 & 6 & 6 \end{bmatrix}^T,\ 3,\ A \right) \\ s' = \left(\begin{bmatrix} 3 & 7 & 6 \end{bmatrix}^T,\ 3,\ B \right) \\ s' = \left(\begin{bmatrix} 3 & 6 & 7 \end{bmatrix}^T,\ 3,\ C \right) \end{cases}. \tag{4.17}$$

The state evolution follows the same pattern as in (4.17) *if the teacher decides to change the observed A into a C.*

4.5 Numerical Experiments

In this section, we evaluate the effectiveness of the proposed framework by demonstrating significant improvements in the student's learning when using a teacher. First, we consider the task of computing the mean of multinomial and binomial data. These examples allow us to derive explicit solutions for both the batch and online setting and analyze the inner workings of the framework. We also apply the framework to a biological parameter estimation problem to showcase its applicability to more complex scenarios. The simulations were performed using Python 3.7 on a 1.90 GHz CPU.

4.5.1 Results for Binomial Data

In these experiments, we sample observations $y_k \in \mathcal{Y} = \{0, 1\}$ according to a Bernoulli distribution

$$\theta_0 = p_0(y_k = 1) = 1 - p_0(y_k = 0), \tag{4.18}$$

with the true parameter set to $\theta_0 = 0.5$. We perform the algorithm over 50 experiments and with $N = 10$, and compare our results with the batch case presentedd in [52].

Figure 4.2 illustrates the outcomes of the optimal online correctional learning policy obtained from the MDP in (4.14) for a binomial setting. One advantage of exploring this case, is that we can prove the optimality of the policy learned by the teacher, as stated in Section 4.4. This can be seen from the fact that the error of the student's estimate, when provided with the corrected sequence of observations (shown in orange in the figure), always equals the minimum achievable error for the original sequence (depicted in black). The exact expression for this error is given by (2.26), repeated here for convenience

$$e(N, \theta_0, b, \hat{\theta}) = \max\left\{ \|\theta_0 - \hat{\theta}\|_1 - \frac{2b}{N}, e_{\min} \right\}. \tag{4.19}$$

The term $e_{\min}$ represents the minimum attainable error as defined in (2.25):

$$e_{\min}(N, \theta_0) = \left\| \theta_0 - \frac{[\theta_0 N]}{N} \right\|_1. \tag{4.20}$$

Here, the notation $[\cdot]$ without subscript means rounding to the nearest integer, subject to the constraint $\mathbb{1}^T \theta_{\min} = 1$, where $\theta_{\min} = \frac{[\theta_0 N]}{N}$. Note that the error (4.19) is never greater than the error associated with the original sequence of observations (shown in blue in the figure).

By analyzing the policy values obtained using the dynamic programming algorithm, we find that the teacher consistently chooses the optimal policy given

by

$$\mu^* = \begin{cases} a_k = \text{keep } y_k, & \text{if } b_k \leq 0 \text{ or } [x]_{y_k} \leq \left[[\theta_0]_{y_k} N\right] \\ a_k = \text{change into } \tilde{y}_k = 1 - y_k, & \text{otherwise.} \end{cases} \tag{4.21}$$

This policy coincides with the policy computed using batch correctional learning [52]. The teacher's strategy is to delay spending the budget for as long as possible, only correcting a sample when there is an excess of one particular outcome. For example, consider the results shown in Figure 4.2 that correspond to the corrected sequence $\{1, 1, 0, 1, 1, 1, 0, 0, \underline{0}, 1\}$. In this case, the teacher decided to change the underlined value, which was originally a 1, into a 0 regardless the value of the next incoming sample. This finding is in contrast to the multinomial case, which will be discussed in the following subsection, where a specific outcome must be chosen for correction.

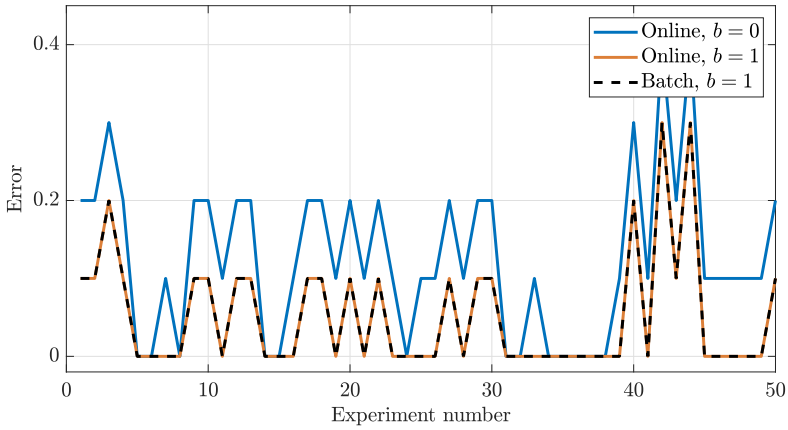


Figure 4.2: The figure shows the student's estimation error in two scenarios: with the help of the teacher (in orange) and without the help of the teacher (in blue). The orange line coincides with the black curve, which represents the theoretical best estimate for that particular situation, as given by (4.19) with $b = 1$ and $N = 10$. As the budget b increases, the orange and black curves converge towards the minimum error $e_{\min}$ defined in (4.23).

4.5.2 Results for Multinomial data

We now consider the outcomes $y_k \in \mathcal{Y} = \{0, \ldots, M-1\}$ to be sampled i.i.d. from a multinomial distribution, which is a generalization of the binomial distribution. Figure 4.3 illustrates the results of the proposed MDP for performing correctional learning in an online setting. The figure shows the student's estimation error based

on the original observation sequence (without help from the teacher) in blue, and the estimation error based on the corrected sequence (with help from the teacher) in orange. We compute the estimates θ as the mean of the observations:

$$[\theta]_i = \frac{1}{N}\sum_{k=1}^{N} \mathbb{I}(y_k = i). \tag{4.22}$$

In this case, we assume that an observation is randomly sampled $N = 5$ times from a multinomial distribution with the true parameter vector $\theta_0 = [0.4, 0.3, 0.3]$ over 50 experiments.

As opposed to the binomial case, we cannot derive a closed-form solution for the minimum attainable error. In the multinomial case, this error is given by the batch error, which can be computed using (8) in [52] with the ℓ_1-norm in the objective function. However, the minimum error independent of $\hat{\theta}$ and b can be computed as follows:

$$e_{\min}(N, \theta_0) = \left\| \theta_0 - \frac{[\theta_0 N]}{N} \right\|_1 = 0.2, \tag{4.23}$$

which is achieved by $\theta^* = [0.4; 0.4; 0.2]$ or $[0.4; 0.2; 0.4]$. The notation $[\cdot]$ should be interpreted in the same way as defined in (4.20).

Intuitively, one would expect the teacher's optimal policy to be to delay spending its budget for as long as possible. In the binomial case, the optimal online policy learned is given by (4.21), which coincides with the policy computed using batch correctional learning. In the multinomial case, the optimal policies differ only a limited number of scenarios when unexpected samples with low rewards are obtained. Note that in experiment 11 (indicated by an arrow in the figure), the altered estimate $\tilde{\theta}$ is worse than the original $\hat{\theta}$ (corresponding to $b = 0$). In that experiment, the teacher chose to perform the following correction

$$\{1, 2, 0, 2, ?\} \to \{1, 2, 0, 0, ?\}, \tag{4.24}$$

that is, changing the fourth observation of a 2 into a 0. This is because this yieled a larger expected value. In this scenario, receiving a 1 or a 2 at time step $k = 5$ had a high probability and maximum reward, but the less likely observation, 0, was received instead.

Figure 4.4 shows that, as expected, the variance of the estimate decreases as the number of observations increases. However, as the budget of the teacher increases, the variance is further decreased. This result illustrates the conclusions from Theorem 4.1.

4.5.3 Biological Parameter Estimation

Biological internal models have played a significant role in exploring and validating neuroscientific theories. For instance, they have helped in understanding the

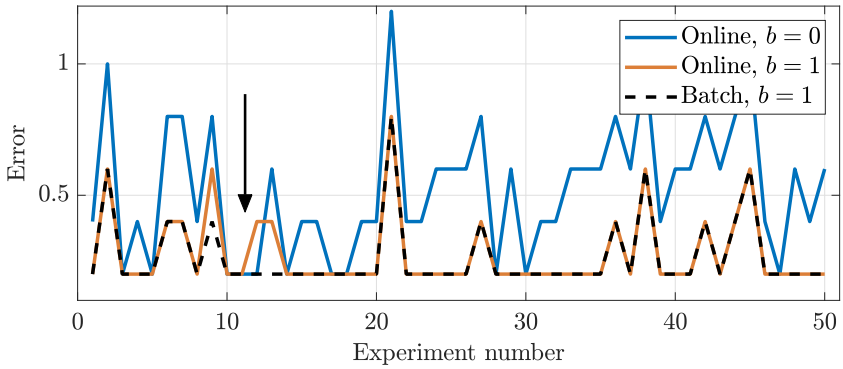


Figure 4.3: The figure illustrates the estimation errors from 50 separate experiments. It is evident that the teacher's policy significantly reduces the error, making the online scenario approach the performance observed in the batch case.

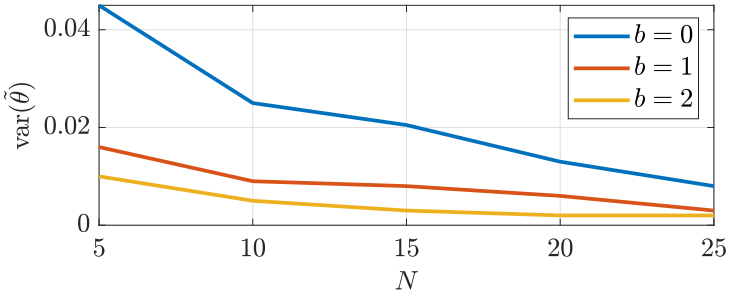


Figure 4.4: The variance of the estimate decreases with increasing b. The case $b = 0$ corresponds to when the teacher cannot assist the student.

cerebellum's involvement in motor control [125], and for predicting and treating neurological diseases [126].

In [127], Lourenço successfully apply this online correctional learning framework to estimate biological neural parameters by observing the behaviour of biological agents, such as animals and humans. However, it should be noted that this contribution falls outside the scope of this thesis and has therefore been omitted here. The interested reader is referred to the original publication [127] for the experimental setup and results.

4.5.4 Other Applications

The two previous examples demonstrate how the framework can be applied to situations involving observations sampled from a system or actions performed by

an agent. These settings extend to a large variety of problems, including assisted language learning or improved hypothesis testing, and bring together a variety of fields such as input design and active learning. When training neural networks, for example, correcting the inputs could be compared to input design methods presented in Section 4.1.1. For reinforcement learning tasks, a teacher could use online correctional learning to accelerate the learning of the student in real time. The framework can also be easily adopted to an adversarial setting, where the teacher finds the perturbation of the observations that maximizes the impact on the student's estimate – e.g. data poisoning [45, Section 6.1].

4.6 Chapter Summary

In this chapter, we extended the concept of correctional learning to an online setting. Specifically, we explored how the teacher can modify the student's observed data in real-time while operating under a budget constraint, aiming to improve the student's learning process. We derived a theoretical upper bound on the reduction in variance of the student's estimation error when the teacher provides assistance. We formulated an MDP to model the process and used dynamic programming to find the optimal correctional learning policy for an online setting.

We demonstrated the effectiveness of our approach through two examples. We showed the improvement in estimation when employing binomial and multinomial data, as depicted in Figures 4.2 and 4.4. Additionally, although outside the scope of this thesis, the framework has been applied to a biological parameter estimation scenario, highlighting its success in more complex settings, as demonstrated in the work [127].

Appendix - Chapter 4

4.A Bounding the Decrease in Variance

Let us first present two known lemmas used to solve Theorem 4.1.

Lemma 4.1. *Let X be a non-negative random variable (r.v.). Then,*

$$\mathbb{E}[X] = \int_0^\infty P(X \geq \tau) d\tau.$$

Proof. Let F be the CDF of X, i.e., $F(\tau) = P(X \leq \tau)$. Then, by integration by parts

$$\begin{aligned}
\mathbb{E}[X] &= \int_0^\infty \tau dF(\tau) = -\int_0^\infty \tau d[\underbrace{1 - F(\tau)}_{=P(X>\tau)}] \\
&= -\tau[1 - F(\tau)]\Big|_{\tau=0}^{\infty} + \int_0^\infty d[\underbrace{1 - F(\tau)}_{=P(X>\tau)}] \\
&= \int_0^\infty P(X > \tau) d\tau.
\end{aligned}$$

Note that $P(X \geq \tau) = P(X > \tau) + P(X = \tau)$, where $P(X = \tau) > 0$ for at most a countable number of values of τ, so

$$\int_0^\infty P(X = \tau) d\tau = 0$$

and

$$\mathbb{E}[X] = \int_0^\infty P(X \geq \tau) d\tau.$$

□

Lemma 4.2. *Let X be a r.v., and λ a constant. Then,*

$$\mathbb{E}[(X - \lambda)^2] = \int_0^\infty P(|X - \lambda| \geq \sqrt{\tau}) d\tau.$$

Proof. Use Lemma 4.1 with X replaced by $(X-\lambda)^2$. □

Let us now restate Theorem 4.1 and prove it in three parts.

Theorem 4.1. *Let $X_1, \dots, X_N$ be i.i.d. r.v.'s in $\{0, 1, \dots, M-1\}$ with mean μ, and $Y = X_1 + \cdots + X_N$. Let $b \in \{0, 1, \dots, N\}$, and*

$$\tilde{Y} = \underset{\{Z \in \{0,\dots,N\} : |Y-Z| \le b\}}{\arg\min} |Z - N\mu|.$$

Then, $\mathrm{var}(\tilde{Y}/N) \le M^2 \exp\left(-\frac{2b^2}{NM^2}\right)$. *Let us further assume that* $X_i \sim \mathbb{U}(\{0, \dots, M-1\})^\dagger$. *Then,*

$$\frac{\mathrm{var}(\tilde{Y}/N)}{\mathrm{var}(Y/N)} \le \frac{6M}{5M+1} \exp\left(-\frac{2b^2}{NM^2}\right).$$

Proof. Let us start by computing $\mathrm{var}(\tilde{Y})$. Using Hoeffding's inequality, we have that

$$\begin{aligned} P(|\tilde{Y} - N\mu| \ge \sqrt{\tau}) &= P(\tilde{Y} \ge N\mu + \sqrt{\tau}) + P(\tilde{Y} \le N\mu - \sqrt{\tau}) \\ &= P(Y \ge N\mu + \sqrt{\tau} + b) + P(Y \le N\mu - \sqrt{\tau} - b) \\ &\le 2\exp\left(-\frac{2(\sqrt{\tau}+b)^2}{NM^2}\right). \end{aligned} \tag{4.25}$$

Therefore, from Lemma 4.2,

$$\begin{aligned} \mathrm{var}(\tilde{Y}) = \mathbb{E}[(X-\lambda)^2] &\le 2\int_0^\infty \exp\left(-\frac{2(\sqrt{\tau}+b)^2}{NM^2}\right) d\tau \\ &= 2\int_0^\infty \exp\left(-\frac{2u^2}{NM^2}\right) \cdot 2(u-b)du \\ &= 4\int_0^\infty u \exp\left(-\frac{2u^2}{NM^2}\right) du - 4b\int_0^\infty \exp\left(-\frac{2u^2}{NM^2}\right) du \\ &\le 4\int_{\frac{2b^2}{NM^2}}^\infty \frac{NM^2}{4} e^{-v} dv = NM^2 \exp\left(-\frac{2b^2}{NM^2}\right). \end{aligned} \tag{4.26}$$

[†]This assumption is not crucial, but provides a special case that is easier to understand.

Let us now compute $\mathrm{var}(Y)$. If $X_i \sim \mathbb{U}(\{1, \dots, M-1\})$, then

$$\begin{aligned}
\mathrm{var}(Y) &= \sum_{k=0}^{M} \left(k - \frac{M}{2}\right)^2 \frac{1}{M+1} \\
&= \frac{1}{M+1} \sum_{k=0}^{M} (k^2 - kM + M^2) \\
&= \frac{1}{M+1} \left(M^2(M+1) - M\frac{M(M+1)}{2} + \frac{1}{6}M(M+1)(2M+1)\right) \\
&= M^2 - \frac{M^2}{2} + \frac{1}{6}M(2M+1) = \frac{5M^2+M}{6}.
\end{aligned} \tag{4.27}$$

Finally, we conclude that

$$\frac{\mathrm{var}(\tilde{Y}/N)}{\mathrm{var}(Y/N)} \leq \frac{M^2 \exp\left(-\frac{2b^2}{NM^2}\right)}{\frac{5M^2+M}{6}} = \frac{6M}{5M+1} \exp\left(-\frac{2b^2}{NM^2}\right). \tag{4.28}$$

□

Chapter 5

Optimal Transport for Correctional Learning

In the previous chapter, we introduced online correctional learning as an effective approach for parameter estimation in system identification problems. In this chapter, we return to the offline setting and generalize the original formulation by incorporating tools from optimal transport. This generalization improves the framework by enabling the estimation of more complex parameters, and makes it more versatile by accommodating multiple intervention strategies for the teacher.

5.1 Introduction

Correctional learning has shown promising results in both offline and online settings, as demonstrated in [52] and Chapter 4, respectively. Nevertheless, the framework still suffers from some limitations. First of all, the derived performance guarantees hold only for simple systems. To describe real-world phenomena, however, one typically requires more complex distributions. For example, a Gaussian distribution can be used to describe biological data such as the heights of people. To model the probability of failure of an appliance, we can use a Weibull distribution. Another disadvantage is that the teacher's policy follows explicitly from the solution, leaving no room for alternative intervention strategies to be considered.

In this chapter, we introduce an alternative approach to correctional learning using tools from optimal transport [80]. Optimal transport is a mathematical framework concerned with finding the most efficient way to transport mass from one location to another, according to some cost function. Historically, optimal transport has been widely used in finance and logistics [128], but recent advances have made it an increasingly popular tool in fields such as systems, control and estimation [82, 129]. In machine learning, optimal transport has found use in a number of applications, including shape reconstruction [130], multi-label classification [131], and brain decoding [132]. Moreover, recent work in robotics demonstrates how

optimal transport can be applied to mapping problems to enable robots to operate in new environments [133], and for policy fusion in reinforcement learning to speed up the process of a robot learning a new task [134].

In the context of correctional learning, we note that the optimal corrections can be viewed as a transportation of probability mass from an initial distribution into a target distribution. Furthermore, by assuming that the estimator depends on the samples only through their empirical measure, we can pose the correctional learning problem as an optimization program in terms of distribution functions – i.e., as an optimal transport problem. In contrast to [52] and our online approach in Chapter 4, this novel formulation considers the samples implicitly through their distribution, which not only enables the estimation of more complex parameters, but also allows for the consideration of alternative intervention strategies.

The main contributions of this chapter are:

- *A generalized correctional learning framework:* we leverage the principles of optimal transport and propose a novel formulation of correctional learning. With this new framework, we can expand the range of applications to consider more sophisticated tasks that involve complex systems.

- *Multiple teacher policies:* in standard learning settings, a teacher agent may exhibit several intervention strategies. We show how our new formulation allows for the consideration of multiple teacher policies to fit different tasks.

- *Evaluation of performance:* we demonstrate the benefits of our optimal-transport approach by applying the framework on three different test cases. Specifically, we show how the framework can be used to estimate the parameters of more complex distributions such as the Gaussian and the Weibull. We also apply the framework to update a robot's reward function in an inverse reinforcement learning setting.

5.1.1 Related Work

In the context of system identification and estimation, our framework can be placed around other works that also use tools from optimal transport. For example, in [46], the authors use optimal transport for state tracking of linear ensembles. More specifically, they propose an optimal-transport approach for estimating the states of multiple subsystems based on their joint output. Other related works include [135], where they propose an optimal transport formulation of the ensemble Kalman filter, and [136], where the authors study the use of optimal transport distances as objective functions for parameter estimation in dynamical systems.

Beyond system identification, optimal transport is being increasingly applied to problems in the fields of data science and machine learning. While this work may not fall under the learning category in the stricter sense, its connection to this area is still worth pointing out. For brevity, we will omit further discussion on this, and instead point the interested reader to [137] for an excellent overview of the topic.

5.2 Preliminaries

In this section, we provide a short summary of the correctional learning formulation discussed in Section 2.5, followed by a brief introduction to optimal transport.

5.2.1 Correctional Learning

Consider a model of some data-generating system parameterized by the unknown parameter $\theta \in \Theta$. Let the true system correspond to the value θ_0. In a standard parameter estimation setting, a learner (student) agent aims to estimate θ_0 as $\hat{\theta}$, based on a sequence of observations sampled from the system, $\mathcal{O} = \{y_1, \dots, y_N\}$, distributed according to $p_0^N \in \mathbb{M}_+(\mathcal{Y}^N)$, where $y_k \in \mathcal{Y} \subseteq \mathbb{R}^d$, and $(\mathcal{Y}, \mathcal{B})$ is a measurable space. That is,

$$\hat{\theta} = f(\mathcal{O}) = f(y_1, \dots, y_N), \tag{5.1}$$

where $f\colon \mathcal{Y}^N \to \Theta$ is some estimator function.

In the correctional learning framework, an expert (teacher) agent is introduced to help the the student in its estimation process. The teacher may do so by modifying the original observation sequence $\mathcal{O}$, into a sequence $\tilde{\mathcal{O}}$ that better represents the true characteristics of the system. The modified sequence is then passed on to the student, who forms the altered estimate $\tilde{\theta}$.

However, utilizing expert knowledge might be expensive or limited. The number of allowed interventions might also be restricted for privacy preserving reasons – the more observations the teacher changes, the more likely it is to be discovered. To account for this, the teacher is constrained to not exceed a certain intervention budget b. If $c\colon \mathcal{Y}^N \times \mathcal{Y}^N \to \mathbb{R}_+$ denotes a distance measure between two sequences of N elements, then the teacher must satisfy

$$c(\mathcal{O}, \tilde{\mathcal{O}}) \leq b. \tag{5.2}$$

The cost may be chosen to be any distance metric, e.g. the ℓ_1-norm for discrete observations.

The goal of the teacher agent is to find the optimal modified sequence that minimizes the student's estimation error

$$V(\theta_0, \tilde{\theta}) = \|\theta_0 - \tilde{\theta}\|. \tag{5.3}$$

where $|\cdot|$ is a norm on Θ.

Depending on the setting, this problem can be posed and solved in different ways: see Section 2.5 and Chapter 4 for correctional learning in a batch setting and online setting, respectively. A schematic view of a general correctional learning framework is provided in Figure 5.2.1.

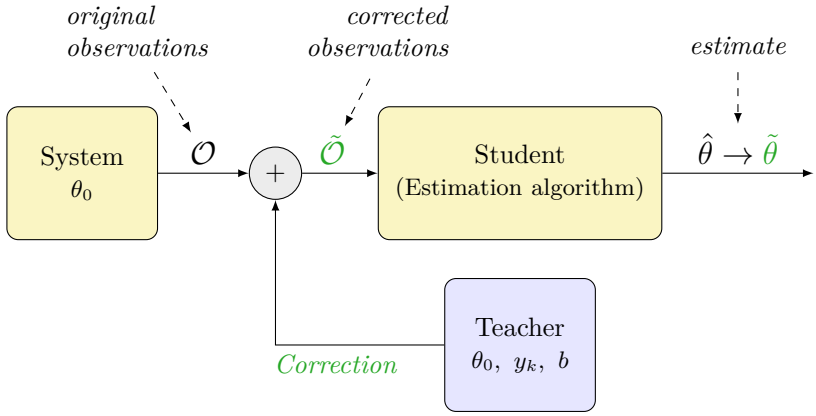


Figure 5.2.1: A schematic view of the correctional learning framework. The teacher knows the true parameter value θ_0 and the original samples y_k. The teacher modifies the original sequence of observations $\mathcal{O}$ into $\tilde{\mathcal{O}}$ by changing at most b samples.

5.2.2 Optimal Transport

Optimal transport is a mathematical framework for finding the most cost-efficient way of transporting (probability) mass from one location to another. Assume, for example, that we have a probability measure $\mu \in \mathbb{M}_+(\mathcal{X})$ that we wish to transform into another probability measure $\nu \in \mathbb{M}_+(\tilde{\mathcal{X}})$, by "moving" small chunks of probability mass with minimal transportation cost.

The cost of transporting one unit of probability mass from location x to location $\tilde{x}$ is quantified by a metric $\tau(x, \tilde{x})$ on $\mathcal{X}$. To compute the total transportation cost, we first define a transportation map $\tau \in \mathbb{M}_+(\mathcal{X} \times \tilde{\mathcal{X}})$, where $d\tau(x, \tilde{x})$ denotes the amount of mass transferred from x to $\tilde{x}$. The objective is to find the optimal transportation map τ that minimizes the total cost. This can be achieved by solving the optimal transport problem

$$\begin{aligned} \min_{\tau \in \mathbb{M}_+(\mathcal{X} \times \tilde{\mathcal{X}})} \quad & \int_{\mathcal{X} \times \tilde{\mathcal{X}}} \tau(x, \tilde{x}) d\tau(x, \tilde{x}) \\ \text{s.t.} \quad & \int_{\tilde{x} \in \tilde{\mathcal{X}}} d\tau(x, \tilde{x}) = d\mu(x), \\ & \int_{x \in \mathcal{X}} d\tau(x, \tilde{x}) = d\nu(\tilde{x}). \end{aligned} \tag{5.4}$$

Here, the two constraints ensure that we do not move more mass than we originally have.

5.3 Correctional Learning as an Optimal Transport Problem

In this section we present the main contribution of this chapter: an optimal transport formulation of the original batch correctional learning problem detailed in Section 2.5. We demonstrate that by using optimal transport, we are able to create a more general framework that enables the estimation of more complex parameters.

5.3.1 General Problem Formulation

Recall the problem setup of correctional learning in Section 5.2.1. In order to establish its connections with optimal transport, we now reframe the formulation in a more general context, emphasizing its relationship with optimal transport.

Assume that the data-generating system is permutation-invariant, or *exchangeable*, in the sense that the distribution p_0 of the samples $\mathcal{O}$ it generates does not change if the samples in $\mathcal{O}$ are permuted (in a deterministic manner). It is then natural to consider estimators that are also permutation-invariant, i.e., that can be described as some function of the samples' empirical measure:

$$\hat{\theta}_N(y_1, \dots, y_N) = J(\hat{p}(y_1, \dots, y_N)), \tag{5.5}$$

where $\hat{p}\colon \mathcal{Y}^N \to \mathbb{M}_+(\mathcal{Y})$ is the empirical measure of the samples, defined as

$$(\hat{p}(y_1, \dots, y_N))(A) := \sum_{k=1}^{N} \frac{1}{N} \mathbb{1}\{y_k = A\}, \quad A \in \mathcal{B}. \tag{5.6}$$

The function $J\colon \mathbb{M}_+(\mathcal{Y}) \to \Theta$ is a fixed function (i.e., independent of N), which we will later assume to be Fréchet-differentiable.

Recall that the samples can be perturbed by the teacher before they reach the student. In the batch setting, the teacher has access to all of the original samples $\mathcal{O}$ before perturbing them into $\tilde{\mathcal{O}} = \{\tilde{y}_1, \dots, \tilde{y}_N\}$, where $\tilde{y}_k \in \tilde{\mathcal{Y}} \subseteq \mathbb{R}^d$. The teacher is subject to a budget constraint, namely

$$\sum_{k=1}^{N} c(y_k, \tilde{y}_k) \leq b. \tag{5.7}$$

The goal of the teacher is still to modify the original sequence $\mathcal{O}$, subject to (5.7), in order to minimize the estimation error

$$\|\theta_0 - \tilde{\theta}\|, \tag{5.8}$$

where $\tilde{\theta}$ is the altered estimate based on $\tilde{\mathcal{O}}$.

Since the estimator in (5.5) depends on the samples only through their empirical distribution, it makes sense to pose the optimization problem to be solved by the

teacher in terms of distribution functions, i.e., as an optimal transport problem

$$\min_{p} \quad \left\| \theta_0 - J\left(\int_{y \in \mathcal{Y}} dp(y, \cdot) \right) \right\|^2 \tag{5.9a}$$

$$\text{s.t.} \quad \int_{(y,\tilde{y}) \in \mathcal{Y} \times \tilde{\mathcal{Y}}} c(y, \tilde{y}) dp(y, \tilde{y}) \leq \frac{b}{N} \tag{5.9b}$$

$$\int_{\tilde{y} \in \tilde{\mathcal{Y}}} \int_{y \in A} dp(y, \tilde{y}) = (\hat{p}(y_1, \ldots, y_N))(A), \quad \forall A \in \mathcal{B}. \tag{5.9c}$$

Here, $p \in \mathbb{M}_+(\mathcal{Y} \times \tilde{\mathcal{Y}})$ is a transportation map that represents the joint measure of the original and modified samples, and $\hat{p}_N$ the empirical distribution of the original samples. It is important to note that this problem is generally infinite-dimensional with linear constraints. However, the function J is not necessarily linear in general. If J is Fréchet-differentiable, and we assume that the budget b is "small" (in the sense that most of the original samples will not be modified), one can use the Taylor approximation of the cost of the modified samples,

$$\begin{aligned} J\left(\int_{y \in \mathcal{Y}} dp(y, \cdot) \right) \approx J(\hat{p}) &+ \int_{y \in \mathcal{Y}} \int_{\tilde{y} \in \tilde{\mathcal{Y}}} (\nabla J(\hat{p}))(\tilde{y}) dp(y, \tilde{y}) \\ &- \int_{y \in \mathcal{Y}} (\nabla J(\hat{p}))(y) d\hat{p}(y), \end{aligned} \tag{5.10}$$

We denote this Taylor approximation by $J_{\text{TA}}(\mathcal{O})$. Substitution into (5.9a) then yields the objective

$$\min_{p} \|\theta_0 - J_{\text{TA}}(\mathcal{O})\|^2. \tag{5.11}$$

Remark 5.1. *Note how the constraints in* (5.9) *now consider the distribution of the samples. This is in contrast to the original formulation in Section 2.5, where each observation is considered individually.*

5.3.2 Discretization of the Continuous Case

One of the most common approaches to solve the optimal transport problem in (5.9) is to discretize it [82]. For simplicity, we will assume that the original samples are independent and identically distributed, with distribution p_0. We start by defining a discretized sample space. Recall that our observation sequence is given by the *multiset*[1]

$$\mathcal{O} = \{y_1, \ldots, y_N\}. \tag{5.12}$$

We can let the *set* of unique values of $\mathcal{O}$ constitute our discretized sample space as

$$\mathcal{S} = \bigcup_{o \subseteq \mathcal{O}} o = \{s_1, \ldots, s_n\} \subseteq \mathcal{O}. \tag{5.13}$$

[1] A sample may occur multiple times in the sequence.

For the continuous case, we note that with probability one,

$$\mathcal{S} = \mathcal{O} \quad \text{and} \quad n = \text{card}(\mathcal{S}) = \text{card}(\mathcal{O}) = N, \tag{5.14}$$

since all the samples in $\mathcal{O}$ are distinct, with probability one. The elements in $\mathcal{S}$ will be called *states*.

Remark 5.2. *We note that there are other methods to determine the states. For instance, they may be fixed to belong to some pre-determined set of values. However, with regards to the nature of the framework of modifying an* observed *sequence, we believe that the suggested approach is reasonable.*

Furthermore, the teacher will be allowed to change the observations in $\mathcal{O}$ into samples from the set

$$\tilde{\mathcal{S}} = \{\tilde{s}_1, \ldots, \tilde{s}_m\}, \tag{5.15}$$

which may or may not coincide with $\mathcal{S}$. Note that $\text{card}(\tilde{\mathcal{S}}) = m$.

We now continue by discretizing (5.9). We note that both the objective in (5.10) as well as our constraints in (5.9) include our decision variable $dp(y, \tilde{y})$ in the integrals, which we cannot sample from. Thus, to approximate the integrals, we use techniques from importance sampling [138]. We consider the proposal distribution

$$d\mu(y, \tilde{y}) = dq(y)dr(\tilde{y}), \tag{5.16}$$

where $dq(y)$ and $dr(\tilde{y})$ are probability measures defined on $\mathcal{S}$ and $\tilde{\mathcal{S}}$, respectively. Note that, for simplicity, μ has been chosen in terms of independent proposal distributions for y and $\tilde{y}$. With this distribution, and restricting p to $\mathcal{S} \times \tilde{\mathcal{S}}$, we rewrite the estimator in (5.10) as

$$\begin{aligned} J_{\text{TA}}(\mathcal{O}) = J(\hat{p}) \\ + \int_{y \in \mathcal{S}} \int_{\tilde{y} \in \tilde{\mathcal{S}}} (\nabla J(\hat{p}))(\tilde{y}) \frac{dp(y, \tilde{y})}{dq(y)dr(\tilde{y})} dq(y)dr(\tilde{y}) \\ - \int_{y \in \mathcal{S}} (\nabla J(\hat{p}))(y) d\hat{p}(y). \end{aligned} \tag{5.17}$$

Using importance sampling, we can discretize the expression in (5.17) as

$$\begin{aligned} J_{\text{TAD}}(\mathcal{O}) = J(\hat{p}) \\ + \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{\partial J}{\partial p_{\tilde{y}_j}}(\hat{p}) \frac{dp(y_i, \tilde{y}_j)}{dq(y_i)dr(\tilde{y}_j)} \\ - \frac{1}{m} \sum_{j=1}^{m} \frac{\partial J}{\partial \tilde{y}_j}(\hat{p}), \end{aligned} \tag{5.18}$$

where we use numerical differentiation to approximate the gradient ∇J. To simplify the notation, we define

$$\alpha \in \mathbb{R}^{n \times m} : \ \alpha_{ij} = \frac{dp(y_i, \tilde{y}_j)}{dq(y_i)dr(\tilde{y}_j)} \tag{5.19}$$

to be our new decision variable. Our objective in (5.11) can then be written as

$$\min_{\alpha} \ \|\theta_0 - J_{\text{TAD}}(\mathcal{O})\|^2. \tag{5.20}$$

We discretize the budget constraint (5.9b) as

$$\frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} c(y_i, \tilde{y}_j)\alpha_{ij} \leq \frac{b}{N}. \tag{5.21}$$

To discretize the constraint in (5.9c), we first utilize the same trick as before and rewrite it as

$$\int_{x \in \mathcal{S}} \frac{dp(y, \tilde{y})}{dq(y)dr(\tilde{y})} dr(\tilde{y}) = \frac{d\hat{p}(y)}{dq(y)}. \tag{5.22}$$

We discretize this as

$$\frac{1}{m} \sum_{j=1}^{m} \frac{dp(y, \tilde{y}_j)}{dq(y_i)dr(\tilde{y}_j)} = \frac{d\hat{p}(y)}{dq(y_i)}. \tag{5.23}$$

Again, since we cannot sample from $p(y, \tilde{y})$, we simply say that the above relation must hold for all values of y, i.e.,

$$\frac{1}{m} \sum_{j=1}^{m} \underbrace{\frac{dp(y_i, \tilde{y}_j)}{dq(y_i)dr(\tilde{y}_j)}}_{\alpha_{ij}} = \frac{d\hat{p}(y_i)}{dq(y_i)}, \quad \forall i. \tag{5.24}$$

All constraints are now written in terms of our new decision variables α_{ij}.

5.3.3 Importance Sampling: Different Approaches

Consider the case when $\tilde{\mathcal{S}} = \mathcal{S}$, and $dq = dr = \hat{p}$. This means that we we will work directly with the observed samples, and the constraint in (5.24) then simplifies to

$$\frac{1}{m} \sum_{j=1}^{m} \frac{dp(y_i, \tilde{y}_j)}{dq(y_i)dr(\tilde{y}_j)} = \frac{d\hat{p}}{dq}(y_i) = \frac{d\hat{p}}{d\hat{p}}(y_i) = 1, \quad \forall i. \tag{5.25}$$

We note that this case is very similar to a discrete setting in the sense that we are limiting the teacher to change the observations into values that have already been seen or encountered. This approach is similar to other resampling techniques and can be viewed as a way of re-weighting the samples to change their importance for the estimation.

We can also sample from dq and dr independently, with $dq \neq dr$. Using this approach, we can impose some prior knowledge on dr, either by defining it to be the true distribution, or some distribution that will yield a more accurate estimate of the parameter we are interested in. However, using this approach, we would not be working directly with the empirical distribution, which means that we would have to perform an interpolation step to figure out how to best change the actual observations. We would also have to use a density estimation technique to enforce the constraint in (5.9c).

5.3.4 Modifying the Sequence

Next we describe how the teacher modifies the sequence based on the α obtained from solving (5.20) with respect to the constraints (5.21) and (5.24). Recall the definition of α in (5.19), and that $dp(y, \tilde{y})$ denotes the amount of probability mass transferred from y to $\tilde{y}$. Then, by applying Bayes' theorem, we compute the *Conditional Probability Mass Function (cpmf)* as

$$p(\tilde{y} \mid y) = \alpha \hat{p}(\tilde{y}), \tag{5.26}$$

which will give us the probability of changing an observation y into $\tilde{y}$.

The teacher's intervention procedure is then as follows. For each sample in $y_k \in \mathcal{O}$, the teacher modifies it according to the cpmf in (5.26). That is,

$$y_k \;\rightarrow\; \tilde{y}_k, \quad \text{where } \tilde{y}_k \sim p(\tilde{y} \mid y_k). \tag{5.27}$$

To ensure that the intervention budget is not exceeded, the teacher generates M_s new sequences. For each new generated sequence that satisfy the budget constraint (i.e., for which the number of corrections is less than or equal to b), an updated estimate is computed. Out of these estimates, the one yielding the lowest estimation error is then chosen to be the optimal one. Should the teacher fail to find a sequence that both improves the estimate and satisfies the budget constraint, it will keep the original sequence.

Naturally, the teacher may follow different intervention policies. Alternative approaches may include changing one sample at a time and then re-solve for a new α following each update. This policy is similar to receding horizon control strategies where we may interpret the budget to be the horizon [61].

Another possible strategy would be for the teacher to always make the change with the highest probability. This would make for a greedy approach [139].

5.4 Numerical results

In this section, we evaluate our framework in three different settings; two theoretical and one applied. For simplicity, we consider $\mathcal{Y} \in \mathbb{R}$ and $\theta_0 \in \mathbb{R}$ in all settings. We evaluate the performance in terms of the absolute error, i.e., $e = |\theta_0 - \tilde{\theta}|$.

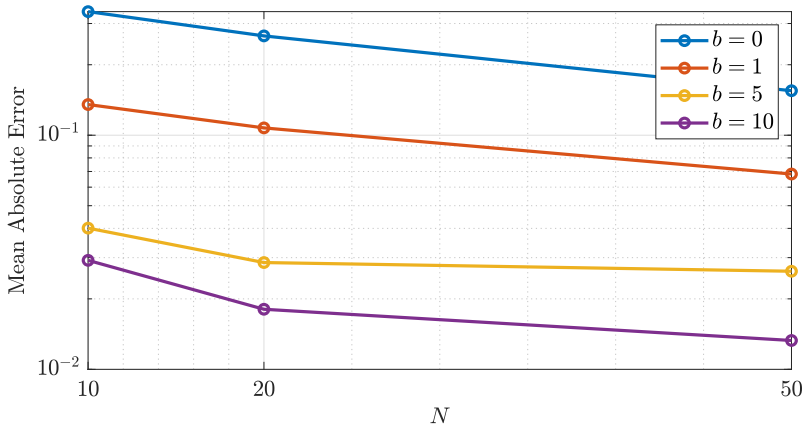


Figure 5.4.1: The absolute estimation error averaged over 100 Monte Carlo simulations for increasing sample sizes and budgets. Note that $b = 0$ corresponds to the case with no teacher intervention.

5.4.1 Variance Estimation of a Gaussian Distribution

In the first experiment, we use the framework to estimate the variance of a Gaussian distribution. We consider the observations to be sampled from the distribution $\mathcal{N}(0, 1)$, so $\theta_0 = \sigma^2 = 1$.

We perform the estimation on three sample sizes, $N = \{10, 20, 50\}$, subject to four different intervention budgets, $b = \{0, 1, 5, 10\}$. In this example, we use a uniform transportation cost, i.e., $c(y, \tilde{y}) = \mathbb{1}\mathbb{1}^T - I$. This means that all changes made are equally expensive. For each sample size and budget, we perform the experiment 100 times and compute the average absolute error. For all configurations, we use $M_s = 1000$. The results are shown in Figure 5.4.1. As expected, the plot shows a decrease in the estimation error as the sample size increase. It also shows that the error is further decreased as the budget increases.

5.4.2 Scale Estimation of a Weibull Distribution

Next we apply the framework to estimate the scale parameter of a Weibull distribution. The probability density function is given by

$$f(x) = \begin{cases} \frac{\epsilon}{\lambda}\left(\frac{x}{\lambda}\right)^{\epsilon-1} e^{-(x-\lambda)^{\epsilon}}, & x \geq 0 \\ 0, & x < 0, \end{cases} \tag{5.28}$$

where $\epsilon > 0$ is called the shape parameter, and $\lambda > 0$ the scale parameter. In this example, we will consider the estimation of λ of a Weibull distribution with $\theta_0 = \lambda_0 = 2$ and $\epsilon = 8$.

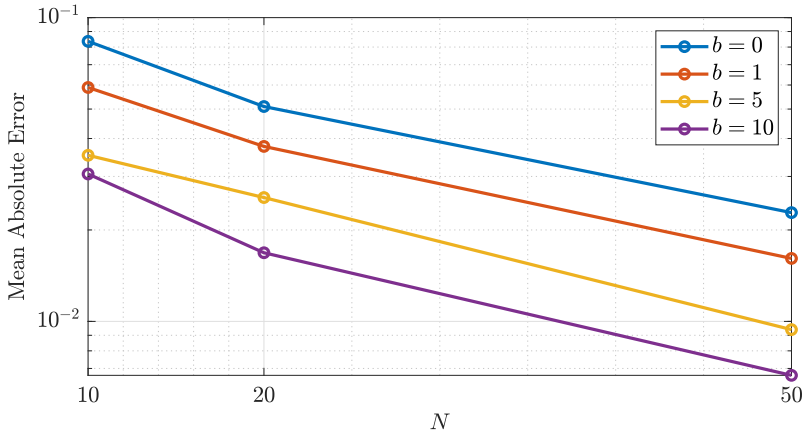


Figure 5.4.2: The absolute estimation error averaged over 100 Monte Carlo simulations for increasing sample sizes and budgets. Note that $b = 0$ corresponds to the case with no teacher intervention.

There are different approaches available for estimating λ, see e.g. [140]. In this experiment, we use the Bayesian two-stage approach derived in [141]. We use the proportional cost

$$c(y, \tilde{y}) = 10 \times \lceil |\tilde{y} - y| \rceil, \tag{5.29}$$

where $\lceil \cdot \rceil$ denotes the ceiling function. As in the previous example, we run the estimation process on the sample sizes $N = \{10, 20, 50\}$ and for the intervention budgets $b = \{0, 1, 5, 10\}$. We use $M_{\text{s}} = 2000$ for all configurations. The averaged estimation errors are shown in Figure 5.4.2. The results are similar to what we observed in the previous experiment, with an improved estimation error for increasing sample sizes and budgets.

5.4.3 Reward Estimation in Inverse Reinforcement Learning

As a final example, we apply our framework to update a robot's reward function in an inverse reinforcement learning setting. Recent work on learning from human interaction shows how physical corrections made by a human (e.g. in the form of applied torque) can improve a robot's learning process [142]. Inspired by their problem setup, we apply our framework in a similar setting.

Consider a robot arm being tasked with moving a coffee cup from one side of a table to the other. To learn the task, the robot gets to observe a set of N trajectories, $\{\xi_1, \ldots, \xi_N\}$, demonstrated by a human. Figure 5.4.3 illustrates some examples of trajectories demonstrated on a robotic arm with seven degrees of freedom implemented in PyBullet. Each trajectory is associated with a total

feature count for each feature $i \in [1, n_\mathrm{f}]$

$$\Phi_i(\xi) = \sum_{x \in \xi} \phi_i(x), \tag{5.30}$$

where $\phi_i(x)$ is the local feature value in a point x along the trajectory ξ. A high feature value corresponds to a good position in space. The features represent different subgoals in performing the task, such as "stay nearby the top of the table" and "avoid the laptop". Based on these features, the robot learns a reward function

$$R = \Theta^T \Phi = \theta_1 \Phi_1 + \dots \theta_n \Phi_n, \tag{5.31}$$

where the weights Θ represent the importance of each feature to the human.

Assume now that the robot has learned a reward function based on the following observations collected over $N = 5$ trajectories with $n_\mathrm{f} = 3$ features

$$\begin{cases} \mathbf{\Phi_1} = \{\Phi_1(\xi_k)\}_{k=1}^5 = \{100,\ 75,\ 50,\ 20,\ 5\} \\ \mathbf{\Phi_2} = \{\Phi_2(\xi_k)\}_{k=1}^5 = \{90,\ 200,\ 10,\ 2,\ 30\} \\ \mathbf{\Phi_3} = \{\Phi_3(\xi_k)\}_{k=1}^5 = \{50,\ 20,\ 3,\ 5,\ 10\} \end{cases} . \tag{5.32}$$

An expert may then apply our framework to improve the robot's learned θ_i's, by modifying the sets $\mathbf{\Phi}_i$ in (5.32) into $\tilde{\mathbf{\Phi}}_i$. As estimator, we consider a slightly modified version of the weight update in [142]:

$$\tilde{\theta}_i = \hat{\theta}_i + \beta \left(\sum_{\Phi \in \mathbf{\Phi}} \Phi - \sum_{\tilde{\Phi} \in \tilde{\mathbf{\Phi}}} \tilde{\Phi} \right), \tag{5.33}$$

where $\beta < 0$ is a step/scaling parameter. Here, we consider $\beta = -0.001$. Note how the feature weights are updated based on the direction of change of the feature values between the original and the modified trajectories. If the altered corrections pass further away from, say, the laptop, the θ_i corresponding to the distance-to-laptop feature will increase.

For this experiment we used $c(y, \tilde{y}) = \mathbb{1}\mathbb{1}^T - I$, $b = 1$, and $M_\mathrm{s} = 1000$. The corrected feature values together with their corresponding updated weight estimate are shown in Table 5.4.1. For reference, we also we also present the true weights and the initial estimates in the same table. The results show that the updated estimates are closer to the true values, compared to the initial estimates.

5.5 Chapter Summary

In this chapter, we presented a generalized formulation of the batch correctional learning framework using optimal transport. We demonstrated that by expressing the correctional learning problem as an optimization program in terms of distribution functions, we obtain a more general and flexible framework better suited for

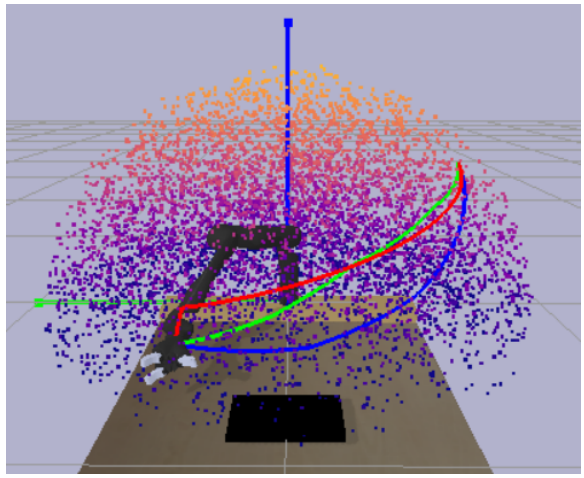


Figure 5.4.3: Here, the robot observes three trajectories with different feature values. The expert may alter some of them to the one that is closer to its preferences. For example, if the robot should avoid the laptop, the expert may change the blue trajectory into the red one, to reflect this.

Table 5.4.1: The corrected feature values and their corresponding estimates. The corrected values are underlined.

True weight	Old estimate	New estimate	Corrected feature values
$\theta_1 = 0.1$	$\hat{\theta}_1 = 0.5$	$\tilde{\theta}_1 = 0.05$	$\tilde{\mathbf{\Phi}}_1 = \{100,\ 75,\ 50,\ 20,\ \underline{50}\}$
$\theta_2 = 1$	$\hat{\theta}_2 = 0.5$	$\tilde{\theta}_2 = 1.1$	$\tilde{\mathbf{\Phi}}_2 = \{\underline{30},\ 200,\ 10,\ 2,\ 30\}$
$\theta_3 = 0.8$	$\hat{\theta}_3 = 0.5$	$\tilde{\theta}_3 = 0.8$	$\tilde{\mathbf{\Phi}}_3 = \{\underline{20},\ 20,\ 3,\ 5,\ 10\}$

estimation of more complex characteristics. We successfully applied the framework on three estimation processes; for the variance estimation of a Gaussian, the scale estimation of a Weibull, and, finally, in an inverse reinforcement setting where we improved a robot's reward function.

Chapter 6

Conclusions and Future Work

This thesis focused on learning-based approaches for different components of the control loop, ranging from neural network-based model predictive control, to cooperative learning for system identification. Augmenting more traditional methods with machine learning tools is pivotal for the further development of fast, efficient and robust control strategies. We conclude this thesis by highlighting our main contributions to this domain and discuss interesting future research directions for its continued exciting evolution.

In Chapter 3, we presented a framework for offline training and evaluation of a neural network for implementing MPC using gradient data. Our primary focus was on investigating the possibility of replacing online MPC optimization solvers with pre-trained neural networks, as such a transition holds the promise of achieving highly efficient and robust real-time implementations.

The main idea is to approximate the MPC mapping from state to control input with a constrained ReLU-based neural network equipped with a convex optimization projection layer to ensure recursive feasibility and asymptotic stability of the resulting trajectories. The main novel aspect lies in incorporating the gradient of the MPC control law with respect to the state input during training of the neural networks.

We used MPT3 [68] and PyTorch [104] to implement the framework. We also used MPT3 to generate training data and to evaluate the resulting controller. A critical element revolves around the generation of training samples, which is related to input design in system identification, see e.g. [143]. This is particularly challenging when dealing with large state spaces. Here we proposed to use a HAR sampler for data generation. We evaluated the resulting controllers based on generated trajectories and normalized cost-functions. The numerical experiments showed the trade-off between the amount of training data and the approximation qualities of the resulting controller.

This framework serves as a first step towards controller identification of MPC using ReLU-networks trained on gradient data. It should be noted that our ap-

proach does not assume any model information beyond the state, control signal and gradient during training. Our framework is therefore not restricted to consider only eMPC or time-invariant MPC problems. Looking ahead, including more model-specific data in the form of model parameters and their respective gradients, or complete control trajectories, in the training data could possibly enhance the framework's effectiveness. However, this would increase the complexity and the need for even more structured training data generation. In addition, extending and evaluating the framework on more advanced MPC problems, including non-linear MPC, presents an interesting direction for future research. The numerical examples provided in this chapter establish a proof of concept; depending on the nature and complexity of the MPC problem, additional challenges will need to be addressed in the future. We believe that this gives a consistent way of approaching the training aspect. Other interesting future directions include adjusting the loss function to account also the the control cost, and not just the NMSE, and exploring whether would it be more meaningful to incorporate the temporal aspect actively into the training process. That is, rather than randomly sampling isolated training datapoints $\{x, u\}$ from the state space, we would consider sampling datapoints from full trajectories.

Chapter 4 introduces an online formulation of the batch correctional learning framework, intially presented in Section 2.5. Inspired by cooperative (learning) problems, correctional learning was developed to tackle challenges associated with unrepresentative data in system identification problems. It offers an alternative strategy for incorporating external or prior knowledge into the estimation process, especially in cases where direct knowledge transmission between agents might be undesirable or even impossible.

The core of the correctional learning framework centers around a teacher-student model. In this setup, an expert (teacher) agent seeks to assist a learner (student) agent in its estimation process by transferring its knowledge in the space of induced probability distributions. In the original offline setting, the teacher analyses and modifies entire batches of sampled data before passing them on to the student. In this process, the teacher is constrained to a fixed intervention budget, reflecting the limited communication between the two agents. However, many real-life processes often demand immediate action as data arrives sequentially. Thus, we identified a need for an *online* counterpart of the framework, where the teacher evaluates (and modifies) each incoming sample immediately and individually, while operating under the same budget constraint.

The main result of this chapter is the formulation of the online correctional learning problem as an MDP. This formulation can readily be adapted to solve many different tasks, simply by adjusting the reward function of the MDP to align with specific task objectives. We employ dynamic programming techniques to determine the opimtal correctional learning policy for the teacher.

As part of our work, we established a theoretical upper bound on the reduction in variance of the student's estimation error in the case of multinomial data, with teacher assistance. Furthermore, we demonstrated the effectiveness of our approach

through three examples. We showcased improved estimation performance when employing binomial and multinomial data. Addtionally, we applied the framework to a biological parameter estimation scenario, demonstrating its success in more complex settings.

This work opens the door to extending our method to various interesting applications, such as correctional reinforcement learning, where the teacher assists the student agent in learning optimal policies. Furthermore, a comparative analysis against related approaches will provide valuable insights. Overcoming the dimensionality problems posed by MDPs will be an important step along the way towards broader applicability and scalability of our proposed framework.

In Chapter 5, we presented a generalized formulation of the batch correctional learning framework using optimal transport. Our novel approach offers significant advancements compared to the existing correctional learning formulations, which exhibit limitations in estimating intricate characteristics and considering alternative intervention strategies.

Our main contribution in this chapter lies in leveraging tools from optimal transport – a mathematical framework for finding the most efficient way to transport mass, according to some cost function. By viewing the teacher's optimal corrections as a form of probability mass transportation, we framed the correctional problem as an optimization program in terms of distribution functions – i.e., as an optimal transport problem. In contrast to previous formulations, this approach considers the samples implicitly through their distribution, thereby enabling the estimation of more complex parameters and the exploration of different teacher policies.

We demonstrated the effectiveness of this formulation through three estimation scenarios: variance estimation of a Gaussian, scale estimation of a Weibull, and in an inverse reinforcement learning setting where we improved a robot's reward function. This novel optimal-transport formulation opens up for several interesting extensions, such as employing correctional learning for differential privacy, handling time-series data, and for balancing biased or skewed learning datasets. Addressing the curse of dimensionality of the discretized problem will be an important step along the way.

References

[1] R. Gross, *Psychology: The Science of Mind and Behaviour*. London, England: Hodder Education, 6 ed., Apr. 2010.

[2] "learn, v.," in *OED Online*, Oxford University Press, Oct. 2021.

[3] A. M. Ozbayoglu, M. U. Gudelek, and O. B. Sezer, "Deep learning for financial applications : A survey," *Applied Soft Computing*, vol. 93, p. 106384, 2020.

[4] R. C. Cavalcante, R. C. Brasileiro, V. L. Souza, J. P. Nobrega, and A. L. Oliveira, "Computational Intelligence and Financial Markets: A Survey and Future Directions," *Expert Systems with Applications*, vol. 55, pp. 194–211, 2016.

[5] A. Sujith, G. S. Sajja, V. Mahalakshmi, S. Nuhmani, and B. Prasanalakshmi, "Systematic review of smart health monitoring using deep learning and artificial intelligence," *Neuroscience Informatics*, vol. 2, no. 3, p. 100028, 2022.

[6] "Meet Flo – The First Period & Ovulation Tracker that Uses Neural Networks," *insideBIGDATA*. `https://insidebigdata.com/2017/05/21/meet-flo-first-period-ovulation-tracker-uses-neural-networks/`. Last accessed on 2023-05-31.

[7] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018.

[8] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2034–2039, 2018.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] H. S. Kang, J. Y. Lee, S. Choi, H. Kim, J. H. Park, J. Y. Son, B. H. Kim, and S. D. Noh, "Smart manufacturing: Past research, present findings, and future directions," *International Journal of Precision Engineering and Manufacturing-Green Technology*, vol. 3, pp. 111–128, Jan. 2016.

[11] M. Bertolini, D. Mezzogori, M. Neroni, and F. Zammori, "Machine learning for industrial applications: A comprehensive literature review," *Expert Systems with Applications*, vol. 175, p. 114820, 2021.

[12] P. K. R. Maddikunta, Q.-V. Pham, P. B, N. Deepa, K. Dev, T. R. Gadekallu, R. Ruby, and M. Liyanage, "Industry 5.0: A survey on enabling technologies and potential applications," *Journal of Industrial Information Integration*, vol. 26, p. 100257, 2022.

[13] K. Hunt, D. Sbarbaro, R. Żbikowski, and P. Gawthrop, "Neural Networks for Control Systems—A Survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[14] B. Recht, "A Tour of Reinforcement Learning: The View from Continuous Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, May 2019.

[15] T. Duriez, S. L. Brunton, and B. R. Noack, *Machine Learning Control – Taming Nonlinear Dynamics and Turbulence*. Springer International Publishing, 2017.

[16] Z.-P. Jiang, T. Bian, and W. Gao, *Learning-Based Control: A Tutorial and Some Recent Results*. Now Foundations and Trends, 2020.

[17] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-Based Model Predictive Control: Toward Safe Learning in Control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 269–296, 2020.

[18] S. Moe, A. M. Rustad, and K. G. Hanssen, "Machine Learning in Control Systems: An Overview of the State of the Art," in *Artificial Intelligence XXXV* (M. Bramer and M. Petridis, eds.), (Cham), pp. 250–265, Springer International Publishing, 2018.

[19] W. E, "A Proposal on Machine Learning via Dynamical Systems," *Communications in Mathematics and Statistics*, vol. 5, pp. 1–11, Mar. 2017.

[20] Q. Li and S. Hao, "An optimal control approach to deep learning and applications to discrete-weight neural networks," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 2985–2994, PMLR, 10–15 Jul 2018.

[21] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert, "Multi-level residual networks from dynamical systems view," in *International Conference on Learning Representations*, 2018.

[22] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," *Inverse Problems*, vol. 34, p. 014004, dec 2017.

[23] K. J. Åstrom and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. USA: Princeton University Press, 2008.

[24] N. S. Nise, *Control Systems Engineering*. Nashville, TN: John Wiley & Sons, 7 ed., Sept. 2013.

[25] L. Ljung, *System Identification: Theory for the User*. Prentice Hall information and system sciences series, Prentice Hall PTR, 1999.

[26] D. Gedon, N. Wahlström, T. B. Schön, and L. Ljung, "Deep state space models for nonlinear system identification∗∗this research was partially supported by the wallenberg ai, autonomous systems and software program (wasp) funded by knut and alice wallenberg foundation and the swedish research council, contracts 2016-06079 and 2019-04956.," *IFAC-PapersOnLine*, vol. 54, no. 7, pp. 481–486, 2021. 19th IFAC Symposium on System Identification SYSID 2021.

[27] A. Chiuso and G. Pillonetto, "System identification: A machine learning perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 281–304, 2019.

[28] W. Liu, J. C. Principe, and S. S. Haykin, *Kernel adaptive filtering*. Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Communications and Control, Hoboken, NJ: Wiley-Blackwell, Feb. 2010.

[29] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 96–114, 2006.

[30] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Systems Magazine*, vol. 32, no. 6, pp. 76–105, 2012.

[31] P. Antsaklis, "Neural networks for control systems," *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 242–244, 1990.

[32] W. T. Miller, R. S. Sutton, and P. J. Werbos, eds., *Neural Networks for Control*. Neural Network Modeling and Connectionism, London, England: MIT Press, Mar. 1995.

[33] M. Hagan and H. Demuth, "Neural networks for control," in *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, vol. 3, pp. 1642–1656 vol.3, 1999.

[34] R. Sutton, A. Barto, and R. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems Magazine*, vol. 12, no. 2, pp. 19–22, 1992.

[35] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[36] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2020.

[37] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," in *2016 9th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 2, pp. 198–201, 2016.

[38] C. Molnar, *Interpretable Machine Learning*. Lulu.com, 2 ed., 2022.

[39] C. Dwork and A. Roth, *The Algorithmic Foundations of Differential Privacy*. Now Foundations and Trends, 2014.

[40] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[41] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari, "Approximating Explicit Model Predictive Control Using Constrained Neural Networks," in *2018 Annual American Control Conference (ACC)*, pp. 1520–1527, 2018.

[42] S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari, "Large Scale Model Predictive Control with Neural Networks and Primal Active Sets," *arXiv preprint arXiv:1910.10835*, 2019.

[43] E. Maddalena, C. da S. Moraes, G. Waltrich, and C. Jones, "A Neural Network Architecture to Learn Explicit MPC Controllers from Data," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11362–11367, 2020. 21st IFAC World Congress.

[44] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, pp. 136–145, PMLR, 2017.

[45] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, 2019.

[46] Y. Chen and J. Karlsson, "State tracking of linear ensembles via optimal mass transport," *IEEE Control Systems Letters*, vol. 2, no. 2, pp. 260–265, 2018.

[47] K. Åström and P. Eykhoff, "System identification—a survey," *Automatica*, vol. 7, pp. 123–162, 3 1971.

[48] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön, "Deep learning and system identification," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1175–1181, 2020. 21st IFAC World Congress.

[49] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[50] F. Liu and P. Demosthenes, "Real-world data: A brief review of the methods, applications, challenges and opportunities," *BMC Medical Research Methodology*, vol. 22, p. 287, 11 2022.

[51] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Proceedings of the 1st Conference on Fairness, Accountability and Transparency* (S. A. Friedler and C. Wilson, eds.), vol. 81 of *Proceedings of Machine Learning Research*, pp. 77–91, PMLR, 23–24 Feb 2018.

[52] I. Lourenço, R. Mattila, C. R. Rojas, and B. Wahlberg, "Cooperative system identification via correctional learning," *19th IFAC Symposium on System Identification*, vol. 54, no. 7, pp. 19–24, 2021.

[53] G. Goodwin, J. Doná, and M. Seron, *Constrained Control and Estimation: An Optimisation Approach*. Communications and Control Engineering, Springer, 2005.

[54] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[55] E. Kerrigan, *Robust Constraint Satisfaction: Invariant Sets and Predictive Control*. PhD thesis, Department of Engineering, University of Cambridge, Cambridge, 2000.

[56] T. Glad and L. Ljung, *Control Theory: Multivariable and Nonlinear Methods*. 11 New Fetter Lane, London EC4P 4EE: Taylor & Francis, 1st ed., 2000.

[57] W. Kwon and S. Han, *Receding Horizon Control: Model Predictive Control for State Models*. Advanced Textbooks in Control and Signal Processing, Springer London, 1 ed., 2006.

[58] J. Zabczyk, *Mathematical Control Theory: An Introduction*. Systems & Control: Foundations & Applications, Cham: Springer Nature, 2 ed., 2020.

[59] E. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Texts in Applied Mathematics, Springer New York, NY, 2 ed., 2013.

[60] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[61] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. USA: Cambridge University Press, 1st ed., 2017.

[62] P. Scokaert and J. Rawlings, "Constrained linear quadratic regulation," *IEEE Transactions on Automatic Control*, vol. 43, no. 8, pp. 1163–1169, 1998.

[63] P. Grieder, F. Borrelli, F. Torrisi, and M. Morari, "Computation of the constrained infinite time linear quadratic regulator," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 6, pp. 4711–4716 vol.6, 2003.

[64] G. Stathopoulos, M. Korda, and C. N. Jones, "Solving the Infinite-Horizon Constrained LQR Problem Using Accelerated Dual Proximal Methods," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1752–1767, 2017.

[65] P. Bemporad, *Explicit Model Predictive Control*, pp. 1–9. London: Springer London, 2013.

[66] A. Alessio and A. Bemporad, *A Survey on Explicit Model Predictive Control*, pp. 345–369. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[67] M. Kvasnica, J. Holaza, B. Takács, and D. Ingole, "Design and Verification of Low-Complexity Explicit MPC Controllers in MPT3 (Extended version)," in *2015 European Control Conference (ECC)*, 2015.

[68] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-Parametric Toolbox 3.0," in *2013 European Control Conference (ECC)*, pp. 502–510, 2013.

[69] L. Ljung, *Modeling and identification of dynamic systems*. Lund: Studentlitteratur, second edition ed., 2021.

[70] S. Haykin, *Neural Networks: A Comprehensive Foundation*. USA: Prentice Hall PTR, 1st ed., 1994.

[71] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[72] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[73] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence, Berlin: Springer, 2012.

[74] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[75] G. Zaccone and M. R. Karim, *Deep Learning with TensorFlow - Second Edition: Explore Neural Networks and Build Intelligent Systems with Python*. Packt Publishing, 2nd ed., 2018.

[76] P. Dangeti, *Statistics for Machine Learning: Techniques for Exploring Supervised, Unsupervised, and Reinforcement Learning Models with Python and R*. Packt Publishing, 2017.

[77] W. Di, A. Bhardwaj, and J. Wei, *Deep Learning Essentials: Your Hands-on Guide to the Fundamentals of Deep Learning and Neural Network Modeling*. Packt Publishing, 2018.

[78] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[79] M. Grant, S. Boyd, and Y. Ye, *Disciplined Convex Programming*, pp. 155–210. Boston, MA: Springer US, 2006.

[80] C. Villani, *Topics in Optimal Transport*. American Mathematical Society, 2003.

[81] G. Peyré and M. Cuturi, "Computational optimal transport," *Foundations and Trends in Machine Learning*, vol. 11, pp. 355–607, 2019.

[82] Y. Chen, J. Karlsson, and A. Ringh, "Optimal transport for applications in control and estimation," *IEEE Control Systems Magazine*, vol. 41, pp. 28–33, 8 2021.

[83] G. Cimini, D. Bernardini, S. Levijoki, and A. Bemporad, "Embedded model predictive control with certified real-time optimization for synchronous motors," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 2, pp. 893–900, 2021.

[84] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020.

[85] S. Fahandezh-Saadi and M. Tomizuka, "In proximity of relu dnn, pwa function, and explicit mpc," *arXiv preprint arXiv:2006.05001*, 2020.

[86] S. East, M. Gallieri, J. Masci, J. Koutnik, and M. Cannon, "Infinite-horizon differentiable model predictive control," *arXiv preprint arXiv:2001.02244*, 2020.

[87] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Autom.*, vol. 31, pp. 1443–1451, 1995.

[88] B. Åkesson and H. Toivonen, "A neural network model predictive controller," *Journal of Process Control*, vol. 16, pp. 937–946, 10 2006.

[89] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[90] G. C. Goodwin, J. A. D. Doná, and M. M. Seron, *Constrained Control and Estimation*. Springer London, 2005.

[91] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[92] J. P. Boyle and R. L. Dykstra, "A method for finding projections onto the intersection of convex sets in hilbert spaces," in *Advances in Order Restricted Statistical Inference* (R. Dykstra, T. Robertson, and F. T. Wright, eds.), (New York, NY), pp. 28–47, Springer New York, 1986.

[93] S. S. Pon Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen, "A deep learning architecture for predictive control," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 512–517, 2018. 10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018.

[94] N. Lanzetti, Y. Z. Lian, A. Cortinovis, L. Dominguez, M. Mercangöz, and C. Jones, "Recurrent neural network based mpc for process industries," in *2019 18th European Control Conference (ECC)*, pp. 1005–1010, 2019.

[95] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[96] R. Winqvist, A. Venkitaraman, and B. Wahlberg, "On Training and Evaluation of Neural Network Approaches for Model Predictive Control," *arXiv prepreint arXiv:2005:04112*, 2020.

[97] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[98] G. E. P. Box and K. B. Wilson, "On the experimental attainment of optimum conditions," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 13, no. 1, pp. 1–45, 1951.

[99] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[100] H. O. Mete and Z. B. Zabinsky, "Pattern hit-and-run for sampling efficiently on polytopes," *Operations Research Letters*, vol. 40, no. 1, pp. 6–11, 2012.

[101] Z. B. Zabinsky and R. L. Smith, *Hit-and-Run Methods*, pp. 721–729. Boston, MA: Springer US, 2013.

[102] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[103] G. C. Pereira, P. F. Lima, B. Wahlberg, H. Pettersson, and J. Mårtensson, "Linear time-varying robust model predictive control for discrete-time nonlinear systems," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 2659–2666, 2018.

[104] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[105] D. Tabas and B. Zhang, "Safe and efficient model predictive control using neural networks: An interior point approach," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 1142–1147, 2022.

[106] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Proceedings of the 5th Conference on Robot Learning* (A. Faust, D. Hsu, and G. Neumann, eds.), vol. 164 of *Proceedings of Machine Learning Research*, pp. 1724–1735, PMLR, 08–11 Nov 2022.

[107] S. Mukherjee, J. Drgoňa, A. Tuor, M. Halappanavar, and D. Vrabie, "Neural lyapunov differentiable predictive control," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 2097–2104, 2022.

[108] S. Yang, S. Chen, V. M. Preciado, and R. Mangharam, "Differentiable safe controller design through control barrier functions," *IEEE Control Systems Letters*, vol. 7, pp. 1207–1212, 2023.

[109] A. Didier, R. C. Jacobs, J. Sieber, K. P. Wabersich, and M. N. Zeilinger, "Approximate predictive control barrier functions using neural networks: A computationally cheap and permissive safety filter," *arXiv preprint arXiv:2211.15104*, 2022.

[110] C. Dawson, S. Gao, and C. Fan, "Safe Control With Learned Certificates: A Survey of Neural Lyapunov, Barrier, and Contraction Methods for Robotics and Control," *IEEE Transactions on Robotics*, 2023.

[111] T. X. Ngheim, J. Drgona, C. Jones, Z. Nagy, R. Schwan, B. Dey, A. Chakrabarty, S. Di Cairano, J. A. Paulson, A. Carron, M. Zeilinger, W. S. Cortez, and D. L. Vrabie, "Physics-informed machine learning for modeling and control of dynamical systems," in *American Control Conference (ACC)*, May 2023.

[112] H. Hose, J. Köhler, M. N. Zeilinger, and S. Trimpe, "Approximate non-linear model predictive control with safety-augmented neural networks," *arXiv preprint arXiv:2304.09575*, 2023.

[113] Y. Zhu, "System identification for process control: Recent experience and outlook," *IFAC Proceedings Volumes*, vol. 39, no. 1, pp. 20–32, 2006. 14th IFAC Symposium on Identification and System Parameter Estimation.

[114] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[115] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning," *ACM Computing Surveys*, vol. 50, pp. 1–35, Apr. 2017.

[116] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of the Ninth International Workshop on Machine Learning*, ML92, (San Francisco, CA, USA), p. 249–256, Morgan Kaufmann Publishers Inc., 1992.

[117] J. Wang, P. Zhao, S. C. Hoi, and R. Jin, "Online feature selection and its applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 698–710, 2014.

[118] L. Pronzato, "Optimal experimental design and some related control problems," *Automatica*, vol. 44, no. 2, pp. 303–325, 2008.

[119] H. Hjalmarsson, "System identification of complex and structured systems," *European Journal of Control*, vol. 15, no. 3, pp. 275–310, 2009.

[120] C. Aggarwal, X. Kong, Q. Gu, J. Han, and P. Yu, *Active Learning: A Survey*, pp. 571–605. CRC Press, Jan. 2014. Publisher Copyright: © 2015 by Taylor & Francis Group, LLC.

[121] S. Verma, J. Dickerson, and K. Hines, "Counterfactual explanations for machine learning: A review," *arXiv preprint arXiv:2010.10596*, 2020.

[122] P. Kuusela and D. Ocone, "Learning with side information: Pac learning bounds," *Journal of Computer and System Sciences*, vol. 68, no. 3, pp. 521–545, 2004.

[123] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley, 2 ed., 2006.

[124] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 4 1994.

[125] Q. Welniarz, Y. Worbe, and C. Gallea, "The forward model: A unifying theory for the role of the cerebellum in motor control and sense of agency," *Frontiers in Systems Neuroscience*, vol. 15, 2021.

[126] P. Lanillos, D. Oliva, A. Philippsen, Y. Yamashita, Y. Nagai, and G. Cheng, "A review on neural network models of schizophrenia and autism spectrum disorder," *Neural Networks*, vol. 122, pp. 338–363, 2020.

[127] I. Lourenço, R. Winqvist, C. R. Rojas, and B. Wahlberg, "A teacher-student markov decision process-based framework for online correctional learning," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 3456–3461, 2022.

[128] A. Galichon, *Optimal Transport Methods in Economics*. Princeton University Press, 2016.

[129] I. Haasler, J. Karlsson, and A. Ringh, "Control and estimation of ensembles via structured optimal transport," *IEEE Control Systems Magazine*, vol. 41, pp. 50–69, 8 2021.

[130] J. Digne, D. Cohen-Steiner, P. Alliez, F. de Goes, and M. Desbrun, "Feature-preserving surface reconstruction and simplification from defect-laden point sets," *Journal of Mathematical Imaging and Vision*, vol. 48, pp. 369–382, 2 2014.

[131] C. Frogner, C. Zhang, H. Mobahi, M. Araya, and T. A. Poggio, "Learning with a Wasserstein Loss," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[132] A. Gramfort, G. Peyré, and M. Cuturi, "Fast optimal transport averaging of neuroimaging data," in *Information Processing in Medical Imaging* (S. Ourselin, D. C. Alexander, C.-F. Westin, and M. J. Cardoso, eds.), (Cham), pp. 261–272, Springer International Publishing, 2015.

[133] A. Tompkins, R. Senanayake, and F. Ramos, "Online domain adaptation for occupancy mapping," in *Robotics: Science and Systems*, 07 2020.

[134] J. Tan, R. Senanayake, and F. Ramos, "Renaissance robot: Optimal transport policy fusion for learning diverse skills," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7052–7059, 2022.

[135] A. Taghvaei and P. G. Mehta, "An optimal transport formulation of the ensemble kalman filter," *IEEE Transactions on Automatic Control*, vol. 66, no. 7, pp. 3052–3067, 2021.

[136] Y. Yang, L. Nurbekyan, E. Negrini, R. Martin, and M. Pasha, "Optimal transport for parameter identification of chaotic dynamics via invariant measures," *SIAM Journal on Applied Dynamical Systems*, vol. 22, no. 1, pp. 269–310, 2023.

[137] G. Peyré, M. Cuturi, *et al.*, "Computational optimal transport: With applications to data science," *Foundations and Trends® in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[138] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*. Elsevier Science, 2015.

[139] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[140] M. Teimouri, S. Hoseini, and S. Nadarajah, "Comparison of estimation methods for the weibull distribution," *Statistics: A Journal of and Applied Statistics*, 03 2011.

[141] B. Lakshminarayanan and C. R. Rojas, "A statistical decision-theoretical perspective on the two-stage approach to parameter estimation," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 5369–5374, 2022.

[142] A. Bajcsy, D. P. Losey, M. K. O'malley, and A. D. Dragan, "Learning robot objectives from physical human interaction," in *Conference on Robot Learning*, pp. 217–226, PMLR, 2017.

[143] M. Annergren, C. A. Larsson, H. Hjalmarsson, X. Bombois, and B. Wahlberg, "Application-oriented input design in system identification: Optimal input design for control [applications of control]," *IEEE Control Systems Magazine*, vol. 37, no. 2, pp. 31–56, 2017.