Master's thesis

NTNU
Norwegian University of Science and Technology

Camilla Sterud

Feedback linearizing neural network controllers

Master's thesis in Cybernetics and robotics
Supervisor: Prof. Jan Tommy Gravdahl og Dr. Signe Moe
December 2019

NTNU
Norwegian University of
Science and Technology

Camilla Sterud

Feedback linearizing neural network controllers

Master's thesis in Cybernetics and robotics
Supervisor: Prof. Jan Tommy Gravdahl og Dr. Signe Moe
December 2019

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Preface

For the past 18 weeks, I have spent a substantial part of my waking hours working on this thesis. It is the work that will conclude my time at the Norwegian University of Science and Technology (NTNU), and attempt to show some of what I have learned during the past five years.

While studying engineering cybernetics at NTNU, I have especially enjoyed the courses on advanced control theory. A couple of years ago, I also started dabbling in statistical learning and machine learning. When I was offered to do my thesis in cooperation with SINTEF Digital's Analytics and Artificial Intelligence group, I was given the perfect opportunity to apply two of my favorite topics in one thesis.

This thesis is about combining the expressive power of neural networks with the mathematical soundness of nonlinear control theory. It is an exciting topic within a rapidly developing research field that has been gaining traction in recent years. I hope that the contents of this thesis are of interest to the reader and that my work may provide some insight into how deep learning can be safely and beneficially applied in control.

It is assumed that the reader of this thesis has in-depth knowledge of control theory, Lyapunov theory, and linear algebra. Some experience with deep learning is beneficial, but not strictly necessary, as this topic will be introduced.

One of the contributions of this thesis is a neural network controller that is partially based on the work of Shi et al. [1]. The Python programming language and the Keras library was used to implement and train the neural networks for this controller [2], [3]. The other contribution is an extension of a method for estimating the Lipschitz constant of neural networks, initially developed by Fazlyab et al. [4]. This extended method was implemented in MATLAB 2019b with the CVX library [5], [6].

Both of my supervisors, Dr. Signe Moe and Prof. Jan Tommy Gravdahl, as well as the head of the Analytics and AI group, Anne Marthine Rustad, have offered valuable help by reading through a draft for this thesis, and suggesting improvements.

Acknowledgments

This thesis was done in cooperation with SINTEF Digital, more specifically the Analytics and AI group. While working on this thesis, I have been fortunate to be part of the great work environment at SINTEFs offices in Oslo. I want to thank everyone in the Analytics and AI group that have been eager to discuss my academic challenges, offer their help and go for lunch and coffee breaks. In particular, I want to thank my supervisor from SINTEF, Dr. Signe Moe, who has been available whenever, two offices down, this entire semester. I also want to thank my supervisor from the Department of Engineering Cybernetics, Prof. Jan Tommy Gravdahl, for asking insightful question and answering slightly less insightful ones.

Great thanks are also owed to my family and my partner, who have all offered their support and their proof reading skills.

Summary

Unknown and unmodeled dynamics is a reoccurring topic in the practical application of control systems. When using popular control methods such as model predictive control and feedback linearization, it is assumed that an analytical model of the system dynamics is available and that the model is so accurate that the modeling error is negligible, or at the very least bounded. However, this is not always the case, and often the control system is simplified at the expense of optimal performance.

Currently, there is a push to unify the frameworks of traditional control theory and data-driven modeling. In particular machine learning, and its subfield deep learning, seems to have much to offer the field of control theory. The problem with including deep learning methods in control systems is the resulting loss of transparency. Deep learning is mostly concerned with neural networks, which are vast black-box models that are hard to analyze mathematically. Control systems with machine learning in the loop often lack stability proofs and performance guarantees, which are crucial if they are applied in safety-critical situations.

In this thesis, a feedback linearizing controller, which uses a neural network to estimate unknown dynamics, is suggested. The suggested controller is designed for solving a general trajectory tracking problem for a broad class of two dimensional nonlinear systems. The controller is proven to stabilize the closed-loop system, such that it is input-to-state and finite-gain $\mathcal{L}_p$-stable from the neural network estimation error to the tracking error. Further, the controller is proven to make the tracking error globally and exponentially converge to a ball centered at the origin. The convergence bound is shown to be dependent on the Lipschitz constant of the neural network estimator when the estimate is updated discretely, or the state measurements are affected by bounded noise.

Through experiments on a simulated mass-spring-damper system, the validity of the theoretical results is investigated, for when the system dynamics are undelayed and also when they are time-delayed. In the time-delayed case, convolutional neural networks are used to estimate unknown dynamics. A procedure for estimating the Lipschitz constant of feedforward networks is therefore extended to work for convolutional neural networks.

In the experiments, all tested controllers outperform a feedback controller that does not compensate for unknown dynamics. When the dynamics are undelayed, there is a relationship between the size of the Lipschitz constant of the neural network and a controller's ability to reject noise. There is also a connection between the Lipschitz constant and the change in performance when the update frequency of the neural network estimate declines.

Sammendrag

Når reguleringssystemer anvendes i praksis er ukjent og umodellert dynamikk et tilbakevendende problem. Ved anvendelse av avanserte reguleringstekniske metoder, som modellbasert prediktiv og tilbakekoblingslinеariserende regulering, antas det at matematiske modeller av systemdynamikken er tilgjengelig, og at de tilnærmer det virkelige systemet så godt at modelleringsfeilen er neglisjerbar, eller i det minste begrenset. Dette er imidlertid ikke alltid tilfellet, og man tar isteden til takke med forenklede reguleringssystemer på bekostning av optimal ytelse.

For tiden finnes det store interesser for å kombinere teoriene fra tradisjonell reguleringsteknikk og datadrevne metoder. Spesielt synes feltet maskinlæring, og dets underordnede felt dyp læring, å ha mye å tilby reguleringsteknikken. Problemet med å inkludere metoder fra dyp læring i reguleringssystemer er at systemet blir mindre gjennomskuelig. Dyp læring handler i hovedsak om nevrale nettverk som er vanskelige å analysere matematisk siden de er store, ugjennomskuelige, ulineære modeller. Reguleringssystemer med maskinlæringskomponenter mangler ofte stabilitetsbevis og garantier for sin oppførsel. Dette er uakseptabelt hvis de skal anvendes i situasjoner hvor sikkerhet er sentralt.

I denne oppgaven utformes en tilbakekoblingslinеariserende (feedback linearizing) regulator som bruker et nevralt nettverk til å estimere ukjent dynamikk. Regulatoren har som formål å løse et generelt stifølgingsproblem for en omfattende klasse todimensjonale systemer. Det bevises at regulatoren stabiliserer den lukkede sløyfen, slik at den er pådrag-til-tilstand-stabil (input-to-state stable) og endelig-forsterket $\mathcal{L}_p$-stabil (finite-gain $\mathcal{L}_p$-stable) fra det nevrale nettverkets estimeringsfeil til avviket i stifølgingsproblemet. Videre bevises at regulatoren får avviket til å konvergere eksponentielt og globalt mot en ball sentrert i origo. Konvergensgrensene avhenger av Lipschitzkonstanten til det nevrale nettverket når nettverksestimatet oppdateres sjeldent, eller når tilstandsmålingene påvirkes av støy av begrenset størrelse.

Eksperimenter utføres på et simulert masse-fjær-dempersystem for å undersøke holdbarheten til de teoretiske resultatene, både når systemets dynamikk er uforsinket og når den er tidsforsinket. Når dynamikken er tidsforsinket brukes nevrale konvulsjonsnettverk til å estimere den ukjente dynamikken. En metode for å estimere Lipschitzkonstanten til nevrale forovernettverk utvides til å fungere for nevrale konvulsjonsnettverk.

I eksperimentene utkonkurrerer alle de foreslåtte regulatorene en tilbakekoblingsregulator som ikke kompenserer for den ukjente dynamikken. Når dynamikken er uforsinket finnes det et forhold mellom størrelsen på Lipschitzkonstanten til den nevrale nettverket og regulatorens evne til å undertrykke målestøy. Det finnes

også en kobling mellom Lipschitzkonstanten og endringen i regulatorens ytelse når estimatet fra det nevrale nettverket oppdateres sjeldnere.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

CNN	Convolutional Neural Network, page 15
FFN	Feedforward Network, page 11
GES	Globally Exponentially Stable, page 28
LipSDP	Lipschitz Semidefinite Program, page 19
MAE	Mean Absolute Error, page 9
MSD	Mass-Spring-Damper, page 32
MSE	Mean Squared Error, page 9
NN	Neural Network, page 11
SDP	Semidefinite Program, page 6
SGD	Stochastic Gradient Descent, page 13

Symbols

α_{L2}	L2 regularization parameter, page 10
$\Delta t_{\max}$	Update time of neural network estimator, page 27
η	Learning rate, page 13
Λ	Lipschitz constant, page 16
λ	Least negative closed-loop eigenvalue, page 27
ρ^u	Upper bound on input change, page 27
ρ^x	Upper bound on system state change, page 27
σ	Standard deviation of measuring noise, page 37
θ	Regression model parameters, page 8
ε	Maximal estimation error of neural network estimator, page 26
$J(\cdot)$	Loss function, page 8

N_{MB} Mini batch size, page 14

s^r Receptive field size, page 16

1 Introduction

1.1 Background and motivation

Practical application of complex control systems is not always straight-forward. Theoretical assumptions might be violated, or the required expert knowledge unavailable. A physical model of the plant one wants to control is often required but not always available. For instance, when controlling a small, unmanned aircraft, inaccurately modeled forces might provoke unwanted behavior when operating at differing heights [1]. In the process industry, time delays and nonlinearities that are hard to model are commonplace. Lack of expert knowledge and off-the-shelf solutions makes the industry settle for simple PID controllers, at the cost of performance [7]. Consequently, there is a gap between modern control theory in academia and in industry.

For the last three decades, researches have been eager to utilize the enormous amounts of information that exists after years of collecting sensor data for process monitoring and control purposes [8]. Soft-sensors, which combine measurement data and analytical models in clever ways to make synthetic measurements, and smart maintenance are two profitable research fields that are part of this data exploitation wave [9].

Machine learning is a research field that has exploded in the last decades due to the availability of data and inexpensive computing power. This has been especially beneficial to the subfield of machine learning known as deep learning, which is mostly concerned with large black-box models called neural networks. Deep learning algorithms have been suggested and successfully implemented for natural language processing, operating autonomous vehicles, object classification, and playing video games, among other things [10]–[13].

Although their feats are seemingly impressive, deep learning systems have been proven to generalize less than one could hope, exemplified by their weakness to adversarial attacks [14]. By adding small, deliberate perturbations in the input data, consistently terrible performance can be achieved from networks with high accuracy on the original data [15]. If perturbations in the input data, so small that they are invisible to the human eye, can completely offset a deep learning system, the time is not ripe for letting it pilot an airplane or control an explosive chemical reaction. From a control engineering perspective, stability proofs, safety certificates, and performance guarantees are absolute necessities for applying a controller to safety-critical settings.

The problem with assuring safety in systems based on neural networks often stems from the fact that neural network design relies on the theoretical repre-

sentational power of combining enough nonlinearities, instead of sound physical principles [16]. The internal workings of a neural network are hard to analyze, and their behavior is challenging to predict. Not only is this lack of interpretability a challenge in regards to performance, but it can also be an ethical one when considering various decision systems and classifiers [17].

Several methods for verifying the performance of neural networks have been suggested. Liu et al. summarizes methods for verifying assumptions about neural networks [18]. Others seek to increase robustness to adversarial examples, by adversarial training, optimizing a robust outer loss or penalizing an upper bound of the Lipschitz constant [14], [19], [20]. Though these approaches make the network more likely to be more robust, no proofs of stability or robustness exist.

Lately, there has been an increasing initiative to combine control theory and machine learning in order to bring rigorous mathematical proofs and reasoning based on physics into data-based methods. An example of this rise in popularity is the conference Learning for Dynamics and Control, which was held for the first time in May 2019 at Massachusetts Institute of Technology. This combination of paradigms is part of the up and coming field *explainable artificial intelligence*, and its subfield sometimes referred to as *hybrid analytics*, *hybrid modeling* or *physics-based AI* [21]. In hybrid analytics, the goal is to entwine the thorough understanding and mathematical soundness of physics-based modeling with the accuracy, general applicability, and automatic pattern-identification capabilities of advanced data-driven methods [22]. Figure 1.1 illustrates how hybrid analytics relates to other system analysis approaches.

Many have suggested ways to incorporate learning into the popular model predictive control scheme, for instance, to achieve stability with pre-learned models, and safe exploration for online model learning [23], [24]. Lyapunov theory has been applied in learning systems to ensure safe exploration in reinforcement learning settings, as done by Berkenkamp et al. [25]. Richards et al. use a particular class of neural networks, dubbed Lyapunov neural networks, to estimate safe sets for dynamical systems [26]. Others have let the notion of closed-loop stability inspire new neural network architectures [27]–[29].

In the recent publication of Shi et al., stability and convergence in a drone path-following problem are proven for a controller where a neural network learns unknown dynamics. They claim to be the first to provide stability guarantees for a neural network-based feedback controller that can utilize arbitrarily large networks [1]. This last work has been a great inspiration for this thesis, and the transformation that will be used in the feedback linearization later is taken from this paper.

The Lipschitz constant is a measure of the maximum rate of change of a continuous function, which is what a neural network happens to be. It has mostly been used for increasing robustness in classification problems [20], [30], but Shi et al. proved that the stability of their feedback controller is dependent on the Lipschitz constant of its neural network estimator [1]. Computing the exact Lipschitz constant of general feedforward networks is NP-hard [31], and several methods

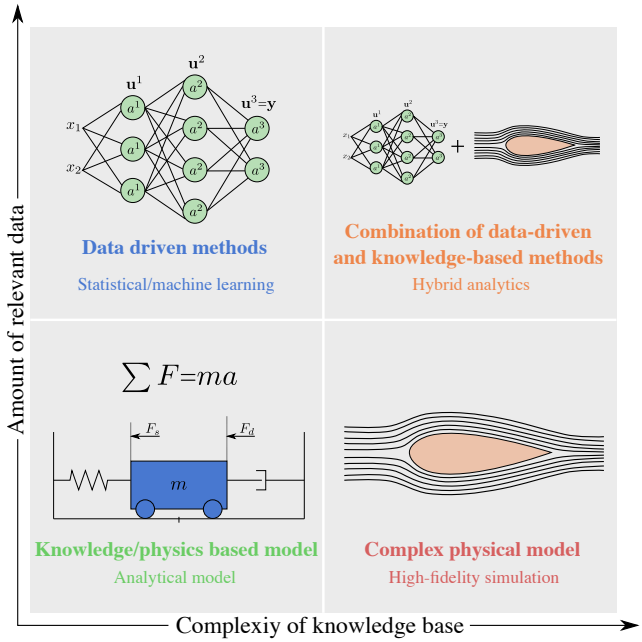


Figure 1.1: Hybrid analytics combines the pattern-recognition capabilities of data-driven methods with the soundness of physics- and knowledge-based models.

for efficient estimation have been proposed [31]–[33]. In particular, Fazlyab et al. propose a method that involves solving a semidefinite program to estimate the Lipschitz constant, which will be expanded in this thesis [4].

1.2 Problem formulation

This thesis is an attempt to further the field of hybrid analytics by incorporating deep learning into nonlinear control. Before such a hybrid approach can be applied to situations where safety is critical, it is essential to make guarantees on stability and performance. Hence, two questions related to the safety of a neural network controller are posed:

1. Can stability and convergence be proven for a neural network controller operating on a general class of nonlinear systems?

2. How can the Lipschitz constant be used to evaluate the robustness of the neural network controller?

The work in this thesis attempts to answer these questions.

1.3 Contributions of this thesis

This thesis makes the following contribution to the understanding of how data-driven learning and control theory can be safely and productively combined:

1. A controller is suggested that gives the closed loop system attractive stability traits, and provably bounds the error in a trajectory tracking problem. The stability and convergence properties of the controller are summarized in two theorems: Theorem 3.1 and Theorem 3.2.

The Lipschitz constant of the neural network in the controller is proven to influence the robustness of the controller, but this is only proven to hold when using a feedforward network on systems without time delay. In order to examine whether the Lipschitz constant has the same influence as in the undelayed case, convolutional neural networks are applied to systems with time-delayed dynamics. This leads to the second contribution of this thesis:

2. The method of Fazlyab et al. for estimating the Lipschitz constant of feedforward networks is extended to convolutional neural networks.

1.4 Thesis outline

The thesis is organized as follows:

Chapter 2 gives an overview of some necessary background theory, including convolution, the Lipschitz constant, semidefinite programming and deep learning. The reader is referred to the appendix and external literature for background theory on nonlinear control.

In Chapter 3, the extension to the method of Fazlyab et al. is presented before the feedforward neural network controller is derived. Two theorems summarize the stability and convergence properties of this controller. Proofs follow. Next, a controller with a convolutional neural network estimator is suggested before neural network architectures, and training schemes are laid forth.

Then, in Chapter 4, experiments testing the controllers on a simulated mass-spring-damper system with an unknown input-nonlinearity are conducted.

Next, in Chapter 5, the results of these experiments are discussed to evaluate the coherence between the results and the predicted behavior from Chapter 3. Further extensions to this work are then suggested, before the thesis is concluded in Chapter 6.

2 Theory

This chapter provides the reader with some necessary background knowledge. First, a short introduction to the 2-norm, causal convolution, and semi-definite programming is given. Machine learning and deep learning are more thoroughly covered, as these are central to the work in this thesis. Further, the Lipschitz constant and its properties are covered, before the method of Fazlyab et al., which will be extended later, is presented.

2.1 The 2-norm

The 2-norm of a vector is defined as follows:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}, \quad \mathbf{x} \in \mathbb{R}^n.$$

Further, the 2-norm of a matrix is defined as the largest singular value of the matrix:

$$\|A\|_2 = \sigma_{\max}(A).$$

Generally, the singular values of any real matrix A are the square roots of the non-negative eigenvalues of both AA^T and A^TA. In particular, the singular values of a positive definite matrix P are just the eigenvalues of P.

In this work, the norm notation $\|.\|$ will always refer to the 2-norm of the argument unless otherwise specified.

2.2 Convolution

The discrete convolution operator, denoted by $*$, computes the response of a relaxed linear system, $\mathbf{h}$, to any input sequence $\mathbf{x}$. The linear system $\mathbf{h}$ is commonly referred to as a filter, in signal processing jargon, or a kernel by data scientists. The one-dimensional convolution operator is defined as in (2.1).

$$(\mathbf{x} * \mathbf{h})[i] = \sum_{k=-\infty}^{\infty} x[k]h[i-k] = \sum_{k=-\infty}^{\infty} h[k]x[i-k], \; \forall \, i \in \mathbb{Z} \tag{2.1}$$

2.2.1 Causal convolution

As shown in (2.2), a convolution operation can be split into a causal and a non-causal part. The output of a causal system cannot depend on future input or output values. When the output signal is causal, the non-causal terms of the convolution in (2.2) are invalid, as it is not possible to look into the future.

$$(\mathbf{x} * \mathbf{h})[i] = \sum_{k=0}^{\infty} h[k]x[i-k] + \sum_{k=-\infty}^{-1} h[k]x[i-k], \; \forall\, i \in \mathbb{Z} \tag{2.2}$$

When working with physical signals, causality is inherent. Also, when $x[i]$ is a time series signal, let $x[i] = 0$, $\forall\, i \leq 0$, and call $\mathbf{x}$ a causal signal. Hence, the convolution of a causal signal with a causal filter reduces to the expression in (2.3) [34].

$$(\mathbf{x} * \mathbf{h})[i] = \sum_{k=1}^{n} h[k]x[i-k] = \sum_{k=1}^{i} x[k]h[i-k], \; \forall\, i \in \mathbb{Z} \tag{2.3}$$

2.2.2 Convolutions as matrix multiplications

Assume that the input signal is of length T, and that the filter is of length N, where $N \leq T$. Then all non-zero outputs of the convolution in (2.3) is given by the matrix multiplication in (2.4). The output $\mathbf{y}$ will be of length $N+T-1$.

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = \begin{bmatrix} h[1] & 0 & 0 & 0 & 0 & \dots & 0 \\ h[2] & h[1] & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \ddots & \vdots \\ h[N] & \dots & h[2] & h[1] & 0 & \dots & 0 \\ 0 & h[N] & \dots & h[2] & h[1] & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & h[N] & \dots & h[2] & h[1] \\ \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & h[2] \\ 0 & \dots & 0 & \dots & 0 & h[N] & \vdots \\ 0 & \dots\dots\dots & & 0 & 0 & 0 & h[N] \end{bmatrix} \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[T] \end{bmatrix} \tag{2.4}$$

2.3 Semidefinite programming

In a semidefinite program (SDP), a linear objective function is minimized while ensuring that some symmetric matrix remains positive semidefinite. This is a convex constraint, and hence, semidefinite programming is a subfield of convex optimization.

Any SDP can be formulated as shown in (2.5), where $M_j = M_j^T, j \in \{1, \ldots, n\}$ [35]. SDPs have been thoroughly studied, as numerous practical optimization problems can be posed as SDPs. For instance, the need to solve linear matrix inequalities is often present when proving robustness for controllers operating in overlapping affine regions. Besides, SDPs express weak duality to their related dual problems and can therefore be solved efficiently using interior point methods [36].

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^T \mathbf{x} \\ &\text{subject to} \quad 0 \leq M(\mathbf{x}) \\ &\mathbf{x} \in \mathbb{R}^n, \quad M(\mathbf{x}) = M_0 + \sum_{j=1}^{n} x_j M_j \end{aligned} \tag{2.5}$$

2.4 Machine learning

In machine learning, the goal is to develop learning algorithms that enable computers to solve a problem based on experience, rather than being explicitly instructed how to solve it. The following quote by Tom Mitchell is a much-used definition of a learning algorithm [37]:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Machine learning is a vast research field that is heavy in statistics, linear algebra, and computer science. In machine learning, it is usually assumed that there exists some true mapping $\mathbf{y} = f(\mathbf{x})$ that one wants to estimate with some model $\hat{\mathbf{y}} = \hat{f}(\mathbf{x})$.

Figure 2.1 illustrates the three categories that machine learning is commonly, but roughly, divided into: Supervised, unsupervised, and reinforcement learning. In supervised learning, there exists a set of examples that are indicative of how the true mapping f behaves. These examples consist of data points $\mathbf{x}_i$, and corresponding ground truth points $\mathbf{y}_i$. The goal is to learn the mapping from these examples. Performance in the supervised setting is evaluated by how similar outputs of $\hat{f}$ are to f. When doing unsupervised learning, no such ground truth is available, and the performance must be measured only based on the available data points. Clustering is a typical application of unsupervised learning, which is concerned with finding similarities between data points. Reinforcement learning is motivated by the way humans learn through interaction and feedback evaluation. The model is left to explore an environment, and its performance is evaluated in terms of a reward it receives for acting the way it does [37].

In this work regression will be performed as a supervised learning problem. Regression is the task of finding statistically significant relationships between a quantitative variable y and covariates $\mathbf{x}$. The regression model $\hat{f}$ can be anything from a polynomial to a deep neural network, as will be discussed later.

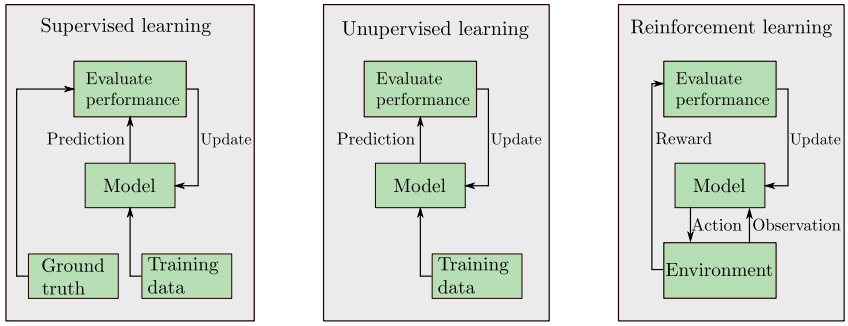


Figure 2.1: It is common to roughly divide machine learning into three approaches: Supervised, unsupervised and reinforcement learning.

2.4.1 The dataset

As the goal in machine learning is to learn from observed data, there is always a dataset that one is interested to learn from. The model should capture the underlying distribution of the dataset so that it is well prepared when faced with unseen data. In the supervised setting, the dataset consists of pairs of data points, and corresponding ground truth points $(\mathbf{x}^i, \mathbf{y}^i)$. Usually, the data is divided into three subsets, called the training set, the validation set, and the test set.

The training set is given to the model for learning. Using this data, the model repeatedly tests its performance and evaluates how to change in order to perform better. This is referred to as training the model. However, the performance of the model on the training set might be deceiving, as it is adjusting itself to fit this data as well as possible. As a sanity check, the model performance is also calculated on the validation set during training. Ideally, the performance on the validation set should be equal to the performance on the training set, as this implies that the model has learned the true underlying distribution of the data, and not only a distribution that fits the training set.

Finally, when the model has finished training, and the final model is at hand, the performance is calculated on the test set. It is important that the test set is not available to the regression model during training, and the performance on the test set should never be used for hyperparameter tuning or selecting model type.

2.4.2 Regression loss functions

It is central in machine learning to have a measure of performance that accurately reflects how well a model is doing its job. For this task a loss function $J(\theta)$ is employed, which is dependent on the model parameters θ. The loss function has the attribute that the parameters corresponding to the global minimum of the loss yield the sought after behavior of the model.

For regression tasks, it is natural to evaluate how close the predictions $\hat{y}_i$ are to

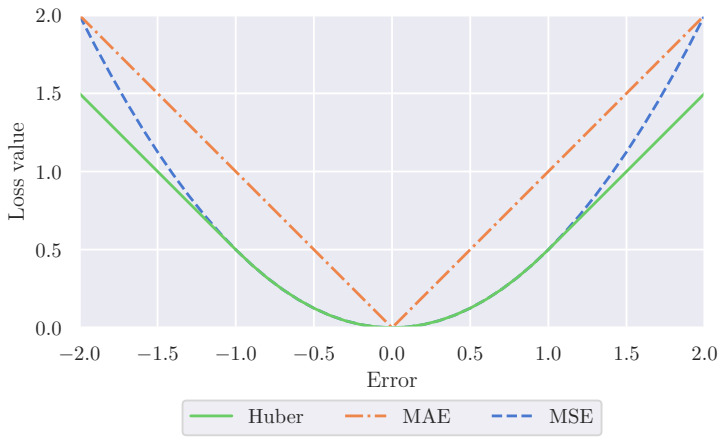


Figure 2.2: The mean squared error, mean absolute error, and the Huber loss with $\delta = 1$.

their corresponding ground truth values y_i, $i \in \{1, 2, \ldots, n\}$. Two measurements of this are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE), which are presented in (2.6) and (2.7), respectively. Both MSE and MAE are valid and common loss functions. However, the MAE is not continuously differentiable, and the MSE is sensitive to outliers, as its derivative is proportional to the error $(\hat{y} - y)$.

$$J^{\text{MSE}} = \frac{1}{n}\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.6}$$

$$J^{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{2.7}$$

A compromise between the MSE and the MAE is the Huber loss, shown in (2.8). The Huber loss is continuously differentiable, and the parameter δ decides where it behaves as the MSE and the MAE. The Huber loss with $\delta = 1$ is illustrated in Figure 2.2.

$$J^{\text{Huber}} = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\frac{1}{2}(\hat{y}_i - y_i)^2, & |\hat{y}_i - y_i| \leq \delta \\ \delta|\hat{y}_i - y_i| - \frac{1}{2}\delta^2, & |\hat{y}_i - y_i| > \delta\end{cases} \tag{2.8}$$

2.4.3 Bias-variance trade-off

Though it will not be proven here, the expected MSE of a regression model $\hat{f}$ at a data point x_0 in the test set can be decomposed as in (2.9). Here ϵ is some zero mean noise affecting the sampling of the true mapping $y_i = f(x_i) + \epsilon$ [38]. Equation (2.9) shows that the performance of the model at a test point is dependent on the variance of ϵ, as well as what is referred to as the bias and variance of the trained model. An optimal estimator would have zero bias and variance, but as it turns out, reducing one usually leads to an increase in the other.

$$\begin{aligned}\mathbb{E}\{(y_0 - \hat{f}(x_0))^2\} &= \Big(y_0 - \mathbb{E}\{\hat{f}(x_0)\}\Big)^2 + \mathbb{E}\{\Big(\mathbb{E}\{\hat{f}(x_0)\} - \hat{f}(x_0)\Big)^2\} + \mathrm{Var}(\epsilon) \\ &= \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2 + \mathrm{Var}(\epsilon)\end{aligned} \tag{2.9}$$

Consider Figure 2.3, where three polynomials of degree one, two and five are fitted to data points sampled from $y = x^2 + \epsilon$. The polynomial of degree one does not have enough flexibility to fit these data points and capture the true mapping. Hence, its bias is too high, and its variance too low. This scenario is known as underfitting. On the other hand, the polynomial of degree five passes through every single training data point, but does a poor job of extrapolating beyond these. This polynomial clearly has high variance and low bias, leading to the situation called overfitting. While the polynomial of degree two is clearly affected by the noise ϵ, its bias and variance seem well balanced.

Generally, it is said that a complex model introduces more variance, while a simpler model has higher bias [39].

2.4.4 L2 regularization

A means for reducing variance in a machine learning model is regularization. Regularization is meant to discourage overfitting and make the model generalize better, but it also increases bias, and makes the model more rigid. The goal is to trade off a small, acceptable increase in bias for a large reduction in variance, overall bettering the performance of the model.

A prevalent form of regularization is called L2 regularization, where the term in (2.10) is added to the loss function during training. L2 regularization is a penalty term that incentivizes small parameters, where $\theta^R \subseteq \theta$ are the parameters that are affected. Typically,bias terms are not included in the L2 regularization term. The weight of the penalty relative to the original loss is controlled with the regularization parameter α_{L2} [39].

$$J^{L2} = \alpha_{L2} \frac{1}{n} \frac{1}{2} \sum_{\theta_j \in \theta^R} \theta_j^2 \tag{2.10}$$

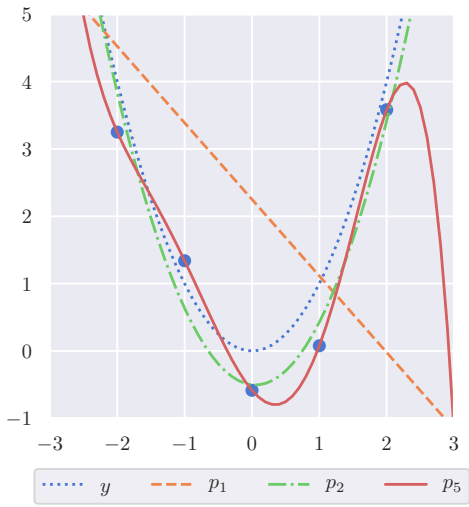


Figure 2.3: Three models are fitted to data points originating from a noisy sampling of $y = x^2$. The three fitted models are polynomials of degree one, two and five. The blue dots represent the training data.

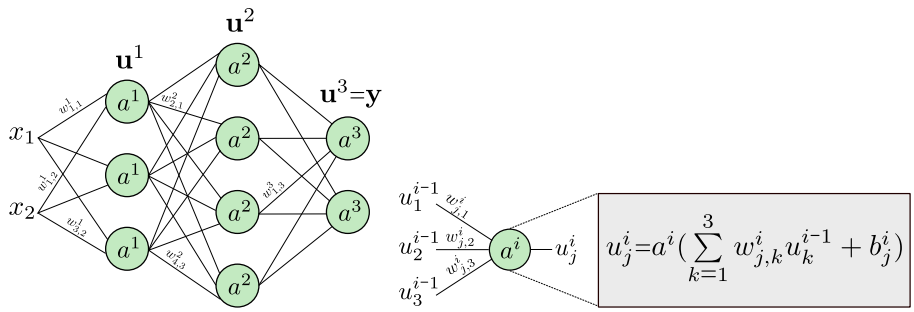
2.5 Deep learning

Deep learning is a sub-field of machine learning where complex, non-linear function approximators called neural networks (NNs) are central. Originally inspired by the human brain, NNs have proven to be excellent estimators for both classification and regression problems. Neural networks have out-competed many traditional signal processing and pattern recognition methods, becoming state of the art in research fields such as natural language processing and computer vision [40].

2.5.1 Feed forward networks

The simplest form of a neural network dates back to the 1960s, and is called a feedforward network (FFN), or a deep feedforward network [40]. A FFN takes an input $\mathbf{x}$ and maps it to an output $\mathbf{y} = f(\mathbf{x}; \theta)$, where θ denotes the network parameters. The mapping can be seen as L nonlinear mappings applied in succession; $f(\mathbf{x}) = f^L(f^{L-1}(...f^2(f^1(\mathbf{x}))))$. Each non-linear mapping is referred to as a layer. The number of layers is known as the depth of the network.

The output of layer i, $\mathbf{u}^i$, is computed as shown in (2.11). First, the output of the previous layer $\mathbf{u}^{i-1}$ is multiplied with a weight matrix W^i and then a bias vector $\mathbf{b}^i$ is added to the product. The resulting vector is then fed to an activation function a^i, which typically is of the type sigmoid, ReLU, or tanh [41]. Often, the activation function of the last layer is chosen to be the identity mapping, so that the output is not restricted.



(a) A FFN with three layers. **(b)** Zoomed in view of a single neuron.

Figure 2.4: Figure 2.4a visualizes a FFN of depth 3. The edges represent multiplication by the entries of the weight matrices, where $w_{j,k}^i$ is the element in row j and column k of W^i. The nodes are neurons where the input is summed, bias added and the activation function applied in accordance with (2.11). Figure 2.4b shows how each neuron computes its output. u_j^i is the j-th element of layer i.

$$\mathbf{u}^i = a^i(W^i \cdot \mathbf{u}^{i-1} + \mathbf{b}^i) \tag{2.11}$$

The FFN can be visualized as a graph where each node, called a neuron, outputs an element of a layer vector $\mathbf{u}^i$. Figure 2.4a shows the graph representation of a three-layer neural network. The edges represent the multiplication of a neuron output with an element of the weight matrix of the next layer. At each neuran, the incoming products are summed, a bias added, and the activation function applied, as shown in Figure 2.4b.

2.5.2 Training feedforward networks

When training a NN, the goal is to minimize some loss function $J(\theta)$ by adjusting the network parameters. Usually, θ consists of the weights and biases of the network. The most popular way to minimize the loss in deep learning is by using some version of gradient descent.

Gradient descent is an algorithm that updates the parameters of a network by perturbing them in the opposite direction of $\nabla_\theta J(\theta)$, which is the gradient of $J(\theta)$ with respect to θ. As is well known, the gradient of a scalar function points in the direction of steepest ascent. An estimate of the gradient can be efficiently computed by applying the chain-rule cleverly in a method known as back-propagation [41].

Back-propagation

Let $\mathbf{y}$ be the output layer of some FFN with L layers. Then, to compute the gradient of the loss J with respect to θ, the derivative of the loss with respect to W^i, $\mathbf{b}^i$ $\forall\, i \in \{1, \ldots, L\}$ must be computed. For clarity, the notation $\mathbf{v}^i = W^i \cdot \mathbf{u}^{i-1} + \mathbf{b}^i$ is adopted, so that $\mathbf{y} = \mathbf{u}^L = a^L(\mathbf{v}^L)$. As seen from (2.12),

applying the chain rule reveals that the derivatives share terms. This opens up for computing the derivatives efficiently, as there is no need to compute every derivative from scratch.

The back-propagation algorithm builds a computational graph that defines how to calculate all the necessary derivative terms. In 1986, Rumelhart et al. showed how back-propagation of errors could be used for learning [42]. Today, calculating the gradients of neural networks during training is almost exclusively done with back-propagation [41].

$$\begin{aligned}
\frac{\partial J}{\partial W^L} &= \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{v}^L}\frac{\partial \mathbf{v}^L}{\partial W^L} = \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{v}^L}[\mathbf{u}^{L-1}]^T \\
\frac{\partial J}{\partial \mathbf{b}^L} &= \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{v}^L}\frac{\partial \mathbf{v}^L}{\partial \mathbf{b}^L} = \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{v}^L} \\
\frac{\partial J}{\partial W^{L-1}} &= \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{u}^{L-1}}\frac{\partial \mathbf{u}^{L-1}}{\partial \mathbf{v}^{L-1}}[\mathbf{u}^{L-2}]^T \\
&\vdots \\
\frac{\partial J}{\partial W^1} &= \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{u}^{L-1}}\cdots\frac{\partial \mathbf{u}^2}{\partial \mathbf{u}^1}\frac{\partial \mathbf{u}^1}{\partial \mathbf{v}^1}\mathbf{x}^T \\
\frac{\partial J}{\partial \mathbf{b}^1} &= \frac{\partial J}{\partial \mathbf{u}^L}\frac{\partial \mathbf{u}^L}{\partial \mathbf{u}^{L-1}}\cdots\frac{\partial \mathbf{u}^2}{\partial \mathbf{u}^1}\frac{\partial \mathbf{u}^1}{\partial \mathbf{v}^1}
\end{aligned} \tag{2.12}$$

There are many different possible schemes for performing gradient descent, and only some of them are explained here. Assume the supervised setting, so that n training examples $(\mathbf{x}^i, \mathbf{y}^i)$ are available.

Batched gradient descent

In what is known as batched, or vanilla, gradient descent, the gradient is computed based on all available training examples. The parameters are then updated as shown in (2.13), where η, the learning rate, decides how much the parameters are perturbed. Here, $X = \begin{bmatrix}\mathbf{x}^1 & \dots & \mathbf{x}^n\end{bmatrix}$ and $Y = \begin{bmatrix}\mathbf{y}^1 & \dots & \mathbf{y}^n\end{bmatrix}$.

$$\theta = \theta - \eta\nabla_\theta J(\theta; X, Y) \tag{2.13}$$

The number of times the parameters are updated using the same subset of the training data is known as the number of epochs N_E of training. Using all training examples to compute the gradient is often infeasible due to memory constraints. Further, it does not allow online learning; all training examples must be available before training starts [43].

Stochastic gradient descent

Stochastic gradient descent (SGD) performs updates by calculating the gradient of one training example at a time, as indicated by (2.14). Performing stochastic

updates enables online learning, but as the gradient is estimated based on one single example, the variance of the updates is high. High variance results in the updates not always changing the parameters towards the closest local minimum, and typically when using SGD, the loss will not steadily decrease but fluctuate heavily. Convergence of this algorithm can be achieved at the same rate as with batched gradient descent by steadily decreasing the learning rate throughout the training [43].

$$\theta = \theta - \eta \nabla_\theta J(\theta; \mathbf{x}^i, \mathbf{y}^i) \tag{2.14}$$

Mini-batch gradient descent

As a compromise between batched gradient descent and SGD, mini-batched gradient descent performs updates on small subsets of the training examples at a time. If the mini-batch is of size N_{MB}, N_{MB} samples are drawn randomly from the n training samples. Increasing the number of samples reduces the variance of stochastic gradient descent, while still being computationally feasible [43]. Let the subset of training examples drawn for the mini-batch be denoted by X_{MB}, Y_{MB}. The resulting update rule is shown in (2.15).

$$\theta = \theta - \eta \nabla_\theta J(\theta; X_{\text{MB}}, Y_{\text{MB}}) \tag{2.15}$$

Adaptive moment estimation

Many improvements to gradient descent, mostly aimed at reducing variance and speeding up convergence, have been suggested. One such method is Adaptive Moment Estimation (Adam) that makes use of moving averages of the first and second moments of the gradient [44].

Instead of updating the weights in the direction of the last gradient, Adam updates them in the direction of the moving average of the gradients m_t, shown in (2.16). This way, the update direction becomes less sensitive to the statistical properties of the current gradient estimate and more dependent on the average direction over the last updates. When using gradient descent, some parameters might experience consistently large gradients, while others are barely updated, as the same learning rate is applied to all elements of θ. Scaling the update of each parameter by the amount the parameter has been updated previously, smooths out this difference. In Adam, this is done by scaling the learning rate by the moving average of the gradient estimate squared, seen in (2.17).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta J(\theta) \tag{2.16}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta))^2 \tag{2.17}$$

As Kingma and Ba point out in their paper, the estimated moments (2.16) and (2.17) are biased towards zero, which they counteract by using the bias-corrected

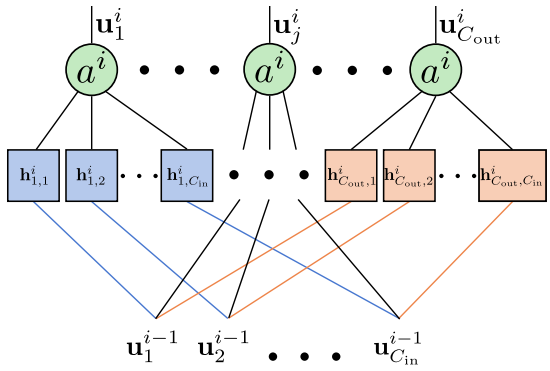


Figure 2.5: A one-dimensional convolutional layer has C_{in} input channels and C_{out} output channels. Here, rectangles represent convolution, and circles represent summation, bias addition and application of the activation function. Each of the output channels is computed as in (2.20).

moments in (2.18). The final update rule of Adam is shown in (2.19), where ϵ is a small number added to avoid division by zero. The authors suggest the following values for the parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$.

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2} \tag{2.18}$$

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \nabla_\theta J(\theta) \tag{2.19}$$

2.5.3 Convolutional neural networks

A convolutional neural network (CNN) is a neural network where at least one of the layers performs a convolution operation on its input. Here, CNNs with one-dimensional filters for convolution in the time dimension is considered.

A one-dimensional convolutional layer is defined by the number of input channels C_{in}, the number of output channels C_{out}, the filter length N, and the choice of padding of the input. A vanilla convolutional layer is illustrated in Figure 2.5

The padding parameter of a convolutional layer decides what part of the convolution to calculate. There are three common modes of padding, usually referred to as *valid*, *same* and *causal*. Only the causal padding mode is considered here. When the padding is causal, only the first T terms of the convolution is performed, where T is the input length. This is equivalent to removing the last N rows of the matrix in (2.4).

The input to a convolutional layer consists of C_{in} vectors $\mathbf{u}^i \in \mathbb{R}^T$, where T is the number of time-steps. Each of the C_{out} output channels of the layer are

calculated as in (2.20), where a^i denotes some activation function, and the bias b^i_j is added element-wise.

An important size measure of a CNN is its receptive field size s^r, which is defined in Definition 2.1. An illustration of the receptive field of a two-layer CNN with one input and one output channel for each layer is shown in Figure 2.6.

$$\mathbf{u}^i_j = a^i(b^i_j + \sum_{k=1}^{C_{\text{in}}} \mathbf{h}^i_{j,k} * \mathbf{u}^{i-1}_k),\ j \in \{1, 2, \ldots, C_{\text{out}}\} \tag{2.20}$$

Definition 2.1 (Receptive field)
The receptive field size of a CNN layer is the number of elements of the network input that influence the layer output. Let a CNN have L_c undilated convolutional layers, where each layer applies filters of length N_i, $i \in \{1, \ldots, L_c\}$. The receptive field size s^r_i of layer i is computed by solving the recursion

$$s^r_i = s^r_{i-1} + N_i - 1, \quad s^r_0 = 1.$$

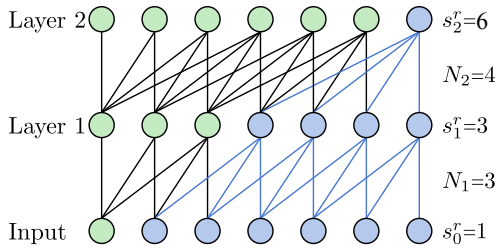


Figure 2.6: The receptive field of a CNN with two layers with filter lengths 3 and 4, and one input and one output channel for each layer.

2.6 The Lipschitz constant

The Lipschitz condition is given by (2.21). If this condition holds for all x, y in a neighborhood of some point x_0, the function f is said to be locally Lipschitz in this neighborhood. If (2.21) hold for all values of x and y, f is globally Lipschitz and Lipschitz continuous. A Lipschitz continuous function is illustrated in Figure 2.7.

Every Λ that satisfy (2.21) in some neighborhood, is called a Lipschitz constant of f in that neighborhood. The smallest Lipschitz constant gives a bound on how much the output of f can change, given some change in the input [45].

$$\left\| f(x) - f(y) \right\| \leq \Lambda \left\| x - y \right\| \tag{2.21}$$

If the Lipschitz constant of a function is known, a worst case evaluation of the effect of a perturbation z to the input can be estimated as

$$\left\|f(x+z) - f(x)\right\| \leq \Lambda \left\|z\right\|.$$

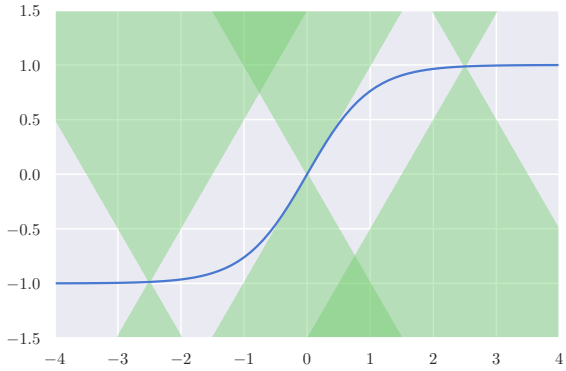


Figure 2.7: A Lipschitz continuous function will always stay outside the infinite green cone, as the center of the cone moves along the trajectory of the function. The slope of the cone is equal to the tightest Lipschitz constant of the function. Here, $\tanh(x)$, which has Lipschitz constant $\Lambda = 1$, is shown with a cone of slope 1 centered at three different points of its trajectory. As seen here, $\tanh(x)$ will always stay outside the cone.

2.6.1 Naive bounds on the Lipschitz constant for feedforward networks

Lemma 2.1 (Lemma 3.1 from Khalil, 2013 [45])
Let $f : \mathcal{D} \to \mathbb{R}^m$ be continuous for some domain $\mathcal{D} \subset \mathbb{R}^n$. Suppose that $\frac{\partial f}{\partial x}$ exists and is continuous on $\mathcal{D}$. If, for a convex subset $\mathcal{W} \subset \mathcal{D}$, there is a constant $\Lambda \geq 0$ such that

$$\left\|\frac{\partial f}{\partial x}\right\| \leq \Lambda$$

for all $x \in \mathcal{W}$, then

$$\left\|f(x) - f(y)\right\| \leq \Lambda \left\|x - y\right\|, \forall\, x, y \in \mathcal{W}.$$

Lemma 2.2
Let $f : \mathcal{D}_f \to \mathbb{R}^m$ be continuous in some domain $\mathcal{D}_f \subset \mathbb{R}^n$. Let $g : \mathcal{D}_g \to$ Image(g) be continuous in some domain $\mathcal{D}_g \subset \mathbb{R}^p$, where Image$(g) \subseteq \mathcal{D}_f$. Assume that f is Lipschitz continuous in $\mathcal{D}_f$ with Lipschitz constant Λ_f, and that g is Lipschitz continuous in $\mathcal{D}_g$ with Lipschitz constant Λ_g. Then

$$\begin{aligned}\left\|f(y_1) - f(y_2)\right\| = \left\|f(g(x_1)) - f(g(x_2))\right\| \leq \Lambda_f \left\|g(x_1) - g(x_2)\right\| \leq \Lambda_f \Lambda_g \left\|x_1 - x_2\right\| \\ \Rightarrow \left\|f(x_1) - f(x_2)\right\| \leq \Lambda_f \Lambda_g \left\|x_1 - x_2\right\|.\end{aligned}$$

Hence, f is Lipschitz continuous in $\mathcal{D}_g$ with Lipschitz constant $\Lambda_f \Lambda_g$.

Consider the FFN layer presented in Section 2.5, and again adopt the notation $\mathbf{u}^i = a^i(\mathbf{v}^i)$. The gradient of a layer output $\mathbf{u}^i$ with respect to its input layer $\mathbf{u}^{i-1}$ is

$$\begin{aligned}\frac{\partial \mathbf{u}^i}{\partial \mathbf{u}^{i-1}} &= \frac{\partial a^i}{\partial \mathbf{v}^{i-1}} W^i \\ &= \operatorname{diag}\left(\begin{bmatrix} \frac{\partial a^i}{\partial v_1^{i-1}} & \frac{\partial a^i}{\partial v_2^{i-1}} \cdots \frac{\partial a^i}{\partial v_{n_i}^{i-1}} \end{bmatrix}\right) W^i.\end{aligned}$$

Recall that the activation functions of neural networks are typically of type ReLU, tanh or sigmoid. All these functions are Lipschitz continuous and their derivatives are contained in the interval $[0, 1]$. Thus, the theoretical maximum of the gradient above is achieved when $(\partial a^i)/(\partial v_j^{i-1}) = 1 \; \forall \, j \in \{1, 2, \ldots, n_i\}$. Hence, the partial derivatives are bounded, and by Lemma 2.1, the FFN layer has a Lipschitz constant $\Lambda^i = \left\|W^i\right\|$:

$$\begin{aligned}\left\| \frac{\partial \mathbf{u}^i}{\partial \mathbf{u}^{i-1}} \right\| &= \left\| \operatorname{diag}(\begin{bmatrix} \frac{\partial a^i}{\partial v_1^{i-1}} & \frac{\partial a^i}{\partial v_2^{i-1}} \cdots \frac{\partial a^i}{\partial v_{n_i}^{i-1}} \end{bmatrix}) W^i \right\| \\ &\leq \left\| \operatorname{diag}(\begin{bmatrix} \frac{\partial a^i}{\partial v_1^{i-1}} & \frac{\partial a^i}{\partial v_2^{i-1}} \cdots \frac{\partial a^i}{\partial v_{n_i}^{i-1}} \end{bmatrix}) \right\| \left\| W^i \right\| \\ &\leq \|\mathbf{I}\| \left\| W^i \right\| \leq \left\| W^i \right\|.\end{aligned}$$

Thus, by Lemma 2.2 a FFN with L layers is Lipschitz continuous in its input space with a naive upper bound on the Lipschitz constant given by (2.22). A lower bound on the Lipschitz constant of the same FFN is given in (2.23), as proven by Combettes and Pesquet. The naive upper bound is provably quite loose, but serves as a sanity check [46].

$$\Lambda_N^U = \prod_{i=1}^{L} \left\| W^i \right\| \tag{2.22}$$

$$\Lambda_N^L = \left\| \prod_{i=1}^{L} W^i \right\| \tag{2.23}$$

2.6.2 Spectral normalization

Spectral normalization is a novel regularization technique developed for stabilizing the training process of Generative Adversarial Networks [47]. The technique

is easily applied to FFNs, and allows us to fix the naive upper bound on the Lipschitz constant (2.22) to some value $\gamma > 0$.

Recall from Section 2.1 that the 2-norm, also called the spectral norm, of a matrix is equal to the largest singular value of the matrix. Hence, the naive upper bound on the Lipschitz constant is simply a product of the largest singular values of the layer weight matrices. Miyato et al. therefore suggest normalizing the weight matrices by their largest singular value, such that the normalized matrix has 2-norm equal to 1:

$$\bar{W} = \frac{W}{\sigma_{\max}(W)} \Rightarrow \left\|\bar{W}\right\| = 1$$

Further, exchanging the weight matrices for their normalized counterparts multiplied by $\gamma^{\frac{1}{L}}$ leads to a naive upper bound equal to γ:

$$\Lambda_N^U = \prod_{i=1}^{L} \left\| \frac{W^i}{\sigma_{\max}(W)} \right\| \gamma^{\frac{1}{L}} = \gamma.$$

The largest singular values of the weight matrices can be estimated using the power iteration method, as suggested by Miyato et al..

2.6.3 Efficient estimation of the Lipschitz constant for feed-forward networks: LipSDP

Here, the method called LipSDP, for Lipschitz Semidefinite Program, is described. It was developed by Fazlyab et al. for estimating the Lipschitz constant of FFNs [4].

Definition 2.2 (Slope-restricted function)
Let $f : \mathbb{R} \to \mathbb{R}$ satisfy (2.24), where $0 \leq \alpha < \beta < \infty$. Then f is said to be slope restricted on $[\alpha, \beta]$.

$$\alpha \leq \frac{\varphi(y) - \varphi(x)}{y - x} \leq \beta, \; \forall \, x, y \in \mathbb{R} \tag{2.24}$$

Let φ be a slope-restricted non-linearity, as described in Definition 2.2. The criterion for slope-restriction (2.24) can be rewritten into the form of (2.25a), which is equivalent to (2.25b) as long as $\lambda \geq 0$.

$$0 \leq \begin{bmatrix} x - y \\ \varphi(x) - \varphi(y) \end{bmatrix}^T \begin{bmatrix} -2\alpha\beta & \alpha + \beta \\ \alpha + \beta & -2 \end{bmatrix} \begin{bmatrix} x - y \\ \varphi(x) - \varphi(y) \end{bmatrix} \tag{2.25a}$$

$$0 \leq \begin{bmatrix} x - y \\ \varphi(x) - \varphi(y) \end{bmatrix}^T \lambda \begin{bmatrix} -2\alpha\beta & \alpha + \beta \\ \alpha + \beta & -2 \end{bmatrix} \begin{bmatrix} x - y \\ \varphi(x) - \varphi(y) \end{bmatrix} \tag{2.25b}$$

The idea from (2.25b) can be extended to reflect the interaction between elements of a vector of non-linearities $\phi(\mathbf{x}) = \begin{bmatrix}\varphi(x_1) & \varphi(x_2) & \dots & \varphi(x_n)\end{bmatrix}$. If all all elements of $\phi(\mathbf{x})$ are outputs of the same slope-restricted function on $[\alpha, \beta]$ applied on n different inputs, $\phi(\mathbf{x})$ satisfies (2.27). Here, $T \in \mathcal{T}_n$, were $\mathcal{T}_n$ is described in (2.26), and $\mathbb{S}^n$ is the set of symmetric matrices of size $n \times n$.

$$\mathcal{T}_n = \{T \in \mathbb{S}^n | T = \sum_{i=1}^{n} \lambda_{ii} e_i e_i^T + \sum_{1 \le i < j \le n} \lambda_{ij}(e_i - e_j)(e_i - e_j)^t, \lambda_{kl} \ge 0\} \quad (2.26)$$

$$0 \le \begin{bmatrix}\mathbf{x}-\mathbf{y} \\ \phi(\mathbf{x}) - \phi(\mathbf{y})\end{bmatrix}^T \begin{bmatrix}-2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T\end{bmatrix} \begin{bmatrix}\mathbf{x}-\mathbf{y} \\ \phi(\mathbf{x}) - \phi(\mathbf{y})\end{bmatrix} \quad (2.27)$$

Consider the L-layer FFN applying the same activation function a at every layer, except for the output layer which has no activation:

$$g(\mathbf{x}) = \mathbf{u}^L = W^L a^{L-1}(\dots a^2(W^2 a^1(W^1\mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2))\dots) + \mathbf{b}^L$$

As in Section 2.5, $\mathbf{u}^i$ denotes the output of layer i. This FFN can be rewritten to the compact form of (2.28), where $\mathbf{z}^T = \begin{bmatrix}\mathbf{x}^T & (\mathbf{u}^1)^T & \dots & (\mathbf{u}^{L-1})^T\end{bmatrix}$, and A, B, C and $\mathbf{b}$ are as in (2.29).

$$B\mathbf{z} = a(A\mathbf{z} + \mathbf{b}), \quad g(\mathbf{x}) = C\mathbf{z} + \mathbf{b}^L \quad (2.28)$$

$$A = \begin{bmatrix} W^1 & 0 & \dots & 0 & 0 \\ 0 & W^2 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & W^{L-1} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & \dots & 0 & I_{n_1} & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & I_{n_2} & \ddots & \vdots \\ \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 & I_{n_L} \end{bmatrix}$$
$$C = \begin{bmatrix}0 & \dots & 0 & W^L\end{bmatrix}, \quad \mathbf{b}^T = \begin{bmatrix}(\mathbf{b}^1)^T & (\mathbf{b}^2)^T & \dots & (\mathbf{b}^{L-1})^T\end{bmatrix} \quad (2.29)$$

By realizing that $B\mathbf{z} = a(A\mathbf{z} + \mathbf{b})$ is just a slope restricted function applied to some input vector $A\mathbf{z} + \mathbf{b}$, the compact version of the FFN can be inserted into (2.27) to yield

$$0 \le \begin{bmatrix}A\mathbf{z}^1 - A\mathbf{z}^2 \\ a(A\mathbf{z}^1 + \mathbf{b}) - a(A\mathbf{z}^2 + \mathbf{b})\end{bmatrix}^T \begin{bmatrix}-2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T\end{bmatrix} \begin{bmatrix}A\mathbf{z}^1 - A\mathbf{z}^2 \\ a(A\mathbf{z}^1 + \mathbf{b}) - a(A\mathbf{z}^2 + \mathbf{b})\end{bmatrix}$$
$$0 \le (\mathbf{z}^1 - \mathbf{z}^2)^T \begin{bmatrix}A \\ B\end{bmatrix}^T \begin{bmatrix}-2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T\end{bmatrix} \begin{bmatrix}A \\ B\end{bmatrix} (\mathbf{z}^1 - \mathbf{z}^2)$$

Now, assume that some $T \in \mathcal{T}_n$ and $\Lambda > 0$ can be found such that (2.30) is fulfilled. Then, the right hand side of (2.30) must be non-negative, as the left hand side is non-negative by the slope restriction condition. From the derivation shown in (2.31), it can be concluded that Λ is a Lipschitz constant for the FFN.

$$\begin{aligned}(\mathbf{z}^1 - \mathbf{z}^2)^T \begin{bmatrix} A \\ B \end{bmatrix}^T \begin{bmatrix} -2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} (\mathbf{z}^1 - \mathbf{z}^2) \\ \leq (\mathbf{z}^1 - \mathbf{z}^2)^T \begin{bmatrix} \Lambda^2 I_{n_0} & 0 & \dots & 0 \\ 0 & \vdots & \ddots & \vdots \\ \vdots & 0 & \dots & 0 \\ 0 & \dots & 0 & -(W^L)^T W^L \end{bmatrix} (\mathbf{z}^1 - \mathbf{z}^2)\end{aligned} \tag{2.30}$$

$$0 \leq (\mathbf{z}^1 - \mathbf{z}^2)^T \begin{bmatrix} \Lambda^2 I_{n_0} & 0 & \dots & 0 \\ 0 & \vdots & \ddots & \vdots \\ \vdots & 0 & \dots & 0 \\ 0 & \dots & 0 & -(W^L)^T W^L \end{bmatrix} (\mathbf{z}^1 - \mathbf{z}^2) \tag{2.31a}$$

$$0 \leq (x^1 - x^2)^T \Lambda^2 (x^1 - x^2) - (\mathbf{u}_1^{L-1} - \mathbf{u}_2^{L-1})^T (W^L)^T W^L (\mathbf{u}_1^{L-1} - \mathbf{u}_2^{L-1}) \tag{2.31b}$$

$$\left(W^L(\mathbf{u}_1^{L-1} - \mathbf{u}_2^{L-1})\right)^T W (\mathbf{u}_1^{L-1} - \mathbf{u}_2^{L-1}) \leq \Lambda^2 (x^1 - x^2)^T (x^1 - x^2) \tag{2.31c}$$

$$\left\| g(x^1) - g(x^2) \right\|^2 \leq \Lambda^2 \left\| x^1 - x^2 \right\|^2 \tag{2.31d}$$

To find the smallest Λ satisfying (2.30), the SDP in (2.33) is solved, where $M(\Lambda, T)$ is as in (2.32). This is a SDP with $1 + \frac{N(N+1)}{2}$ decision variables, where $N = \sum_{j=1}^{L-1} n_j$ is the number of neurons in the FFN. Solving a SPD with $\mathcal{O}(N^2)$ decision variables is infeasible for large networks without extraordinary amounts of computational power. Therefore, the authors of [4] suggest replacing T with a diagonal matrix, reducing the number of variables to $N + 1$. This relaxation does not invalidate the proofs of finding a Lipschitz constant for the FFN, though it will be slightly looser than the one found with $T \in \mathcal{T}_N$.

$$M(\Lambda, T) = \begin{bmatrix} A \\ B \end{bmatrix}^T \begin{bmatrix} -2\alpha\beta T & (\alpha+\beta)T \\ (\alpha+\beta)T & -2T \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} + \begin{bmatrix} -\Lambda^2 I_{n_0} & 0 & \dots & 0 \\ 0 & \vdots & \ddots & \vdots \\ \vdots & 0 & \dots & 0 \\ 0 & \dots & 0 & (W^L)^T W^L \end{bmatrix} \tag{2.32}$$

$$\min_{\Lambda,T} \Lambda \quad \text{s.t.} \quad M(\Lambda, T) \preceq 0, \quad \Lambda > 0, \quad T \in \mathcal{T}_N \tag{2.33}$$

2.7 Nonlinear control theory

Established results from nonlinear control theory are extensively used in this thesis. All necessary information to follow the reasoning can be found in Nonlinear Systems by Khalil (2015) [45]. For ease of reading, the definitions, lemmas and theorems that are used later are reiterated in Appendix A.

3 Methodology and Theoretical Results

In this chapter, a method for estimating the Lipschitz constant of CNNs is presented. Next, a feedback linearizing controller with a FFN estimating unknown dynamics is suggested. Two theorems summarize the stability and convergence properties of the FFN controller. A CNN controller for time-delayed systems is also suggested. Then, a mass-spring-damper system that will be used for experiments is introduced, before the NNs that are trained for the experiments are presented.

3.1 LipSDP for convolutional neural networks with temporal convolutions

The LipSDP method, presented in Section 2.6.3, can be applied to convolutional neural networks by rethinking them as structured feedforward networks. By doing this, an efficient and accurate method for estimating the Lipschitz constant of CNNs is gained.

Recall formula (2.20) for calculating the output of a convolutional layer with C_{in} input channels and C_{out} output channels:

$$\mathbf{u}_j^{i+1} = a^{i+1}(\mathbf{b}_j^{i+1} + \sum_{k=1}^{C_{\text{in}}} \mathbf{h}_{j,k}^{i+1} * \mathbf{u}_k^i),\ j \in \{1, 2, \ldots, C_{\text{out}}\} \qquad \text{(2.20 revisited)}$$

Technically, this equation can be fed an input of any length T:

$$\mathbf{u}_k^i = \begin{bmatrix} u_k^i[1] & u_k^i[2] & \ldots & u_k^i[T] \end{bmatrix}^T.$$

However, if T is larger than the receptive field size (Definition 2.1) s^r of the last convolutional layer of the network, the exact same outputs can be restored by feeding the CNN several inputs

$$\begin{bmatrix} u_k^i[1] & \ldots & u_k^i[s^r] \end{bmatrix}^T$$
$$\begin{bmatrix} u_k^i[2] & \ldots & u_k^i[s^r+1] \end{bmatrix}^T$$
$$\vdots$$
$$\begin{bmatrix} u_k^i[T-s^r+1] & \ldots & u_k^i[T] \end{bmatrix}^T.$$

In the case that $T < s^r$, the first $s^r - T$ elements of the input vector might be set to zero. This justifies the assumption that the length of the input is always equal to the receptive field size without loss of generality.

Under this assumption, the filters from (2.20) can be rewritten in the fashion of (2.4), by dropping the last N rows. Then each filter $\mathbf{h}_{j,k}^{i+1}$ of length N_i has its matrix equivalent $H_{j,k}^{i+1} \in \mathbb{R}^{s^r \times s^r}$ as shown in (3.1).

$$H_{j,k}^{i+1} = \begin{bmatrix} h_{j,k}^{i+1}[1] & 0 & 0 & 0 & 0 & \dots & 0 \\ h_{j,k}^{i+1}[2] & h_{j,k}^{i+1}[1] & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \ddots & \vdots \\ h_{j,k}^{i+1}[N_i] & \dots & h_{j,k}^{i+1}[2] & h_{j,k}^{i+1}[1] & 0 & \dots & 0 \\ 0 & h_{j,k}^{i+1}[N_i] & \dots & h_{j,k}^{i+1}[2] & h_{j,k}^{i+1}[1] & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & h_{j,k}^{i+1}[N_i] & \dots & h_{j,k}^{i+1}[2] & h_{j,k}^{i+1}[1] \end{bmatrix} \tag{3.1}$$

By adopting this causal matrix form of the filters and stacking the input vectors vertically, (2.20) can be rewritten as a matrix multiplication that returns the output stacked vertically. The final expression for this is (3.2). This shows that a convolutional layer is just a feed forward layer with structured weight matrices.

$$\begin{aligned} \mathbf{u}_j^{i+1} &= a^{i+1}(\mathbf{b}_j^{i+1} + \sum_{k=1}^{C_{\text{in}}} H_{j,k}^{i+1}\mathbf{u}_k^i) \\ &= a^{i+1}(\mathbf{b}_j^{i+1} + \begin{bmatrix} H_{j,1}^{i+1} & H_{j,2}^{i+1} & \dots & H_{j,C_{\text{in}}}^{i+1} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^i \\ \mathbf{u}_2^i \\ \vdots \\ \mathbf{u}_{C_{\text{in}}}^i \end{bmatrix}) \end{aligned}$$

$$\begin{bmatrix} \mathbf{u}_1^{i+1} \\ \mathbf{u}_2^{i+1} \\ \vdots \\ \mathbf{u}_{C_{\text{out}}}^{i+1} \end{bmatrix} = a^{i+1}(\begin{bmatrix} \mathbf{b}_1^{i+1} \\ \mathbf{b}_2^{i+1} \\ \vdots \\ \mathbf{b}_{C_{\text{out}}}^{i+1} \end{bmatrix} + \begin{bmatrix} H_{1,1}^{i+1} & H_{1,2}^{i+1} & \dots & H_{1,C_{\text{in}}}^{i+1} \\ H_{2,1}^{i+1} & H_{2,2}^{i+1} & \dots & H_{2,C_{\text{in}}}^{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ H_{C_{\text{out}},1}^{i+1} & H_{C_{\text{out}},2}^{i+1} & \dots & H_{C_{\text{out}},C_{\text{in}}}^{i+1} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^i \\ \mathbf{u}_2^i \\ \vdots \\ \mathbf{u}_{C_{\text{in}}}^i \end{bmatrix}) \tag{3.2}$$

$$\mathbf{u}^{i+1} = a^{i+1}(\mathbf{b}^{i+1} + H_{\text{all}}^{i+1}\mathbf{u}^i)$$

When only a prediction of the next time step is desired, it is common to flatten the output of the last convolutional layer and add one or more fully connected layers. The flattened output either has the form of (3.2), or is shaped as

$\mathbf{u}^i = \begin{bmatrix} u_1^i[1] & u_2^i[1] & \dots & u_{C_{\text{out}}}^i[1] & u_1^i[1] & \dots u_{C_{\text{out}}}^i[T] \end{bmatrix}^T$. Either way, it is always possible to reshape the weight matrix of the following fully connected layer such that its output is $\mathbf{u}^{i+1} = a^{i+1}(W^{i+1}\mathbf{u}^i + \mathbf{b}^{i+1})$.

The LipSDP program for CNNs was implemented in MATLAB using CVX, which is a package for specifying and solving convex programs [5], [6].

3.2 Feedback linearizing neural network controller

In this section, a neural network controller based on feedback linearization is derived. It is proven that the resulting closed-loop system is finite gain $\mathcal{L}_p$ stable, input-to-state stable and that it converges to an error ball centered at the origin whose radius is possible to influence. It is also shown that the Lipschitz constant of the neural network affects the noise rejection capabilities of the controller. Finally, it is shown how these results change when the assumption that the controller can be updated continuously is violated. These results are presented in two theorems, Theorem 3.1 and Theorem 3.2, which are proven in Section 3.2.2 and Section 3.2.3.

3.2.1 Controller design

Consider all systems of the form (3.3), where c_{11} and c_{12} are known constants, and $d(\mathbf{x}, u)$ is a continuous, unknown, nonlinear mapping. The objective is making x_1 track the trajectory $x_1^r(t)$, or equivalently, driving (3.4a) to zero. It is assumed that the trajectory is twice continuously differentiable, and that its derivatives are known.

$$\begin{aligned} \dot{x}_1 &= c_{11}x_1 + c_{12}x_2 \\ \dot{x}_2 &= d(\mathbf{x}, u) + bu \end{aligned} \tag{3.3}$$

Define the reference value (3.5), such that the composite variable z_2 can be defined as in (3.4b), where Γ is a freely chosen parameter. Rearranging (3.4b) yields the dynamics of z_1 in (3.6a). Taking the derivative of z_2 with respect to time results in (3.6b). The dynamics of $\mathbf{z} = \begin{bmatrix} z_1 & z_2 \end{bmatrix}$ is a form of error dynamics of (3.3), when the objective is trajectory tracking of $x_1^r(t)$.

It should be noted that when $c_{11} = 0$ and $c_{12} = 1$, x_1 can be seen as the position, and x_2 as the velocity, of (3.3). If z_1 is driven to the origin, z_2 is the difference between x_2 and $\dot{x}_1^r(t)$, which is an intuitive velocity error for the system.

$$z_1 = x_1 - x_1^r(t) \tag{3.4a}$$

$$z_2 = \frac{1}{c_{12}}(\dot{z}_1 + \Gamma z_1) = \frac{1}{c_{12}}(\dot{x}_1 - \dot{x}_1^r(t) + \Gamma z_1) = x_2 - x_2^r(t) \tag{3.4b}$$

$$x_2^r(t) = -\frac{1}{c_{12}}(c_{11}x_1 - \dot{x}_1^r(t) + \Gamma z_1) \tag{3.5}$$

By Definition A.1, systems of the form (3.3) are feedback linearizable under the change of variables $\mathbf{z} = \begin{bmatrix} x_1 - x_1^r(t) & x_2 - x_2^r(t) \end{bmatrix}$, given that $d(\mathbf{x}, u)$ is continuous. This diffeomorphism transforms the system into the form of (A.1), as shown in (3.7). The pair (A, B) is controllable for all $c_{12} \neq 0$.

$$\dot{z}_1 = -\Gamma z_1 + c_{12} z_2 \tag{3.6a}$$

$$\dot{z}_2 = d(\mathbf{x}, u) + bu - \dot{x}_2^r(t) \tag{3.6b}$$

$$\begin{aligned} \dot{\mathbf{z}} &= \begin{bmatrix} -\Gamma & c_{12} \\ 0 & 0 \end{bmatrix} \mathbf{z} + \begin{bmatrix} 0 \\ b \end{bmatrix} \left(u + \frac{1}{b}\left[d(\mathbf{x}, u) - \dot{x}_2^r(t)\right]\right) \\ \dot{\mathbf{z}} &= A\mathbf{z} + B(u - \frac{1}{b}\left[-d(\mathbf{x}, u) + \dot{x}_2^r(t)\right]) \end{aligned} \tag{3.7}$$

Let $\hat{d}(\mathbf{x}(t), u(t))$ be the output of a neural network estimating $d(\mathbf{x}, u)$ by mapping the states and input to a real number: $\hat{d} : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$. Define the estimation error of the neural network as $v(t) = d(\mathbf{x}, u) - \hat{d}(\mathbf{x}, u)$. Assuming that new estimates are continuously available, the controller (3.8) is proposed, where $K = \begin{bmatrix} k_1 & k_2 \end{bmatrix}$. Applying this controller to (3.7) yields the closed loop dynamics in (3.9).

$$u = -K\mathbf{z} - \frac{1}{b}\left[\hat{d}(\mathbf{x}(t), u(t)) - \dot{x}_2^r(t)\right] \tag{3.8}$$

$$\dot{\mathbf{z}} = (A - BK)\mathbf{z} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} v(t) \tag{3.9}$$

Theorem 3.1 (Closed-loop stability and convergence of a continuous NN controller.)
Let $\hat{d} : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$ be a neural network estimating the unknown dynamics of (3.3)*, and define $v(t) = d(\mathbf{x}, u) - \hat{d}(\mathbf{x}, u)$. Assume that*

(i) The error of the NN estimate is bounded: $\|v(t)\| \leq \varepsilon$.

(ii) The estimator $\hat{d}(\mathbf{x}, u)$ is Lipschitz continuous with Lipschitz constant Λ.

(iii) The estimate of the unknown dynamics can be updated continuously.

Let the input u of (3.3) *be given by the continuous NN controller* (3.8)*, where K and Γ satisfy* (3.10)*.*

$$k_1 = -\frac{c_{12}}{b}, \quad \Gamma > 0, \quad \Gamma k_2 > \frac{c_{12}^2}{b} \tag{3.10}$$

Then, the resulting closed loop system (3.9) *is input-to-state and finite-gain* $\mathcal{L}_p$*-stable from* $v(t)$ *to* $\mathbf{z}(t)$*.*

The error norm converges globally and exponentially to a ball centered at the origin:

$$\lim_{t\to\infty} \left\|\mathbf{z}(t)\right\| \leq \left|\frac{\varepsilon}{\lambda}\right|, \tag{3.11}$$

where λ *denotes the least negative eigenvalue of* $A - BK$*.*

Further, if the measurements of $\mathbf{x}$ *are affected by some additive bounded noise* $\mathbf{q}^x = [q_1 \quad q_2]^T$ *satisfying* $|q_1| \leq q_1^m$*,* $|q_2| \leq q_2^m$*, the error norm converges to*

$$\lim_{t\to\infty} \left\|\mathbf{z}(t)\right\| \leq \left|\frac{1}{\lambda}\left(\Lambda \left\|\mathbf{q}^x\right\| + B^q + \varepsilon\right)\right| \tag{3.12}$$

where $B^q = |c_{12} + \frac{bk_2+1}{c_{12}}(c_{11} + \Gamma)|q_1^m + |k_2 b|q_2^m$*.*

Even though the forward pass of a neural network is relatively quick, the assumption that the controller in (3.8) can be updated continuously might be violated. Therefore, the scenario where a new estimate can only be made every $\Delta t_{\max}$ seconds is considered. This results in the controller (3.13), where $\Delta t \leq \Delta t_{\max}$ is the time since the last estimate was made. It is assumed that the linear term of the controller can be updated continuously. Applying (3.13) to (3.7) yields the perturbed system (3.14), where $w(t) = d(\mathbf{x}(t), u(t)) - \hat{d}(\mathbf{x}(t-\Delta t), u(t-\Delta t))$.

$$u(t) = -K\mathbf{z}(t) - \frac{1}{b}[\hat{d}(\mathbf{x}(t-\Delta t), u(t-\Delta t)) - \dot{x}_2^r(t)] \tag{3.13}$$

$$\dot{\mathbf{z}} = (A - BK)\mathbf{z} + \begin{bmatrix}0\\1\end{bmatrix} w(t) \tag{3.14}$$

Theorem 3.2 (Closed-loop convergence of a discrete NN controller.)
Define $w(t) = d(\mathbf{x}(t), u(t)) - \hat{d}(\mathbf{x}(t-\Delta t), u(t-\Delta t))$*, and assume that*

(i) The error of the NN estimate is bounded: $\left\|d(\mathbf{x}(t), u(t)) - \hat{d}(\mathbf{x}(t), u(t))\right\| \leq \varepsilon$*.*

(ii) The estimator $\hat{d}(\mathbf{x}, u)$ *is Lipschitz continuous with Lipschitz constant* Λ*.*

(iii) The change in the system state is bounded: $\left\|\mathbf{x}(t) - \mathbf{x}(t-\Delta t)\right\| \leq \rho^x$ *for some small time-step* $\Delta t \leq \Delta t_{max}$*.*

(iv) The change in the system input is bounded: $\left\|u(t) - u(t-\Delta t)\right\| \leq \rho^u$ *for some small time-step* $\Delta t \leq \Delta t_{max}$*.*

Let u be given by the discretely updated NN controller (3.13) *where K and Γ satisfy* (3.10)*, and let λ denote the least negative eigenvalue of $A - BK$. Then, the resulting closed loop system* (3.14) *is input-to-state and finite-gain $\mathcal{L}_p$-stable from $w(t)$ to $\mathbf{z}(t)$.*

The error norm converges globally and exponentially to an error-ball centered at the origin:

$$\lim_{t\to\infty} \left\|\mathbf{z}(t)\right\| \leq \left| \frac{(\Lambda(\rho^x + \rho^u) + \varepsilon)}{\lambda} \right|. \tag{3.15}$$

Further, if the measurements of $\mathbf{x}$ are affected by some additive bounded noise $\mathbf{q}^x = [q_1 \quad q_2]^T$ satisfying $|q_1| \leq q_1^m$, $|q_2| \leq q_2^m$, the error norm converges to

$$\lim_{t\to\infty} \left\|\mathbf{z}(t)\right\| \leq \left| \frac{1}{\lambda} \left(\Lambda(\rho^x + \rho^u + \|\mathbf{q}^x\|) + B^q + \varepsilon\right) \right|, \tag{3.16}$$

where $B^q = |c_{12} + \frac{bk_2+1}{c_{12}}(c_{11} + \Gamma)|q_1^m + |k_2 b| q_2^m$.

3.2.2 Proof of Theorem 3.1

If the nonlinearity $d(\mathbf{x}, u)$ is perfectly estimated by $\hat{d}(\mathbf{x}, u)$, the control law (3.8) transforms (3.3) into the form of an autonomous system with a single trivial equilibrium point in $\mathbf{z} = 0$, as shown in (3.17). Choosing K and Γ such that (3.10) is satisfied forces $A - BK$ to be a symmetric negative definite matrix. Hence, (3.17) is a linear, autonomous system with strictly negative eigenvalues, and is therefore globally exponentially stable (GES). Consequently, finding a Lyapunov function $V = c_1 \left\|\mathbf{z}\right\|^2$ satisfying the requirements in (A.4) is trivial.

$$\dot{\mathbf{z}} = (A - BK)\mathbf{z} \tag{3.17}$$

Convergence

When the nonlinearity is unknown, (3.9) can be seen as (3.17) perturbed by the bounded term $v(t)$. By Lemma A.1, the perturbed system (3.9) converges globally and exponentially to some error ball centered at the origin. Therefore, the NN controller in (3.8) is guaranteed to drive the system towards the set-point at an exponential rate.

Let λ be the largest eigenvalue of $(A - BK)$ (the smallest in magnitude). As $(A - BK)$ is real symmetric negative definite, λ is real and strictly negative. In addition, $\mathbf{x}^T(A - BK)\mathbf{x} \leq \lambda \mathbf{x}^T\mathbf{x}$ [48]. To further investigate what $\mathbf{z}$ converges to, the Lyapunov function $V = \frac{1}{2}\left\|\mathbf{z}\right\|^2 = \frac{1}{2}\mathbf{z}^T\mathbf{z}$ is used. Taking the derivative of V yields

$$\begin{aligned}\dot{V} &= \mathbf{z}^T(A-BK)\mathbf{z} + \mathbf{z}^T \begin{bmatrix}0\\1\end{bmatrix} [d(\mathbf{x}(t),u(t)) - \hat{d}(\mathbf{x}(t),u(t))] \\ &\leq \lambda \mathbf{z}^T\mathbf{z} + \|\mathbf{z}\| \left\|v(t)\right\| \\ &\leq \lambda \|\mathbf{z}\|^2 + \|\mathbf{z}\| \varepsilon \\ \dot{V} &\leq 2\lambda V + \sqrt{2V}\varepsilon.\end{aligned}$$

Define $W = \sqrt{V} = \frac{1}{\sqrt{2}} \|\mathbf{z}\|$, so that

$$\dot{W} = \frac{\dot{V}}{2\sqrt{V}} \leq \lambda W + \frac{\varepsilon}{\sqrt{2}}. \tag{3.18}$$

With initial condition $\mathbf{z}(t_0) = \mathbf{z}_0$, the solution to the differential inequality (3.18) is

$$W(t) \leq \frac{1}{\sqrt{2}}(\|\mathbf{z}_0\| + \frac{\varepsilon}{\lambda})e^{\lambda(t-t_0)} - \frac{\varepsilon}{\sqrt{2}\lambda}.$$

As $\frac{\mathrm{d}}{\mathrm{d}t}\|\mathbf{z}\| = \sqrt{2}\dot{W}$, it can be concluded by the comparison lemma that

$$\left\|\mathbf{z}(t)\right\| \leq \|\mathbf{z}_0\| e^{\lambda(t-t_0)} + \frac{\varepsilon}{\lambda}(e^{\lambda(t-t_0)} - 1). \tag{3.19}$$

This shows that $\mathbf{z}(t)$ will converge to the ball centered at the origin with radius $\left|\varepsilon/\ \lambda\right|$ at an exponential rate. Hence, the tightness of the estimate $\hat{d}$ and the poles of $A - BK$ bound the final error.

Input-to-state and $\mathcal{L}_p$-stability

If (3.19) holds, it must also hold that

$$\left\|\mathbf{z}(t)\right\| \leq \|\mathbf{z}_0\| e^{\lambda(t-t_0)} - \frac{\varepsilon}{\lambda}.$$

By Definition A.2 and Definition A.4, $\gamma(\varepsilon) = -\varepsilon/\lambda$ belongs to class $\mathcal{K}$, and $\beta(\|\mathbf{z}_0\|, t-t_0) = \|\mathbf{z}_0\| e^{\lambda(t-t_0)}$ is a class $\mathcal{KL}$ function. As $\varepsilon = \sup_{t_0 \leq \tau \leq t} \left\|v(\tau)\right\|$, system (3.9) is input-to-state stable from $v(t)$ to $\mathbf{z}(t)$ by Definition A.5.

Further, as $\mathbf{z}$ globally and exponentially converges to an error ball of radius r, it is clear that $\left\|\mathbf{z}(t)\right\| \leq \max(r, \|\mathbf{z}_0\|)$ for all $t \in \{t_0, \infty\}$. As mentioned before, the unforced system is GES, and it is trivial to find a Lyapunov function satisfying the requirements (A.4). Using this, together with the assumption that $\|v\| \leq \varepsilon$, proves that (3.9) is finite gain $\mathcal{L}_p$ stable from $v(t)$ to $\mathbf{z}(t)$ by Theorem A.1.

Noise rejection

It is possible to evaluate how robust the controller (3.8) is to noise. Assume the measurement of $\mathbf{x}$ is affected by additive noise $\mathbf{q}^x = \begin{bmatrix} q_1 & q_2 \end{bmatrix}^T$, where $|q_1| \leq q_1^m$ and $|q_2| \leq q_2^m$. This implies that $\mathbf{z}$ is affected by the noise $\mathbf{q}^z = \begin{bmatrix} q_1 & 1/c_{12}(c_{11}+\Gamma)q_1 + q_2 \end{bmatrix}^T$. Hence, Γ amplifies the noise affecting z_2. The noisy controller input when the noise has been added to the computation of $K\mathbf{z}$ and $\dot{x}_2^r$ is

$$u(\mathbf{x}+\mathbf{q}^x) = -K\mathbf{z} - \frac{1}{b}(\hat{d}(\mathbf{x}+\mathbf{q}^x, u) - \dot{x}_2^r) - \frac{c_{12}}{b}q_1 - (k_2 + \frac{1}{b})(\frac{1}{c_{12}}(c_{11}+\Gamma)q_1 + q_2) + \frac{1}{b}q_2.$$

Using the Lyapunov function from earlier, and letting Λ denote the Lipschitz constant of $\hat{d}$, it follows that

$$\begin{aligned}
\dot{V} &\leq \mathbf{z}^T(A-BK)\mathbf{z} + \mathbf{z}^T \begin{bmatrix} 0 \\ 1 \end{bmatrix} [d(\mathbf{x}, u) - \hat{d}(\mathbf{x} + q^x, u)] \\
&\quad + \|\mathbf{z}\| \, (|c_{12} + \frac{bk_2+1}{c_{12}}(c_{11}+\Gamma)|q_1^m + |k_2 b| q_2^m) \\
&\leq \lambda \mathbf{z}^T\mathbf{z} + \|\mathbf{z}\| \, (\left\| d(\mathbf{x}, u) - \hat{d}(\mathbf{x}, u) \right\| + \left\| \hat{d}(\mathbf{x}, u) - \hat{d}(\mathbf{x} + \mathbf{q}^x, u) \right\|) \\
&\quad + \|\mathbf{z}\| \, |c_{12} + \frac{bk_2+1}{c_{12}}(c_{11}+\Gamma)|q_1^m + |k_2 b| q_2^m) \\
&\leq \lambda \mathbf{z}^T\mathbf{z} + \|\mathbf{z}\| \, (\Lambda \, \|\mathbf{q}^x\| + |c_{12} + \frac{bk_2+1}{c_{12}}(c_{11}+\Gamma)|q_1^m + |k_2 b| q_2^m + \varepsilon).
\end{aligned}$$

By the same reasoning as before, $\mathbf{z}$ converges at an exponential rate to the circle with radius $\left| (\Lambda \, \|\mathbf{q}^x\| + |c_{12} + \frac{bk_2+1}{c_{12}}(c_{11}+\Gamma)|q_1^m + |k_2 b| q_2^m + \varepsilon)/\lambda \right|$. From this it can be concluded that the size of Λ, as well as the choice of K, impacts the effect the noise has on the final error.

3.2.3 Proof of Theorem 3.2

As before, let $\lambda < 0$ be the largest eigenvalue of $(A - BK)$, and Λ the Lipschitz constant of the NN. The perturbation term in (3.14) satisfies

$$\begin{aligned}
\left\| w(t) \right\| &= \left\| d(\mathbf{x}(t), u(t)) - \hat{d}(\mathbf{x}(t-\Delta t), u(t-\Delta t)) \right\| \\
&\leq \left\| d(\mathbf{x}(t), u(t)) - \hat{d}(\mathbf{x}(t), u(t)) \right\| + \left\| \hat{d}(\mathbf{x}(t), u(t)) - \hat{d}(\mathbf{x}(t-\Delta t), u(t-\Delta t)) \right\| \\
&\leq \epsilon + \Lambda \left\| \mathbf{x}(t) - \mathbf{x}(t-\Delta t) \right\| + \Lambda \left\| u(t) - u(t-\Delta t) \right\| \\
\left\| w(t) \right\| &\leq \epsilon + \Lambda(\rho^x + \rho^u).
\end{aligned}$$

Again the Lyapunov function $V = \frac{1}{2}\|\mathbf{z}\|^2 = \frac{1}{2}\mathbf{z}^T\mathbf{z}$ is employed.

$$\begin{aligned}\dot{V} &= \mathbf{z}^T(A - BK)\mathbf{z} + \mathbf{z}^T \begin{bmatrix}0\\1\end{bmatrix} w(t) \\ &\leq \lambda \mathbf{z}^T\mathbf{z} + \|\mathbf{z}\| \left(\Lambda(\rho^x + \rho^u) + \varepsilon\right) \\ \dot{V} &\leq 2\lambda V + \sqrt{2V}(\Lambda(\rho^x + \rho^u) + \varepsilon)\end{aligned}$$

By using Lemma A.2 in the same way as in the previous section, it is concluded that

$$\left\|\mathbf{z}(t)\right\| \leq \|\mathbf{z}_0\| \, e^{\lambda(t-t_0)} + \frac{(\Lambda(\rho^x + \rho^u) + \varepsilon)}{\lambda}(e^{\lambda(t-t_0)} - 1).$$

Hence, the system still converges to an error ball centered at the origin at an exponential rate, but the radius of the error ball now depends on the upper bounds on the change in state and input and the Lipschitz constant of $\hat{d}$.

Input-to-state and $\mathcal{L}_p$-stability

The stability proofs are identical to those proving Theorem 3.1 when $v(t)$ is exchanged for $w(t)$, and ε for $\epsilon = \Lambda(\rho^x + \rho^u) + \varepsilon$.

Noise rejection

The proof for noise rejection is also analogous to the one for Theorem 3.1, and will not be repeated.

3.3 Time-delayed system with convolutional neural network controller

Consider the time-delayed system (3.20), where h_i are continuous mappings $h : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$. The time delays τ_i are unknown, but it is assumed that the largest time delay τ_{n_τ} is relatively small. For this system, a controller that can utilize short term historical data to make x_1 follow x_1^r is needed.

In spite of recurrent architectures being the default for many data-scientists when meeting sequence modeling problems, CNNs are employed for this task, as they have been demonstrated to work just as well [49]. Their architecture is also less complicated and easier to analyze.

As in Section 3.2.1, the diffeomorphism $z = [x_1 - x_1^r \quad x_2 - x_2^r]$ is applied. The suggested controller is shown in (3.21), where $\hat{g}$ is made by a CNN with three input channels which produces one output at each time-step. The number of

time-steps n_T to look back, and the interval between each sample $\Delta T = T_j - T_{j-1}$ must be decided by the control designer.

$$\begin{aligned}\dot{x}_1 &= c_{11}x_1 + c_{12}x_2 \\ \dot{x}_2 &= bu + d(\mathbf{x}, u) + \sum_{i=1}^{n_\tau} h_i(x(\tau_i), u(\tau_i)) \\ &= bu + g(x(t), x(t-\tau_1)\ldots, x(t-\tau_{n_\tau}), u(t), u(t-\tau_1), \ldots, u(t-\tau_{n_\tau}))\end{aligned} \tag{3.20}$$

$$\begin{aligned}u(t) = -K\mathbf{z}(t) - \frac{1}{b}[&\hat{g}(\mathbf{x}(t-T_1)\ldots, \mathbf{x}(t-T_{n_T}), u(t-T_1), \ldots, u(t-T_{n_T})) \\ &- \dot{x}_2^r(t)]\end{aligned} \tag{3.21}$$

If it is assumed that the estimation error is bounded, such that $|g - \hat{g}| < \epsilon$, the same proof as in Section 3.2.2 can be used to show that the system converges to ϵ/λ. In Section 4.5 and Section 4.6, the performance of this controller is empirically investigated, and it is explored whether it expresses similar behavior to the FFN controller designed in Section 3.2.

3.4 Mass-spring-damper datasets

The mass-spring-damper (MSD) system in (3.22) is used to investigate the behavior of the previously suggested controllers. The MSD is illustrated in Figure 3.1. Here, x_1 is the position of the mass, and x_2 is its velocity. The input non-linearity is given by $d(\mathbf{x}, u)$, while F_s and F_d are known forces generated by the spring and the damper, respectively. The mass rolls on wheels that experience no friction. The spring force is given by Hooke's law, and the damping is assumed to be proportional to the velocity, as shown in (3.23). The spring and damper coefficients are such that when the input nonlinearity is not present, the system is under-damped. All coefficients of the MSD are listed in Table 3.1.

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{b}{m}u - \frac{1}{m}F_d(x_2) - \frac{1}{m}F_s(x_1) + d(\mathbf{x}, u)\end{aligned} \tag{3.22}$$

$$F_s = c_s x_1, \quad F_d = c_d x_2 \tag{3.23}$$

For this system, the NN controller (3.8) derived in Section 3.2 turns into (3.24), where $\mathbf{z} = \begin{bmatrix} x_1 - x_1^r & x_2 - x_2^r \end{bmatrix}$ and $x_2^r(t) = \dot{x}_1^r - \Gamma z_1$.

$$u = -K\mathbf{z} - \frac{m}{b}[\hat{d}(\mathbf{x}(t), u(t)) - \frac{1}{m}F_d(x_2) - \frac{1}{m}F_s(x_1) - \dot{x}_2^r(t)] \tag{3.24}$$

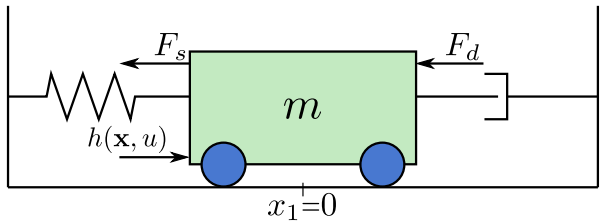


Figure 3.1: The mass spring damper system that is used as a test system for the suggested NN controllers.

Table 3.1: The coefficients of the MSD from (3.22).

Coefficient	b	m	c_s	c_d
Value	1	1	1	0.72

Two datasets were created by simulating the MSD in MATLAB 2019b with Simulink [5]. Each set consists of 500 simulations of the PID controller in (3.25) attempting to make the mass follow a sine curve with amplitude and period sampled from a uniform distribution, as in (3.26). Each simulation runs for 20 s and is sampled at 10 Hz. Hence, the sets consists of 100 000 samples, which are split 60/15/25 into training, validation and test set.

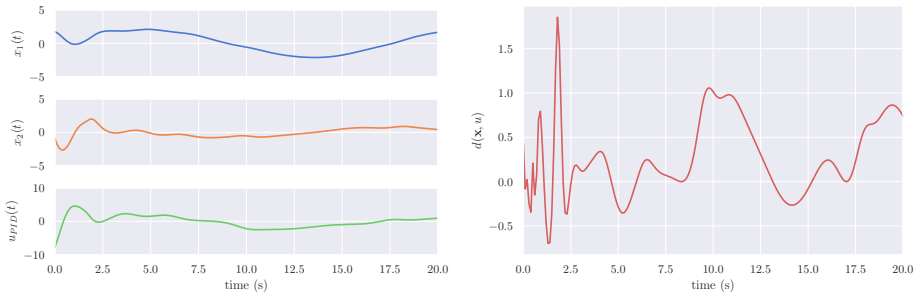
Set 1 is affected by the input nonlinearity given in (3.27), and Set 2 by the one in (3.28). One simulation from each dataset is plotted in Figure 3.2.

$$u_{\text{PID}} = 5(x_1(t) - x_1^r(t)) + \int_0^t (x_1(\tau) - x_1^r(\tau))\mathrm{d}\tau + x_2(t) \tag{3.25}$$

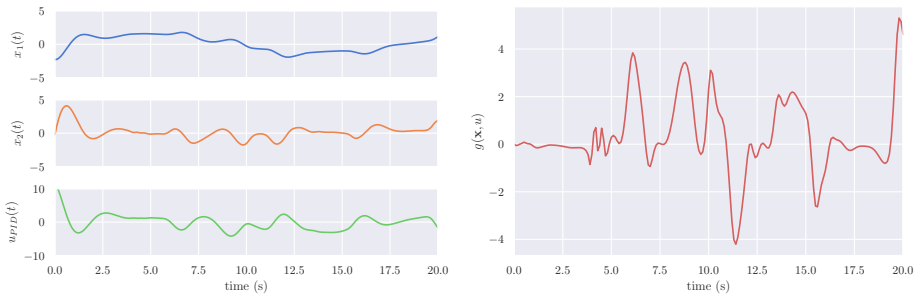
$$x_1^r(t) = A\sin(\frac{2\pi}{T}t), \quad A = 5w_1,\ T = 19w_2 + 1, \quad w_1,\ w_2 \sim \mathcal{U}(0, 1) \tag{3.26}$$

$$d(\mathbf{x}, u) = \sin(u)(\sin(x_1) + \sin(x_2)) \tag{3.27}$$

$$d(\mathbf{x}, u) = (\sin(x_1) + \sin(x_2))\sum_{i=0}^{39} \frac{1}{i+1}\sin(u(t - 0.1i)) \tag{3.28}$$



(a) One 20 s simulation from Set 1.



(b) One 20 s simulation from Set 2.

Figure 3.2: One simulation from each of the datasets.

3.5 Training neural networks

Here, the networks that were trained for testing the suggested NN controllers (3.8) and (3.21) are presented. It should be noted that little time was spent tuning hyperparameters and testing different network architectures and optimizers, as it was not seen as important in regards to the goal of this thesis. Hence, all FFNs and all CNNs respectively have the same architecture.

The neural networks were implemented and trained using the Keras library for Python 3, which provides easy-to-use functionality for working with neural networks in Python [2], [3]. A summary of the FFN architecture is given in Table 3.2, and the CNN architecture is summarized in Table 3.3.

All networks were trained with a Huber loss function with $\delta = 1$ and the Adam optimizer with a learning rate of 1×10^{-3}. Each network was trained for 1600 epochs, with a batch size of 32. After these epochs, both training and validation loss had both stabilized for all the networks. After training, predictions were made on the test sets, and the largest prediction error was stored to use for estimating convergence bounds later.

Three FFNs and three CNNs were trained. These are named in Table 3.4 and Table 3.5, respectively. SpectNorm 10 and SpectNorm 1 were trained with spectral normalization with target Lipschitz constant 10 and 1, respectively. As seen in Table 3.4, the LipSDP program from (2.33) reveals that the Lipschitz constant of the network may vary from the target value of the spectral normalization. The CNNs called L2 1×10^{-3} and L2 1×10^{-2} were trained with L2 regularization with $\alpha_{L2} = 1 \times 10^{-3}$ and $\alpha_{L2} = 1 \times 10^{-2}$, respectively. L2-regularization was applied in order to decrease the size of the network weights, and thereby also the Lipschitz constant of the CNNs.

The CNNs receive samples that are spaced evenly in time with $\Delta T = 0.1\,\mathrm{s}$ between each sample of each input. The receptive field size of the CNNs is $s^r = 49$.

Table 3.2: The FFN architecture that is used for all FFNs in this thesis. Each FFN has 801 trainable parameters.

Layer type	Input shape	Output shape	Parameters	Activation
Fully connected	$(N_{MB}, 3)$	$(N_{MB}, 32)$	128	tanh
Fully connected	$(N_{MB}, 32)$	$(N_{MB}, 16)$	528	tanh
Fully connected	$(N_{MB}, 16)$	$(N_{MB}, 8)$	136	tanh
Fully connected	$(N_{MB}, 8)$	$(N_{MB}, 1)$	9	Identity

Table 3.3: The CNN architecture that is used for all CNNs in this thesis. The CNNs receive inputs where there are $\Delta T = 0.1\,\mathrm{s}$ between each sample, and have a receptive field size of $s^r = 49$.

Layer type	Input shape	Output shape	Filter length	Parameters	Activation
Temporal convolution	$(N_{MB}, 29, 3)$	$(N_{MB}, 29, 8)$	25	608	tanh
Temporal convolution	$(N_{MB}, 29, 8)$	$(N_{MB}, 29, 4)$	25	164	tanh
Flatten	$(N_{MB}, 16)$	$(N_{MB}, 116)$		0	Identity
Fully connected	$(N_{MB}, 116)$	$(N_{MB}, 16)$		1872	tanh
Fully connected	$(N_{MB}, 16)$	$(N_{MB}, 1)$		17	Identity

Table 3.4: The three FFNs used in the controllers in Experiment 1 and 2.

Controller	LipSDP bound	Test loss	Max error on test set
FFN	69.311	0.017	2.463
SpectNorm 10	5.960	0.023	1.958
SpectNorm 1	0.726	0.103	2.262

Table 3.5: The three CNNs used in the controllers in Experiment 3 and 4.

Controller	LipSDP bound	Test loss	Max error on test set
CNN	658.516	0.070	4.293
L2 1×10^{-3}	55.218	0.107	5.149
L2 1×10^{-2}	10.041	0.318	4.060

4 Experiments and Results

Four experiments are conducted to test the controllers suggested in Section 3.2 and Section 3.3. Their performance is compared with a feedback controller that does not compensate for the unknown dynamics.

4.1 Experiment setup

In Experiment 1 and 2, three NN controllers are tested on the MSD from Section 3.4 with the undelayed input nonlinearity (3.27). The controllers use the networks described in Table 3.4 to estimate the unknown dynamics. In Experiment 3 and 4, controllers with the three CNNs from Table 3.5 are tested on the MSD with the delayed input nonlinearity (3.28). From here on the controllers will be referred to by the name of the network they use as an estimator.

The CNN and FFN controllers are given in (4.1) and (4.2), respectively. The controller parameters were chosen to be $\Gamma = 2$ and $K = \begin{bmatrix} -1 & 2.5 \end{bmatrix}$, such that the eigenvalues of $A - BK$ are -3.281 and -1.219. Other parameter values were considered, but an increase in λ leads to an increase in noise amplification, and a decrease in λ results in a slow controller. The chosen parameter values seem to balance these behaviors well.

$$u(t) = -K\mathbf{z}(t) - \frac{m}{b}[\hat{d}(\mathbf{x}(t-\Delta t), u(t-\Delta t)) - \frac{1}{m}F_s(x_1) - \frac{1}{m}F_d(x_2) - \dot{x}_2^r(t)] \quad (4.1)$$

$$\begin{aligned} u(t) = -K\mathbf{z}(t) - \frac{m}{b}[& -\frac{1}{m}F_s(x_1) - \frac{1}{m}F_d(x_2) - \dot{x}_2^r(t) \\ & + \hat{g}(\mathbf{x}(t), \mathbf{x}(t-\Delta T), \ldots, \mathbf{x}(t-(s^r-1)\Delta T), \\ & \quad u(t), u(t-\Delta T), \ldots, u(t-(s^r-1)\Delta T))] \end{aligned} \quad (4.2)$$

The controllers are exposed to 500 trajectory tracking tasks three times with three different levels of measuring noise. In each trajectory tracking task the controller attempts to follow a random sine wave reference, generated in the same way as the trajectories for the datasets, explained in Section 3.4. The measurement noise on both x_1 and x_2 is sampled from a truncated normal distribution with zero mean and standard deviation σ, truncated at $\pm\sigma$.

The empirical averages of $|z_1|$ and $\|\mathbf{z}\|$ are used as performance indicators. The averages are calculated over the entire simulation time, and averaged over the 500 tasks with the same measurement noise levels.

Retrospective bounds were calculated using (3.12) for Experiments 1 and 3, and (3.16) in Experiment 2 and 4. The maximum error of the NN estimators on the test set, listed in Table 3.4 and Table 3.5, was used as an approximation of ε. The limits on the change in states and input, ρ^x and ρ^u, were estimated by finding the maximum $L2$-distance between consecutive samples of $\mathbf{x}$ and u while running the experiments. From the truncated noise distribution it is known that $q_1^m = q_2^m = \sigma$.

The Lipschitz constants of the CNNs are estimated by the LipSDP program described in Section 3.1. The LipSDP values end up between the naive lower and upper bounds described in Section 2.6.1, as expected.

4.2 Feedback controller

For comparison, a controller that does not have an estimator of the unknown dynamics is tested on all the 500 scenarios. This controller is equivalent to the controllers in (4.1) and (4.2) with estimators that always predicts 0, and has Lipshitz constant $\Lambda = 0$. The performance of the feedback controller is displayed in Table 4.1, where $\mathbb{E}\{.\}$ denotes the empirical average.

Table 4.1: Feedback controller: Expected mean error for the controllers from (4.1) and (4.2) when the estimator always outputs 0. The controller was tested on 500 trajectory tracking problems three times with varying levels of measuring noise.

	Noise limit					
	$\sigma = 0$		$\sigma = 0.1$		$\sigma = 0.25$	
Input nonlinearity	$\mathbb{E}\{\|z_1\|\}$	$\mathbb{E}\{\|\mathbf{z}\|\}$	$\mathbb{E}\{\|z_1\|\}$	$\mathbb{E}\{\|\mathbf{z}\|\}$	$\mathbb{E}\{\|z_1\|\}$	$\mathbb{E}\{\|\mathbf{z}\|\}$
Undelayed	0.376	0.893	0.380	0.902	0.438	1.053
Delayed	0.231	0.513	0.236	0.237	0.255	0.579

4.3 Experiment 1: Continuously updated feed forward network controllers

In this experiment, the scenario where $\Delta t_{\max} \simeq 0$ was emulated by forcing Simulink to perform the FFN forward pass every 0.01 s. Mean error and estimated convergence bounds are presented in Table 4.2. The performance of the controllers in the case without measuring noise is visualized in Figure 4.1. When noise is not present, all three controllers outperform the pure feedback controller.

Table 3.4 shows that the performance of the controllers is largely unaffected by the size of the Lipschitz constant of the FFN, even though the convergence bounds

clearly are. SpectNorm 1 performs slightly worse than FFN and SpectNorm 10 when there is no measuring noise, which is not surprising, as it had the highest loss on the test set after training. When measuring noise is added, however, the performance evens out.

Table 4.2: Experiment 1: Expected mean error and retrospective convergence bounds for the controllers in Table 3.4 when the forward pass is calculated every $\Delta t_{\max} = 0.01\,\mathrm{s}$. Each controller was tested on 500 trajectory tracking problems three times with varying levels of measuring noise.

	Noise limit								
Controller	$\sigma = 0$			$\sigma = 0.1$			$\sigma = 0.25$		
	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound
FFN	**0.102**	0.179	2.020	**0.135**	0.264	12.467	0.299	0.682	54.266
SpectNorm 10	0.105	0.187	1.606	0.136	0.266	3.154	**0.299**	0.681	9.343
SpectNorm 1	0.136	0.269	1.855	0.151	0.308	2.801	0.301	0.686	6.583

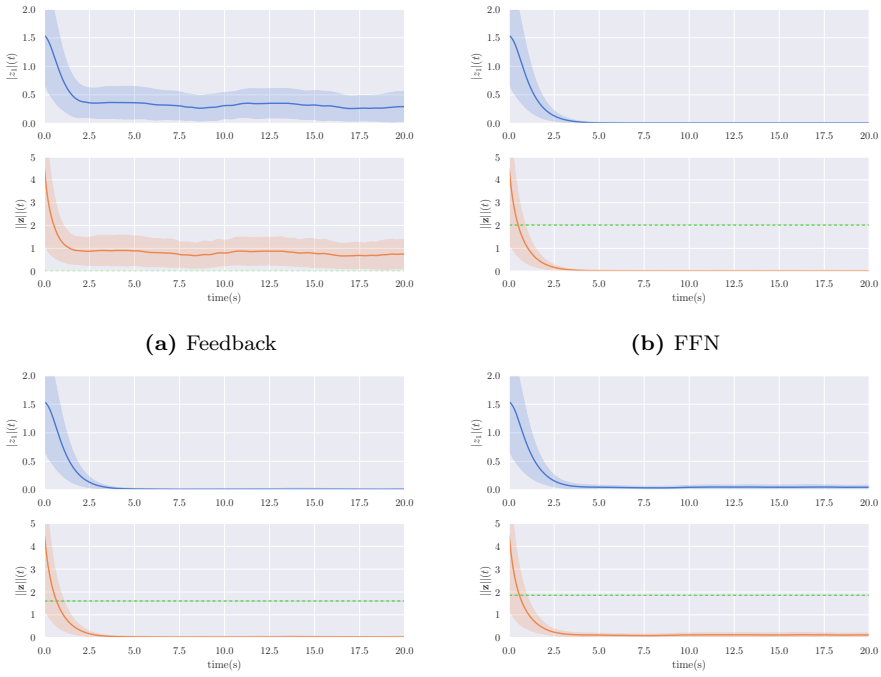
4.4 Experiment 2: Discretely updated feed forward network controllers

When the forward pass is only calculated every $\Delta t_{\max} = 1\,\mathrm{s}$, expected mean error and estimated convergence bounds are as in Table 4.3. Table 4.3 reveals that, as before, the bounds are tighter when the Lipschitz constant is low, but that low bounds have no great effect on performance. The performance of the controllers in the case without measuring noise is visualized in Figure 4.2.

Figure 4.3 displays the data from Table 4.2 and Table 4.3 together with the performance of the pure feedback controller from Table 4.1. From this plot it is clear that all the controllers outperform the feedback controller. Even though SpectNorm 1 has the highest expected tracking error when measuring noise is not present, it is barely affected by the transition to discrete updates. SpectNorm 1 also experiences a smaller drop in performance when measuring noise is added, than SpectNorm10 and FFN do.

Table 4.3: Experiment 2: Expected mean error and retrospective convergence bounds for the controllers in Table 3.4 when the forward pass is calculated every $\Delta t_{\max} = 1\,\mathrm{s}$. Each controller was tested on 500 trajectory tracking problems three times with varying levels of measuring noise.

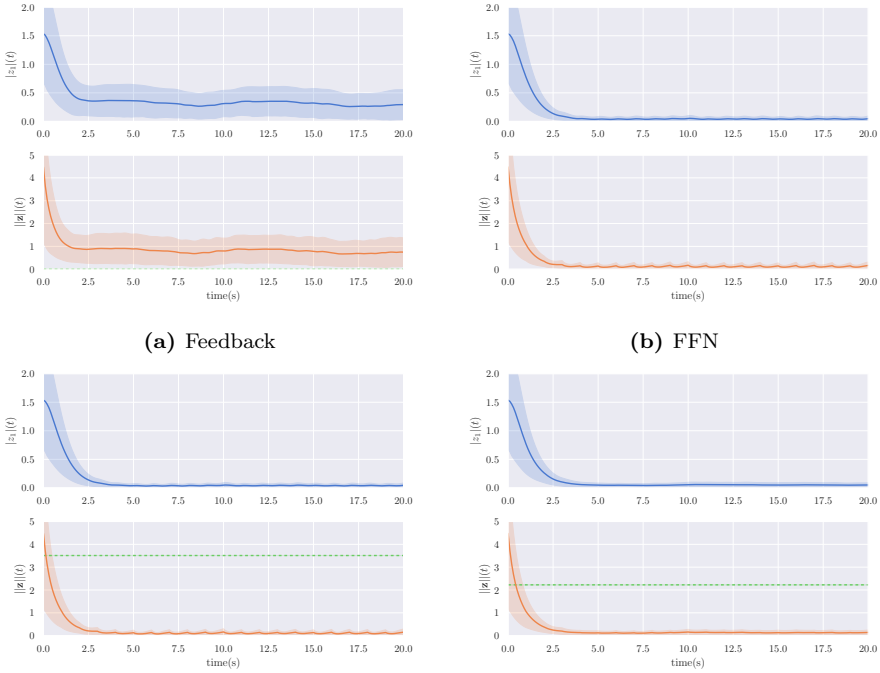
	Noise limit								
Controller	$\sigma = 0$			$\sigma = 0.1$			$\sigma = 0.25$		
	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound
FFN	0.131	0.265	24.744	0.152	0.319	82.745	0.307	0.708	272.699
SpectNorm 10	**0.129**	0.259	3.509	**0.149**	0.311	8.157	0.306	0.703	24.366
SpectNorm 1	0.139	0.279	2.226	0.153	0.315	3.322	**0.300**	0.687	7.970



(a) Feedback **(b)** FFN

(c) SpectNorm 10 **(d)** SpectNorm 1

Figure 4.1: Experiment 1: The three controllers described in Table 3.4 were tested on 500 trajectory tracking problems. This depicts the expected performance of the controllers when $\Delta t_{\max} = 0.01\,\mathrm{s}$. The solid lines mark the empirical mean, and the shaded area around the lines represent $\pm$ one standard deviation. A plot showing the error of the feedback controller is included for comparison. The measuring noise is zero in these plots.



(a) Feedback

(b) FFN

(c) SpectNorm 10

(d) SpectNorm 1

Figure 4.2: Experiment 2: The three controllers described in Table 3.4 were tested on 500 trajectory tracking problems. This depicts the expected performance of the controllers when $\Delta t_{\max} = 1\,\mathrm{s}$. The solid lines mark the empirical mean, and the shaded area around the lines represent $\pm$ one standard deviation. A plot showing the error of the feedback controller is included for comparison. The measuring noise is zero in these plots.

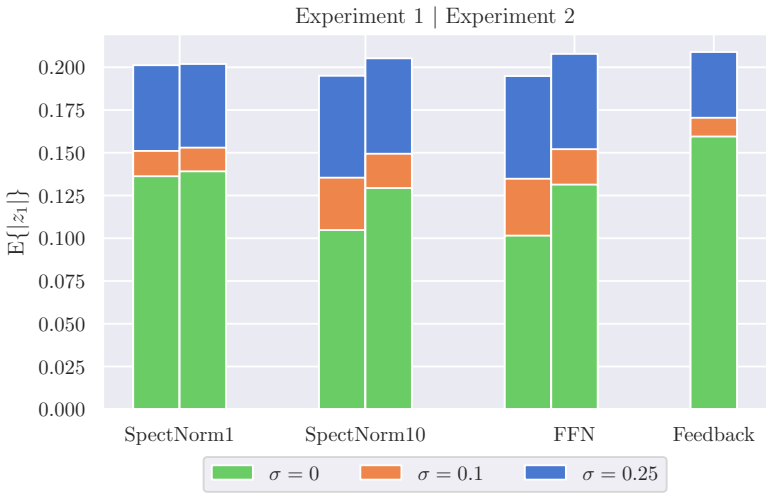


Figure 4.3: A visualization of the data from Tables 4.1 to 4.3. The left part of each bar presents the results from Experiment 1, while the right presents results from Experiment 2. SpectNorm 1 is less affected by the switch from continuous and discrete operation than the other two controllers, and is also relatively less affected by measuring noise.

4.5 Experiment 3: Continuously updated convolutional neural network controllers

In this experiment, the CNN forward pass is calculated every $\Delta t_{\max} = 0.1\,\mathrm{s}$. As the CNN is designed to take a vector of length $s^r = 49$ as input, where all elements are 0.1 s apart in time, this is the practical equivalent of continuous operation for this controller.

The performance of the three CNN controllers from Table 3.5 is summarized in Table 4.4. All the controllers outperform the pure feedback controller, which is clearly seen in Figure 4.5. L2 1×10^{-2} is the best performing controller out of the three, even though it had the highest loss on the test set. Figure 4.4 shows the operation of the controllers when the noise level is 0. Here, it can be seen that not only does L2 1×10^{-2} have a lower expected error, but the standard deviation of the error is also much smaller, than it is for Feedback, CNN and L2 1×10^{-3}.

4.6 Experiment 4: Discretely updated convolutional neural network controllers

When the CNN controllers are updated discretely, the expected mean error and estimated bounds are as in Table 4.5. Figure 4.5 reveals that neither of the CNN controllers handles the transition to discrete operation very well, and when noise

is added to the measurements, the performance is worse than the pure feedback controller.

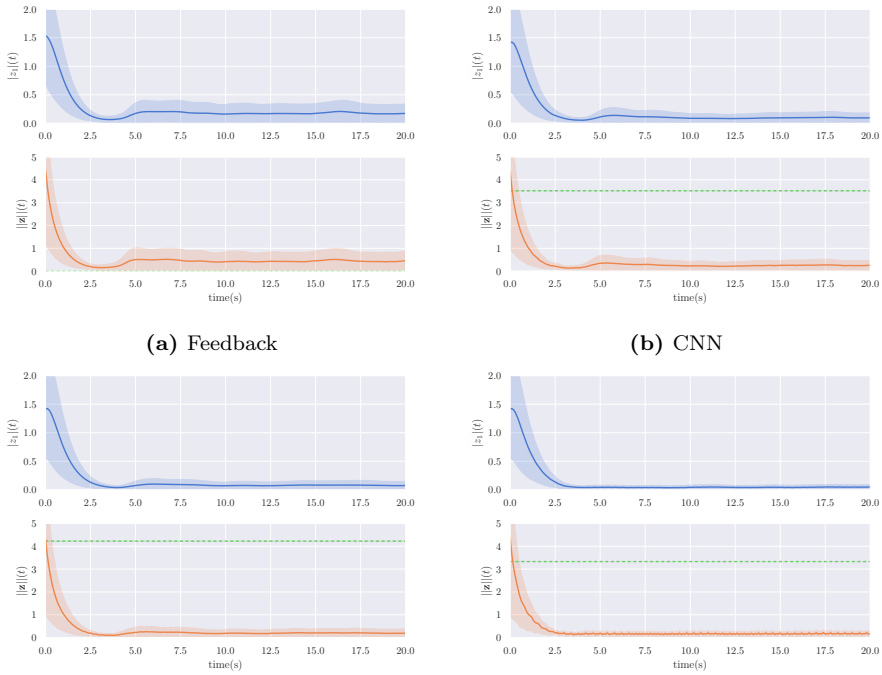
The relationship seen in Experiments 1 and 2 between the Lipschitz constant and the corresponding robustness to noise and discrete operation cannot be observed for the CNN controllers.

Table 4.4: Experiment 3: Expected mean error and retrospective convergence bounds for the controllers in Table 3.5 when the forward pass is calculated every $\Delta t_{\max} = 0.1\,\mathrm{s}$. Each controller was tested on 500 trajectory tracking problems three times with varying levels of measuring noise.

	Noise limit								
Controller	$\sigma = 0$			$\sigma = 0.1$			$\sigma = 0.25$		
	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound
CNN	0.174	0.374	3.521	0.178	0.390	80.766	0.306	0.714	389.74
L2 1×10^{-3}	0.157	0.327	4.223	0.160	0.342	11.490	0.304	0.709	40.554
L2 1×10^{-2}	**0.128**	0.304	3.330	**0.146**	0.334	5.356	**0.302**	0.708	13.460

Table 4.5: Experiment 4: Expected mean error and retrospective convergence bounds for the controllers in Table 3.5 when the forward pass is calculated every $\Delta t_{\max} = 1\,\mathrm{s}$. Each controller was tested on 500 trajectory tracking problems three times with varying levels of measuring noise.

	Noise limit								
Controller	$\sigma = 0$			$\sigma = 0.1$			$\sigma = 0.2$		
	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound	$\mathbb{E}\{\lvert z_1\rvert\}$	$\mathbb{E}\{\lVert\mathbf{z}\rVert\}$	Bound
CNN	0.206	0.475	1211.320	0.239	0.557	1765.396	**0.354**	0.844	3349.609
L2 1×10^{-3}	0.213	0.498	83.526	0.290	0.700	262.22	0.430	1.057	399.664
L2 1×10^{-2}	**0.192**	0.450	12.317	**0.222**	0.525	33.467	0.378	0.914	59.470



(a) Feedback **(b)** CNN

(c) L2 1×10^{-3} **(d)** L2 1×10^{-2}

Figure 4.4: Experiment 3: The three controllers described in Table 3.5 were tested on 500 trajectory tracking problems. This depicts the expected performance of the controllers when $\Delta t_{\max} = 0.1\,\mathrm{s}$. The solid lines mark the empirical mean, and the shaded area around the lines represent $\pm$ one standard deviation. A plot showing the error of the feedback controller is included for comparison. The measuring noise is zero in these plots.

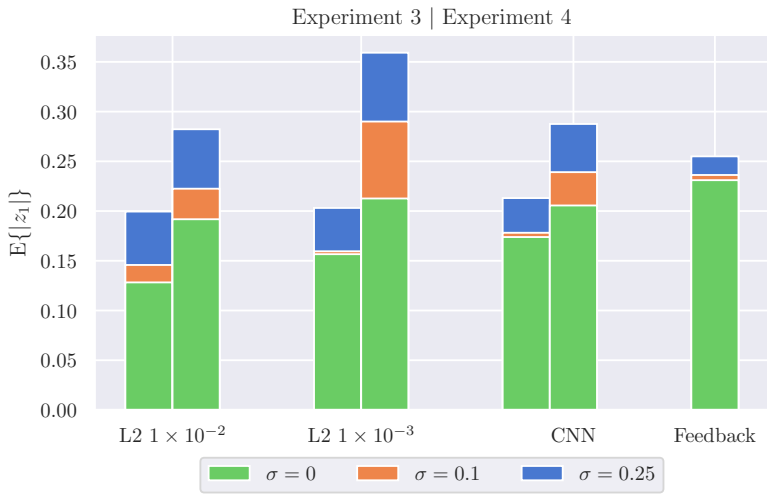


Figure 4.5: A visualization of the data from Tables 4.1, 4.4 and 4.5. The left part of each bar presents the results from Experiment 3, while the right presents results from Experiment 4.

5 Discussion and Further Work

Here, the results of experiments 1-4 as well as the validity of Theorem 3.1 and Theorem 3.2 are discussed. Suggestions for how to proceed and extend this work are then laid forth.

5.1 Discussion

The results of experiments 1 and 2 confirm that the convergence bounds derived in Section 3.2 apply in practice. The controllers outperform the pure feedback controller that does not compensate for the unknown dynamics, even when the NN estimate is rarely updated, and the state measurements are affected by noise.

The experiments reveal that the error bounds are quite loose and that a low bound does not necessarily imply better performance overall. Nonetheless, the experiments indicate that a lower Lipschitz constant reduces the relative effect of rarely updating the neural network estimate, as well as the controller's sensitivity to measuring noise.

Although the performance of the CNN controllers from experiments 3 and 4 do not seem to be correlated with the Lipschitz constant in the same way as the FFN controllers, they consistently outperform the pure feedback controller when updated every 0.1 s. The error stays well within the retrospective bounds in these experiments as well. Nonetheless, the bounds are too loose for this to be a convincing indication of whether such bounds generally hold for this kind of controller.

The original plan was to implement spectral normalization for CNNs as well, but there was no time for this. Therefore, L2-regularization was used as a means to train CNNs with differently sized Lipschitz constants. Both spectral normalization and L2-regularization are methods that reduce the variance of neural networks, possibly at the cost of accuracy, as demonstrated by the test loss of the NNs recited in Table 3.4 and Table 3.5. However, it is clear from experiments 1 and 2 that this bias-increase can be beneficial for tackling noise and discrete updates. In experiments 3 and 4, the controller with the most heavily regularized model outperformed the two others, even though it had the highest test loss.

One of the cornerstones of the results in this thesis is the boundedness of the estimation error, which was assumed in both Theorem 3.1 and Theorem 3.2. In

the experiments, it was assumed that ε could be approximated by the maximum prediction error on the test set. This assumption is reasonable, as the samples in the test set seem to be representative of the data the estimator will meet during operation. However, if the test set is small, or gathered from operation in a limited area of the state space, this approximation might be optimistic.

If the unknown dynamics are known to be locally Lipschitz in the area of operation, with Lipschitz constant L, an argument can be made for having higher trust in the estimation error bound of a NN with a small Lipschitz constant Λ. Let the maximum prediction error on the test set be $\hat{\varepsilon}$, observed at some test sample $\hat{x}$. When the estimator makes a prediction on a new, unseen observation $\tilde{x}$, the estimation error satisfies the following:

$$\begin{aligned}\left\|d(\tilde{x}) - \hat{d}(\tilde{x})\right\| &= \left\|d(\tilde{x}) - d(\hat{x}) + d(\hat{x}) + \hat{d}(\hat{x}) - \hat{d}(\hat{x}) - \hat{d}(\tilde{x})\right\| \\ &\leq \left\|d(\tilde{x}) - d(\hat{x})\right\| + \left\|d(\hat{x}) - \hat{d}(\hat{x})\right\| + \left\|\hat{d}(\hat{x}) - \hat{d}(\tilde{x})\right\| \\ &\leq (L + \Lambda)\left\|\tilde{x} - \hat{x}\right\| + \hat{\varepsilon}.\end{aligned}$$

However, decreasing Λ makes the NN less flexible, as the variance is reduced, and the bias increased, which might contribute to an increase of ε. Hence, the control designer must consider when the NN grows too rigid to learn the dynamics accurately.

Another challenge is the assumptions made in Theorem 3.2 that the convergence bounds are dependent on the maximum rate of change of the state and input. It is relatively common to assume that the input change is bounded, but for the state change to be upper bounded, it must be required that the dynamics are bounded as well. This does not hold globally when parts of the dynamics are, for instance, linear or polynomial. If applying this controller, one must be certain that either, the update rate of the NN estimate is faster than the system dynamics, or the dynamics are bounded.

5.2 Further work

A significant advantage of using this controller for trajectory tracking is that $\dot{x}_2$ can be measured at a high sampling rate with an accelerometer. From these measurements, it is possible to get ground truth values of the unkown dynamics, by utilizing $d(\mathbf{x}, u) = bu - \dot{x}_2$. This real life data would probably be affected by noise, as well as uncertainity in the knowledge of b. Seeing what effect noisy training data has on the controller would be interesting. Also, it is reasonable to believe that restricting the Lipschitz constant of the estimators will be beneficial in this situation, as argued by Shi et al. [1].

The class of systems that the suggested controller can be applied to can easily be extended to include systems with more than two states. As long as the unknown dynamics affect the derivatives of states that also have a known, linear input

term in their derivatives, the method from Section 3.2 can be directly extended to include these systems.

It is possible that the convergence bound can be tightened by, for instance, using a different Lyapunov function, or changing the controller design slightly. The controller parameters k_2 and Γ were chosen by trial and error, and it would be a good idea to develop a scheme that chooses them more intentionally. These parameters decide the size of λ, which sets the rate of convergence. Also, the convergence bound is directly dependent on the size of both k_2 and Γ when the measurements are noisy. It should be possible to find values that balance convergence rate and noise rejection intelligently.

The LipSDP program that was extended to include CNNs worked as expected, where the predicted Lipschitz constant lies between the naive lower and upper bound. However, as the originally suggested program has $\mathcal{O}(N^2)$ decision variables, the simpler version where T is diagonal had to be applied, as discussed in Section 2.6.3. The matrix in (3.2), which is the weight matrix of a CNN layer when the CNN is written as a FFN, consists of lower triangular blocks of size $s^r \times s^r$. Also, each diagonal of these lower triangular blocks is just one repeated number. It might be possible to exploit this special structure to make the LipSDP program more efficient, and perhaps more accurate.

6 Conclusion

This work attempts to further the sound unification of data-driven methods and traditional control theory.

A feedback linearizing controller with a neural network estimating unknown dynamics is suggested. The controller is proven to stabilize the closed-loop error dynamics in a general two-dimensional trajectory tracking problem. Moreover, the controller globally and exponentially drives the error to a ball centered at the origin. A specific upper bound on the radius of the error ball is found. Measuring noise and the update rate of the neural network estimate influence the size of the convergence bound.

Experiments on a simulation of a mass-spring-damper system affected by an unknown input nonlinearity confirm the theoretical findings. The convergence bounds are dependent on the Lipschitz constant of the neural network estimator. Experiments show that the Lipschitz constant influences the noise rejection capabilities of the controller, as well as its ability to handle a low update rate of the neural network estimate.

A method for approximating the Lipschitz constant of feedforward neural networks is extended to convolutional neural networks, and a controller with a convolutional neural network estimator is tested on a system with time-delayed input nonlinearity. Experiments with this controller indicate that the Lipschitz constant does not have the same influence as in the undelayed case. However, all the suggested controllers consistently outperform a feedback controller, which does not compensate for unknown dynamics.

References

[1] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *International Conference on Robotics and Automation, ICRA 2019, (Montreal, QC, Canada, May 20-24, 2019)*, IEEE, 2019, pp. 9784–9790, ISBN: 978-1-5386-6027-0.

[2] "Python 3.6", `https://www.python.org/`.

[3] F. Chollet *et al.*, "Keras 2.3.1", `https://keras.io`, 2015.

[4] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. J. Pappas, "Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks", 2019. arXiv: `1906.04893 [cs.LG]`.

[5] "Matlab version 9.7.0 (r2019b)", Natick, Massachusetts, USA, 2019.

[6] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1", `http://cvxr.com/cvx`, Mar. 2014.

[7] National Instruments. (2019). Pid theory explained, [Online]. Available: `http://www.ni.com/en-no/innovations/white-papers/06/pid-theory-explained.html` (visited on 11/11/2019).

[8] P. Kadlec, B. Gabrys, and S. Strandt, "Data-driven soft sensors in the process industry," *Computers & Chemical Engineering*, vol. 33, no. 4, pp. 795–814, 2009. DOI: `https://doi.org/10.1016/j.compchemeng.2008.12.012`.

[9] J. Lee, H.-A. Kao, and S. Yang, "Service innovation and smart analytics for industry 4.0 and big data environment," *Procedia CIRP*, vol. 16, pp. 3–8, 2014, ISSN: 2212-8271. DOI: `https://doi.org/10.1016/j.procir.2014.02.001`.

[10] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars", Apr. 2016. arXiv: `1604.07316 [cs.CV]`.

[11] L. Deng and Y. Liu, "Deep Learning in Natural Language Processing", 1st ed. Springer Singapore, 2018, ISBN: 9789811338489.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information*

Processing Systems 2012, NIPS 2012, (Lake Tahoe, Nevada, United States, December 3-6, 2012), 2012, pp. 1097–1105.

[13] T. P. Lillicrap, J. J. Hunt, A. e. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", Sep. 2015. arXiv: 1509.02971 [cs.LG].

[14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks", Dec. 2013. arXiv: 1312.6199 [cs.CV].

[15] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, (Las Vegas, NV, USA, June 27-30, 2016)*, IEEE Computer Society, 2016, pp. 2574–2582, ISBN: 978-1-4673-8851-1.

[16] K. Hornik, ""approximation capabilities of multilayer feedforward networks"," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[17] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Comput. Surv.*, vol. 51, no. 5, 93:1–93:42, Aug. 2018.

[18] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for Verifying Deep Neural Networks", Mar. 2019. arXiv: 1903.06758 [cs.LG].

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, (San Diego, CA, USA, May 7-9, 2015)*, 2015.

[20] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, (Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018)*, PMLR, 2018, pp. 5286–5295.

[21] D. Gunning and D. W. Aha, "Darpa's explainable artificial intelligence program," *AI Magazine*, vol. 40, no. 2, pp. 44–58, 2019. [Online]. Available: https://search.proquest.com/docview/2258093718?accountid=12870.

[22] S. Rahman, A. Rasheed, and O. San, "A hybrid analytics paradigm combining physics-based modeling and data-driven modeling to accelerate incompressible flow solvers," *Fluids*, vol. 3, no. 3, p. 50, 2018.

[23] D. Limon, J. Calliess, and J. Maciejowski, "Learning-based nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7769–7776, 2017. DOI: https://doi.org/10.1016/j.ifacol.2017.08.1050.

[24] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," in *57th IEEE Conference on Decision and Control, CDC 2018, (Miami, FL, USA, December 17-19, 2018)*, IEEE, 2018, pp. 6059–6066, ISBN: 978-1-5386-1395-5.

[25] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural*

Information Processing Systems 2017, (Long Beach, CA, USA, December 4-9 2017), 2017, pp. 908–918.

[26] S. M. Richards, F. Berkenkamp, and A. Krause, "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Proceedings of Machine Learning Research*, ser. Proceedings of Machine Learning Research, vol. 87, PMLR, 2018, pp. 466–476.

[27] B. Chang, M. Chen, E. Haber, and E. H. Chi, "AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks", Feb. 2019. arXiv: 1902.09689 [stat.ML].

[28] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, (Montréal, Canada, December 3-8 2018)*, 2018, pp. 6571–6583.

[29] M. Ciccone, M. Gallieri, J. Masci, C. Osendorfer, and F. Gomez, "Naisnet: Stable deep networks from non-autonomous differential equations," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, (Montréal, Canada, December 3-8 2018)*, 2018, pp. 3025–3035.

[30] H. Qian and M. N. Wegman, "L2-Nonexpansive Neural Networks", Feb. 2018. arXiv: 1802.07896 [cs.AI].

[31] K. Scaman and A. Virmaux, "Lipschitz regularity of deep neural networks: Analysis and efficient estimation," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, (Montréal, Canada, December 3-8 2018)*, 2018, pp. 3839–3848.

[32] R. Balan, M. Singh, and D. Zou, "Lipschitz Properties for Deep Convolutional Networks", Jan. 2017. arXiv: 1701.05217 [cs.LG].

[33] D. Zou, R. Balan, and M. Singh, "On Lipschitz Bounds of General Convolutional Neural Networks", Aug. 2018. arXiv: 1808.01415 [cs.IT].

[34] J. G. Proakis and D. K. Manolakis, "Digital Signal Processing", 4th ed. Edinburgh Gate: Pearson, 2014, vol. 4, pp. 43–150, ISBN: 9781292025735.

[35] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[36] H. Wolkowicz, R. Saigal, and L. Vandenberghe, "Handbook of semidefinite programming: theory, algorithms, and applications". Springer Science & Business Media, 2012, vol. 27, ISBN: 9781461543817.

[37] T. M. Mitchell, "Machine Learning". McGraw-Hill Education, 1997, vol. 76, ISBN: 9780070428072.

[38] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction", 2nd ed. Springer-Verlag, 2009, vol. 12, ISBN: 9780387848570.

[39] T. H. Gareth James Daniela Witten and R. Tibshirani, "An Introduction to Statistical Learning with Applications in R", 1st ed. Springer Science & Business Media, 2013, ISBN: 9781461471370.

[40] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

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

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[43] S. Ruder, "An overview of gradient descent optimization algorithms", Sep. 2016. arXiv: 1609.04747 [cs.LG].

[44] 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)*, 2015.

[45] H. K. Khalil, "Nonlinear Systems", 3rd ed. Edinburgh Gate: Pearson, 2015, vol. 4, pp. 87–90, ISBN: 9781784490133.

[46] P. L. Combettes and J.-C. Pesquet, "Lipschitz Certificates for Neural Network Structures Driven by Averaged Activation Operators", Mar. 2019. arXiv: 1903.01014 [math.OC].

[47] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *6th International Conference on Learning Representations, ICLR 2018, (Vancouver, BC, Canada, April 30 - May 3, 2018)*, OpenReview.net, 2018.

[48] D. S. Bernstein, "Scalar, Vector, and Matrix Mathematics: Theory, Facts, and Formulas", 2nd ed. Princeton University Press, 2009, p. 467, ISBN: 9780691140391.

[49] S. Bai, J. Zico Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *arXiv e-prints*, arXiv:1803.01271, arXiv:1803.01271, Mar. 2018. arXiv: 1803.01271 [cs.LG].

A Definitions, lemmas and theorems from Khalil

A.1 Feedback linearization

Definition A.1 (Definition 13.1 from Khalil [45])
A nonlinear system

$$\dot{x} = f(x) + G(x)u$$

where $f : \mathcal{D} \to \mathbb{R}^n$ and $G : \mathcal{D} \to \mathbb{R}^{n \times p}$ are sufficiently smooth on a domain $\mathcal{D} \subset \mathbb{R}^n$, is said to be feedback linearizable if there exists a diffeomorphism $T : \mathcal{D} \to \mathbb{R}^n$ such that $\mathcal{D}_z = T(\mathcal{D})$ contains the origin and the change of variables $z = T(x)$ transforms the system into the form

$$\dot{z} = Az + B\gamma(x)[u - \alpha(x)] \tag{A.1}$$

with (A, B) controllable and $\gamma(x)$ nonsingular for all $x \in \mathcal{D}$.

By sufficiently smooth in Definition A.1 we mean that the mappings must be continuous, and that partial derivatives are as smooth as might be required by later applications.

A.2 Perturbed systems

Let (A.3) be the perturbed version of the nominal system (A.2), where $g(t, \mathbf{x})$ is some bounded perturbation term.

$$\dot{\mathbf{x}} = f(t, \mathbf{x}) \tag{A.2}$$

$$\dot{\mathbf{x}} = f(t, \mathbf{x}) + g(t, \mathbf{x}) \tag{A.3}$$

Lemma A.1 (Lemma 9.2 from Khalil [45])
Let $x = 0$ be an exponentially stable equilibrium point of the nominal system (A.2). Let $V(t, x)$ be a Lyapunov function of the nominal system that satisfies

$$\begin{aligned} c_1 \|\mathbf{x}\|^2 \leq V(t, x) \leq c_2 \|\mathbf{x}\|^2 \\ \frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}} f(t, \mathbf{x}, 0) \leq -c_3 \|\mathbf{x}\|^2 \\ \left\| \frac{\partial V}{\partial \mathbf{x}} \right\| \leq c_4 \|\mathbf{x}\| \end{aligned} \tag{A.4}$$

in $[0,\infty)\times\mathcal{D}$*, where* $\mathcal{D}=\{x\in\mathbb{R}^n|\ \|\mathbf{x}\|<r\}$*. Suppose the perturbation term* $g(t,x)$ *satisfies*

$$\left\|g(t,\mathbf{x})\right\|\leq\delta<\frac{c_3}{c_4}\sqrt{\frac{c_1}{c_2}}\theta_r$$

for all $t\geq 0$*, all* $\mathbf{x}\in\mathcal{D}$*, and some positive constant* $\theta<1$*. Then, for all* $\left\|\mathbf{x}(t_0)\right\|<r\sqrt{c_1/c_2}$*, the solution* $\mathbf{x}(t)$ *of the perturbed system* (A.3) *satisfies*

$$\left\|\mathbf{x}(t)\right\|\leq k\exp(-\gamma(t-t_0))\left\|\mathbf{x}(t_0)\right\|,\ \forall t_0\leq t<t_0+T$$

and

$$\left\|\mathbf{x}(t)\right\|\leq b,\ \forall t\geq t_0+T$$

for some finite T*, where*

$$k=\sqrt{\frac{c_2}{c_1}},\ \gamma=\frac{(1-\theta)c_3)}{2c_2},\ b=\frac{c_4}{c_3}\sqrt{\frac{c_2}{c_1}}\frac{\delta}{\theta}.$$

A.3 $\mathcal{L}_p$-stability

The space $\mathcal{L}_p^m$ for $1\leq p<\infty$ is defined as the set of all piecewise continuous functions $\mathbf{u}:[0,\infty)\to\mathbb{R}^m$ such that

$$\|\mathbf{u}\|_{\mathcal{L}_p}=\left(\int_0^\infty\|\mathbf{u}(t)\|^p\mathrm{d}t\right)^{\frac{1}{p}}<\infty.$$

We define the extended space $\mathcal{L}_{p,e}^m$ as $\mathcal{L}_{p,e}^m=\{u|u_\tau\in\mathcal{L}_p^m,\forall\,\tau\in[0,\infty)\}$, where u_τ is a truncation of u defined by

$$u_\tau(t)=\begin{cases}u(t), & 0\leq t\leq\tau\\ 0, & t>\tau.\end{cases}$$

Definition A.2 (Definition 4.2 from Khalil [45])
A continuous function $\alpha:[0,\infty)\to[0,\infty)$ *is said to belong to class* $\mathcal{K}$ *if it is strictly increasing and* $\alpha(0)=0$*.*

Definition A.3 (Definition 5.2 from Khalil [45])
A mapping $H:\mathcal{L}_{p,e}^m\to\mathcal{L}_{p,e}^q$ *is* $\mathcal{L}_p$ *stable if there exists a class* $\mathcal{K}$ *function* α*, defined on* $[0,\infty)$*, and a nonnegative constant* β *such that*

$$\left\|(Hu)_\tau\right\|_{\mathcal{L}_p}\leq\alpha(\|u_\tau\|)_{\mathcal{L}_p}+\beta$$

for all $u\in\mathcal{L}_{p,e}^m$ *and* $\tau\in[0,\infty)$*. It is finite-gain* $\mathcal{L}_p$ *stable if there exist nonnegative constants* γ *and* β *such that*

$$\left\|(Hu)_\tau\right\|_{\mathcal{L}_p} \leq \gamma \left\|u_\tau\right\|_{\mathcal{L}_p} + \beta$$

for all $u \in \mathcal{L}_{p,e}^m$ *and* $\tau \in [0, \infty)$.

Theorem A.1 (Theorem 5.1 from Khalil, paraphrased [45])
Consider the system (A.5) *and take* r *and* r_u *such that* $\|\mathbf{x}\| \leq r$ *in* $\mathcal{D}$ *and* $\|\mathbf{u}\| \leq r_u$ *in* $\mathcal{D}_u$. *Suppose that* $\mathbf{x} = 0$ *is an exponentially stable equilibrium point of the unforced system* $\dot{\mathbf{x}} = f(t, \mathbf{x}, 0)$ *and that a Lyapunov function satisfying the requirements of* (A.4) *exists.*

If the origin is globally exponentially stable and in addition $\mathcal{D} = \mathbb{R}^n$ *and* $\mathcal{D}_u = \mathbb{R}^m$, *then, for each* $\mathbf{x}_0 \in \mathbb{R}^n$, *the system* (A.5) *is finite-gain* $\mathcal{L}_p$ *stable for each* $p \in [1, \infty]$.

$$\begin{aligned} \dot{\mathbf{x}} &= f(t, \mathbf{x}, \mathbf{u}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{y} &= h(t, \mathbf{x}, \mathbf{u}), \quad \mathbf{x} \in \mathbb{R}^n,\ \mathbf{u} \in \mathbb{R}^m \end{aligned} \tag{A.5}$$

A.4 Input-to-state stability

Definition A.4 (Definition 4.2 from Khalil [45])
A continuous function $\beta : [0, \infty) \times [0, \infty) \to [0, \infty)$ *is said to belong to class* $\mathcal{KL}$ *if, for each fixed* s, *the mapping* $\beta(r, s)$ *belongs to class* $\mathcal{K}$ *with respect to* r *and, for each fixed* r, *the mapping* $\beta(r, s)$ *is decreasing with respect to* s *and* $\beta(r, s) \to 0$ *as* $s \to \infty$.

Definition A.5 (Definition 4.7 from Khalil [45])
The system (A.5) *is said to be input-to-state stable if there exist a class* $\mathcal{KL}$ *function* β *and a class* $\mathcal{K}$ *function* γ *such that for any initial state* $\mathbf{x}(t_0)$ *and any bounded input* $u(t)$, *the solution* $\mathbf{x}(t)$ *exists for all* $t \geq t_0$ *and satisfies*

$$\left\|\mathbf{x}(t)\right\| \leq \beta(\left\|\mathbf{x}(t_0)\right\|, t - t_0) + \gamma\left(\sup_{t_0 \leq \tau \leq t} \left\|\mathbf{u}(\tau)\right\|\right).$$

A.5 Comparison principle

Lemma A.2 (Lemma 3.4 from Khalil [45])
Consider the scalar differential equation

$$\dot{u} = f(t, u),\ u(t_0) = u_0$$

where $f(t, u)$ *is continuous in* t *and locally Lipschitz in* u, *for all* $t \geq 0$ *and all* $u \in J \subset \mathbb{R}$. *Let* $[t_0, T)$ *(*T *could be infinity) be the maximal interval of existence of the solution* $u(t)$, *and suppose* $u(t) \in J$ *for all* $t \in [t_0, T)$. *Let* $v(t)$ *be a continuous function that satisfies the differential inequality*

$$\dot{v}(t) \leq f(t, v(t)), \quad v_0 \leq u_0$$

with $v(t) \in J$ *for all* $t \in [t_0, T)$. *Then,* $v(t) \leq u(t)$ *for all* $t \in [t_0, T)$.

NTNU
Norwegian University of
Science and Technology