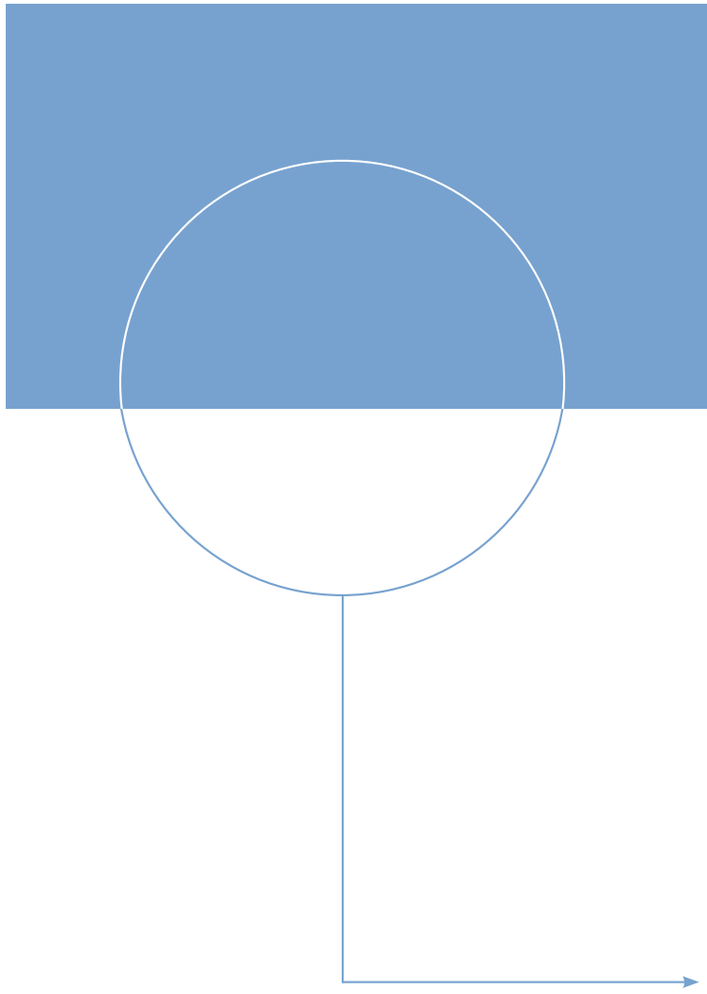

Master's thesis

2021

Edmond Peci

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

Edmond Peci

Robustness and Stability of Long Short-Term Memory Recurrent Neural Networks

June 2021

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Robustness and Stability of Long Short-Term Memory Recurrent Neural Networks

Edmond Peci

Cybernetics and Robotics
Submission date: June 2021
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Esten Ingar Grøtli, Signe Moe, Mark Haring, Katrine Seel

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This master thesis is a result of 20 weeks of work from mid-January to early June, and is carried out at Norwegian University of Science and Technology (NTNU), the Department of Engineering Cybernetics. The master thesis is partly an extension of the specialisation project with course code TTK4550, with the aim of looking further into the concept of persistency of excitation as a way to robustifying deep learning models. A few parts of the preliminaries in Chapter 2 are re-used from this specialisation project, and will be clearly marked. The specialisation and master thesis topic was proposed by SINTEF Digital in association with the machine learning project "Towards autonomy in process industries: Combining data-driven and model-based methods for estimation and control (TAPI)" [1]. Project partners are NTNU, SINTEF, Hydro, Elkem, Yara and Borregaard.

Studying cybernetics and robotics, I found courses on nonlinear control systems particularly challenging and interesting. In the later years, machine learning, and especially deep learning, has caught my interest. This thesis made it possible to combine two of the fields I have great admiration for.

The reader is assumed to have broad knowledge in control theory, including Lyapunov analysis. Advanced concepts will be briefly presented in the preliminaries. Some knowledge of basic concepts in machine learning is advantageous, but deep learning will be introduced from the basics.

There are two main contributions in this thesis. Firstly, the persistency of excitation principle, based on the work of [2], is analysed and extended to long short-term memory recurrent neural networks. Two training procedures inspired by persistency of excitation are proposed. The long short-term memory neural network is altered to account for the exogenous signals injected and the two training procedures are evaluated with regard to robustness to perturbations in input. Secondly, constraints on the parameters of this persistently exciting-inspired long short-term memory neural network will be presented for ensuring that it is input-to-state stable. This work is inspired by [3], in which a regular LSTM is analysed. The programming language Python [4] and the deep learning library PyTorch [5] are used to implement the neural networks, the training procedures and input-to-state stability constraints. All neural networks are trained on CPUs, constraining the deepness of the neural networks.

All of my supervisors, Prof. Jan Tommy Gravdahl at the Department of Engineering Cybernetics, NTNU, and Esten Ingar Grøtli, Signe Moe, Mark Haring and Katrine Seel at SINTEF Digital, have all offered great help in reading through parts of this thesis and suggesting improvements.

Trondheim, 2021-07-06

Edmond Peci

Acknowledgment

This master thesis was done in cooperation with SINTEF Digital. I want to thank all my supervisors, Esten Ingar Grøtli, Mark Haring, Signe Moe and Katrine Seel from SINTEF Digital and Prof. Jan Tommy Gravdahl at the Department of Engineering Cybernetics, NTNU, for their continuous availability, frequent meetings and clarifications. This has been of great help in keeping continuity in the work throughout the past 20 weeks.

I also want to thank my family for their patience, understanding and motivation throughout this thesis.

Summary

This master thesis aims at getting a deeper understanding of the robustness and stability of long short-term memory recurrent neural networks in view of perturbations in input and perturbation in initial conditions. Recurrent neural networks may approximate dynamic systems due to their ability to maintain a memory of the past. It is important to consider, apart from robustness to perturbation in input, stability when recurrent neural networks are used to model dynamic systems that exhibits stability properties. In a regression setting, they have a wide spectrum of application areas, many of which are considered safety-critical. The introduction of adversarial examples complicates the matter. Adversarial examples are inputs that are specifically designed to produce large errors in the output of a neural network. It is as such of high importance to study the ambiguous terrain of robustness and stability of recurrent neural networks in order to shorten the gap between theoretical models in academia and actual real-world application.

Input perturbation robustness is studied from the perspective of persistency of excitation, an important condition in system identification. Persistency of excitation is relevant for systems that possess signals (i.e. inputs) that interact with some unknown system parameters. It is important that the input signal is rich enough in order to produce good parameter estimates. It has recently been interest in studying if this principle is applicable to deep learning as means to produce robust parameter estimates. The common technique of attempting to robustify neural networks, norm penalty regularisation, is shown to be equivalent to a robust optimisation objective. Robust optimisation seeks to deal with optimisation problems in which uncertain data is present, constrained by some uncertainty set. The norm penalty and robust optimisation equivalence pave the way for a large class of robust linear optimisation problems. The long short-term memory recurrent neural network is altered to account for the persistency of excitation principle. Two training procedure, attempting to persistently excite neural network parameters by injecting exogenous signals, are adapted and discussed for the persistently exciting-inspired long short-term memory recurrent neural network.

Stability of the altered long short-term memory neural network will be studied from the perspective of input-to-state stability. The neural network is represented in state-space form. Discrete input-to-state stability concepts are used to propose sufficient constraints on the parameters of the altered long short-term memory neural network for establishing the input-to-state stability property.

The robustness and stability of the altered long short-term memory neural network is evaluated on a cascaded tanks system identification problem. The experiments show that injecting exogenous signals into a long short-term memory recurrent neural network lead to noticeably better performance compared to regular norm regularisation techniques. The enforcement of the sufficient stability conditions results in some deterioration in performance, which may evidently be mitigated by increasing network capacity.

Sammendrag

Denne masteroppgaven har som mål å studere robusthet og stabilitet i lys av den populære rekurrente nevrale nettverkstypen 'lang kortidsminne' (long short-term memory) i møte med perturbert inngangsdata and pertubasjoner av nettverkets initial tilstander. Rekurrente nevrale nettverk kan approksimere dynamiske systemer som følger av modellens mulighet til å vedlikeholde et minne av fortiden. Det er viktig å studere, bortsett fra robusthet mot perturbasjoner i input, stabilitet når et rekurrent nevralt nettverk benyttes til å modellere dynamiske systemer som innehar stabilitetsegenskaper. Rekurrente nevrale nettverk i regressjonsoppgaver har mange applikasjonsmuligheter, mange av dem sikkerhetskritiske. Introduksjonen av 'Motstander eksempler' (adversarial examples) begrenser mange felt som kan dra nytte av disse modellene. 'Motstander eksempler' er inngangsdata som er spesifikt designet for å produsere frem avvik i nettverkets utgang. Det er derfor viktig å studere robusthet og stabilitet av kunstig rekurrente nevrale nettverk for å redusere avstanden mellom teoretiske modeller i akademia og applikasjon i industrien.

Robusthet mot perturbering av inngangsdata blir studert i lys av konseptet vedvarende eksitasjon (persistency of excitation), en viktig betingelse i systemidentifikasjon. Vedvarende eksistasjon er relevant for systemer som innehar signaler (inngangsverdier) som interagerer med parametre i modellen. Det er viktig at signalene bærer nok informasjon om systemet for produsere gode parameter estimat. Vedvarende eksitasjon har nylig blitt foreslått som et verktøy for å robustifisere dype nevrale nettverk. En tradisjonell måte for å forsøke å øke robustheten til kunstige nevrale nettverk, norm regularisering, vises å være ekvivalent et robust optimeringsproblem. Robust optimering er et felt innenfor matematisk optimalisering som søker løsninger på problemer med data som innehar usikkerhet, begrenset av en usikkerhetsmengde. Norm regularisering og robust optimering ekvivalensen baner vei for en rik klasse av robuste lineære optimeringsproblemer. Det rekurrente nevrale nettverket, kort langtidsminne, blir modifisert for å kunne anvende vedvarende eksitasjon-inspirerte treningsprosedyrer. To treningsprosedyrer som begge forsøker å sørge for vedvarende eksitasjon ved å injisere utenforstående signaler blir tilpasset og diskutert for det modifiserte rekurrente nevrale nettverket.

Stabilitet av det modifiserte rekurrente nevrale nettverket, kort langtidsminne, blir studert i lys av stabilitetskonseptet inngang-til-tilstand stabilitet (input-to-state stability). Vi representerer det nevrale nettverket som en ulineær tilstandsrepresentasjon. Diskret inngang-til-tilstand stabilitetskonsepter blir avendt for å foreslå tilstrekkelige betingelser på det nevrale nettverkets parametre for å sørge for inngang-til-tilstand stabilitetsegenskapen.

Det modifiserte rekurrente nevrale nettverkets robusthet og stabilitet blir evaluert på et to-tank system. Eksperimentene viser at det å injisere eksogene signaler inn i et kort langtidsminne rekurrent nevralt nettverk fører til en merkbar forbedring i prediksjonsevne sammenlignet med mer tradisjonelle metoder som norm regularisering. Håndhevelsen av inngang-til-tilstand betingelsene fører til en degradering i nettverkets prediksjonsevne. Prediksjonsevnen kan tilsynelatende forbedres ved å øke nettverkets kapasitet.

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial neural network

CNN Convolutional neural network

DNN Deep neural network

FGSM Fast gradient sign method

FNN Feedforward neural network

ISS Input-to-state stability

LSTM Long short-term memory

MSE Mean square error

NLP Natural language processing

NRO Nonlinear robust optimisation

PE Persistency of excitation

PGD Projected gradient descent

ResNet Residual neural network

RNN Recurrent neural network

SGD Stochastic gradient descent

Chapter 1

Introduction

Artificial Neural Networks (ANNs) are mathematical models inspired by the human brain. Among the first works in the field of neural networks, one finds the neurophysiologist Warren McCulloch and mathematician Walter Pitts research on artificial neurons, mathematical functions that are the very building block of the networks. In the past years, neural networks have been a hot topic in the field machine learning. The successful application of deep neural networks[1] in computer vision [13], speech recognition, text generation, and in the latter years control theoretical problems [14], have induced a resurgence in studying their capabilities and applying the models in an increasingly amount of tasks. This resurgence has been fueled by more publicly available labeled datasets, increased computing power and memory and prominent research in the field of optimisation theory.

1.1 Background

Deep artificial neural networks (DNN) are at its core universal function approximators. Deeper networks with an increasing amount of parameters have been crucial in achieving state of the art performance in several fields, among them computer vision and language processing. With the advancement of research in neural networks and subsequently their potential role in safety critical systems, such as autonomous vehicles, the black-box nature of deep neural networks is highlighted as an important concern with regard to their robustness to input noise. Understanding the output of a neural network based on its input and network architecture, is not a matter of course due to the model complexity and non-determinism during training. The black-box concern with regard to robustness was reinforced when some daunting properties of neural networks were presented in [15], most importantly their sensitivity to input that has been slightly perturbed. The perturbation is done systematically with the goal of maximising a loss criteria with regard to the input. The perturbed input is most often termed an adversarial example and is in a sense small, but worst-case perturbation.

[1] Deep neural networks are neural networks with multiple layers, i.e. stacked neural networks. Deep learning is a machine learning field that utilise deep artificial neural networks in producing machine learning models.

Problem Formulation

Employing artificial neural networks in real-world interacting applications, for instance in autonomous vehicles, require a high degree of confidence that the models we deploy are robust to noise and perturbations in input. The notion of safety-critical systems is expanding as new technology emerge. The non-robust responses of neural networks to input perturbations restrain the vast amount of applicable areas that may benefit from neural networks. Small perturbations of the input, may result in large changes in the network's output. It is not a matter of course to determine what is causing these deviances, which may in turn lead to confusion, and in worst case, dangerous actions from system operators, controllers or other equipment that is acting on the network's output. Since the first mentioning of adversarial examples in view of deep learning, researchers have looked at robustifying commonly used neural network architectures, such as convolutional neural networks (CNNs), feedforward neural networks (FNNs) and recurrent neural networks (RNNs). Several years later, there is no unambiguous solution to the issue. Moreover, employing neural networks as part of safety-critical systems, it is often not sufficient to empirically show that the network is "well-behaved". It is a necessity with formal guarantees that the system is indeed stable for a class of inputs and initial conditions. This thesis will look at robustness and stability in view of long short-term memory (LSTM) recurrent neural networks.

Related work

Among the first mentioning of adversarial examples appear in [15], in which some novel counter-intuitive properties of ANNs are introduced. They show that state-of-the art DNNs such as AlexNet [16], that generalise well on object recognition tasks, fail to correctly classify examples that are subjected to small perturbations in pixel values. The perturbations applied to the test examples are non-random, as they are generated by maximising some loss metric with respect to the input. They study adversarial examples in light of Lipschitz continuity of a FNN. In light of this, they propose that regularisation techniques that penalise the Lipschitz upper bounds of each layer may result in more robust networks. In [17], the effect of adversarial examples on several ANNs working on multivariate time series data is considered. Among these neural networks, one finds RNNs, and more specifically, a special type of these networks called LSTM neural networks. The experiments show that all the evaluated neural networks are vulnerable to adversarial examples. They highlight the consequences with regard to safety-critical systems such as healthcare, prognostics and cybersecurity, and cost-critical domains such as finance and energy sector.

The landscape of remedies against adversarial examples is huge and ambiguous. A robust optimisation-related approach is taken in [2] for robustifying neural networks in a classification setting. In adaptive control and system identification, the condition persistency of excitation is of high importance. The condition ensure that the excitement of the parameters in a model cannot decay too rapidly. Failing to satisfy persistency of excitation,

algorithms, such as stochastic gradient descent methods, may provide non-robust parameter estimates due to the algorithm not obtaining the necessary amount of information. Inspired by this condition, they present a new training procedure for ensuring robustness of neural networks, based on reinterpreting norm parameter regularisation. Exogenous perturbations, or disturbances, are injected into each layer during training as a way to enrichen the input signal, attempting to persistently excite the parameters. They show that a CNN trained with this training procedure is more robust to perturbations in input, particularly adversarial examples. In [12], this training procedure is studied in a regression setting. FNNs trained with persistently exciting parameters are noticeably more robust to adversarial perturbations, and perform equally well when no perturbations are present on a number of multivariate datasets.

In [3], they study input-to-state stability (ISS) in view of a LSTM neural networks. ISS is a stability concept that is frequently applied to nonlinear control systems with external inputs. They show that under some constraints on the RNN parameters, the network is ISS. This result is utilised in [18] to design a LSTM neural network-based controller based on model predictive control.

What Remains to be Done?

The discovery of adversarial examples that may alter a neural network's output drastically sparked a surge in studying robustness of neural networks. Employing non-robust neural networks in cost-critical domains introduce a vulnerability. Adversarial examples, often imperceptible, may result in drastic changes in a neural network's output, affecting all components dependant on it. This is even more critical in safety-critical systems.

A profuse amount of work studying remedies against adversarial examples have been in view of artificial neural networks in a classification setting [19]. In the recent years some work has also emerged in the regression setting for FNNs [12, 20, 21], but to a lesser extent. An informal list of all papers on adversarial examples available at the distribution service arXiv is given in [22], exemplifying this imbalance. The research of robustness in RNNs is even less developed. This is not surprising. Deep learning has had its breakthrough in computer vision, in which they are used for classification purposes. Adversarial examples that are more or less indistinguishable to the original input, are easily crafted by adjusting pixel values in directions that maximise a loss metric. RNNs on the other hand has had a breakthrough in natural language processing. It is substantially more difficult to find adversarial examples that are indistinguishable to the original input due to the data being words and sentences. A perturbed input ought therefore to be a misspelling (for example changing one letter in a word). RNNs are, however, also used in a regression setting for modeling system dynamics due to its appealing recurrent nature [3, 23, 24], and are just as susceptible to perturbations in input as CNNs and FNNs [17].

The two concepts robustness and stability are seemingly tied together. Neural network robustness denotes that a network generalise well to unseen input (which may be slightly

perturbed). Robust optimisation has been frequently used as a tool to produce neural network models that are robust to perturbations in input. Stability, in the sense of dynamic system theory, studies the system's robustness to small perturbations of initial conditions on the output of the dynamic system. It is important to consider stability when a neural network is used to model dynamic systems that exhibits stability properties in the dynamic system theory sense [3]. This ensures that the effect of initial conditions vanish and future state trajectories are bounded given bounded inputs.

This work aims at contributing to robustness and stability in view of a particular class of RNNs, namely the LSTM neural network. Two different approaches for possibly aiding in producing robust neural network will be in focus: robust optimisation and nonlinear stability theory.

1.2 Objectives

The objectives of this thesis are partially an extension of [12][2], mentioned as related work, in which the training procedure for attempting to ensure persistently exciting parameters from [2] is successfully implemented for regression tasks using FNNs. A proposed area for future research in [12] is to investigate the use of the technique for other architectures, such as RNNs. In this thesis, the RNN variant LSTM is studied.

Secondly, the stability of a LSTM neural network for attempting to ensure persistently exciting neural network parameters will be analysed with the stability paradigm input-to-state stability (ISS).

A hybrid approach, combining both robust optimisation (and hereby robustness in light of persistently excitement of the parameters) and stability, will be the main objectives of this thesis. The objectives are rooted in the two questions 1) Can the concept of persistency of excitation robustify LSTM neural networks? 2) Can input-to-state stability be achieved for a LSTM neural network with persistently exciting parameters? The following objectives are proposed as a result of these questions,

1. Extend the training procedure of [2] for long short-term memory recurrent neural networks.

2. Propose alternative training procedures for evoking persistently exciting neural network parameters, primarily using theory from robust nonlinear optimisation.

3. Implement the training procedures on long short-term memory recurrent neural network and empirically evaluate their benefits on prediction tasks with different number of features.

4. Analyse the input-to-state stability of a long short-term memory recurrent neural network for attempting to persistently excite its parameters.

[2] This is the project thesis conducted by the author autumn 2020, as mentioned in the preface.

5. Investigate the effect that the stability constraints have on a long short-term memory recurrent neural network trained with the training procedure from [2], both in terms of robustness and stability.

1.3 Contributions

The thesis firstly studies the concept of persistency of excitation in a RNN setting. The work of [2] is the main inspiration, from which a training procedure for FNNs will be in focus. The LSTM neural network is altered in Section 3.1 in order to be able to apply the training procedure for attempting to persistently excite the neural network parameters. The training procedure from [2] is discussed in view of the LSTM recurrent neural network in Section 3.1.1. An alternative training procedure, rooted in nonlinear robust optimisation theory, will also be studied as part of the persistency of excitation viewpoint in Section 3.1.2.

In the second part of the thesis, the altered LSTM neural network will be studied from a stability perspective. The main inspiration for this part stem from [3], where sufficient conditions are derived for a regular LSTM neural network for ensuring the ISS property. The main contribution with regard to stability is summarised in Theorem 3.2.2, in which neural network parameter constraints are proposed for ensuring that the altered LSTM recurrent neural network is ISS.

1.4 Outline

Chapter 2 lays the foundation for the theoretical preliminaries. The chapter starts with introducing concepts from linear algebra, particularly norms. Secondly, neural networks and deep learning is introduced, covering neural network architectures, the universal approximation theorem and neural network optimisation. The concept of neural network robustness is also treated in detail together with common tools for attempting to robustify them, such as ℓ_2 norm regularisation. The concept of persistency of excitation is also introduced and discussed in relation to neural networks. Lastly, nonlinear stability theory concepts are presented, particularly input-to-state stability

Chapter 3 presents the main contributions (theoretical methodology). The regular LSTM neural network equations are altered to account for the principle of persistency of excitation. Two training procedures acting on the altered equations are presented. Lastly, an input-to-state stability analysis on the altered LSTM equations is presented.

Chapter 4 describes the practical methodology. The dataset generation process is described together with the resulting dataset. Neural network configuration aspects are discussed. Lastly, four experiments are described, rooted in the objectives of the thesis.

Chapter 5 presents results and discusses the findings.

Chapter 6 concludes the thesis and provide recommendations for further work.

Chapter 2

Preliminaries: linear algebra, deep learning, robust optimisation and stability theory

Chapter 2 will provide the theoretical framework necessary for discussing robustness and stability of deep learning algorithms, particularly RNNs. Initially, some important results from linear algebra will be presented in Section 2.1. ANNs and an overview of the most important components in the optimisation of a neural network is given in Section 2.2. The concept of robustness in neural networks will be treated in Section 2.3. Following, persistency of excitation in light of neural networks, with its relevance to robustness, is introduced in Section 2.4. The chapter ends with nonlinear stability theory in Section 2.5.

Please note that parts of this chapter stem from a specialisation project [12] conducted by the author during autumn 2020. If the reader is familiar with the work, these parts may be skipped. The relevant subsections are repeated in this thesis with some minor changes due to their relevance to the work in this master thesis. The subsections that are included from [12], will be clearly marked with a dagger symbol (†) in each subsection title and a double dagger (‡) symbol at the end of each subsection.

2.1 Linear Algebra

This chapter will introduce some results from linear algebra, particularly norm functions. For a more in-depth review of the fundamentals of linear algebra, we refer to [25, Chapter 2], which is the main reference for the material in Section 2.1.

2.1.1 Vector norms

The reference for the vector norm material is [25, Chapter 2]. A vector norm is a function on a vector space that provide a distance measure. They serve the same purpose as the absolute

value in the scalar case. The real coordinate space $\mathbb{R}^n$ together with a norm on $\mathbb{R}^n$ constitute a metric space

Definition

A vector norm is defined in Definition 2.1.1. As we see, three conditions ought to be satisfied: positivity, triangle inequality/subadditivity and homogeneity.

Definition 2.1.1. *A vector norm on $\mathbb{R}^n$ is a function $\|\| : \mathbb{R}^n \mapsto \mathbb{R}$ satisfying the following properties:*

$$\begin{aligned} &\|x\| \geq 0 && \forall x \in \mathbb{R}^n \\ &\|x+y\| \geq \|x\| + \|y\| && \forall x, y \in \mathbb{R}^n \\ &\|kx\| = \|k\|\|x\| && \forall k \in \mathbb{R}, x \in \mathbb{R}^n \end{aligned}$$

Special class of norms: p-norms

Different norm classes are distinguished by subscripts on $\|\cdot\|$. A useful class of vector norms that will be used extensively are the p-norms. The p-norms are defined in eq. (2.1)-(2.2),

$$\|x\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p} \quad \forall p \geq 1 \leq \infty \tag{2.1}$$

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i| \qquad p \to \infty \tag{2.2}$$

Some frequently used p-norms are the 1-, 2- and ∞-norms, given in eq. (2.3), (2.4) and (2.2),

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n| \tag{2.3}$$

$$\|x\|_2 = (|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}} \tag{2.4}$$

Vector norm properties

Hölder inequality is a fundamental inequality concerning p-norms, given in eq. (2.5),

$$\left\|x^T y\right\| \leq \|x\|_p \left\|y\right\|_q \quad \frac{1}{p} + \frac{1}{q} = 1 \tag{2.5}$$

By choosing p=q=1, we get a special case of this inequality called *Cauchy-Schwarz inequality*. This is given in eq. (2.6),

$$\left\|x^T y\right\| \leq \|x\|_2 \left\|y\right\|_2 \tag{2.6}$$

There exists constants k_1, k_2 such that all norms on $\mathbb{R}^n$ are equivalent. Assume $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ are norms on $\mathbb{R}^n$. In such a case, the inequality given in eq. (2.7) holds for all $x \in \mathbb{R}^n$,

$$k_1 \|x\|_\alpha \leq \|x\|_\beta \leq k_2 \|x\|_\alpha \tag{2.7}$$

2.1.2 Matrix norms

Matrix norms are important in analysing algorithms or systems involving matrices. For example in linear system theory, evaluating how sensitive a system is to noise/data error, the matrix norm (more specifically, the spectral norm, which will be defined in the coming section) of the state matrix, provides a measure of the amplification of noise. Similar to vector norms, matrix norms provide a measure of distance on matrix space. We will adapt the notation of [25] and denote vector with lowercase symbols, and matrices with uppercase symbols.

Definition

The definition of a matrix norm ought to be equivalent to the definition of a vector norm. This is due to $\mathbb{R}^{m\times n}$ is isomorphic, i.e. identical in structure, to $\mathbb{R}^{mn}$ [25, Chapter 2.3].

Definition 2.1.2. *A vector norm on $\mathbb{R}^{m\times n}$ is a function $\|\|:\mathbb{R}^{m\times n}\mapsto\mathbb{R}$ satisfying the following properties:*

$$\begin{aligned}
&\|A\|\geq 0 && \forall A\in\mathbb{R}^{m\times n}\\
&\|A+B\|\geq\|A\|+\|B\| && \forall A,B\in\mathbb{R}^{m\times n}\\
&\|kA\|=\|k\|\|A\| && \forall k\in\mathbb{R}, A\in\mathbb{R}^{m\times n}
\end{aligned}$$

Matrix p-norms

Just as with vector norms, we use subscripts to identify different classes of matrix norms. A important class of matrix norms are the p-norms, defined in Definiton 2.1.3

Definition 2.1.3 (p-norm)**.** *The p-norm of a matrix $\boldsymbol{A}\in\mathbb{R}^{m\times n}$ is the p-norm of the largest vector that is produced by applying the matrix $\boldsymbol{A}$ to a unit p-norm vector $x\in\mathbb{R}^n$:*

$$\|A\|_p=\sup_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p}=\max_{\|x\|_p=1}\|Ax\|_p \tag{2.8}$$

The spectral norm is defined by letting p=2 in Definition 2.1.3. An alternative formulation of the spectral norm is given in eq. (2.9).

$$\|A\|_2=\sqrt{\lambda_{max}(A^TA)}=\sigma_1(A) \tag{2.9}$$

where $\sigma_1(A)$ denotes the largest singular value of the matrix A. A drawback of the 2-norm compared to the 1-norm and ∞-norms, is that it is more convoluted to compute. A common method is to apply the singular value decomposition to the matrix. Applying singular value decomposition is computationally heavy, and thus in practice, the largest singular value is most often estimated using the power iteration method. We refer to [25] for a description of the method.

2.1.3 Hadamard product

The Hadamard product [25, Chapter 12] appears frequently in machine learning and denotes a pointwise product between two matrices.

Definition 2.1.4. *Given two matrices with the same dimension $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times n}$, the Hadamard product $(A \odot B) \in \mathbb{R}^{m\times n}$ is a matrix with the same dimensions as its operands, where each element is given by the product given by eq. (2.10),*

$$(A \odot B) = (A)_{ij}(B)_{ij} \tag{2.10}$$

where $(A)_{ij}$ denotes the element at column i and row j for matrix A, and likewise for matrix B.

Note that the Hadamard product is represented by a wide number of symbols: $\odot, \circ, \otimes$. In this thesis, the symbol $\odot$ will be used. Example 2.1.1 shows the Hadamard product between two 2×2 matrices.

Example 2.1.1.

$$A \odot B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} \\ a_{21}\,b_{21} & a_{22}\,b_{22} \end{bmatrix}. \tag{2.11}$$

It is sometimes useful to get rid of the Hadamard product operator when doing vector and matrix operations. Indeed, the relation in eq. (2.12) is true:

$$a \odot b = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \odot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{bmatrix} = \begin{bmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = Ab \tag{2.12}$$

where $A = \text{diag}(a_1, a_2, \dots, a_n) \in \mathbb{R}^{n\times n}$ is a diagonal matrix where $A_{ii} = a_i \forall\, i \in [1, n]$.

2.2 Artificial neural networks and deep learning †

Deep learning is a powerful artificial intelligence tool based on the interconnection of multiple units called artificial neurons, on a layer-to-layer basis, forming an ANN. DNNs express that there are at least more than one layers between the input units and the output unit(s). The networks are inspired by the complex decision making of the human mind. Consider the primary visual cortex of the brain, consisting of hundreds of millions of neurons, with billions of connections between them. Humans are able to understand what their eyes show with little effort. Even handwritten symbols with bad handwriting are (often) interpreted correctly as a result of years of practicing. Deep learning with ANNs exhibits some of these characteristics of practicing. In traditional computer vision, features (corners, textures, etc.) in an image are typically hand-crafted [26]. This is a tedious process. ANNs on the other hand, can be thought of as representational learning. Representational learning, also known as feature learning, learn features in the input data by transforming the data [10, 27]. In deep learning context, this is done by the collection of several (hence the name *deep* neural networks) non-linear transformations with the intention of producing good quality features. The two main references for this section are [6, 10] ‡.

2.2.1 Perceptrons and artificial neurons †

A very simplified mathematical model inspired by a biological neuron was invented in the late 1950s by Frank Rosenblatt under the name perceptron [28]. A perceptron is one form of artificial neurons that takes in one or more weighted inputs and compares the sum of the weighted inputs against some threshold (bias). The output of a perceptron is binary (i.e. 0 if the weighted sum is below threshold and 1 if weighted sum is greater than threshold) [6].

The perceptron is a simplistic decision making algorithm. By varying the weights and bias, different models of decision making are produced. For example, if we have an input x_1 that is particularly important for some reason, it may be weighted heavily (i.e. corresponding weight w_1 is given a high weight). By varying weights, and thresholds, different decision makings are made. Naturally, when the input space is large, manual weight adjustments is not feasible. This is where neural networks and representational learning makes its entrance.

In neural networks, one typically has a number of artificial neurons in each layer. Figure 2.1 provides an example of a neural network with a single hidden layer of 4 units. By making small adjustments to the weights in the network, the overall network can change to produce a desired output, be it for a classification task (categorical values), or regression (continuous values). The binary nature of perceptron is not fortunate in this setting. A small change can flip the output from 0 to 1, or vice versa. A way of handling this issue is to pass the weighted sum to a nonlinear function [6]. Some frequently used nonlinear functions include the sigmoid function, rectified linear unit function (ReLU) and the hyperbolic tangent function (tanh). The inclusion of nonlinear activation functions contribute to the expressive nature of neural networks, for which will be discussed in the forthcoming section ‡.

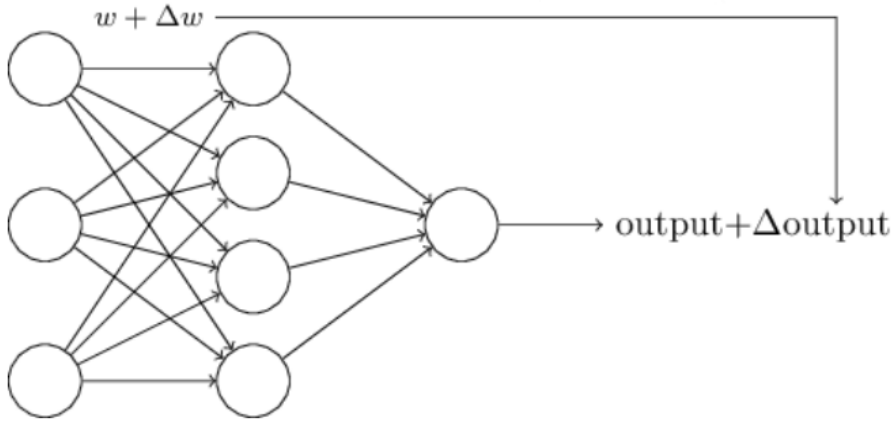

Figure 2.1: A simple neural network with one hidden layer consisting of four perceptrons. Small changes in weighted input causes changes in output. Reprinted from [6] with permission.

2.2.2 Expressiveness of neural networks: the universal approximation theorem †

Neural networks are at its core highly expressive function approximators. The universal approximation theorem of neural networks states that given

- A continuous function
- A compact domain, i.e. finite range

any continuous function can be approximated by a neural network with one hidden layer [6, Chapter 4], such as the network in Figure 2.1. There are multiple proofs of the universal approximation theorem for different kinds of activation functions. Two proofs of the theorem for the popular activation functions sigmoid and the rectified linear unit (reLU) are provided in [29, 30], respectively.

A visual explanation of the universal approximation theorem with the activation function sigmoid is given in [6, Chapter 4] for functions between euclidean spaces. In this section, we only consider the scalar case for simplicity in notation. Consider a neural network with one hidden layer with two artificial neurons in it. The sigmoid function is a s-shaped function, given in eq. (2.13),

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.13}$$

where $z = wx + b$. The weight, w, allows for the function to be made arbitrarily steep, approaching a step function. The bias, b, allows for shifting the function and is analogous to adding a constant to a linear function. Large weight values results in a more steep sigmoid function[1]. An illustration corresponding to a neural network with one hidden layer and large

[1]The sigmoid functions are approximated as step functions for simplicity and intuition. When adding more hidden nodes in the network, it is easier to see the effect with seemingly step functions.

weight and bias values in the upper neuron (we do not consider the lower neuron for now) of the hidden layer is given in Figure 2.2. All the animations in this section were made with the tool provided by [6, Chapter 4]. Note that the right part of the figure is showing the sum of weighted input and bias from the hidden layer, and not the network output, which is given by the function $\sigma(w_1\sigma(s_1) + w_2\sigma(s_2) + b)$, where σ is given in eq. (2.2).

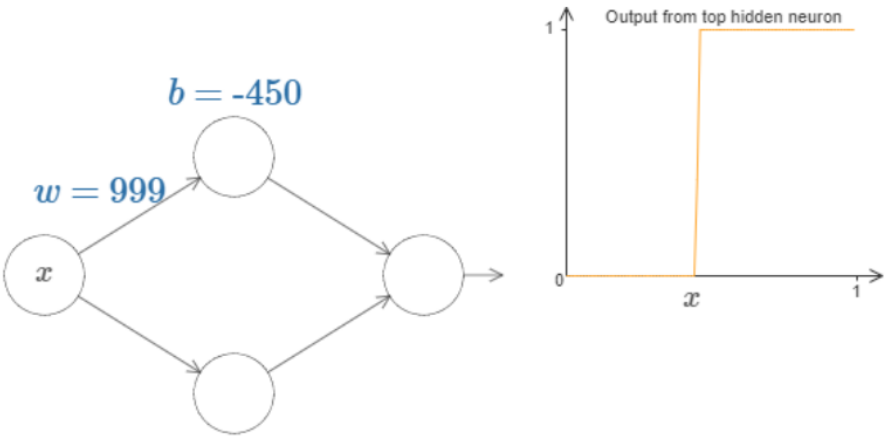

Figure 2.2: A simple neural network with one hidden layer (left) and the output of the weighted input and bias (right) for values $w = 999$ and $b = -450$.

For simplicity, the weight and bias is combined into one shifting variable, $s_i = -\frac{b_i}{w_i}$, $i \in G$, where G is the set of nodes in the network. Including the second artificial neuron (the lower node), the graph is extended with a new sigmoid approximating step function. The upper neuron shifts and scales the steepness of the first step function (i.e. first "stair" in the staircase function given in Figure 2.3). Likewise, the second neuron shifts and scales the steepness of the extension/second staircase. The weights (w_1 and w_2 in Figure 2.3) from the hidden units to the output scales the function in horisontal direction.

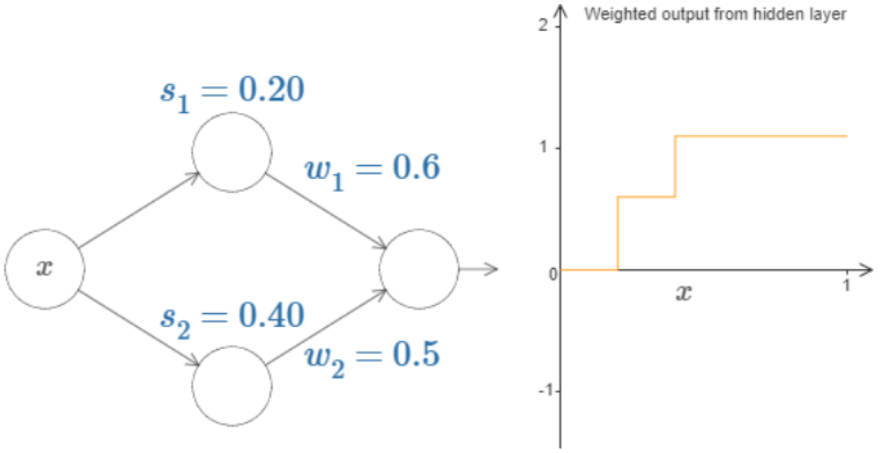

Figure 2.3: A simple neural network with one hidden layer (left) and the output of the weighted input and bias (right) for $s_1 = 0.2$, $s_2 = 0.4$, $w_1 = 0.6$ and $w_2 = 0.5$

If w_1 and w_2 in Figure 2.3 are set to a and $-a$ respectively, where a is any real number, the function becomes approximately a rectangular function, which starts at s_1 and ends at s_2, with height a and $-a$. Adding two more hidden units, and keeping the symmetrical weights as described for w_1 and w_2, a new rectangular shape is added. This is described in Figure 2.4, where the weights w_1 and w_2 are renamed to a single variable h, representing the function range (y-axis). Note the shifting variable and height symmetry between the two upper nodes and the two latter nodes.

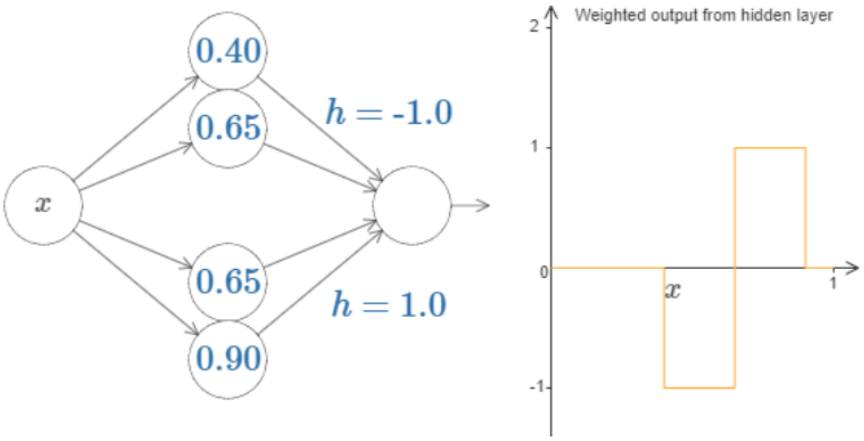

Figure 2.4: A simple neural network with one hidden layer (left) and the output of the weighted input and bias (right) for $s_1 = 0.4$, $s_2 = 0.65$, $s_3 = 0.65$, $s_4 = 0.9$, $h_1 = -1.0$ and $h_2 = 1.0$.

With this empirical evidence, intuitively, building a wider layer appears to give more expressiveness. To illustrate this, consider the function $f(x) = 0.2 + 0.4x^2 + 0.3\sin(15x) + 0.05\cos(50x)$ in the compact domain $[0,1]$. We want to approximate this using a neural network. In the networks above, the analysis is concerned with the weighted sum $\sum_i w_i a_i$, for $i \in G$. However, as was mentioned, the output of the network is actually $\sigma(\sum_i w_i a_i + b)$. This is generally not a problem. The neural network is designed so that the weighted output from the hidden layer is given by $\sigma^{-1} * f(x)$. For simplicity, the output neuron bias is zero. Figure 2.5 shows a possible configuration of weights and biases that yield a very rough approximation of the function $f(x)$, given the network on the left side. Recalling from the previous illustrations, adding more pairs of hidden-layer nodes ought to slice the compact domain into thinner rectangular functions, and subsequently more accurate approximation. An analogous analysis could have been made for the linear rectifier unit. In [6, Ch.4] the analysis is extended to neural networks with more than one input node (i.e. multi-variable function approximation), but the results are similar to one input node ‡.

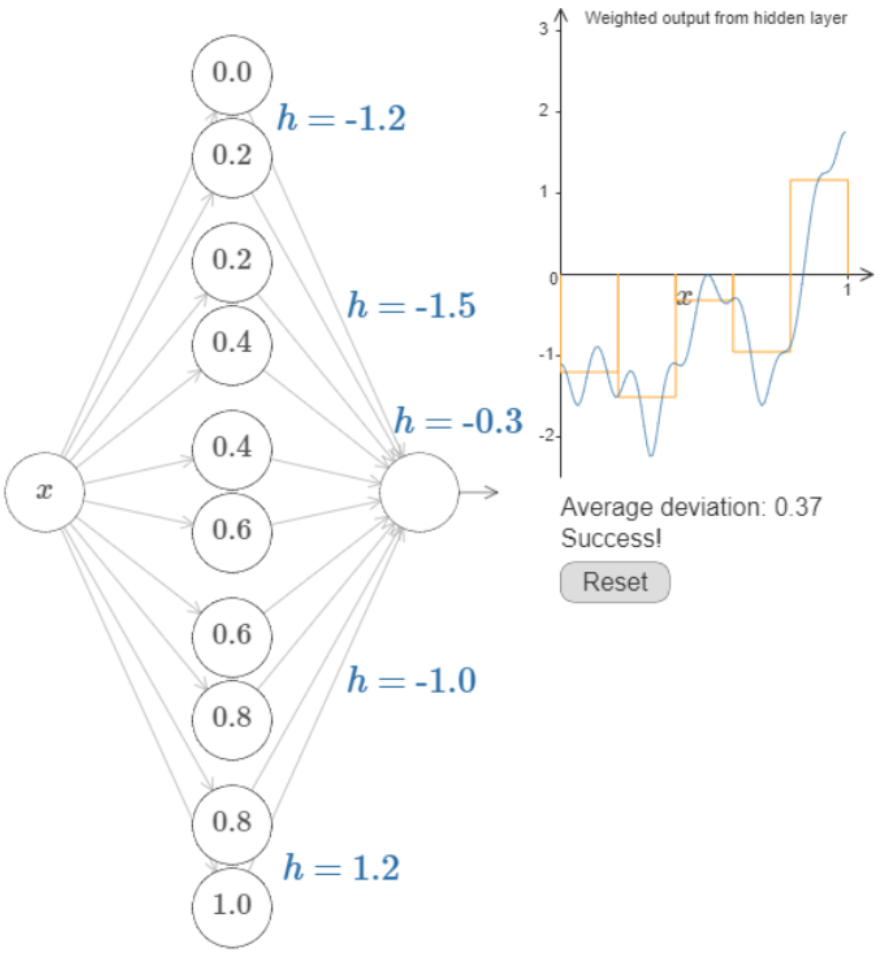

Figure 2.5: A simple neural network with one hidden layer (left) and the output of the weighted input and bias (right) for multiple nodes (values inside circles) and different output weights (h). Each output weight, h, corresponds to every two-pair neurons, starting from the top, i.e. h=-1.2 corresponds to the two uppermost neurons.

2.2.3 Neural network architectures

ANNs come in different flavours. The term ANN denotes the interconnection of nodes, so-called artificial neurons, that make up the neural network. As introduced in Section 2.2.2, neural networks are function approximators at its core, and may be applicable to many disciplines. As such, *specialised* architectures of neural networks have been developed throughout time.

A visual compilation of a few different neural network architectures is included in Figure 2.6. The landscape of neural network architectures is large, and the architecture is dependent on the task at hand. The combination of different architectures contributes to the complexity. It is for instance normal to combine a RNN with a FNN in time series regression, due to the output of a RNN being a sequence (i.e. vector) of values, while in regression we desire a real-valued continuous output variable. Moreover, neural networks also differ in being static or dynamic. Static neural networks are models in which the number of nodes and number

of layers are fixed and static during training, while vice versa for dynamic neural networks. In this thesis, only static neural networks are considered.

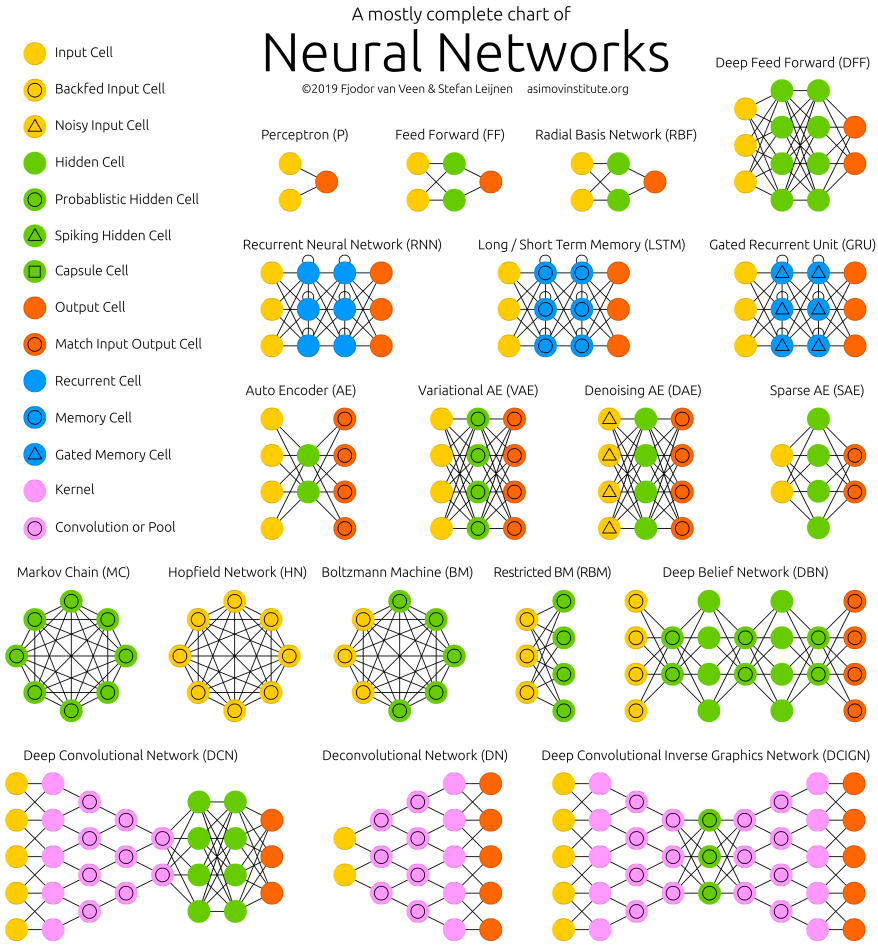

Figure 2.6: An overview of different neural network architectures. (Image courtesy of Fjodor Van Veen [7]).

2.2.4 Recurrent neural networks and the long short-term memory

In this thesis, the RNN will be the architecture in focus. A RNN is a class of neural networks that work on sequential data. Assume we have a S-long sequence of data, $x \in \mathbb{R}^S$. A RNN suitable to process such a sequence is given in eq. (2.14),

$$\begin{aligned} h_k &= f(x_k, h_{k-1}, \theta_h, \theta_x) \\ \hat{y}_k &= g(h_k, \theta_y) \end{aligned} \tag{2.14}$$

for $k \in \{1, 2, \ldots, S\}$. The terms f and g denote transformations in intermediate layers of the network and output layer of the RNN, respectively. h_k is known as the hidden state, θ_h, θ_x and θ_y are the weight matrices corresponding to x, h and the output mapping $\hat{y}$. Note that many details have been abstracted away, and will be uncovered in the coming sections.

A natural questions to ask is why is it advantageous to maintain a memory of the sequence (this memory of the past is often interchangeably used with the term hidden state, introduced in eq. (2.14)). We consider an example from natural language processing (NLP), similar to the one presented in [10, Chapter 10]. Consider the two sentences "Last week it was sunny" and "It was sunny last week". We want to develop a machine learning model to classify the weather. Semantically, these sentences are equivalent. The sentence structure, however, is different. A FNN would have to learn parameters for each words separately. Intuitively, this is problematic for two main reasons. Firstly, it likely requires more parameters in the model, with the possible consequences that the model overfits on the data it is trained on. Secondly, it is more computationally heavy as more parameters ought to be adjusted. A RNN adapt the important concept weight sharing, similar to the architecture CNN [31], and persist information from each element of the sequence of data acted upon.

In the rest of the section, we will first present a brief introduction to computational graphs. These graph-like visualisations are frequently used when describing recurrences, and will appear in coming chapters. In the latter part, the RNN that will be used in this thesis, the LSTM neural network, will be introduced.

Computational graphs for recurrent neural networks

Neural networks are usually visualised informally with a graph-like notation (as seen from Figure 2.6). A computational graph is a way of formalising (and visualising for smaller networks) the construction of a number of computations. It is typically indicated what kind of arithmetic operation that is performed between two nodes and what the resulting output is. In fact, computational graphs are essential to large deep learning software frameworks such as Keras [32] and Pytroch [5]. An adapted example from [10] is provided in Figure 2.7.

RNNs have a slightly peculiar graphical representation due to its recursive computations. They are often depicted with a self-loop. This loop symbolises the recurrent nature and may be unfolded to represent a directed acyclic computational graph, similar to the example in Figure 2.7. We consider the dynamic system given in eq. (2.15), which may be approximated by a RNN [10, Chapter 10].

$$h_k = l(h_{k-1}, x_k; \theta) \tag{2.15}$$

where h_k denotes the system state[2], x_k denotes the system input (external signal), l is some transformation and θ represents some system parameters. The state at time step k is dependent on the state at $k-1$. Note that $k \in \mathbb{Z}_+$ denotes a time step index. If we have a finite amount of time steps, k, the recurrence may be unfolded by applying its definition $k-1$

[2]Note that the system state is usually called the hidden state in RNN terminology.

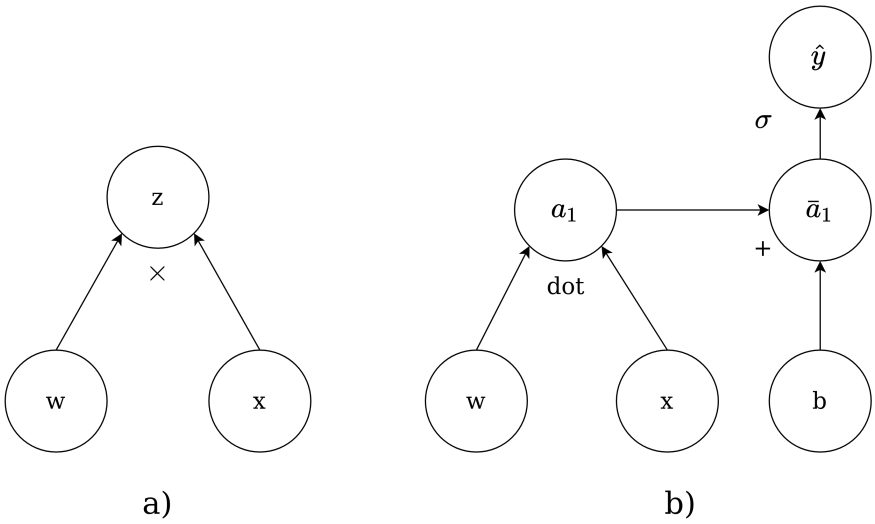

Figure 2.7: Two examples of computational graphs. (a) A graph using the multiplication arithmetic and produce the output $z = w \times x$. (b) A graph representing a forward pass using the dot product, addition and sigmoid function. The output is $\hat{y} = \sigma(wx^T + b)$. Forward pass is an important mathematical operation in neural network computations, and will be described in coming sections.

times. For $k = 4$, we obtain the unfolded system given in eq. (2.16),

$$\begin{aligned} h_4 &= l(h_3, x_4; \theta) \\ &= l(l(h_2, x_3), x_4; \theta) \\ &= l(l(l(h_1, x_2; \theta), x_3; \theta), x_4, \theta) \end{aligned} \tag{2.16}$$

Revisiting computational graphs, the system given in eq. (2.15) is usually represented by a self-loop. As seen, the very same system "unfolded" (eq. (2.16)) may be represented by directed acyclic computational graph, as claimed initially. This relation is visualised in Figure 2.8 for an arbitrary long unfold.

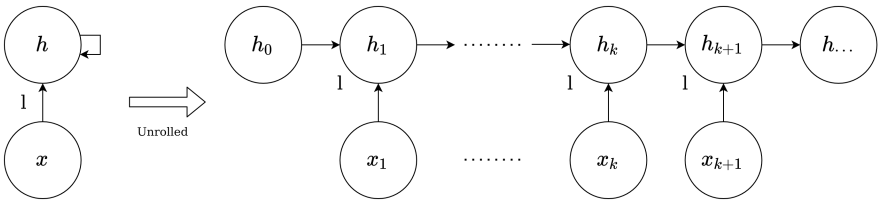

Figure 2.8: The computational graph of a recurrence relation. The left-most graph represents the recurrence relation in its compact form with self-loop. The right-most graph represents the unfolded recurrence relation.

Note that there is a discrepancy between the notation in neural networks and control system theory. In the machine learning, it is customary to denote the input as x. In control system theory, it is common to denote the state by x and the input by u. We will use the machine learning notation in the first part of the thesis due to its compliance with existing theory and results, and re-define the notation when necessary. This will be clearly expressed.

Long short-term memory

RNNs have a bad reputation of being notoriously difficult to train. This reputation is well-justified. The two widely known issues of training RNNs with gradient descent based methods is that the gradients, which are essential ingredients in gradient descent based methods, may vanish or explode [33, 34].

This difficulty is related to the unfolding of a recurrent neural network. Figure 2.8 illustrates how a RNN may be unfolded for a sequence $S \in \mathbb{Z}_+$. The function f was left intentionally unspecified in the previous section. In many cases, this function represents a FNN mapping. The unfolding is as such understood as multiple FNNs passing a memory, represented by the (hidden) state h_k, dependent on the input sequence $x = [x_1 \quad x_2 \quad \cdots \quad x_S]$. For long sequences, the RNN, consisting of FNNs, may get *very* deep. In general, deep neural networks suffer from either vanishing or exploding gradients as a result of the the large amount of multiplication operations necessary during backpropagation [34] (backpropagation will be introduced in coming sections).

For deep CNNs, the architecture type *Residual neural network* (ResNet) [8] has been largely successful. The architecture is based on the addition of shortcut connections that skip one or more neural network layers. These shortcuts act as a highway, allowing gradients to flow through the network (in both directions) without being affected by the nonlinear activation functions, which contribute to the vanishing or exploding of the gradients. The multiple neural networks along the highway may add their respective contributions to the highway, and as such, affect the gradients passing directly through the highway [8]. The concept is illustrated in Figure 2.9.

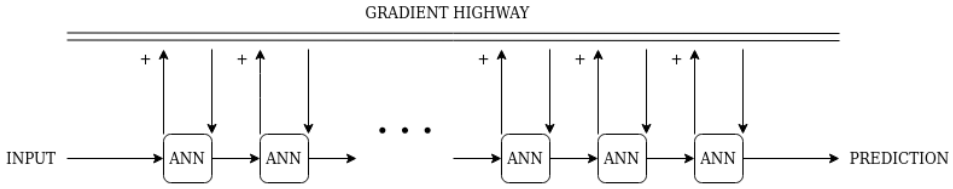

Figure 2.9: The conceptual idea of a residual neural network. The ANN blocks represent an arbitrary neural network architecture. The figure is inspired by the ResNet block from [8].

The internal skip-connection concept is not new, and certainly not restricted to CNNs. A popular RNN from 1997 known as the LSTM neural network [35], employed a similar design. This RNN architecture will mainly be in focus in this thesis due to its success at overcoming central issues constraining RNNs. The LSTM neural network is a special configuration of a RNN that adds an additional state, designated to persisting only *important* information from previous steps. The main question remains: how is the network supposed to truncate unnecessary information?

The LSTM does so by employing four "gates": i_k, f_k, g_k and o_k, visualised in Figure 2.10. The gates are essentially FNNs. The index $k \in \mathbb{Z}_+$ describes the sequence index. The sequence length refers to the length of the input sequences that ought to be processed, and

is equal to the number of unfolds in Figure 2.8. It is an adjustable hyperparameter, i.e. a parameter that alters the learning process. Long sequence lengths give the network more degrees of freedom (and as such "better memory") by having a large number of adjustable parameters. The downside is that the network may poorly generalise to new inputs, i.e. the model is less robust. Overfitting and underfitting are two important concepts in machine learning and will be thoroughly covered in coming sections.

We describe the gates from left to right in Figure 2.10, following [10, Chapter 10]. The forget gate, f_k, is responsible for deciding which information in the vectors x_k and h_{k-1} that is to be discarded. It is a FNN using the sigmoid as activation function. It outputs values in the range of 0 to 1, where values close to 0 emphasise which entries in the vectors x_k and h_{k-1} that may be "forgotten"/truncated, and vice versa for values close to 1. The next two gates are responsible for determining what information from the inputs that is to be stored. The *input* gate, i_k, is also a sigmoid FNN, deciding which values in the input that ought to be updated. The candidate gate, g_k, is a tanh FNN that outputs candidate values that are to be added to the state vector from the previous time step, c_{k-1}. The two gates are combined by multiplication, before added to the state vector, c_k. The *output* gate, o_k, is a sigmoid FNN that outputs a filtered version, h_k, of the updated cell state. The hidden state, h_k, may be passed as output or passed to the next LSTM-module if the network is not done processing the sequence, as indicated in Figure 2.10.

Consider a LSTM neural network with $H \in \mathbb{Z}_+$ number of nodes in the neural network layers (gates), also known as the hidden size. The input element x_k has $D \in \mathbb{Z}_+$ number of features. The dimensions of the input x_k and hidden state h_k of an arbitrary time step are $\mathbb{R}^D$ and $\mathbb{R}^H$, respectively. The operations described above, and illustrated in Figure 2.10, may be described mathematically as in eq. (2.17)-(2.22)[3]

$$i_k = \sigma(U_i x_k + W_i h_{k-1} + b_i) \tag{2.17}$$
$$f_k = \sigma(U_f x_k + W_f h_{k-1} + b_f) \tag{2.18}$$
$$g_k = \tanh(U_g x_k + W_g h_{k-1} + b_g) \tag{2.19}$$
$$o_k = \sigma(U_o x_k + W_o h_{k-1} + b_o) \tag{2.20}$$
$$c_k = f_k \odot c_{k-1} + i_k \odot g_k \tag{2.21}$$
$$h_k = \tanh(c_k) \odot o_k \tag{2.22}$$

where $\odot$ denotes the Hadamard product (see Definition 2.1.4) and σ denotes the sigmoid function (eq. (2.13)). Moreover,

- 1) $U_f \in \mathbb{R}^{H\times D}$, $W_f \in \mathbb{R}^{H\times H}$, $b_f \in \mathbb{R}^H$ are the forget gate kernel weight matrix, recurrent weight matrix and bias matrix, respectively (red box in Figure 2.10).

[3]There are multiple of different variants of the LSTM equations. In this thesis, the LSTM notation in [36] is used.

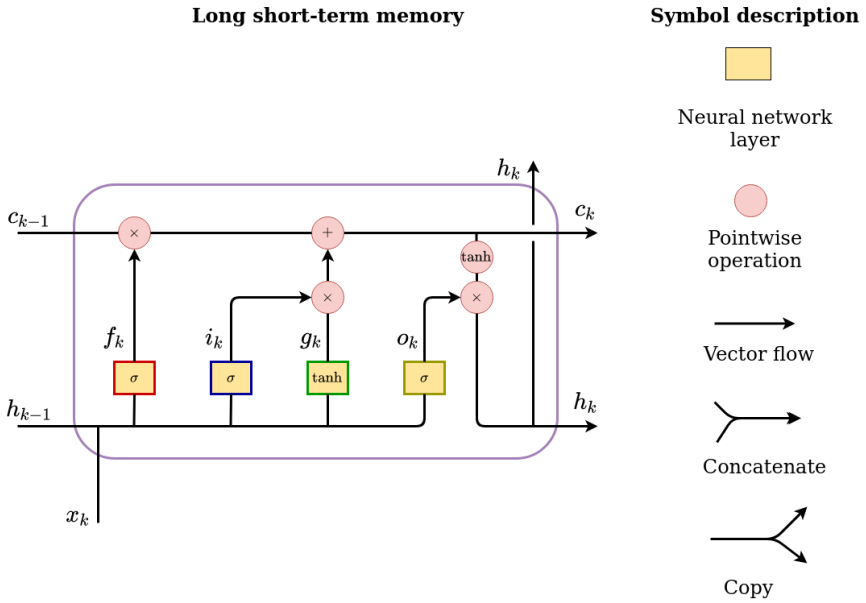

Figure 2.10: (Left) A long short-term memory RNN (the operations within purple boundary). (Right) Symbol description. Image adapted from [9].

- 2) $U_i \in \mathbb{R}^{H\times D}$, $W_i \in \mathbb{R}^{H\times H}$, $b_i \in \mathbb{R}^H$ are the input gate kernel weight matrix, recurrent weight matrix and bias matrix, respectively (blue box in Figure 2.10).
- 3) $U_g \in \mathbb{R}^{H\times D}$, $W_g \in \mathbb{R}^{H\times H}$, $b_g \in \mathbb{R}^H$ are the candidate gate kernel weight matrix, recurrent weight matrix and bias matrix, respectively (green box in Figure 2.10).
- 4) $U_o \in \mathbb{R}^{H\times D}$, $W_o \in \mathbb{R}^{H\times H}$, $b_o \in \mathbb{R}^H$ are the output gate kernel weight matrix, recurrent weight matrix and bias matrix, respectively (yellow box in Figure 2.10).

2.2.5 Deep neural network†

State-of-the art accuracy has been achieved in several domains, among them computer vision, by utilising deeper networks. The universal approximation theorem (see Section 2.2.2) gives us that one single hidden layer is sufficient for approximating any continuous function on a compact domain. A natural question to ask is why go deeper? An important caveat to relying too much on the universal approximation theorem is that it only gives us the fact that it is indeed possible to approximate a continuous function with one hidden layer. Finding the optimal architecture (number of neurons) and hyperparameter selection may be a very challenging task. As such, the universal approximation theorem does not consider objectives outside approximating a continuous function. In most cases, a neural network is not of very much use if it does not generalise well to unseen data. This may be due to the network either overfitting or underfitting. In the case of overfitting, the network often has too many parameters and memorise the training data (including outliers and noise). In the case of underfitting, the opposite may be true, restricting the network from learning the representation of the features.

By going deeper, a network may learn more abstract representations of the data [37, 38], often with less artificial neurons. This is perhaps easiest to visualise in a computer vision setting. In CNNs used for object classification, the first few layers represent concrete features such as horizontal and vertical lines. The weights at the deeper layers capture abstract features by assembling the more concrete features. A visual motivation for this is provided in Figure 2.11, using a pre-trained ResNet18 neural network [39] (18 layers deep). As observed, the above-mentioned concepts are demonstrated by this simple example. In the first layers, the weights emphasize simple features such as horizontal lines, vertical lines (two first filters) and colours (the last few filters). We can certainly observe from the activations that this is indeed a zebra. In the last layer, it is much more difficult to see what the network weight emphasise on, and the activation of the image is now beyond recognisable, as much more complex features have been represented ‡.

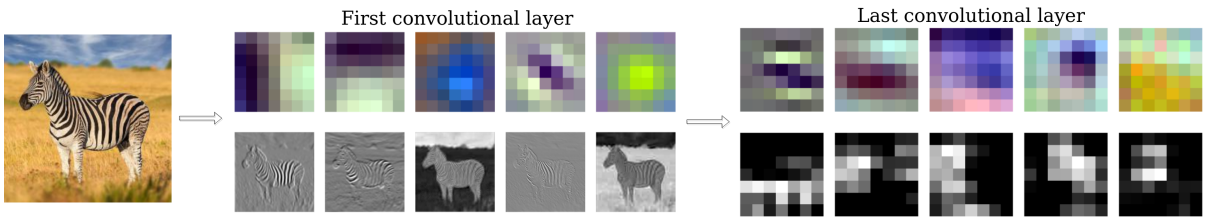

Figure 2.11: Passing a picture as input in a pre-trained ResNet18 convolutional neural network. The upper pictures represent some arbitarily selected filters (weights) of the network and the latter pictures represent the actual activation of the network. These illustrations are presented for the first and last layer of the network, respectively ‡.

2.2.6 Optimisation of a neural network: Optimiser, loss function and backpropagation †

Optimising a neural network for a particular task involves a combination of various components: optimisation method, loss function, a network configuration and training data. An abstract overview of the training procedure of a general neural network is presented in Algorithm 1. In this section, a closer look at the inner loop workings (line 3-6) of the training procedure will be presented. Note that we will in general only consider the scalar case as to not clutter the notation. The principles are analogous in multidimensional space ‡.

Forward pass and loss function †

The first overall step of a training procedure is to produce a prediction, be it a continuous variable (regression) or an integer (classification) representing some class that is to be predicted. This is called the forward pass step (line 3 in Algorithm 1). Consider a neural network

Algorithm 1 An abstraction of a neural network training procedure †

Require: Stopping criteria $\epsilon > 0$, loss function L, optimiser O, training data D, neural network f_θ where θ are the parameters

for epoch $< \epsilon$ **do**
  for (input,label) $\in D$ **do**
    Calculate prediction f_θ(input) ▷ Forward pass
    Calculate loss L(prediction, label)
    Calculate gradients of loss w.r.t. parameters $\nabla_\theta L$ ▷ Backward pass
    Update parameters $O(\nabla_\theta L)$
  end for
  epoch ← epoch + 1
end for

with L layers, $f(x;\theta)$. Given the network parameters θ and input data x, a single scalar forward pass through the network is given in eq. (2.23)-(2.24),

$$a^0 = x \qquad l = 0 \tag{2.23}$$

$$a^{(l)} = \sigma(\theta^{(l)} \cdot a^{(l-1)}) = \sigma(z^l) \quad l = 1, \ldots, L \tag{2.24}$$

where the superscript denotes the layer in focus, σ represents a nonlinear activation function (see Section 2.2.2) and $a^{(l-1)}$ represents the input to layer l. The output of layer l is thus $a^{(l)}$, which is used as input to the next layer, $l+1$, in which the same procedure as in eq. (2.24) is performed with the parameters from layer $l+1$ and $a^{(l)}$ as input. At the last layer, a prediction, $\hat{y} = f(x;\theta)$ is produced. Note that the z-notation in the latter expression of eq. (2.24) is a much used notation, partly due to the easier notation in the back-propagation algorithm, which is to be introduced in this section. It is also common to denote the neural network parameters as a weight matrix, w, and a bias vector b. This is not strictly necessary due to the bias trick, in which the bias vector is appended to the weight matrix (extension as a new column) and the input vector appended with an additional element (1). One may then denote this new matrix that combines the weight and bias parameters as θ for easier notation. See Figure 2.12 for a conceptual representation of the bias trick for a 3×3 matrix.

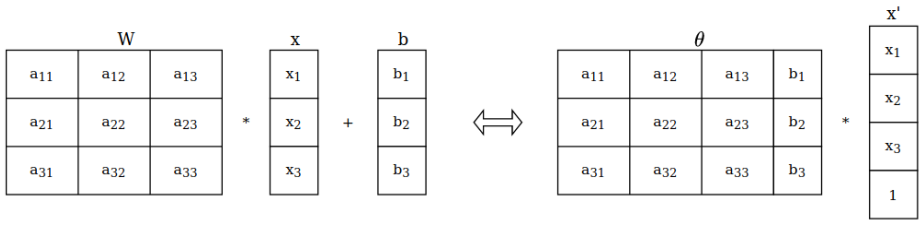

Figure 2.12: The bias trick. Multiplying a matrix $W \in \mathbb{R}^{n \times m}$ with a vector $X \in \mathbb{R}^{1 \times m}$ and adding a vector $b \in \mathbb{R}^{1 \times m}$ is equivalent to augmenting the weight matrix with a 1×n column and augmenting the vector x with an additional element, 1, and then multiplying the resulting pairs ‡.

In order to evaluate how close (or far off) the network prediction is, some measurement

of the error is needed. Loss functions serve this purpose (line 4 in Algorithm 1). A frequently used loss function for regression tasks is the mean square error (MSE) [40]. Given a sample of n data points, x and target values, $y_i, i = \{1, 2, \cdots, n\}$, the MSE is given in eq. (2.25),

$$L(\theta) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 \tag{2.25}$$

where $\hat{y}_i = f(x_i;\theta)$ is the prediction of the neural network on data points x_i.

One may observe from eq. (2.25) that when $\hat{y}_i$ approaches y_i, the loss function is decreasing towards 0, in which the two are equal. MSE possess some convenient properties such as differentiability and inner product inducement. The first ensures that it is differentiable everywhere. The second property yields that the MSE of y_i and $\hat{y}_i$ is the euclidian distance between the two. This is related to the intuition above: the error increases when the euclidean distance between the target and the prediction increases ‡.

The backpropagation algorithm †

One of the essential steps of neural network training is finding the gradients of the loss function with respect to the parameters. These gradients are then used by the optimiser in minimising the loss function, and as such, enable learning. The backpropagation algorithm[4] is a widely used algorithm for this purpose. In a multi-layered neural network, the backpropagation algorithm propagates the information from the loss function backwards in the network in order to compute the gradients on a layer-basis (line 5 in Algorithm 1).

Consider a L-layer deep neural network with parameters θ, the loss function $L(a^{(L)}, y)$, where $a^{(L)}$ is defined in eq. (2.23)-(2.24), and $z = \theta x$. The gradients of a specific layer are found by utilising the chain rule, as shown in eq. (2.26) [6, Chapter 2].

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}\theta^{(L)}} &= \frac{\mathrm{d}L}{\mathrm{d}a^{(L)}}\frac{\mathrm{d}a^{(L)}}{\mathrm{d}z^{(L)}}\frac{\mathrm{d}z^{(L)}}{\mathrm{d}\theta^{(L)}} \\ \frac{\mathrm{d}L}{\mathrm{d}a^{(L-1)}} &= \frac{\mathrm{d}L}{\mathrm{d}a^{(L)}}\frac{\mathrm{d}a^{(L)}}{\mathrm{d}z^{(L)}}\frac{\mathrm{d}z^{(L)}}{\mathrm{d}a^{(L-1)}}\end{aligned} \tag{2.26}$$

where $a^{(L)}$ represents the activation of layer L ‡.

Backpropagation through time

Backpropagation remains an essential ingredients in producing useful neural network models. The backpropagation rules from eq. (2.26) are general purpose. A special case of the backpropagation algorithm has been developed for training RNNs. The application of the general purpose backpropagation algorithm on RNNs is called backpropagation through time (BPTT) [42]. It works by firstly expanding the computational graph of the RNN (see Section 2.2.4) in order to acquire the sequence dependencies. The general purpose backpropagation is then applied to to compute the gradients.

[4] The backpropagation algorithm is associated with automatic differentiation. Automatic differentiation is similar to backpropagation, albeit being a general family of techniques [41].

Assume we are to train the RNN given in eq. (2.14). We have sequential data, $x \in \mathbb{R}^S$, where each element $x_1, \ldots, x_S$ in the vector x correspond to an element in the sequence set $\mathscr{S} = \{1, \ldots, S\}$. Given an appropriate loss function $L(x, y, \theta)$, it is derived in [43] that the gradients of the loss with respect to the parameters θ_h for an arbitrary input element index, $k \in \mathscr{S}$, are as shown in eq. (2.27).

$$\frac{\partial L}{\partial W} = \frac{1}{S} \sum_{k=1}^{S} \frac{\partial L(y_k, \hat{y}_k)}{\partial \hat{y}_k} \frac{\partial g(h_k, \theta_y)}{\partial h_k} \frac{\partial h_k}{\partial W} \tag{2.27}$$

The expression for U is equivalent, only with U inserted for W. The term $\frac{\partial h_k}{\partial W}$ is in [43] shown to equal the term in eq. (2.28)

$$\frac{\partial h_k}{\partial W} = \frac{\partial f(x_k, h_{k-1}, W)}{\partial W} + \sum_{i=1}^{S-1} \left(\prod_{j=i+1}^{S} \frac{\partial f(x_j, h_{j-1}, W)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, W)}{\partial W}. \tag{2.28}$$

We observe from the two terms in eq. (2.27) and eq. (2.28) that the gradients of the loss with respect to the parameters are indeed dependent on the previous sequence elements. Note that the expression for $\frac{\partial h_k}{\partial U}$ is equivalent to the one in eq. (2.28), only with U inserted for W. If the sequence is very large, we see from eq. (2.28) that the computed gradient may vanish or explode, depending on initialisation and optimisation method. Different strategies for dealing with the problem of vanishing or exploding gradients are discussed in [43], among them several time step truncation techniques. We refer to [43] for the interested.

Optimisation methods: gradient descent based optimisation †

The optimiser is responsible for updating the parameters in the neural network (line 6 in Algorithm 1), based on the information (gradients) provided by the backpropagation algorithm. The stochastic gradient descent (SGD), Adam optimiser and variants of them are typically used in deep learning. They are based on the principle of gradient descent: a minimum of a proxy function, $J : \mathbb{R}^n \to \mathbb{R}$, for instance a loss function, is iteratively sought by moving in the direction of the negative gradient, $-\nabla_\theta J(\theta)$. The parameter update rule for the k'th gradient descent iterate is given in (2.29) [44],

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta J(\theta_k) \tag{2.29}$$

where α is the step size. In deep learning, this is typically called learning rate.

For loss functions, for example the MSE in eq. (2.25), the basic gradient descent (batch gradient descent) calculates the gradient with respect to the whole dataset.

$$\nabla_\theta J(\theta) = \frac{1}{n} \sum_{i}^{n} \nabla_\theta L_i(x_i; \theta) \tag{2.30}$$

The number of operations for a single gradient step is $\mathscr{O}(nd)$ [45], where n is the number of samples and d is the input dimensionality . When the dataset (n) is large and/or the number

of features (d) are large, the number of operations nd become large and not feasible for optimisation in deep learning.

The SGD [46] is an approximation of the batch gradient descent aimed at reducing the computational load. The gradient is approximated as shown in (2.31),

$$\frac{1}{n}\sum_{i}^{n}\nabla_\theta L_i(x_i;\theta) \approx \nabla_\theta L_j(x_j;\theta) \tag{2.31}$$

where $j \in \{1,2,3,..,n\}$ is chosen by uniform sampling.

SGD is among the most used optimisers. Several variants of SGD exist, for example SGD with momentum, which seeks to speed up the learning process. It is described more in detail in [10, Chapter 8]. Adaptive learning rate optimisers, such as Adam [47], have become increasingly popular due to it usually working well with less hyperparameter tuning effort, and converge fairly quickly to decent solutions thanks to its adaptive learning rate and momentum. A caveat to these optimisers is that it has been observed that adaptive methods converge to solutions that generalise worse for DNNs [48, 49] ‡.

2.3 Robustness in neural networks

One of the central challenges in producing a *successful* machine learning model is that it generalise well to unseen input, i.e. input that the neural network was not trained on. Adversarial examples may be viewed as *worst-case* unseen input. If the model generalise well, we say it is a robust model. Non-robust models are by large related to model capacity. Restricting the model capacity by reducing the number of parameters seems as a reasonable first step in producing more robust models. Restricting the model too much, however, may lead to a model that underfits, i.e. a too simple model that cannot capture the essential patterns in the data. Adjusting the model capacity merely on empirical experimentation is time consuming due to the yo-yo-like effect, in which one reduce and increase the model capacity in increments. Fortunately, there are tools aimed at providing some general guidelines for producing possibly robust models.

In this section, robustness in neural networks will be explored, together with some common tools for aiding in producing more robust networks. Some of the underlying reason for the lack of robustness in neural networks will be presented in Section 2.3.1. Adversarial examples are worst-case input and an important research topic in deep learning. The reason for their existence will be covered in Section 2.3.2, along with two methods for generating adversarial examples. Norm regularisation and robust optimisation are tools frequently used to call forth robustness in neural network models, and will be presented in Section 2.3.3 and Section 2.3.4.

2.3.1 Model capacity, overfitting and underfitting

A key goal when employing machine learning is to produce models that generalise to new, *unseen* input. This is input that the model was not trained on. If the model performs well on this input, we say it generalises well. In this work, the terms generalisation and robustness are synonymous.

In supervised machine learning, we typically have access to a *labeled* training set. Each training input is associated with its true value. This enables us to compute an error metric on the training set, for which we seek to minimise with respect to the model parameters. The two fields deep learning and optimisation theory have much in common. A key difference, however, is that in addition to minimising an error metric, we desire the test error[5] to be as low as possible for the deep learning case. The test set is a portion of the dataset that has been collected separately from the training set, and is as such considered "unseen" input. It must, however, be generated from an input distribution that we anticipate the machine learning system to come across when deployed. The test error is defined as the expected value of the error on unseen input [10, Chapter 5].

When training a neural network, two properties for a decent neural network based model to possess are listed in [10, Chapter 5], and repeated in the following list:

1. Small training error
2. Training and test error are close to each other (small generalisation gap)

The two properties are frequently used in analysing underfitting and overfitting, both concepts directly related to model capacity. Model capacity is a term used to describe a neural network's ability to fit different functions. Underfitting is a term describing when a model fails to satisfy 1. This is usually a sign of having too *low* model capacity. Overfitting is a term describing when a model fails to satisfy 2. This is usually a sign of having too *high* model capacity. The relations between model capacity, underfitting and overfitting are conceptually presented in Figure 2.13. As the model capacity increases, the generalisation gap also increases. The models with capacity in the right end of the figure are less robust. The models residing in the proximity of the red line are likely to be more robust.

[5] Test error and generalisation error are used interchangeably in the machine learning literature

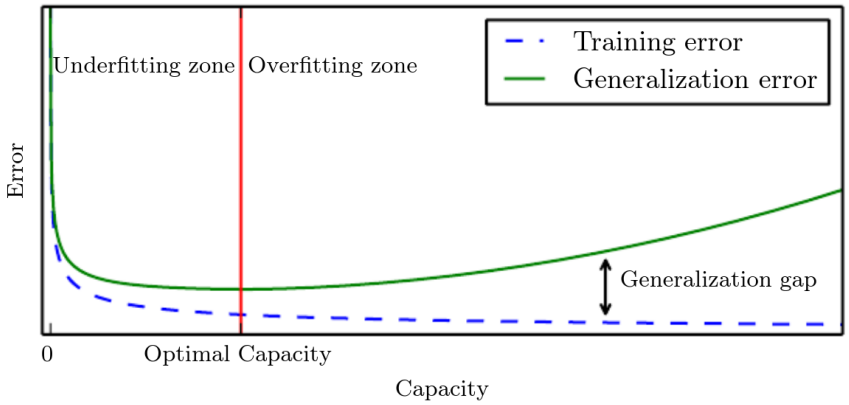

Figure 2.13: A conceptual possible relationship between model capacity and train and test errors. The red line separates the model capacities that are likely to underfit from the models that are likely to overfit. Image courtesy to [10].

Figure 2.14 exemplify the importance of model capacity in a linear regression setting. The goal is to approximate a nonlinear function. In the left-most model of Figure 2.14, the model capacity is too low for fitting this dataset. We say the model suffers from underfitting and the test error will likely be high. The model in the middle is of sufficient capacity and will likely generalise well (training and test errors will be low and close to each other). The right-most model has too high capacity. We have fitted outliers from the training dataset. Outliers have a low probability of appearing in the test set [50], and as such, this model will with high probability have a low training error, but high test error. This is a clear sign of overfitting (high generalisation gap). In the case of neural networks, one does not have the concept of polynomial degree in the same sense. We change the model capacity by increasing the number of parameters (weights and biases) in the neural network, either by building wider networks (more nodes) or deeper networks (more layers). In Section 2.2.2, a visual motivation for how the increasement of nodes in the network lead to better accuracy. However, increasing the number of nodes too much, we get a finer slicing of the function we seek to approximate, but also increase the risk of fitting outliers.

There is no unambiguous way of deciding model capacity. It is highly dependent on the problem at hand and the complexity of the data. Figure 2.13 provides a motivation for what to be aware of when increasing model capacity (we want to keep the generalisation gap small). In the preceding sections, various tools frequently used in producing more robust neural networks will be explored.

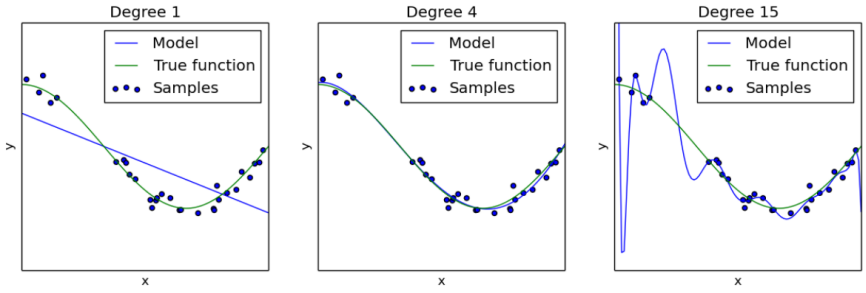

Figure 2.14: Polynomial regression (linear regression where we add polynomial features to the initial linear model) used for approximating a nonlinear function. The left-most model corresponds to a underfitting regression model $Y_1 = \theta_0 + \theta_1 x$. The middle model corresponds to an optimal regression model $Y_2 = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$. The right-most model corresponds to a overfitting regression model $Y_3 = \theta_0 + \sum_{i=1}^{15} \theta_i x^i$. Image courtesy to [11].

2.3.2 Adversarial examples

Adversarial examples entered the deep learning community in 2013 [15]. Adversarial examples are inputs that are slightly perturbed, often imperceptible. It is not the perturbation magnitude that causes the neural network's to exhibit large variations in output, but rather the direction (sign) of the gradients. Training a neural network, the goal is to minimise some loss metric with respect to the model parameters (weights and biases). Adversarial examples on the other hand have a different goal. They are generated by adjusting an input $x \in \mathbb{R}$ by a some value, $\delta \in \mathbb{R}$, with the intent of maximising a loss metric, constrained by an upper bound on the perturbation given by $\left\| x^{adv} - x \right\|_p \leq \epsilon$, where $x^{adv} = x + \delta$ and $\epsilon \in \mathbb{R}_+$ is an adjustable parameter that control the strength of the perturbation. Large ϵ yields large perturbations and bad performance on test data, but render the anomaly easier to detect. Analogously, low ϵ yields lower perturbations and not as drastic worsening of performance. Consequently, the anomaly is more troublesome to pinpoint. For now, the δ remains obscure. We will look into two white-box methods, uncovering this term. For a white-box method, it is assumed that one has access to the model parameters.

Fast gradient sign method (FGSM)

The fast gradient sign method was firstly explored in [51]. The method is intuitive for harnessing adversarial examples. It utilises the gradient of the cost function used to train the neural network by linearising the cost function around the current value of the parameters. Given a neural network with parameters $\theta \in \mathbb{R}^m$, input data $x \in \mathbb{R}^n$ and a cost function used to train the network, $J(x, y, \theta)$. The fast gradient sign method finds a perturbation, δ, as shown in eq. (2.32),

$$x^{adv} = x + \delta = x + \epsilon \cdot sign(\nabla_x J(x, y, \theta)) \tag{2.32}$$

where $\epsilon \in \mathbb{R}_+$ is a tuning parameter. The perturbation is bounded by the ℓ_∞-norm [52], such that $\left\| x^{adv} - x \right\|_\infty \leq \epsilon$. The method is effective due to the relatively effortless acquisition of

the gradients by backpropagation (see Section 2.26). The method was initially used in a classification setting, but may also be used to perturb input in a regression setting, as is done in [12, 53].

Projected gradient descent method (PGD)

The projected gradient descent method (PGD) [54] is another method for harnessing the perturbations used to generate the adversarial examples. It is altogether an iterative variant of the FGSM, in which FGSM is applied a finite amount of times k with a step size $\alpha = \frac{\epsilon}{k}$. The values are clipped for each iterate to make sure that the adversarial examples stay inside a neighbourhood of the original input. The method requires a initialisation due to the iterative nature. The initialisation is usually a random value close to the original input. Given a neural network with parameters $\theta \in \mathbb{R}^n$, input data $x \in \mathbb{R}^n$, a cost function used to train the network, $J(x, y, \theta)$ and an initialisation $x^{(0)} = x$, the projected gradient descent method is then described in eq. (2.33),

$$x_k^{adv} = \text{clip}_{x,\delta}(x_{k-1} + \delta)) = \text{clip}_{x,\epsilon}\left(x_{k-1}^{adv} + \alpha \cdot sign(\nabla_{x_{k-1}^{adv}} J(x_{k-1}^{adv}, y, \theta))\right) \tag{2.33}$$

where $clip_{x,\epsilon}$ is an element-wise clipping operation confining each input element, $x_j \in \mathbb{R}$, in the range $[\max(0, x_j - \epsilon), \min(1, x_j + \epsilon)]$ [54]. PGD is often interchanged with the basic iterative method from [55]. They are both iterative variants of FGSM. The main difference is how the methods are initialised.

Adversarial examples: consequence of linearity in networks

Adversarial examples have been a major headache in the deep learning community since its entry in 2013. What appeared to be a discovery that could be circumvented by "masking", i.e. hiding the gradients (recall from the introduction of FGSM and PGD that the gradients are important ingredients in the techniques generating adversarial examples), is in fact a much more delicate issue. The obvious solution by hiding the internals is not sufficient. Gradient masking techniques for ensuring robustness fail due to the generalisation of adversarial examples. Even if one deny the outside world access to the useful gradients, it turns out that the adversary may train their own model for some task, acquire the gradients and generate adversarial examples. These adversarial examples generalise to other models (for example models that has been "gradient masked") aimed at similar tasks. This has been studied in several works, among them [51, 56].

A natural questions arise: what ingredient in the neural network contribute to the existence of adversarial examples? A hypothesis is that adversarial examples are dot products in high dimensional space [51]. Consider the dot product between a parameter vector, $\theta \in \mathbb{R}^n$ and an adversarial example, $x^{(adv)} \in \mathbb{R}^n$. The dot product is given in eq. (2.34),

$$\theta^T x^{adv} = \theta^T x + \theta^T \delta \tag{2.34}$$

where $\delta \in \mathbb{R}^n$ is discussed in the section introduction, and two methods for generating it has been proposed (FGSM and PGD). In deep learning, eq. (2.34) is passed to a nonlinear activation function, for which it causes the activation to grow by $\theta^T\delta$. The perturbation may be maximised, subjected to the ℓ_∞-norm by letting $\delta = sign(\theta)$. This is nothing but how FGSM and PGD work. Since θ has n dimensions and the average vector element is given by a, then the activation will grow by $\epsilon \cdot a \cdot n$ (recall that ϵ is the upper-bound of the perturbation). As seen, the change in activation function grow linearly with the dimension. By making a number of infinitesimal changes to the high dimensional input, this add up to a big contribution, leading to the dramatic effect of adversarial examples: large changes in output. The main hypothesis is thus that DNNs that employ piece-wise linear activation functions operate in numerous small linear regions [57], and that it is the linearity of neural networks that likely introduce the susceptibility to adversarial examples.

Although there exist no unambiguous solution to the problem adversarial examples, the spark in research has produced many interesting tools for producing more robust neural networks. We will review some of these tools in the preceding sections.

2.3.3 Norm regularisation

Regularisation describes any technique intended for improving the generalisation ability of a model [58]. Norm regularisation adds information in the form of some norm of the parameters to an optimisation objective in attempting to achieve this goal. In this thesis, the popular ℓ_2 norm regularisation is considered [10, Ch.7]. As spoiled by the name, ℓ_2 norm regularisation employs the ℓ_2-norm (see Section 2.1.1 and Section 2.1.2) as a means to restrict the model capacity (the importance of model capacity with regard to underfitting or overfitting is covered in Section 2.3.1). The regularisation technique is not unique to neural networks. It is also applied in a wide range of regression models, such as linear regression. In this section, ℓ_2 norm regularisation will be treated in the general case of feedforward neural networks, before the specialised case of recurrent neural networks is treated.

We will now consider a norm regularised cost function. A cost function denotes the function that the neural network parameters are optimised with regard to, and combine a loss function, for example the mean square error in eq. (2.25), and a regularisation term, $R : \mathbb{R}^n \to \mathbb{R}$. We will consider ℓ_2 norm regularisation, and thus we may express R as $R(\theta) = \lambda \|\theta\|_2^2$. A general ℓ_2-regularised cost function is given in eq. (2.35),

$$J(x;\theta) = L(x;\theta) + R(\theta) = L(x;\theta) + \lambda \|\theta\|_2^2 \tag{2.35}$$

where $\theta \in \mathbb{R}^n$ describes the model parameters, $L : \mathbb{R}^n \to \mathbb{R}$ is an appropriate loss function, $x \in \mathbb{R}^n$ denotes the training data and $\lambda \in [0, \infty)$ is a hyperparameter that adjusts the regularisation strength. A large λ yields stronger regularisation and vice versa.

In order to get an intuition of how norm regularisation affects the neural network parameters during training, let us consider the gradient of eq. (2.35), given in eq. (2.36),

$$\nabla_\theta J(x;\theta) = \nabla_\theta L(x;\theta) + 2\lambda\theta \tag{2.36}$$

Assume we utilise some variant of the gradient descent algorithm (see Section 2.2.6) in order to update the network parameters. The update rule for the k-th iterate is derived in [10] and repeated in eq. (2.37),

$$\begin{aligned} \theta_{k+1} &\leftarrow \theta_k - \alpha(\nabla_\theta L(\theta_k; x) + 2\lambda\theta_k) \\ &\Updownarrow \\ \theta_{k+1} &\leftarrow (1 - 2\lambda\alpha)\theta_k - \alpha(\nabla_\theta L(\theta_k; x)) \end{aligned} \tag{2.37}$$

where $\alpha \in \mathbb{R}$ is commonly denoted as the learning rate (see Section 2.2.6). It is readily seen from eq. (2.37) that ℓ_2 norm penalty is equivalent to decreasing the parameters of the neural network by a constant factor $1 - 2\lambda\alpha$.

The update rule in eq. (2.37) shows how the parameters shrink from iteration to iteration. What is even more interesting is to see how regularisation affects the parameters over the whole training course. Assume there exists a quadratic approximation of a loss function, $\tilde{L}(\theta)$ in the neighbourhood of some model's parameters θ. It is shown in [10] that the regularised solution (minimised cost function) is related to the unregularised solution (minimised loss function) by eq. (2.38),

$$\begin{aligned} \tilde{\theta}^* &= (Q\Lambda Q^T + \lambda I)^{-1} Q\Lambda Q^T \theta^* \\ &= (Q(\Lambda + \lambda I)Q^T)^{-1} Q\Lambda Q^T \theta^* \\ &= Q(\Lambda + \lambda I)^{-1} \Lambda Q^T \theta^* \end{aligned} \tag{2.38}$$

where $Q \in \mathbb{R}^{n \times n}$ is square matrix where the i-th column is the corresponding eigenvector $q_i \in \mathbb{R}^n$ and $\Lambda \in \mathbb{R}^{n \times n}$ is a diagonal matrix whose diagonal entries correspond to the eigenvalues of the hessian. The matrices H, Q and Λ are related by the eigendecomposition, $H = Q\Lambda Q^{-1}$.

From this simple example it is readily seen what effect regularisation has on the optimisation problem of a quadratic cost function. The regularised solution, $\tilde{\theta}^*$ is is a scaled version of the unregularised optimum, θ^*, in which it is rescaled along the axes defined by the eigenvectors of the hessian. In other words, the vector components of θ^* that are aligned with the k-th eigenvector of the hessian, H, is rescaled by a factor of $\frac{\lambda_k}{\lambda_k + 1 + \lambda}$. Regularisation is thus a way of penalising directions along the eigenvectors of the hessian that do not contribute a lot to optimising the cost function.

Norm regularisation and recurrent neural networks

Norm regularisation is an important tool in producing more robust neural network models, and is frequently used when training CNNs and FNNs. As dicussed previously, RNNs work on input sequences and are often harder (see Section 2.2.4) to train. In LSTM neural networks, we have more options with regard to norm regularisation. The LSTM neural network is presented in Section 2.2.4, from which we we observe that we have two types of parameter matrices: input weight matrices (U_j) and recurrent weight matrices (W_j) for $j = \{i, f, g, o\}$. As such, the a norm regularised cost function for a LSTM RNN may include both weight types.

We now rewrite the general cost function in eq. (2.35) to account for the two weight matrices types that LSTM neural networks consist of. Let $U = [U_i, U_f, U_o, U_g]$ represent the matrix collection of all input weight matrices and equivalent for W. Moreover, assume $L(\cdot)$ is some appropriate loss function. The ℓ_2 norm regularised cost function for a LSTM neural network is given in eq. (2.39),

$$\begin{aligned} J(x, h; U, W) &= L(x, h; U, W) + R_1(U) + R_2(W) \\ &= L(x, h; U, W) + \lambda_1 \|U\|_2^2 + \lambda_2 \|W\|_2^2 \end{aligned} \tag{2.39}$$

where R_1 and R_2 are ℓ_2 matrix norm functions (see Section 2.1.2) and λ_1 and λ_2 are corresponding hyperparameters described in Section 2.3.3.

In one influential paper on RNNs and the difficulty of optimising them, [59], they propose that while ℓ_2 regularisation of the recurrent weights (W) may help with mitigating the issue of exploding gradients, it may also prevent the RNN in exhibiting long term memory traces. This claim is based on the explanation that ℓ_2 regularisation term ensures that the largest singular values of the recurrent weight matrices stays smaller than 1. This, however, restricts the model to a single point attractor at the origin, in which signals inserted in the model die out exponentially fast.

2.3.4 Robust optimisation †

At the core of many machine learning tasks is optimisation theory. A vast amount of optimisation approaches have been important in machine learning as a result of the diverse applicability and theoretical insight they give. Robust optimisation is an optimisation branch that deal with optimisation problems involving uncertain data by reinterpreting ideas from convex optimisation theory and duality. The goal is to make the solutions of the optimisation solution robust against bounded uncertainty [60]. This uncertainty may result from the process of obtaining data. It can also come as a result of errors in model specification.

Lately, an increasing amount of neural network robustness analysis have focused on the training procedure, including the dynamics of the optimisation algorithm and loss functions. This is connected to robust optimisation, which will be further clarified in this subsection. ‡

Robust linear programming †

Consider the linear program given in eq. (2.40).

$$\begin{aligned} \text{minimise} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{2.40}$$

in which the above-mentioned uncertainty is present in the objective function matrix $c \in \mathbb{R}^n$, the constraint matrix $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Robust optimisation solves this linear program, in addition to handling deterministic uncertainty in view of an uncertainty set. Instead of

solving the linear program in eq. (2.40), robust optimisation proposes the robust linear program [61] given in eq. (2.41),

$$\begin{aligned} &\text{minimise} && c^T x \\ &\text{subject to} && Ax \leq b \quad \forall (c, A, b) \in \mathcal{U} \end{aligned} \tag{2.41}$$

where $\mathcal{U}$ denotes the uncertainty set that must be specified. There is a trade-off between the size of the uncertainty set and computational effort. Examples of uncertainty sets are polyhedral sets [60], ℓ_p-ball sets [62] and box sets [61] ‡.

Nonlinear robust optimisation

We have seen one particular class of robust optimisation, namely the robust linear program. In this section, we will look at a nonlinear robust optimisation (NRO) problem, which is becoming more and more important in real-world applications. The material in this section is based on the survey from [63]. A few examples of real-world applications presented in the survey are airfoil design and electrical engine design.

The NRO problem is given in eq. (2.42),

$$\begin{aligned} &\underset{x \in \mathcal{X}}{\text{minimise}} && f(x) \\ &\text{subject to} && c(x; u) \leq 0 \quad \forall (d) \in \mathcal{U}(x) \end{aligned} \tag{2.42}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is some nonlinear function, $x \in \mathbb{R}^n$ denotes the decision variables, $\mathcal{X}$ is the feasible set, $d \in \mathbb{R}^m$ are uncertainty parameters and $\mathcal{U}(x)$ is the uncertainty set. A few examples of uncertainty set has been given in Section 2.3.4.

A special case of the NRO problem given in eq. (2.42) is a bilevel optimisation problem (min-max problem) that arise when implementation errors ought to be conidered. These types of problems may be expressed as given in eq. (2.43).

$$\underset{x \in \mathcal{X}}{\text{minimise}} \max_{d \in \mathcal{U}(x)} \quad f(x + d) \tag{2.43}$$

where x, $\mathcal{X}$, d and $\mathcal{U}(x)$ are defined equally as for eq. (2.42). The uncertainty set for this particular formulation is in [63] defined as $\mathcal{U} = \{d \in \mathbb{R}^n | \left\| d \right\|_2 \leq \eta\}$, where $\eta \geq 0 \in \mathbb{R}$ is a scalar that adjusts the magnitude of the perturbations we want to be robust against.

2.3.5 Robust optimisation and regularisation synergy †

Certain regularised problems, such as the one discussed in Section 2.3.3, are as a matter of fact equivalent to robust linear optimisation problems [60]. This means that many regularised problems have an inherent "hidden robustness" which opens up the possibility to design new optimisation objective functions and consequently algorithms that also consider robustness to data uncertainty.

Among the the first exploration of the inherent "hidden robustness" in regularised problems is found in [64]. They show that the ridge regression[6] in eq. (2.44),

$$\min_{\theta} \left\| y - \theta^T x \right\|_2^2 + \lambda \left\| \theta \right\|_2^2 \tag{2.44}$$

is equivalent to the robust optimisation formulation given in eq. (2.45),

$$\min_{\theta} \max_{d:\|\mathscr{U}\|_F \leq \lambda} \left\| y - \theta^T (x+d) \right\|_2^2 \tag{2.45}$$

where $\mathscr{U}(x)$, x, d are defined as in Section 2.3.4, λ have been dicussed in Section 2.3.3, $\theta \in \mathbb{R}$ denotes the model parameters, $\|\cdot\|_F$ is the frobenius norm [25, Chapter 2] and $y \in \mathbb{R}$ is the target value.

They argue in [60] that the computational cost of eq. (2.45) is comparable to eq. (2.44). Some of the perks of formulating a ridge regression as a robust optimisation is that by varying the uncertainty set, $\mathscr{U}(x)$, one may produce a large class of regularisation-like training procedures, some of which are presented and discussed in [60] ‡.

2.4 Persistency of excitation and robustness †

Persistency of excitation is an important condition in system identification and adaptive control. A definition of the condition is given in [65].

Definition 2.4.1. *Persistency of excitation*
A vector $\psi : \mathbb{R}_{\geq 0} \to \mathbb{R}^{2n}$ *is persistently exciting (PE) if* $\exists\ \alpha_1, \alpha_2$ *and* $\delta > 0$ *such that* $\forall\ t_0 \geq 0$

$$\alpha_1 I \leq \int_{t_0}^{t_0+\delta} w(t) w^{\mathsf{T}}(t) dt \leq \alpha_2 I$$

The concept of persistency of excitation arises in systems where an input signal, or an action of some sort, excites parameters in a system. If the excitement of the parameters decay too rapidly, an online learning algorithm such as the stochastic gradient descent, may fail to estimate the parameters correctly due to not receiving the necessary amount of information [2]. For system identification and adaptive control, the importance of this condition is prominent in order to identify the correct parameters. Estimating good parameters is dependent on the "richness" of the input signal, ensuring that all modes of the system are excited. Put in other words, the input signal must contain enough information about the system we desire to estimate. More noticeably, in view of neural network, is its connection to stochastic gradient methods. Assume the stochastic gradient descent algorithm in Section 2.2.6 is to be used in estimating the unknown parameters θ of a convex function $f(\theta)$ and that for some reason, the function is attenuated by $\delta \in (0,1)$, $f(\theta) \leftarrow \delta f(\theta)$. If δ decays too rapidly, stochastic gradient methods will not necessarily converge to the optimal θ [2] ‡.

[6] Ridge regression is also known as Tikhonov regularization or as ℓ_2 regularisation with mean square error.

2.4.1 Persistency of excitation in neural networks †

In this section, persistency of excitation in view of neural networks is discussed. A motivation for why persistency of excitation is difficult to satisfy for deep FNN is provided. The motivation is then extended to RNNs.

Consider a L-layer deep FNN with input $x \in \mathbb{R}^n$ given in eq.

$$f(x;\theta) = (f_L \circ f_{L-1} \circ \cdots \circ f_1)(x) = f_L(f_{L-1}(f_{L-2}(\cdots(f_1(x)))) \tag{2.46}$$

where f_i, $i \in [1,L]$, represents the i-th layer operation of the network with corresponding parameters θ_i. The network is trained using a stochastic gradient method (see Section 2.2.6). In [2], they take the stance there are no guarantees that the convergence of the standard training yield the optimal parameters, θ. As a consequence, the network may not be robust with regard to unseen data. A motivation for the abovementioned claim is in [2] based on a two-layer FNN with parameters θ_i, $i \in [1,2]$, and the activation function ReLU,

$$f(x;\theta) = \theta_2^T \phi(\theta_1 x)$$

where $\phi(x) = x^+ = \max(0,x)$. The following theorem from [2] provides a foundation for persistency of excitation in view of neural networks.

Theorem 2.4.1. *Given the function to be estimated $f : \mathbb{R}^n \to \mathbb{R}^m$, a data set of size $|D|$, a set of points $\{x_i\}_{i\in D}$ and some number of hidden nodes (o) in the intermediate layers. Assume the neural network with one hidden layer and ReLU as activation function, $f(x;\theta,b)$, with parameters $\theta_1 \in \mathbb{R}^{o\times m}$, $\theta_2 \in \mathbb{R}^{n\times o}$, $b \in \mathbb{R}^o$ is trained by solving the optimisation problem in (2.47),*

$$\min_{\theta_1,\theta_2,b} \frac{1}{2} \sum_{i\in D} \left\| f(x_i) - \theta_2^T \phi(\theta_1 x) \right\|_2^2 \tag{2.47}$$

with a stochastic gradient descent algorithm from Section 2.2.6. Moreover, assume the training converges from a random initialisation to some stationary point $(\tilde{\theta}_1, \tilde{\theta}_2, \tilde{b})$ with a static learning rate, α. Let $(\tilde{\theta}_1)_j$ and $(\tilde{b})_j$ denote the j-th rows of $\tilde{\theta}_1$ and $\tilde{b}$ at equilibrium and define the set of training points activating the j-th node in the hidden layer as,

$$D_j = \{i \in D : (\tilde{\theta}_1)_j x_i + \tilde{b}_j > 0\}$$

Given all these assumptions, the Lipschitz constant of the estimated function $\tilde{f}(x;\tilde{\theta},\tilde{b})$ is almost surely upper bounded by,

$$L\left(\alpha,\bar{\mu},\underline{\lambda},\tilde{b},n_{active}^{max}\right) := n_{active}^{max}\left(\sqrt{\frac{2}{\alpha\underline{\lambda}}}\right)\left(\frac{2\bar{\mu}\left\|\tilde{b}\right\|_\infty}{\underline{\lambda}} + \sqrt{\frac{1}{\underline{\lambda}}\left|\frac{2}{\alpha} - \left\|\tilde{b}\right\|_\infty^2\right|}\right) \tag{2.48}$$

in which $n_{active}^{max} = \max_{x\in\mathbb{R}^n} \sum_{j=1}^{o} 1 \forall\, j \in D_j$ represents the maximum number of nodes in the hidden-layer that an input point may activate, $\underline{\lambda} = \min_{j\in[o]} \lambda_{\min}\left(\sum_{i\in D_j} x_i x_i^\intercal\right)$ represents a lower bound for the smallest eigenvalue of the covariance matrix of the input points activating the same artificial neuron in the hidden-layer, and $\bar{\mu} = \max_{j\in[o]} \left\|\sum_{i\in D_j} x_i\right\|_2$ is a upper bound on the ℓ_2-norm.

Proof. See [2, Appendix A] ■

If the training converges, the theorem above provides an Lipschitz upper bound for the function estimate, and as such the network, from its input to its output. The Lipschitz bound is dependent on several factors as seen in eq. (2.48) and described in Theorem 2.4.1. A key condition is that every neuron in the hidden layer ought to be excited from every dimension, i.e. the training data activating hidden-layer neurons must to have full rank. This will in turn provide persistency of excitation for the parameters. The issue is that these are conditions that are very difficult to satisfy, and likely violated for two reasons: a lot of data is low dimension by nature, for example audio and raw images. The second caveat is that even if the input data is of full rank, it has been shown that stochastic gradient descent algorithms generate low-rank signals in the hidden layers of the network [66]. Consequently, persistency of excitation is difficult to satisfy for DNNs by merely adding more data. New techniques are likely necessary in order to attempt to persistently excite the network parameters and provide more robust parameter estimates ‡.

2.4.2 Persistency of excitation and recurrent neural networks

Theorem 2.4.1 provide sufficient conditions for upper-bounding the Lipschitz constant of a 2-layer FNN. RNNs are a special class of neural networks that work on sequences of data. As discussed in Section 2.2.4, they consists of multiple FNNs for each sequence element. For example, the LSTM module employ four FNNs for each sequence element, indicated by the colour boxes in Figure 2.10. As such, by the same argumentation of Section 2.4.1, it is likely to be just as difficult to satisfy the conditions given in 2.4.1 for RNNs and subsequently LSTM neural networks.

2.4.3 A loss objective for persistently exciting neural network parameters †

In Section 2.4.1 we discuss why persistent excitation is difficult to satisfy for *deep* neural networks. Tools and techniques for introducing robustness in neural networks are presented in sections 2.3.3 and 2.3.4, and more importantly, the equivalence between ℓ_2 norm penalty regularisation and a robust optimisation formulation. This equivalence is restricted to the squared ℓ_2-norm. One of the training procedures in focus in this work, presented firstly in [2], extends the equivalence to a larger class of norm regularisation terms. It is aimed at inducing perturbations into the neural network during training. The hypothesis is that by doing so, this will give the trainable parameters in the deeper layers persistent excitation when trained with stochastic gradient descent based algorithms, and consequently, increase the neural network robustness.

Given inputs, $x \in \mathbb{R}^n$ and the corresponding target values $y \in \mathbb{R}^n$, consider the two optimisation problems given in eq. (2.49) and eq. (2.50),

$$\min_{\theta} \quad (y - \theta^\top x)^2 + \lambda \left\| \theta \right\|_p^m \tag{2.49}$$

$$\min_{\theta} \quad \frac{1}{2}\left(y - \min_{d:\|d\|_q \leq \eta} \theta^\top (x+d) \right)^2 + \frac{1}{2}\left(y - \max_{d:\|d\|_q \leq \eta} \theta^\top (x+d) \right)^2 \tag{2.50}$$

where $\|\cdot\|_p$ and $\|\cdot\|_q$ are dual norms, $m \in [1, \infty)$ denotes a constant and $\lambda \in \mathbb{R}_+$ and $\eta \in \mathbb{R}_+$ are some positive scalars. In [2] it is shown that $\forall \lambda$, there exists a η such that the solutions of the two optimisation problems are equal. Likewise, $\forall \eta$, there exists a λ such that the solutions of the two optimisation problems are equal.

The latter formulation, given in eq. (2.50), seeks to ensure that the ℓ_q-norm constrained points around some training point, x_i (the ith element of the vector), are assigned a value close to the corresponding target, y_i, of the training point. As such, the problem is implicitly solved with a richer data set. The input of the linear mapping, x, is perturbed by a disturbance, d, such that the output of the forward pass is minimised and maximised. This may stimulate persistency of excitation of the parameters of the model [2].

This is one of the training procedure that will be in main focus in the coming sections. It will be discussed and adapted in view of the LSTM neural network in Chapter 3. Algorithm 1 may be updated to account for this change in the general training procedure. An updated version is given in Algorithm 2. We now require a uncertainty set, $\mathscr{U}$. Examples of different uncertainty sets are presented in Section 2.3.4. As indicated by eq. (2.50), the uncertainty set in the loss function formulation is some l_q-norm ball. The disturbances d in eq. (2.50), constrained by a uncertainty set, ought to be found for each layer (line 3-4 in Algorithm 2). Note that the subscripts are added to distinguish which part of the objective function that is considered. The perturbations are optimised in the same manner as the regular parameters. Firstly, forward pass the addition of the input and initial disturbance. Secondly, calculate the inner-term of the loss objective in eq. (2.50) (i.e. the term $\theta^\top(x+d)$). Thirdly, apply backpropagation on this term with respect to the disturbance parameters (see Section 2.2.6). Lastly update the disturbances parameters ‡.

2.5 Nonlinear systems and stability

This section introduce stability concepts in a nonlinear system setting. The two main sources for this section are [67] and [68], and we refer to these sources for in-depth coverage of the topics presented here. Section 2.5.1 introduce nonlinear state-space representations. In Section 2.5.2 a technique called spectral normalisation will be briefly covered. Lastly, Section 2.5.3 introduce the concept of input-to-state stability in the discrete case.

Algorithm 2 A neural network training procedure for persistently exciting parameters †

Require: Stopping criteria $\epsilon > 0$, cost function L, optimiser O, training data D, neural network $f_{(\theta;d)}$ with parameters θ and d, , set of allowable perturbations $\mathscr{U}$

for epoch $< \epsilon$ **do**
  for (input,label) $\in D$ **do**
    Estimate $d_{min} \in \mathscr{U}$ and update d in $f_{(\theta;d)}$
    Estimate $d_{max} \in \mathscr{U}$ and update d in $f_{(\theta;d)}$
    Calculate prediction $f_{\theta;d}$(input) ▷ Forward pass
    Calculate loss C(prediction, label)
    Calculate gradients of loss w.r.t. parameters $\nabla_\theta L$ ▷ Backward pass
    Update parameters $O(\nabla_\theta L)$
  end for
  epoch ← epoch + 1
end for

2.5.1 Nonlinear systems and state-space representations

A nonlinear dynamic system is often modeled by a finite number of coupled first-order ordinary differential equations [67], given in eq. (2.51),

$$\begin{aligned}
\dot{x}_1 &= f_1(t, x_1, x_2, \cdots, x_n, u_1, u_2, \cdots, u_m) \\
\dot{x}_2 &= f_2(t, x_1, x_2, \cdots, x_n, u_1, u_2, \cdots, u_m) \\
&\vdots \\
\dot{x}_n &= f_n(t, x_1, x_2, \cdots, x_n, u_1, u_2, \cdots, u_m)
\end{aligned} \tag{2.51}$$

where $\dot{x}_i$ denotes the derivative of the *state variable* x_i with respect to *time variable* t, and $u_i \forall\, i \in [1, m]$ are given *input variables*. The state variables are in some ways the memory of the past of the dynamic system. A more compact formulation of the dynamic system in eq. (2.51) is given in eq. (2.52),

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix}, \qquad u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_m \end{pmatrix}, \qquad f(t,x,u) = \begin{pmatrix} f_1(t,x,u) \\ f_2(t,x,u) \\ \vdots \\ \vdots \\ f_n(t,x,u) \end{pmatrix} \tag{2.52}$$

from which we may express the n ordinary differential equations in the n-dimensional vector differential equation [67, Chapter 1], given in (2.53),

$$\dot{x} = f(t, x, u) \tag{2.53}$$

where $f \in \mathbb{R}^n$ is a continuous nonlinear function, $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$. In the sense of system theory and state-space representations, there is usually associated an output equation with

the state differential equation given in eq. (2.53) [69]. The observation equation is included in eq. (2.54),

$$y = h(t, x, u) \tag{2.54}$$

in which $y(t) \in \mathbb{R}^q, 1 \leq q \leq n$ denotes the output of the system and h is a continuous nonlinear function. The output vector include variables that are of interest in analysing the dynamic system (often in the sense of nonlinear *control* systems), for example variables that may be measured [69]. The representation in eq. (2.53) and eq. (2.54) are in the literature often referred to as *state space model* [67].

The representation above is given for continuous time state-space representations. We will view the LSTM neural network as discrete-time nonlinear systems, as a result of the discrete nature of the sequence elements. The two equations eq. (2.53) and eq. (2.54) may be expressed in the discrete-time as given in eq. (2.55) and eq. (2.56),

$$x_{k+1} = f(k, x_k, u_k) \tag{2.55}$$

$$y_k = h(k, x_k, u_k) \tag{2.56}$$

where $k \in \mathbb{Z}_+$ is an integer representing the discrete-time step.

2.5.2 Spectral normalisation

Spectral normalisation is a technique introduced in [70] for stabilising the training of a special class of neural networks called generative adversarial networks [71]. As the name suggests, it scales the network weight matrix, W, in a neural network by a factor of the spectral norm, also known as the largest singular value of a matrix (see Section 2.1.2), $\sigma(W)$, as expressed in eq. (2.57),

$$W_{SN} = \frac{W}{\sigma(W)} \tag{2.57}$$

By scaling each layer, [70] shows that the Lipschitz norm of the generative neural network as a whole is bounded from above by 1 by making use of the composite nature of a FNN architecture. This fact has been used in providing stability guarantees for controllers acting on dynamics estimated by FNNs [72].

The concept of using spectral normalisation as an aid to bound the Lipschitz norm of a neural network by some upper bound is not directly transferable to LSTM neural networks. They do not posses the same composite nature as FNNs. The spectral normalisation technique may still be applied in bounding the weight matrices of a LSTM neural networks, but it is not apparent what kind of bound one achieves on the LSTM neural network.

Nonetheless, bounding the parameters is an important concept in stabilising neural networks, and RNNs are no exception. In [73], bounds on the LSTM gates are presented for ensuring the network is stable (no exploding or vanishing gradients).

2.5.3 Input-to-state stability

In this section, *discrete* input-to-state stability [74] will be introduced. This stability paradigm will be used to analyse the stability of a persistently-exciting inspired LSTM neural network. Important definitions will be treated in the first part of the section, before the definition of input-to-state stability is presented. The three main references for this section are [74], [67] and particularly [68].

Stability theory is of high importance in control systems engineering and remains an essential requirement for cost- and safety critical applications. As pointed out in [74], there are two main approaches to stability, broadly speaking. The first is the state-space approach associated with Lyapunov analysis (see [67, Chapter 4]. The second is the operator approach, also known as the input-output stability (see [67, Chapter 5]). ISS seeks to merge the two aforementioned views of stability. It is a stability concept that is concerned with a system's robustness to disturbances.

In order to define input-to-state stability, a few concepts ought to be explored. The comparison functions, class $\mathcal{K}$, class $\mathcal{K}_\infty$ and class $\mathcal{KL}$ are continuous functions that are used in stability theory to characterise stability properties. They appear frequently in definitions of ISS. The three relevant comparison functions are defined in Definition 2.5.1 and Definition 2.5.2, respectively.

Definition 2.5.1 ($\mathcal{K}$ and $\mathcal{K}_\infty$ functions [67, Chapter 4])**.** *A class $\mathcal{K}$ (kappa) function is a continous function $\alpha : [0, a) \to [0, \infty)$ and belongs to class $\mathcal{K}$ if:*

$$\begin{cases} \alpha(0) = 0 \\ \alpha(r) \text{ is strictly increasing, i.e. } \quad \frac{\partial \alpha}{\partial r} > 0 \quad \forall r \end{cases} \tag{2.58}$$

If in addition $a \to \infty$ and $\alpha(r) \to \infty$ as $r \to \infty$, then α is a class $\mathcal{K}_\infty$ function.

Definition 2.5.2 ($\mathcal{KL}$ function [67, Chapter 4])**.** *A class $\mathcal{KL}$ (kappa-ell) function is a continous function $\beta : [0, a) \times [0, \infty) \to [0, \infty)$ and belongs to class $\mathcal{KL}$ if:*

for each fixed s,

- *$\beta(r, s)$ is a class $\mathcal{K}$ function with respect to r.*

for each fixed r

- *$\beta(r, s)$ decreasing with respect to s.*
- *$\beta(r, s) \to$ as $s \to \infty$.*

With the definitions of the comparison functions, the concept of ISS is defined in Definition 2.5.3.

Definition 2.5.3 (ISS [68])**.** *The system described by eq.* (2.55)-(2.56) *is input-to-state (ISS) stable if* $\exists$ *a class* $\mathcal{KL}$ *function,* β*, and a class* $\mathcal{K}$ *function,* γ*, such that for each bounded input* $u \in \ell_\infty^m$*, any initial state* $x_0 \in \mathbb{R}^n$ *and* $\forall\, k \in \mathbb{Z}_+$*, the following holds,*

$$\left\| x\left(k, x_0, u\right) \right\|_2 \leq \beta\left(\left\| x_0 \right\|_2, k \right) + \gamma\left(\| u \|_\infty \right) \tag{2.59}$$

Lyapunov functions are frequently used in studying stability properties of nonlinear dynamic systems. The definition of an input-to-state Lyapunov function is given in (2.5.4).

Definition 2.5.4 (ISS-Lyapunov function [68])**.** *Let* $V : \mathbb{R}^n \to \mathbb{R}_+$ *be a continuously differentiable function such that the following holds:*

$$\alpha_1\left(\left\| x_0 \right\|_2 \right) \leq V(x_0) \leq \alpha_2\left(\left\| x_0 \right\|_2 \right) \tag{2.60}$$

$$V(f(x_0, u)) - V(x_0) \leq -\alpha_3\left(\left\| x_0 \right\|_2 \right) + \sigma\left(\| u \|_2 \right) \tag{2.61}$$

$\forall\, x_0 \in \mathbb{R}^n$ *and* $\forall\, u \in \mathbb{R}^m$*, where* α_1 *and* α_2*,* α_3 *are class* $\mathcal{K}_\infty$ *functions and* σ *is a class* $\mathcal{K}$ *function. Then,* $V(\cdot)$ *is called an ISS-Lyapunov function for the system given in eq.* (2.55)-(2.56)*.*

Note that σ in Definition 2.5.4 is not the sigmoid function.

Theorem 2.5.1 from [68] propose a connection between an ISS-Lyapunov function and input-to-state stability.

Theorem 2.5.1 (ISS and Lyapunov synergy [68])**.** *Consider a system,* $\sum$*, on the form given by eq.* (2.55)-(2.56)*. If* $\sum$ *has a ISS-Lyapunov function (Definition 2.5.4), then* $\sum$ *is input-to-state stable (ISS).*

Chapter 3

Methodology and theoretical results

Chapter 3 will provide the main theoretical contributions with regard to both robustness and stability in view of the LSTM neural network, presented in Section 2.2.4. The robustness and stability analyses are founded in mathematical optimisation and dynamic control system theory, respectively. The robustness analysis in Section 3.1 will firstly alter the familiar LSTM equations from Section 2.2.4 to account for different types of training procedures, motivated by the concept of persistency of excitation. Two training procedures inspired by persistency of excitation will be explored. The first training procedure, option 1, is concerned with the formulation from [2] as a means to enrichen the input signal. The second training procedure, option 2, utilises the nonlinear robust optimisation problem from Section 2.3.4 in perturbing the input signal, attempting to enriching it. The stability analysis in Section 3.2 is concerned with providing formal bounds on the parameters in order to say something about the stability of the RNN. In particular, the stability properties of the altered LSTM are analysed by using the concept of ISS, inspired by the work of [3]. A persistently exciting-inspired LSTM neural network is expressed as a state-space model, for which certain parameter constraints are presented to ensure that the state-space model is input-to-state stable.

3.1 Robustness in view of Persistency Of Excitation

An important topic in machine learning is to produce robust models. A robust model is a model that generalises well to input the model is not trained on. In this section, two training procedures for aiding in persistently exciting the parameters of some neural network will be interpreted and adapted for the LSTM neural network (see Section 2.2.4 for an introduction to the standard LSTM neural network). The concept of persistency of excitation is described in Section 2.4.

In order to apply the training procedures that will be presented, the standard LSTM described by eq. (2.17)-(2.22) ought to be altered. We add so-called perturbations, whose in-

tention is to enrichen the input signal x. The altered LSTM is given in eq. (3.1)-(3.6),

$$i_{k_{\text{PE}}} = \sigma\left(U_i(x_k + d_{u_k}) + W_i(h_{k-1_{\text{PE}}} + d_{w_k}) + b_i\right) \tag{3.1}$$
$$f_{k_{\text{PE}}} = \sigma\left(U_f(x_k + d_{u_k}) + W_f(h_{k-1_{\text{PE}}} + d_{w_k}) + b_f\right) \tag{3.2}$$
$$o_{k_{\text{PE}}} = \sigma\left(U_o(x_k + d_{u_k}) + W_o(h_{k-1_{\text{PE}}} + d_{w_k}) + b_o\right) \tag{3.3}$$
$$g_{k_{\text{PE}}} = \tanh\left(U_g(x_k + d_{u_k}) + W_g(h_{k-1_{\text{PE}}} + d_{w_k}) + b_g\right) \tag{3.4}$$
$$c_{k_{\text{PE}}} = f_{k_{\text{PE}}} \odot c_{k-1_{\text{PE}}} + i_{k_{\text{PE}}} \odot g_{k_{\text{PE}}} \tag{3.5}$$
$$h_{k_{\text{PE}}} = \tanh(c_{k_{\text{PE}}}) \odot o_{k_{\text{PE}}} \tag{3.6}$$

where $U_j, W_j, b_j,\ j = \{i, f, o, g\}$ denote the parameter matrices of the respective gates. The four gates and the two states, $c_{k_{\text{PE}}}$ and $h_{k_{\text{PE}}}$ are described in detail in Section 2.2.4. Compared to the regular LSTM, we now add parameters, $d_{u_k} \in \mathbb{R}^{F\times B}$ and $d_{w_k} \in \mathbb{R}^{H\times B}$, to the input of the LSTM neural network. The disturbances d_{u_k} are used to excite the input parameters (U-matrices) for each time step, k, while the disturbances d_{w_k} are used to excite the hidden state parameters (W-matrices) for each time step, k. The equations in eq. (3.1)-(3.6) describe the altered LSTM workings of one element of the input sequence.

The training procedure for training networks on the form given in eq. (3.1)-(3.6) must now, in addition to optimising a cost function with respect to the parameters of the network, also optimise the cost function with respect to these disturbances, forming a bilevel optimisation[1] problem. We name this network architecture "PE LSTM", indicating that it is a LSTM neural network model with parameters for applying the persistently exciting-principles discussed in Section 2.4. Note that the subscript PE is included to distinguish the representation from the regular LSTM neural network from Section 2.2.4.

As discussed in section 2.2.5, increasing depth may be beneficial for a neural network's generalisation capabilities. Assume the input, $x \in \mathbb{R}^S$, is of sequence length S such that for all vector elements, $x = \begin{bmatrix} x_1 & x_2 & \cdots & x_S \end{bmatrix}^T$. When using deep LSTMs, the hidden state sequence of the shallowest LSTM neural network layer, i.e. $h^{(0)} = \begin{bmatrix} h_1 & h_2 & \cdots & h_S \end{bmatrix}^T$, is fed to the deeper LSTM layer as new input. The superscript denotes the specific PE LSTM layer. A L-layer deep PE LSTM-based neural network, $f^{(L)}$, is presented in eq. (3.7)-(3.9),

$$h^{(0)} = \text{PE-LSTM}^{(0)}(x, d_u^{(0)}, d_w^0; U^{(0)}, W^{(0)}, b^{(0)}) \tag{3.7}$$
$$h^{(l)} = \text{PE-LSTM}^{(l)}(h^{(l-1)}, d_u^{(l)}, d_w^{(l)}; U^{(l)}, W^{(l)}, b^{(l)}), \quad l = 1, 2, 3, .., L-1 \tag{3.8}$$
$$f^{(L)} = \text{FEEDFORWARD}(h^{L-1}; \theta) \tag{3.9}$$

where $f^{(L)} : \mathbb{R}^S \to \mathbb{R}$ is a FNN (the last layer of the overall neural network) with parameters $\theta \in \mathbb{R}^S$. A RNN works on a sequence of elements and outputs a S-sequence long vector. The feedforward neural network's sole role is to map the sequence to a real-valued scalar. Note

[1] A bilevel optimisation problem is a special kind of optimisation in which one problem is embedded into another.

that the formulation abstracts the inner-workings of the PE LSTM module (eq. (3.1)-(3.6)) on each sequence by the term PE-LSTM. Moreover, FEEDFORWARD, abstracts the inner-workings of a FNN.

A multilayer LSTM neural network may present itself as a confusing construct due to the unrolling that RNNs undergo in each layer, in combination with stacked LSTM neural networks in deeper layers. The concept of unrolling is treated in detail in Section 2.2.4. Figure 3.1 is provided to visualise a L-layer deep LSTM neural network. The left part of the figure captures the unrolled abstraction of the neural network, similar to the one described by eq. (3.7)-(3.8). Note that in the figure, a regular LSTM neural network is visualised, but the principle is identical for the PE LSTM neural network. As indicated by the figure, a multilayer LSTM neural network is multiple LSTM modules stacked on top of each other, where each hidden state element from the previous layer act as input to the corresponding LSTM neural network in the deeper layers. The internals of the LSTM-blocks are visualised in Figure 2.10. In a regression setting, the sequence output from the deepest layer, $h_0^{(L)}, h_1^{(L)}, \cdots, h_S^{(L)}$, is passed to a FNN for the above-mentioned mapping.

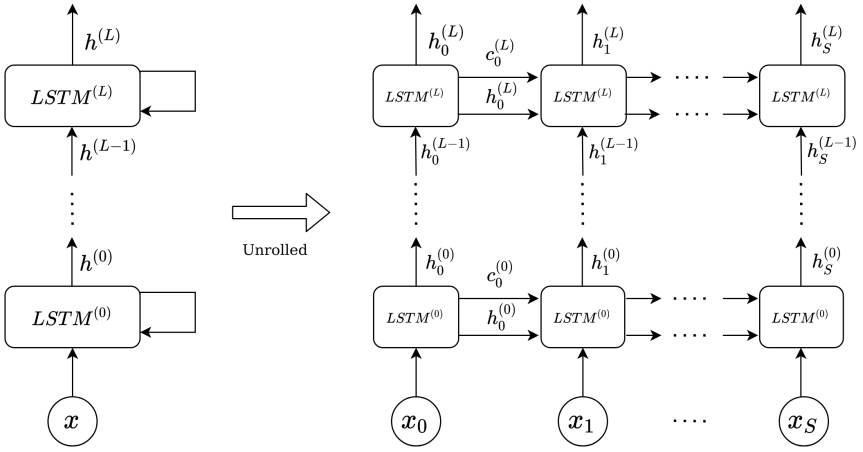

Figure 3.1: A L-layer deep LSTM neural network. The superscripts denote the specific layers that the LSTM modules belong to. The subscripts denote the sequence element. (Left) A compact computational graph of the multilayer LSTM neural network. (Right) The corresponding unrolled LSTM neural network.

In Section 3.1.1 and Section 3.1.2, two training procedures acting on the PE LSTM-based neural network given in eq. (3.7)-(3.9) will be presented.

3.1.1 Persistency of excitation of LSTM: option 1

A loss objective for persistently exciting parameters of a linear regression objective is presented in Section 2.4.3 based on the work of [2]. This is based on reinterpreting norm regularisation (eq. (2.49)) as a way to enrichen the input signals of regression models (eq. (2.50)). This shows that if one for each training point perturb the input of a linear mapping such that the mapping's output is both maximised and minimised within some uncertainty set,

one may attempt to persistently excite the parameters of the model. The technique is applied to CNNs in [2], where it was firstly introduced, and reinterpreted in a regression setting for FNNs in [12]. Analogously, by perturbing the intermediate LSTM linear layers, that is, the affine operations $\theta^T x$, where $\theta \in \mathbb{R}^n$ are the network parameters and $x \in \mathbb{R}^n$ is the input, one may apply the training objective and potentially persistently excite the parameters of a *recurrent* neural network (LSTM).

We recall from the standard LSTM equations that the four gates in the module consist of the aforementioned affine operation, before passing the result to the nonlinear activation functions. This is utilised when we add the so-called perturbations, whose role is to enrichen the input signals over all layers. The altered LSTM neural network is given in eq. (3.1)-(3.6). We will now look at how those perturbations may be generated.

The optimisation objective given in eq. (2.50) provides a way to generate the inserted perturbations. The objective is adapted to recurrent LSTM networks and presented in eq. (3.10),

$$\min_{\theta} \quad \frac{1}{2}\left(y - \min_{d_u, d_w \in \mathscr{U}} f^{(L)}(\theta; h^{L-1})\right)^2 + \frac{1}{2}\left(y - \max_{d_u, d_w \in \mathscr{U}} f^{(L)}(\theta; h^{L-1})\right)^2 \tag{3.10}$$

where $f^{(L)}(\theta; h^{(L-1)})$ and $h^{(L-1)}$ are given by the neural network described in eq. (3.7)-(3.9) and $y \in \mathbb{R}$ represents some target value. Moreover, $\mathscr{U}$ denotes the uncertainty set, described in Section 2.3.4. The uncertainty set used in this thesis will be the ℓ_∞ ball with radius $\eta \in \mathbb{R}_+$. The radius is a tunable hyperparameter. We denote the models produced by this training procedure PE LSTM Opt-1, indicating that it is the option 1 of persistently exciting LSTM.

The procedure from [2] is proved for a linear regression problem and applied to a neural network with perturbation parameters, similar to (3.7)-(3.9). In this work we study the LSTM neural network, which employ an additional state, h_k, with corresponding weight matrices W_j for $j = \{i, f, o, g\}$. In Section 2.4.3, the equivalence of the regularised objective and persistently exciting objective utilised in eq. (3.10), is introduced for the input x and weight matrix U_j for $j = \{i, f, o, g\}$. For RNNs, one may also norm regularise the recurrent weight matrix W_j for $j = \{i, f, o, g\}$, as given in eq. (2.39). The proof in [2] is based on dual norm theory and does not consider perturbing the hidden state as the work is only concerned with FNNs. It is possible to extend the equivalence given in eq. (2.49) and eq. (2.50) for the hidden state and the recurrent weight matrix as is done in the original proof of [2]. We then use Wh_{k-1} and $\|W\|_p$ instead of Ux_k and $\|U\|_p$. In this thesis, however, we will assume $d_{w_k} = 0$. In Section 2.3.3 it is discussed how regularisation of the *recurrent weights*, W, may prevent a RNN from exhibiting long term memory traces. We will therefore refrain from regularising the recurrent weights. By the equivalence of the regularised objective in eq. (2.49) and the persistently exciting objective in eq. (2.50), $d_{w_k} = 0$ follows. It is, however, included in the PE LSTM equations eq. (3.1)-(3.6) as it is possible to perturb also the hidden state in light of that the recurrent weight matrices may be regularised. This also leaves the option open for experimenting with perturbing the hidden state.

3.1.2 Persistency of excitation of LSTM: option 2

The training procedure for enrichening the input signals of a LSTM neural network studied in Section 3.1.1 is based on theory from Section 2.4.3. In this section, an alternative training procedure based on a NRO problem will be explored. In Section 2.3.4, a robust nonlinear objective was presented in eq. (2.43). This special case of the general NRO problem seeks to encapsulate implementation errors and seeks to minimise an objective under worst case implementation errors. We take the viewpoint that by adding these perturbations, the input signal is enriched, and as such may excite the signals passing through the network.

The NRO objective is adapted to work on the PE LSTM neural network described by eq. (3.1)-(3.6), and the multilayer PE LSTM described in eq. (3.7)-(3.9). The training objective is given in eq. (3.11),

$$\min_{\theta} \max_{d_u, d_w \in \mathscr{U}} \ell(f^{(L)}(\theta; h^{L-1}), y) = \min_{\theta} \max_{d_u, d_w \in \mathscr{U}} \left(y - f^{(L)}(\theta; h^{L-1}) \right)^2 \tag{3.11}$$

where $f^{(L)}(\theta; h^{L-1})$ and h^{L-1} is given by the neural network described in eq. (3.7)-(3.9) and $y \in \mathbb{R}$ represents some target value. Just as for the training procedure in Section 3.1.1, $\mathscr{U}$ denotes the uncertainty set. The uncertainty set used in this thesis will be the ℓ_∞ ball with radius $\eta \in \mathbb{R}_+$, a new tunable hyperparameter. This training procedure has some resemblances of the training procedure presented in Section 3.1.1. They are both bilevel optimisation problems, i.e. an optimisation problem embedded into another. Moreover, both are inspired by persistency of excitation, which is in essence an expression of providing system identification models rich enough input signals. The first training procedure, however, is derived from dual norm theory. The training procedure in eq. (3.11) is at its core a robust optimisation problem re-framed in neural network setting. It attempts to find parameters under worst-case disturbances by perturbing the input to each layer during training.

3.2 Stability of the long short-term memory network

This section will treat the stability of the PE LSTM neural network in view of nonlinear system theory. The two concepts stability and robustness are tied together. A robust neural network is generally robust to perturbations in input. A stable neural network give us confidence that it is well-behaved during training. Moreover, proving that the estimated dynamics are stable is important if the dynamics estimated display stability properties [3], ensuring that the effect of perturbations in initial conditions vanishes. This section explores the PE LSTM neural network from a stability perspective. The PE LSTM neural network is firstly expressed as a state-space representation in Section 3.2.1. Constraints on parameter matrices (both kernel and recurrent) will in turn be presented as a way to ensure ISS for the PE LSTM neural network.

The inspiration for this section stems from [3], discussed in Section 1.1. The main difference is that the analysis in [3] is done on a regular LSTM representation. The analysis in

this section is concerned with the persistently exciting-inspired LSTM representation, firstly introduced in eq. (3.1)-(3.6).

Note that a change of notation will be adapted in order to accompany the notation of the nonlinear system theory and avoid confusion when utilising theorems, definitions and lemmas from Section 2.5. Note that this is not the mainstream notation in machine learning research, but it is occasionally adapted when working in the cross-section of machine learning and nonlinear system theory. We now denote u_k the input of the PE LSTM neural network, instead of x_k. Moreover, x_k now denotes the system state, i.e. $x_{1_k} = c_k$ and $x_{2_k} = h_k$.

3.2.1 State-space form of the long short-term memory neural network

The ISS definitions and theorems in Section 2.5.3 are presented for nonlinear systems on state-space representation (see Section 2.5.1). In this section, the PE LSTM neural network described by eq. (3.1)-(3.3) will be written as a state-space representation. The PE LSTM neural network is included in eq. (3.12)-(3.17) due to its importance, in a slightly rewritten form,

$$i_{k_{\text{PE}}} = \sigma(U_i u_k + U_i d_{u_k} + W_i h_{k-1_{\text{PE}}} + W_i d_{w_k} + b_i) \tag{3.12}$$
$$f_{k_{\text{PE}}} = \sigma(U_f u_k + U_f d_{u_k} + W_f h_{k-1_{\text{PE}}} + W_f d_{w_k} + b_f) \tag{3.13}$$
$$o_{k_{\text{PE}}} = \sigma(U_o u_k + U_o d_{u_k} + W_o h_{k-1_{\text{PE}}} + W_o d_{w_k} + b_o) \tag{3.14}$$
$$g_{k_{\text{PE}}} = \tanh(U_g u_k + U_g d_{u_k} + W_g h_{k-1_{\text{PE}}} + W_g d_{w_k} + b_g) \tag{3.15}$$
$$c_{k_{\text{PE}}} = f_{k_{\text{PE}}} \odot c_{k-1_{\text{PE}}} + i_{k_{\text{PE}}} \odot g_{k_{\text{PE}}} \tag{3.16}$$
$$h_{k_{\text{PE}}} = \tanh(c_{k_{\text{PE}}}) \odot o_{k_{\text{PE}}} \tag{3.17}$$

where the different terms are described in Section 2.2.4 and Section 3.1. In order to achieve a discrete state-space representation ($x_{k+1} = f(x_k, u_k)$), we augment the initial condition of the dynamic system to the first element of the input (or some other values), and achieve that the input at time step k is aligned with the state at time step k. The state space vector of the PE LSTM neural network for an arbitrary time step, k, is defined in eq. (3.18).

$$x_k = \begin{bmatrix} c_{k_{\text{PE}}}^T & h_{k_{\text{PE}}}^T \end{bmatrix}^T \tag{3.18}$$

where $c_{k_{\text{PE}}} \in \mathbb{R}^H$ and $h_{k_{\text{PE}}} \in \mathbb{R}^H$ are the internal and hidden states of the LSTM RNN (see Section 2.2.4). The resulting dimension of the state vector in eq. (3.18) is $\mathbb{R}^{2H}$.

The input of the PE LSTM is given by $u_k \in \mathbb{R}^D$, $d_{u_k}\mathbb{R}^D$ and $d_{w_k}\mathbb{R}^H$, defined in eq. (3.19).

$$\bar{u}_k = \begin{bmatrix} {u_k}^T & d_{u_k}^T & d_{w_k}^T \end{bmatrix}^T \tag{3.19}$$

where the notation $\bar{u}_k$ is used to distinguish it from the original input signal. The terms D and H are described in Section 2.2.4 and represents the number of input features and the hidden size (number of artificial neurons), respectively. The resulting dimension of the state-space input is $\mathbb{R}^{2D+H}$.

Lastly we describe the output of the network as a whole. As discussed in eq. (3.1), we need a FNN to map the predicted sequence to a real-valued variable. The output of the PE LSTM neural network as a whole is given in eq. (3.20),

$$y_k = W_{out} \cdot h_{k_{\text{PE}}} \tag{3.20}$$

where $W_{out} \in \mathbb{R}^H$ denotes the FNN weight matrix.

The state vector consists of two elements. The compact formulation (see eq. (2.52)) of the system described by the PE LSTM equations is given in eq. (3.21) and eq. (3.22),

$$\begin{aligned} f_1 = c_{k+1_{\text{PE}}} &= f_{k_{\text{PE}}} \odot c_{k_{\text{PE}}} + i_{k_{\text{PE}}} \odot g_{k_{\text{PE}}} \\ &= \sigma(f_{f_k}(\bar{u}_k, h_k)) \odot c_{k_{\text{PE}}} + \sigma(f_{i_k}(\bar{u}_k, h_k)) \odot \tanh(f_{g_k}(\bar{u}_k, h_k)) \end{aligned} \tag{3.21}$$

$$\begin{aligned} f_2 = h_{k+1_{\text{PE}}} &= \tanh\left(c_{k+1_{\text{PE}}}\right) \odot o_{k_{\text{PE}}} \\ &= \tanh\left(c_{k+1_{\text{PE}}}\right) \odot \sigma(f_{o_k}(\bar{u}_k, h_k)) \end{aligned} \tag{3.22}$$

where $f_{j_k}(\bar{u}_k, h_k), j \in \{i, f, o, g\}$ are defined in eq. (3.23)-(3.26),

$$f_{f_k}(\bar{u}_k, h_k) = U_f u_k + U_f d_{u_k} + W_f h_{k_{\text{PE}}} + W_f d_{w_k} + b_f \tag{3.23}$$

$$f_{i_k}(\bar{u}_k, h_k) = U_i u_k + U_i d_{u_k} + W_i h_{k_{\text{PE}}} + W_i d_{w_k} + b_i \tag{3.24}$$

$$f_{g_k}(\bar{u}_k, h_k) = U_g u_k + U_g d_{u_k} + W_g h_{k_{\text{PE}}} + W_g d_{w_k} + b_g \tag{3.25}$$

$$f_{o_k}(\bar{u}_k, h_k) = U_o u_k + U_o d_{u_k} + W_o h_{k_{\text{PE}}} + W_o d_{w_k} + b_o \tag{3.26}$$

These are defined in order to ease the notation. In the remaining sections, the function argument $(\bar{u}_k, h_k)$ will not be repeated as to simplify notation and readability. We indicate that the functions are dependent on the state h_k and the input u_k by the subscript k.

We are now ready to present the state-space formulation of the PE LSTM neural network given in eq. (3.12)-(3.17). Given the state in eq. (3.18), the proposed input in eq. (3.19) and the output in eq. (3.20), the state-space representation of the PE LSTM is described by eq. (3.27)-(3.29),

$$x_{k+1} = f(k, x_k, \bar{u}_k) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{3.27}$$

$$= \begin{bmatrix} \sigma(f_{f_k}) \odot c_{k_{\text{PE}}} + \sigma(f_{i_k}) \odot \tanh(f_{g_k}) \\ \tanh\left[\sigma(f_{f_k}) \odot c_{k_{\text{PE}}} + \sigma(f_{i_k}) \odot \tanh\left(f_{g_k}\right)\right] \odot \sigma(f_{o_k}) \end{bmatrix} \tag{3.28}$$

$$y_k = h(k, x_k, \bar{u}_k) = W_{out} \cdot h_{k_{\text{PE}}} \tag{3.29}$$

3.2.2 Input-to-state stability analysis on a persistently exciting LSTM

In this section, constraints on the parameters of a PE LSTM neural network (see Section 3.1) for ensuring ISS will be presented. The main result is presented in Theorem 3.2.2.

In this work, we take a nonlinear dynamic system perspective. A PE LSTM neural network is represented as a state-space representation in Section 3.2.1. This enables us to utilise discrete ISS concepts from Section 2.5.3 as means to bound the parameter matrices. The result is derived by using an ISS-Lyapunov function, defined in Definition 2.5.4. Choosing a Lyapunov function candidate is not a matter of course and a difficult problem in itself. Inspired by the work of [3], a p-norm function is utilised. This makes the analysis substantially more manageable and convenient due to the properties possessed by norm functions, some of which presented in Section 2.1. Before preceding with the theorem, an assumptions and a lemma ought to be explored.

Assumption 3.2.1. *Let $\mathscr{S} \subset \mathbb{Z}_+$ define the set containing each integer in the sequence length. Assume that each element of $\bar{u}_k$, given in eq. (3.19), is in the range of $[-1,1] \forall k \in \mathscr{S}$.*

Assumption 3.2.1 is often satisified as a result of input saturation or input normalisation. The latter is especially relevant for neural networks. When training neural network, it is common to scale the input values to avoid that certain features with large values compared to other features becomes dominating. The convergence time will often be noticeably faster if the average of each input sample in the training set is close to zero [75].

For convenience, we define Lemma 3.2.1. Note that the lemma only considers the scalar case since the $\tanh(x)$ used in the LSTM neural network operates element-wise.

Lemma 3.2.1. *The activation function $|\tanh(x)|$ may be bounded by $|x|$ as given in eq. (3.30),*

$$|\tanh(x)| \leq |x| \quad \forall x \in \mathbb{R} \tag{3.30}$$

Proof. See Section A.1 in Appendix A. ■

We are now ready to define the main theorem. Theorem 3.2.2 proposes constraints on the PE LSTM neural network parameters such that the neural network is ISS.

Theorem 3.2.2. *Consider the state-space PE LSTM neural network in eq. (3.27)-(3.29). If the conditions given in eq. (3.31) and eq. (3.32) are satisfied, the PE LSTM is input-to-state (ISS) stable with respect to the inputs u_k, d_{u_k}, d_{w_k} and augmented b_{g_k}.*

$$\left\|\sigma(\bar{f}_{f_k})\right\|_1 + \left\|\sigma(\bar{f}_{o_k})\right\|_1 \cdot \left\|\sigma(\bar{f}_{f_k})\right\|_1 < 1 \tag{3.31}$$

$$\left(\left\|\sigma(\bar{f}_{i_k})\right\|_1 + \left\|\sigma(\bar{f}_{o_k})\right\|_1 \cdot \left\|\sigma(\bar{f}_{i_k})\right\|_1\right) \left\|W_g\right\|_1 < 1 \tag{3.32}$$

where $\bar{f}_{f_k}$, $\bar{f}_{i_k}$, $\bar{f}_{g_k}$ and $\bar{f}_{o_k}$ denotes the diagonal matrices defined in eq. (3.33)-(3.36),

$$\bar{f}_{f_k} = diag(f_{f_k}^{(1)}, \ldots, f_{f_k}^{(H)}) \tag{3.33}$$

$$\bar{f}_{i_k} = diag(f_{i_k}^{(1)}, \ldots, f_{i_k}^{(H)}) \tag{3.34}$$

$$\bar{f}_{g_k} = diag(f_{g_k}^{(1)}, \ldots, f_{g_k}^{(H)}) \tag{3.35}$$

$$\bar{f}_{o_k} = diag(f_{o_k}^{(1)}, \ldots, f_{o_k}^{(H)}) \tag{3.36}$$

where $f_j^{(p)}, p \in \{1,\dots,H\}$ *denotes the p-th entry of the vectors* $f_j, j \in \{f_k, i_k, g_k, o_k\}$*, defined in eq.*(3.23)-(3.26)*.* H *denotes the hidden size of the network.*

Proof. See Section A.2 in Appendix A. ■

Remark 1. *In the case of a multilayer LSTM (i.e. stacked LSTM modules as shown in Figure 3.1), the hidden state* h_k *of the shallowest LSTM (layer 0) is fed as the input to the next LSTM and so on and so forth for deeper layers, forming a cascaded system. As long as the conditions in Theorem 3.2.2 are satisfied on a layer-to-layer basis for each sequence element, the overall network is input-to-state stable if the input constraint assumption given in Assumption 3.2.1 is relaxed to restrict the input to* $[-1,1] \forall k \in \mathscr{S}$*. In such a case, the intermediate layer-to-layer input,* h_k*, satisfies the same assumptions as the original input due eq.* (3.17) *(the function range of hyperbolic tangent is* $[-1,1]$*), and by [68, Corollary 4.2], the cascaded system is input-to-state stable if each subsystem (i.e. LSTM layer) is input-to-state stable.*

It is beneficial to express these equations in terms of parameter matrices. Firstly, by Assumption 3.2.1, we know that $\left\|U_j u_k\right\| \leq 1\left\|U_j\right\| \forall j \in \{i,o,f,g\}$. Due to eq. (3.17), we see that the upper bound on the range of $h_{k_{\text{PE}}}$ is 1, since it is dependent on the range of the hyperbolic tangent ($[-1,1]$). As such, $\left\|W_j h_k\right\| \leq 1\left\|W_j\right\| \forall j \in \{i,o,f,g\}$. Moreover, we have that $\bar{f}_{j_k}, j \in \{i,o,f,g\}$ are diagonal matrices, as defined in Theorem 3.2.2. The 1-norm of a diagonal matrix is equal to the largest absolute value entry in the diagonal, since all off diagonal elements are zero. This nothing but the infinity norm of the vectors $f_{j_k}, j \in \{i,o,f,g\}$, which in turn may be upper-bounded as shown in eq. (3.37),

$$\begin{aligned}\left\|\sigma(\bar{f}_{j_k})\right\|_1 &\leq \sigma\left(\left\|f_{j_k}\right\|_\infty\right) = \sigma\left(\left\|U_j u_k + U_j d_{u_k} + W_j h_{k_{\text{PE}}} + W_j d_{w_k} + b_j\right\|_\infty\right) \\ &\leq \sigma\left(\left\|U_j + U_j d_{u_k} + W_j h_{k_{\text{PE}}} + W_j d_{w_k} + b_j\right\|_\infty\right) \\ &\leq \sigma\left(\left\|U_j\right\|_\infty + \left\|W_j\right\|_\infty + \left\|U_j\right\|_\infty \|d_{u_k}\|_\infty + \left\|W_j\right\|_\infty \|d_{w_k}\|_\infty + \left\|b_j\right\|_\infty\right)\end{aligned} \tag{3.37}$$

$\forall j \in \{i,o,f,g\}$. Moreover, the first inequality of eq. (3.37) is based on the function range (y-axis) of the sigmoid function (σ), being 0 to 1 for all of its function arguments, as seen from eq. (2.13).

Notice that $\|d_{u_k}\|_\infty \leq \eta$ and $\|d_{w_k}\|_\infty \leq \eta$ (this is our uncertainty set discussed in Section 3.1.1 and Section 3.1.2). Following the discussion in Section 3.1.1 regarding the disturbances d_{w_k}, we set set the disturbances corresponding to the hidden state equal to 0. Inserting the upper-bound of $\|d_{u_k}\|$ in eq. (3.37), one end up with the simplified expression in eq. (3.38),

$$\begin{aligned}\left\|\sigma\left(\bar{f}_{j_k}\right)\right\|_1 &\leq \sigma(\left\|U_j\right\|_\infty + \left\|W_j\right\|_\infty + \left\|U_j\right\|_\infty \eta + \left\|b_j\right\|_\infty) \\ &= \sigma\left(\left\|U_j\right\|_\infty \cdot \underbrace{(1+\eta)}_{\bar{\eta}} + \left\|W_j\right\|_\infty + \left\|b_j\right\|_\infty\right) \\ &= \sigma\left(\bar{\eta}\left\|U_j\right\|_\infty + \left\|W_j\right\|_\infty + \left\|b_j\right\|_\infty\right) \quad \forall j \in \{i,o,f,g\}\end{aligned} \tag{3.38}$$

To summarise, we impose the conditions given in eq. (3.39) and eq. (3.40) on the PE LSTM neural network parameters in order to ensure the ISS stability property.

$$\sigma\left(\bar{\eta}\left\|U_f\right\|_\infty + \left\|W_f\right\|_\infty + \left\|b_f\right\|_\infty\right) + \sigma\left(\bar{\eta}\|U_o\|_\infty + \|W_o\|_\infty + \|b_o\|_\infty\right)\cdot\sigma\left(\bar{\eta}\left\|U_f\right\|_\infty + \left\|W_f\right\|_\infty + \left\|b_f\right\|_\infty\right) < 1 \quad (3.39)$$

$$\left[\sigma\left(\bar{\eta}\|U_i\|_\infty + \|W_i\|_\infty + \|b_i\|_\infty\right) + \sigma\left(\bar{\eta}\|U_o\|_\infty + \|W_o\|_\infty + \|b_o\|_\infty\right)\cdot\sigma\left(\bar{\eta}\|U_i\|_\infty + \|W_i\|_\infty + \|b_i\|_\infty\right)\right]\left\|W_g\right\|_1 < 1 \quad (3.40)$$

where $\bar{\eta} = 1+\eta$, and η defines the radius of the ℓ_∞-norm ball uncertainty set.

These constraints may be enforced during training by for example scaling the parameter matrices, to ensure the conditions are satisfied *after* each parameter update. This ensures that the network is trained with the scaled parameters (forward pass and backpropagation). The conditions ensure that the network's reachable set is bounded and that the effect of initialisation asymptotically vanishes [3].

Lastly, we discuss the similarities and differences with regard to the related work of [3]. In [3], a norm-based Lyapunov function is recommended (particularly the 1-norm) due to some desirable properties introduced in Section 2.1. This Lyapunov is also used in the analysis of the PE LSTM neural network from Section 3.1. As such, the conditions in both works are in the form of some norm function. The main difference is that the analysis in [3] is done on a regular LSTM representation (such as the one presented in Section 2.2.4). The conditions in Theorem 3.2.2 are concerned with the PE LSTM representation, firstly introduced in eq. (3.1)-(3.6). This is different than the regular LSTM in that two new inputs are generated and consequently, the state space representation is different. The conditions in Theorem 3.2.2 take into account this architectural change. The reworked bounds in eq. (3.37) include the uncertainty set of the persistently exciting training procedures, described in Section 3.1.1 and Section 3.1.2.

Chapter 4

Experiments: Description of setup and cases

Chapter 4 outlines practical methodology, including the system used to generate the dataset, neural network configuration aspects and a description of all experiments employed for evaluating the theoretical methods explored in Chapter 3. A cascaded tank system dataset will be used for training the neural networks. The data generation process and the resulting dataset will be discussed and presented in Section 4.1. In Section 4.2, implementation details will be briefly dicussed. Section 4.3 presents motivation and justification for decisions regarding the configuration of the neural networks. Lastly, Section 4.4 describes the different experiments that are to be conducted.

We now present the semantic of some terms that will be appearing in the coming sections. The notion *model* denotes a neural network with its respective hyperparameter selection and training procedure. For each model, 10 neural networks will be trained with the same model configuration, but different parameter initialisations. We denote the different trained neural networks of a model as iterations, e.g. "model type 1, iteration 1" denotes the trained neural network from the first training iteration of "model type 1".

4.1 Data generation and dataset

The dataset that will be used to benchmark the proposed training procedures in Section 3.1 and the stability constraints in Section 3.2.2 stem from a two-tank system (cascaded tanks) with free outlets, supplied by a pump. This is a common benchmarking problem in nonlinear system identification and is presented in [76] among three common system identification problems.

Applying the Bernoulli's principle together with mass conservation, it may be shown that

the liquid levels in the cascaded tank system are given by eq. (4.1) [76],

$$\begin{aligned}\frac{\mathrm{d}h_1}{\mathrm{d}t} &= -\frac{a_1\sqrt{2g}}{A_1}\sqrt{h_1} + \frac{1}{A_1}ku(t)\\ \frac{\mathrm{d}h_2}{\mathrm{d}t} &= -\frac{a_2\sqrt{2g}}{A_2}\sqrt{h_2} + \frac{a_1\sqrt{2g}}{A_2}\sqrt{h_1}\end{aligned} \tag{4.1}$$

where h_1 and h_2 are system states and denote the liqud level of the upper and lower tank, respectively. Tank areas are denoted by A_1 and A_2, while a_1 and a_2 represent the effluent areas. The symbol g denotes the gravitational constant and the voltage to input flow conversion constant is represented by k.

Although there is a publicly available dataset of this process, we will simulate the dynamic system represented by eq. (4.1) and generate the dataset. This gives us flexibility with regard to input signals, the sampling time and dataset size. The system given in (4.1) is simulated using Python [4] and the library SciPy [77]. System specific parameters are given in Table 4.1

Table 4.1: Parameters of the cascaded tanks system.

a_1	a_2	A_1	A_2	k
0.5	0.5	1	1	1

A simulation time of 300 seconds is used with a time granulity (resolution) of 0.01 seconds, resulting 30000 datapoints. The input is generated by varying the amplitude of a square wave signal, where the amplitude is sampled from a uniform distribution. The clock period is chosen as 3 ms and the sampling period is 5 seconds. The resulting dataset is visualised in Figure 4.1 for the first 10000 time samples.

As we observe from eq. (4.1), the dataset provides us with two estimation subjects/target values: h_1 and h_2. The first is estimated based on merely one feature, namely the input u. The second subject is estimated based on two features, u and h_1. This is easily observed from eq. (4.1) or deducted based on inspecting the plant set-up.

4.2 Software implementation considerations

The implementation of all software is done with the help of the deep learning framework PyTorch [5]. The implementation of the PE LSTM neural network, discussed in Section 3.1, is realised by adding a parallel container of parameters, intended for storing the perturbations that are added to the input during the forward pass (see Section 2.2.6) of the neural network. The training procedure is described in Algorithm 2, and the signal flow is visualised in Figure 4.2. Note that we use the notation from Section 3.2.2 with regard to inputs and states. Comparing this figure with the standard LSTM neural network in Figure 2.10, we see that we

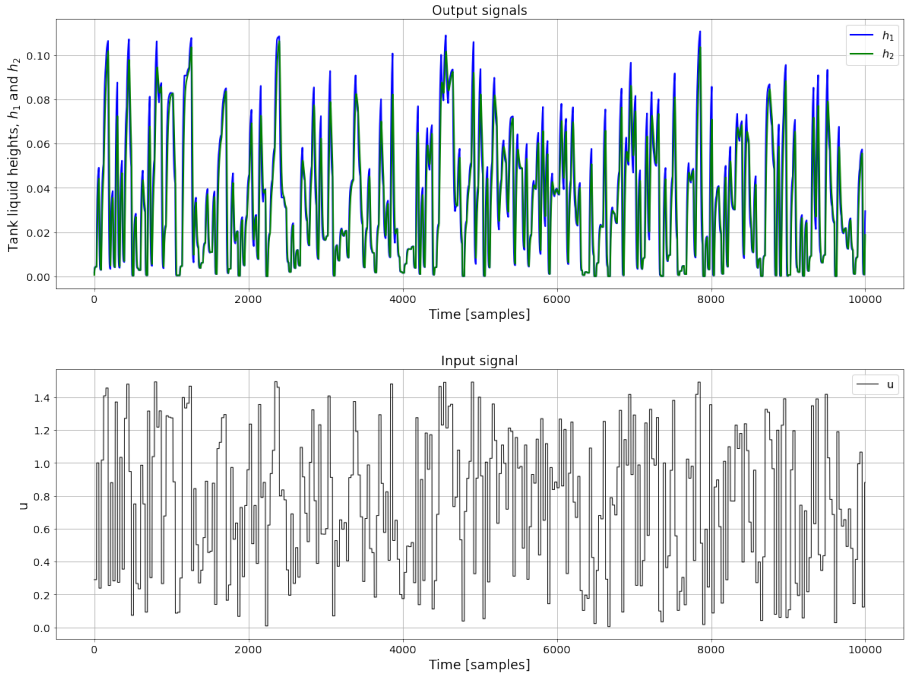

Figure 4.1: Simulation of a cascaded tanks system. (Lower) The input sequence. (Upper) The resulting system output corresponding to the input sequence in lower figure.

now seek to approximate two optimisation problems. The optimisation problems are estimated by using backpropagation. The optimisation process is described in detail in Section 2.4.3. The approximated perturbations are then added to the original inputs u_k and state h_k as means to enrichen the block-input signals. This is then fed into the standard LSTM module and backpropagation is performed with respect to the model parameters. We refer to the software framework, specifically the README-file, for specific software implementation details and tutorials for how to train models and run experiments.

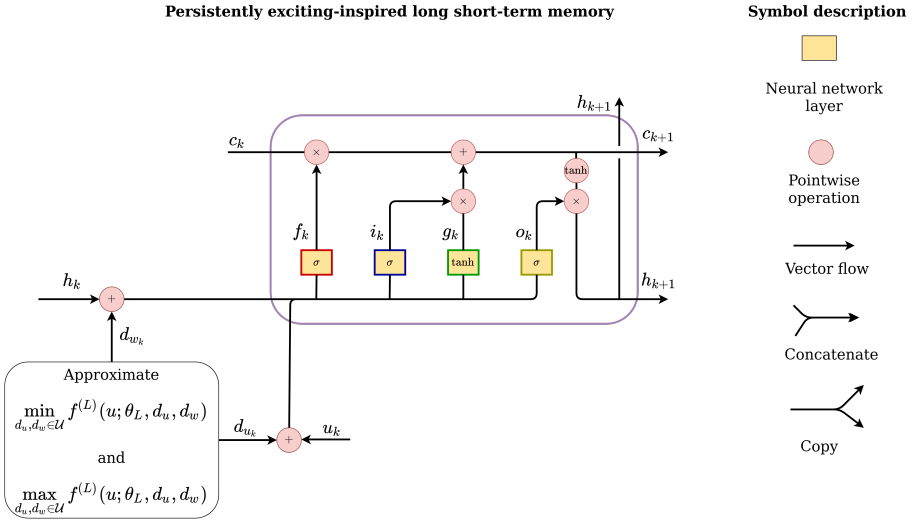

Figure 4.2: (Left) A long short-term memory RNN (the operations within purple boundary) with incoming PE perturbations. (Right) Symbol description. Image adapted from [9].

4.3 Recurrent neural network configuration

This section provides motivation and justification for specific configuration choices related to the LSTM neural networks that will be used in the experiments. Specific configurations will be presented in Section 4.4.

The development of neural network models entails performing a number of configurations. One must configure the neural network itself (number of layers and number of neurons in each layer), the dataset (splitting it into training, validation and test subset) and configure the training procedure. This includes tuning parameters whose values are used to alter the learning process, also known as hyperparameters, and choosing the optimiser, which seeks to minimise the training objective (see Section 2.2.6).

4.3.1 Configuring the network architecture

Each network will be trained with multiple layers. The sequence length is generally fixed at $S = 15$ time steps, with a few exceptions. The number of nodes will be tuned with two conflicting considerations in mind: large enough capacity to fit the data, but low enough as to avoid overfitting. In Section 2.2.2, a visual motivation is provided for both in light of the universal approximation theorem. Increasing the number of nodes slices the compact domain (i.e. the domain of the function we ought to estimate) into thinner rectangular functions, and potentially leads to a better function approximation. Increasing the number of nodes too much, though, may lead to fitting noise and outlier points as well. Configuring the network layers, we will start with one node in each layer, and gradually increase the number of nodes until satisfactory performance. This reduces the chance of overfitting by starting from low-capacity models.

4.3.2 Configuring the dataset

The dataset described in Section 4.1 will be split into a training, validation and test set, respectively. The split ratio will be 60%, 20% and 20% respectively, following the Pareto principle [78] (80% for train and validation and 20% for test). As described in Section 4.1, the dataset consists of 30 000 datapoints. Table 4.2 summarise the size of each dataset category (train, validation and test), including the total size of the dataset.

Table 4.2: Cascaded tank system dataset size information.

Target variable	Total size	Train size	Validation size	Test size
h_1	30 000	19200	4800	6000
h_2	30 000	19200	4800	6000

4.3.3 Configuring the training procedure

Training a RNN entails initialising the parameters, choosing an optimiser algorithm and tuning a number of hyperparameters. The hyperparameters play a major role in the learning process and are of significant importance. We have briefly mentioned a few hyperparameters in previous sections, such as the learning rate (Section 2.2.6) and the norm regularisation parameter (Section 2.3.3). In this section, these configuration steps will be discussed.

Initialisation

The initalisation of parameters is done by the framework software PyTorch, in which all parameters are initialised from $\mathcal{U}(-\sqrt{(z)}, \sqrt{(z)})$, where $z = \frac{1}{\text{hidden size}}$. Hidden size denotes the number of nodes in each layer. Note that when training different models and subsequently iterations of the respective models, we use the same parameter initialisation for all models for a specific iteration. For example, defining model 1 and model 2 with their respective configuration, we initialise the first iteration of model 1 and model 2 equally, and likewise for the second iteration and so on. This is done to keep the comparisons as fair as possible and avoid that potential "bad" initialisations result in a skewed impression of a particular model.

Optimiser

The optimiser utilised for all training iterations is the Adam optimiser (see Section 2.2.6). Adam optimiser is an efficient optimiser as one needs to tune less optimiser-specific hyperparameters, and is as such a common choice in benchmarking and testing. Additionally, due to momentum and adaptive learning rate adjustment, it is likely to converge fairly quickly to decent parameter estimates, as discussed in Section 2.2.6.

Feature scaling

Feature scaling has been previously discussed in relation to Assumption 3.2.1. One of the main reasons for scaling features is to avoid that certain features with large value ranges dominate during training. In this thesis, min-max scaling [79] will be used. Min-max scaling transforms all data points over all features into a common range, for example $[0, 1]$ or $[-1, 1]$. The downside to this scaling technique is that the method may be sensitive to outliers. This thesis will use either $[0, 1]$ or $[-1, 1]$ scaling depending on performance. Each neural network will be trained with each scaling range, and the worst results will be reported in Appendix C for completeness.

Hyperparameters

The hyperparameters are important due to their impact on the training procedure, and as such the resulting parameter estimates. The validation dataset is utilised for tuning the different parameters and monitoring the learning process with regard to overfitting. In general,

as discussed in Section 2.3.1, if the error on the training dataset decreases while at the same time the error on the validation dataset increases, it is a sign of overfitting.

Neural network hyperparameter tuning does generally not have any physical interpretation. As such, it is a matter of applying and observing, experience and general guidelines. The hyperparameters are tuned in three increments (e.g. 0.1, 0.01 and 0.001 for optimiser learning rate), for which the hyperparameter with the best parameter estimates is chosen. We evaluate the goodness of fit of the hyperparameters by comparing the mean square error over the whole test dataset and observing the validation error versus the training error. Some hyperparameters are unique to the different models. The ℓ_2 regularisation penalty parameter is for example only relevant to the models that apply ℓ_2 regularisation.

4.4 Experiments

This section describes experiments aimed at evaluating the training procedures and stability constraints derived and discussed in Chapter 3. The experiments are rooted in the objective list in Section 1.2.

4.4.1 Experiment 1: Robustness in view of persistency of excitation

The first experiment aims at evaluating the two training procedures from Section 3.1 as means to robustify LSTM neural networks. Three LSTM neural networks will be trained. One is trained with option 1 (we call this PE LSTM Opt-1) of the persistently exciting-inspired training procedure, introduced in Section 3.1.1. The second model will be trained with option 2 (we call this PE LSTM Opt-2) of the persistently exciting-inspired training procedure, described in Section 3.1.2. In order to have a common comparison ground, a baseline model will be included. The baseline model is a ℓ^2 norm penalised LSTM (ℓ_2 norm regularisation is described in Section 2.3.3). The reason for choosing this baseline is twofold. Norm regularisation is a traditional technique for attempting to produce robust neural network models. Secondly, in [12], a similar experiment is carried out on FNNs where multiple baselines are included, all of which performed similar or equivalent to ℓ^2 norm regularisation. All models used in experiment 1 are summarised in Table 4.3.

Table 4.3: All model types used in experiment 1

Model name	Characteristics
LSTM ℓ_2	ℓ_2-regularisation
PE LSTM Opt-1	Persistency of excitation-inspired
PE LSTM Opt-2	Persistency of excitation-inspired

As mentioned initially, experiment 1 aims at evaluating the robustness of LSTM-based neural networks produced by the training procedures inspired by the persistency of excita-

tion principle. A common benchmark is to evaluate models against worst-case perturbations, i.e. adversarial examples [2, 12, 20, 53]. Adversarial examples are treated in detail in Section 2.3.2. The two methods for generating adversarial examples will be the fast gradient sign method (FGSM) and project gradient descent method (PGD).

The configuration of the LSTM neural networks is done according to the methodology described in Section 4.3. The hyperparameters of the neural networks are summarised in Appendix B in Table B.1 and Table B.2. Lastly, the loss function used is the MSE, described in Section 2.2.6. This is a frequently used measure of difference in regression.

4.4.2 Experiment 2: Performance evaluation of input-to-state stability constraints

Note that in experiment 2, experiment 3 and experiment 4, only the best performing PE training procedure from experiment 1 (the option 1 training procedure from Section 3.1.1), will be considered in order to reduce the number of model comparisons. Moreover, we will only focus on the prediction problem corresponding to the target value h_2, due to the nature of the following experiments being to evaluate the ISS constraints. Moreover, this prediction problem has more features in the input, and deep learning models are likely to have more purpose (see Section 2.2.5 for an overview of deep learning and the number of input features).

ISS stability is a stability paradigm for dynamic systems used to address the question of robustness with regard to disturbances. In Theorem 3.2.2, constraints on the parameters of a persistently exciting LSTM are proposed based on a discrete ISS analysis. Experiment 2 is included to evaluate the effect that the constraints, given in eq. (3.39) and eq. (3.40), have on neural networks with regard to prediction capacity and robustness to perturbations in input. We will use test data perturbed by adversarial noise just as in experiment 1. The constraints are enforced as shown in Algorithm 3. The scaling constants, Λ_1 and Λ_2 must be specified prior to training. Large scaling constants yield faster convergence, but also more aggressive scaling and potentially worse performance due to the parameters being decreased aggressively during training.

Algorithm 3 Enforcement of PE LSTM input-to-state stability constraints

Require: Layer kernel weights U_f, U_o, U_i
Require: Layer recurrent weights W_f, W_o, W_i, W_g
Require: Layer biases b_i, b_o, b_f
Require: Scaling constants Λ_1, Λ_2
  $C_1 \leftarrow$ left hand side of eq. (3.39)
  $C_2 \leftarrow$ left hand side of eq. (3.40)
  while $not\ C_1 < 1$ and $not\ C_2 < 1$ **do**
    $U_f, W_f, U_o, W_o, b_f, , \leftarrow U_f/\Lambda_1, W_f/\Lambda_1, U_o/\Lambda_1, W_o/\Lambda_1$
    $W_i, W_g, U_i, b_i, b_w \leftarrow W_i/\Lambda_2, W_g/\Lambda_2, U_i/\Lambda_2, b_i/\Lambda_2, b_w/\Lambda_2$
  end while

PE LSTM neural networks with different configurations in architecture and hyperparameter selection will be presented. PE LSTM Opt-1 (no ISS constraints enforced) from Table 4.3 will be used as the baseline with regard to prediction capacity (MSE) and adversarial example robustness. In addition, a new PE LSTM model will be trained with spectral normalisation applied (see Section 2.5.2). We denote this model as PE LSTM Opt-1 Spectral norm. The first model with ISS-constraints enforced (we denote this PE LSTM Opt-1 ISS-1) will be based on the same architecture as PE LSTM Opt-1 (the same amount of layers, artificial neurons and sequence length/time steps). The second model with ISS-constraints enforced (we denote this PE LSTM Opt-1 ISS-2) will be more experimental with regard to the number of artificial neurons and sequence length/time steps. The three new models introduced for experiment 2 are summarised in Table 4.4, together with the baseline from experiment 1.

Table 4.4: All models used in Experiment 2

Model name	Characteristics
PE LSTM Opt-1	Input perturbation robustness
PE LSTM Opt-1 Spectral norm.	Input perturbation robustness and spectral normalisation
PE LSTM Opt-1 ISS-1	Input perturbation robustness and ISS stability
PE LSTM Opt-1 ISS-2	Input perturbation robustness and ISS stability

The configuration of all models is done according to the methodology described in Section 4.3. The hyperparameters of the neural networks are summarised in Appendix B in Table B.3. The error metric used is the mean square error (see Section 2.2.6), just as in Experiment 1. Moreover, in accordance with Remark 1, all multilayer models trained with ISS-constraints ought to be scaled in the range $[-1, 1]$.

4.4.3 Experiment 3: Input-to-state for persistently excited LSTM - Stability parameter

Experiment 3 is inspired by the work of [73]. A heuristic approach is proposed to estimate the stability parameter, L, (i.e. upper-bound on Lipschitz constant) of a LSTM. It is defined as given in eq. (4.2),

$$\frac{\left\| f(x,u) - f(x',u) \right\|}{\left\| x - x' \right\|} \leq L \tag{4.2}$$

$\forall x, x'$ and u satisfying the Assumptions imposed in Section 3.2.2. Moreover, $L \in \mathbb{R}_+$ is the parameter to be estimated. Note that in our case, f is described by the PE LSTM neural network given in eq. (3.7)-(3.9). This experiment is included to evaluate the effect that different initial conditions have on the estimated dynamic systems.

To estimate L in eq. (4.2), we generate 5990 input sequences from the test dataset. The initial state is randomly sampled from a normal distribution $x, x' \hookrightarrow \mathcal{N}(0, 0.3)$. For each input sequence, we re-initialise 10 times and choose the maximum estimate. The three models

used for prediction purposes on the test dataset, PE LSTM Opt-1 Spectral norm., PE LSTM Opt-1 ISS-1 and PE LSTM Opt-1 ISS-2, are described in Section 4.4.2.

4.4.4 Experiment 4: Input-to-state for persistently excited LSTM - Gain function estimates

The constraints in Theorem 3.2.2 may be conservative. Experiment 4 is included to investigate how the constraints affect the ISS constrained models specifically. This analysis is conducted by comparing the gain functions (γ) in the ISS definition, Definition 2.5.3, for the two models PE LSTM Opt-1 ISS-1 and PE LSTM Opt-1 ISS-2. A randomly generated selection of input sequences and initial states will be used. The two networks are trained with the same hyperparameters, but differ both in architecture and sequence length.

Particularly, the inequality (2.59) in Definition 2.5.3 will be used to compute the gain functions. If the PE LSTMs are indeed ISS, according to Definition 2.5.3, the following must hold,

$$\left\| x\left(k, x_0, u, d_u, b_g\right)\right\|_2 \leq \beta\left(\left\|x_0\right\|_2, k\right) + \gamma_1\left(\|u\|_\infty\right) + \gamma_2\left(\left\|d_u\right\|_\infty\right) + \gamma_3\left(\left\|b_g\right\|_\infty\right) \tag{4.3}$$

for each bounded input $u_i \in \ell_\infty^m$, $i = \{1,2,3\}$, any initial state $x_0 \in \mathbb{R}^n$ and $\forall k \in \mathbb{Z}_+$. Recall from Section 2.5.3 that β represents a class $\mathcal{KL}$ function γ represents a $\mathcal{K}$ function. Since β is a class $\mathcal{KL}$-function, by the range of the class of $\mathcal{KL}$-function, given in Definition 2.5.2, eq. (4.3) may be lower-bounded as given in eq. (4.4),

$$\left\| x\left(k, x_0, u, d_u, b_g\right)\right\|_2 \leq \gamma_1\left(\|u\|_\infty\right) + \gamma_2\left(\left\|d_u\right\|_\infty\right) + \gamma_3\left(\left\|b_g\right\|_\infty\right) \tag{4.4}$$

The comparison functions derived in Section A.2 as a part of the input-to-state analysis are summarised in eq. (4.5)-(4.11),

$$\alpha_1(\|x\|_2) = \|x\|_2 \tag{4.5}$$
$$\alpha_2(\|x\|_2) = \sqrt{D}\|x\|_2 \tag{4.6}$$
$$\alpha_3(\|x\|_2) = \sqrt{2H}\|x\|_2 \tag{4.7}$$
$$\sigma_u\left(\|u\|_2\right) = b\sqrt{D}\left\|U_g\right\|_1\left\|u_k\right\|_2 \tag{4.8}$$
$$\sigma_{d_u}\left(\left\|d_u\right\|_2\right) = b\sqrt{D}\left\|U_g\right\|_1\left\|d_{u_k}\right\|_2 \tag{4.9}$$
$$\sigma_{d_w}\left(\left\|d_w\right\|_2\right) = b\sqrt{H}\left\|W_g\right\|_1\left\|d_{h_k}\right\|_2 \tag{4.10}$$
$$\sigma_{b_g}\left(\left\|b_g\right\|_2\right) = b\sqrt{H}\left\|b_g\right\|_2 \tag{4.11}$$

where all α-functions are class $\mathcal{K}_\infty$-functions and all σ-functions are $\mathcal{K}$-functions. The term b is defined in eq. (A.21) and D and H represent the number of features in the input and the number of hidden nodes, respectively.

As seen from eq. (4.4) and eq. (4.5)-(4.11), the comparison functions of the ISS-stability definition are not directly related to the comparison functions of the ISS-Lyapunov definition. In [68], they show how one may compute the corresponding γ-functions in eq. (4.4) from the ISS-Lyapunov comparison functions, given in eq. (4.5)-(4.11). From the proof of Theorem 2.5.1 and Remark **3.6** in [68], it is shown that for any $\rho \in \mathcal{K}_\infty$ so that $\mathrm{Id} - \rho$ (Id represents the any identity function, f(x)=x, on some set) is of class $\mathcal{K}$, the gain functions γ may be computed as in eq. (4.12),

$$\gamma = \alpha_1^{-1} \circ \alpha_2 \circ \alpha_3^{-1} \circ \rho^{-1} \circ \sigma \tag{4.12}$$

where $(f \circ g)(x)$ denotes function composition, i.e. $f(g(x))$ and f^{-1} represents the inverse of function f. Moreover, ρ may be defined as $\rho(r) = r - \arctan(r)$. This is a class $\mathcal{K}_\infty$ by Definition 2.5.1. In such a case, using the identity function $f(r) = r$, we have that $f(r) - \rho(r) = r - (r - \arctan(r)) = \arctan(r)$. Since $\arctan(r)$ is a class $\mathcal{K}$-function [67, Chapter 4.4], the assumption is indeed satisfied for our defined ρ. The general expression for γ in eq. (4.12), may be used to define our input-specific γ-functions by inserting eq. (4.8), eq. (4.9) and eq. (4.11) for σ in eq. (4.12). The resulting comparison functions are given in eq. (4.13)-(4.15),

$$\gamma_1 = \alpha_1^{-1} \circ \alpha_2 \circ \alpha_3^{-1} \circ \rho^{-1} \circ \sigma_u \tag{4.13}$$

$$\gamma_2 = \alpha_1^{-1} \circ \alpha_2 \circ \alpha_3^{-1} \circ \rho^{-1} \circ \sigma_{d_u} \tag{4.14}$$

$$\gamma_3 = \alpha_1^{-1} \circ \alpha_2 \circ \alpha_3^{-1} \circ \rho^{-1} \circ \sigma_{b_g} \tag{4.15}$$

The sum of the defined comparison functions in eq. (4.13)-(4.15) must be greater than or equal to $\left\| x\left(k, x_0, u, d_u, b_g\right) \right\|_2$ for each bounded input u, d_u and b_g, any initial state x_0 and $\forall k \in \mathbb{Z}_+$, according to the definition of input-to-state stability, given in Definition 2.5.3. This is the equivalent to $\gamma_1 + \gamma_2 + \gamma_3 - \left\| x\left(k, x_0, u, d_u, b_g\right) \right\|_2 \geq 0$. The smaller the difference, the smaller the gains, γ. The test set is used to generate 5900 input sequences. The initial state is randomly sampled from a normal distribution $x \hookrightarrow \mathcal{N}(0, 0.3)$. For each input sequence and initial state, $\gamma_1 + \gamma_2 + \gamma_3 - \left\| x\left(k, x_0, u, d_u, b_g\right) \right\|_2$ is calculated. The minimum difference over the 5900 input sequences and initial states is stored, and this process is repeated 100 times for the 10 training iterations of each model, PE LSTM Opt-1 ISS-1 and PE LSTM Opt-1 ISS-2, described in Section 4.4.2.

Lastly, remark that verifying even simple properties of deep neural networks have been shown to be an NP-complete problem [80]. This experiment is thus not included to "prove" input-to-state stability heuristically, but merely to investigate how the ISS-constraints in Theorem 3.2.2 affect models with difference architectures and working on different sequence lengths, in light of the two ISS models presented in Section 4.4.2.

Chapter 5

Results and Dicussion

Chapter 5 presents results from the conducted experiments in Section 5.1. The results are discussed in Section 5.2.

5.1 Main results

The results are organised in a section-to-section manner based on the four experiments, starting with the results from experiment 1.

5.1.1 Experiment 1

The mean square *test* error over the whole test set of the predicted liquid level of tank 1 (h_1) is summarised in Table 5.1. The two test cases considered are when the test input is not perturbed in any way and when the test input is perturbed by adversarial examples. We see that the two models LSTM ℓ_2 and PE LSTM Opt-1 perform quite similar in all test cases for the hyperparameters given in Table B.1 in Appendix B. The LSTM ℓ_2 is 8.40% worse than the PE LSTM Opt-1 model in the nominal test case of no perturbation. It is only 5.46% worse when the input is perturbed by the FGSM-method (See section 2.3.2) with a perturbation magnitude of $\epsilon = 0.01$. On the contrary, the LSTM ℓ_2 model performs 1.15% better than the counterpart, PE LSTM Opt-1, in presence of adversarial noise with a perturbation magnitude of $\epsilon = 0.1$. Both the models produce parameters that are relatively stable from training iteration to training iteration, as indicated by the standard deviations in parenthesis. The PE LSTM Opt-2 model performs particularly bad compared to the two other models with large spread in mean square test errors for the 10 training sessions. The results are analogous when considering the second method for perturbing the input, namely the PGD (described in Section 2.3.2).

The mean square *test* error over the whole test set of the predicted liquid level of tank 2 (h_2) is summarised in Table 5.2. The two test cases considered are as described for the prediction problem related to tank 1. As we see, the differences between the models are more significant for this estimation problem. The best performing models is the PE LSTM Opt-1,

i.e. the neural networks trained with the training procedure described in Section 3.1.1. The PE LSTM Opt-1 has 40% lower mean square test error compared to the second-best model, PE LSTM Opt-2, in the no perturbation case, and 39% and 41% lower mean square test error than PE LSTM Opt-2 when the input is perturbed with FGSM with the two perturbation magnitudes $\epsilon = 0.01$ and $\epsilon = 0.1$, respectively. The results are analogous when considering input perturbed by the PGD method.

Table 5.1: (Experiment 1) Test error (in 1×10^{-6}) of the prediction of liquid level of tank 1 (h_1) of the cascaded tank system with min-max scaling in the range $[0,1]$. The evaluation metric used is the mean square error (MSE). The average test MSE (outside parenthesis) and the standard deviation (inside parenthesis) stem from 10 training sessions producing in total 10 neural networks for each model type. The models are tested in two scenarios. The first scenario is when the test data is not perturbed in any way. The second scenario is when the test data is perturbed. Two methods are used to perturb the test data: FGSM and PGD, with two different perturbation strengths (ϵ). The best results for the different situations are highlighted in bold.

Model	***No perturbation***	***FGSM***		***PGD***	
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.1$
LSTM ℓ_2	11.9 (0.853)	18.3 (1.03)	**173(2.30)**	18.3 (1.03)	**166(2.01)**
PE LSTM Opt-1	**10.9(0.732)**	**17.3(0.934)**	175(2.11)	**17.2(0.935)**	167 (2.56)
PE LSTM Opt-2	18.3 (4.16)	26.5 (5.04)	201 (14.7)	26.5 (5.04)	195 (15.5)

Table 5.2: (Experiment 1) Test error (in 1×10^{-6}) of the prediction of liquid level of tank 2 (h_2) of the cascaded tank system with min-max scaling in the range $[-1,1]$. The evaluation metric used is the mean square error (MSE). The average test MSE (outside parenthesis) and the standard deviation (inside parenthesis) stem from 10 training sessions producing in total 10 neural networks for each model type. The models are tested in two scenarios. The first scenario is when the test data is not perturbed in any way. The second scenario is when the test data is perturbed. Two methods are used to perturb the test data: FGSM and PGD, with two different perturbation strengths (ϵ). The best results for the different situations are highlighted in bold.

Model	***No perturbation***	***FGSM***		***PGD***	
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.1$
LSTM ℓ_2	10.5 (2.70)	14.7 (3.72)	92.7 (21.6)	14.7 (3.72)	92.0 (21.4)
PE LSTM Opt-1	**7.43(1.79)**	**10.1(2.21)**	**57.1(7.64)**	**10.1(2.21)**	**57.0(7.68)**
PE LSTM Opt-2	8.90 (3.69)	12.4 (4.61)	81.5 (19.2)	12.4 (4.61)	80.5 (19.1)

Figure 5.1 and Figure 5.2 show the prediction of the liquid level of tank 1 and tank 2 versus the target (i.e. true) values and the corresponding training and validation losses, respec-

tively. The model-type used to produce the predictions is the best-performing model from Table 5.1, in which the sample of the model closest to the mean is chosen from one of the 10 training sessions. The two models LSTM ℓ_2 and PE LSTM Opt-1 have quite similar performances, and therefore we include two figures for the estimation of h_1. The predicted value is estimated by using one feature in the input.

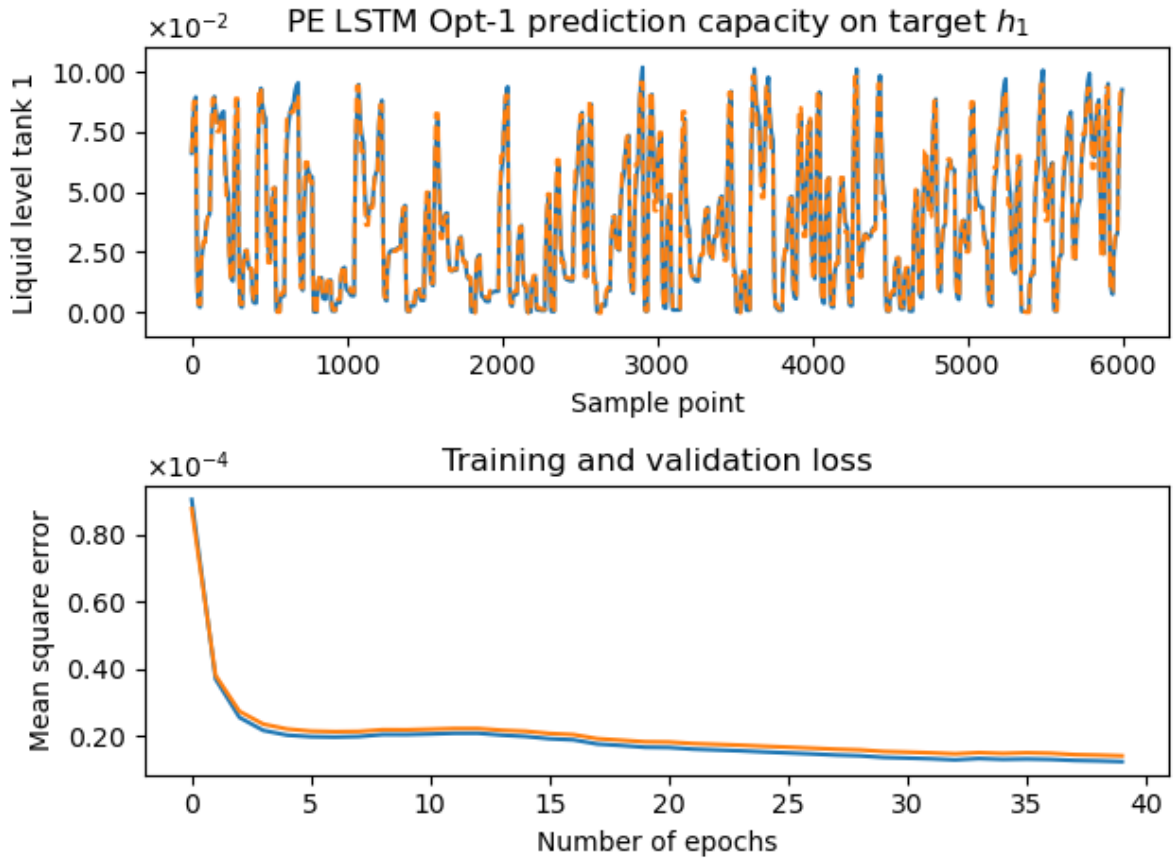

Figure 5.1: (Upper) Experiment 1 prediction capacity of the PE LSTM Opt-1 model given in Table 4.3. Target value is the liquid level of tank 1 (h_1). Orange dashed lines indicate the predicted values. Blue solid line indicate target values. (Lower) Corresponding train loss (blue line) and validation loss (orange line) for each epoch.

Figure 5.3 show the prediction of the liquid level of tank 2 versus the target (i.e. true) value and the corresponding training and validation losses. The model-type used to produce the predictions is the best-performing model from Table 5.2 (PE LSTM Opt-1), in which the iteration of the model closest to the mean is chosen from one of the 10 training sessions. The predicted value is estimated by using two features in the input.

As mentioned in Section 4.3, the worse results corresponding to the alternative scaling ranges are included in Appendix C in Table C.1 and Table C.2, corresponding to the prediction of h_1 and h_2, respectively.

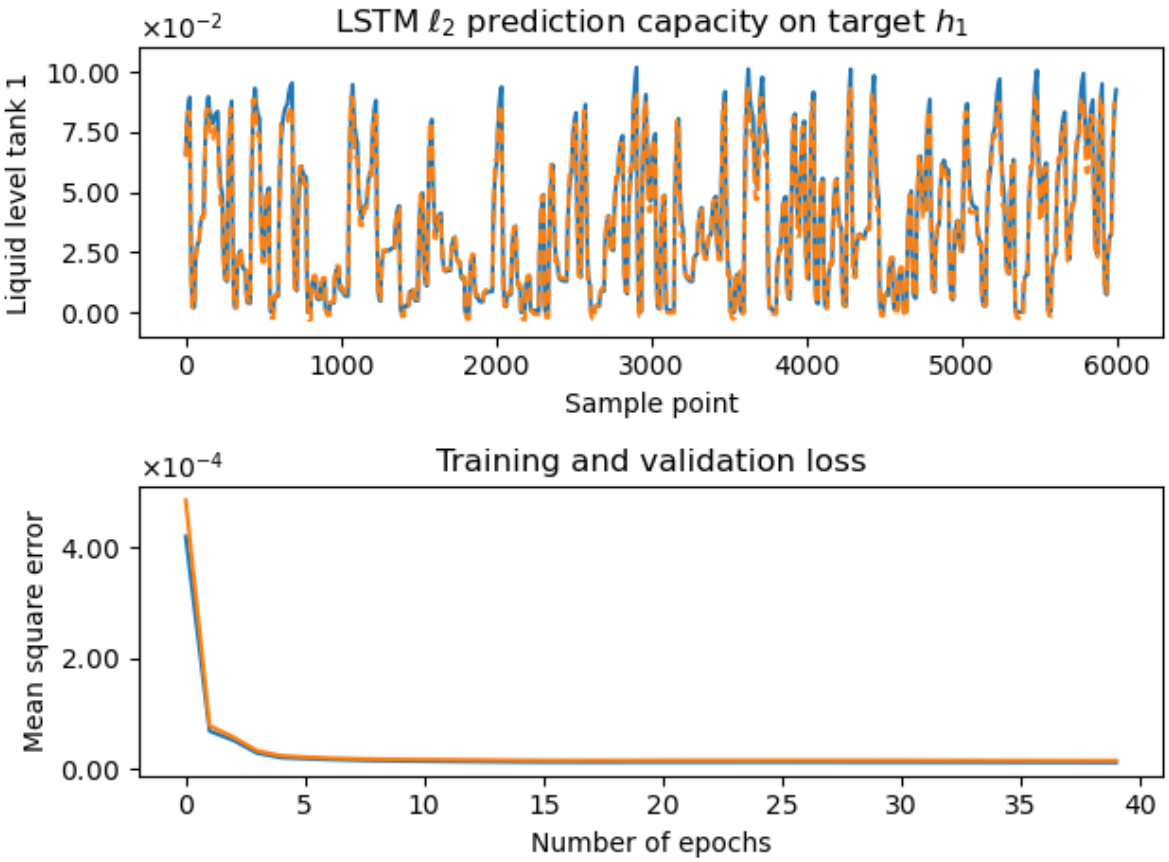

Figure 5.2: (Upper) Experiment 1 prediction capacity of the LSTM ℓ_2 model given in Table 4.3. Target value is the liquid level of tank 1 (h_1). Orange dashed lines indicate the predicted values. Blue solid line indicate target values. (Lower) Corresponding train loss (blue line) and validation loss (orange line) for each epoch.

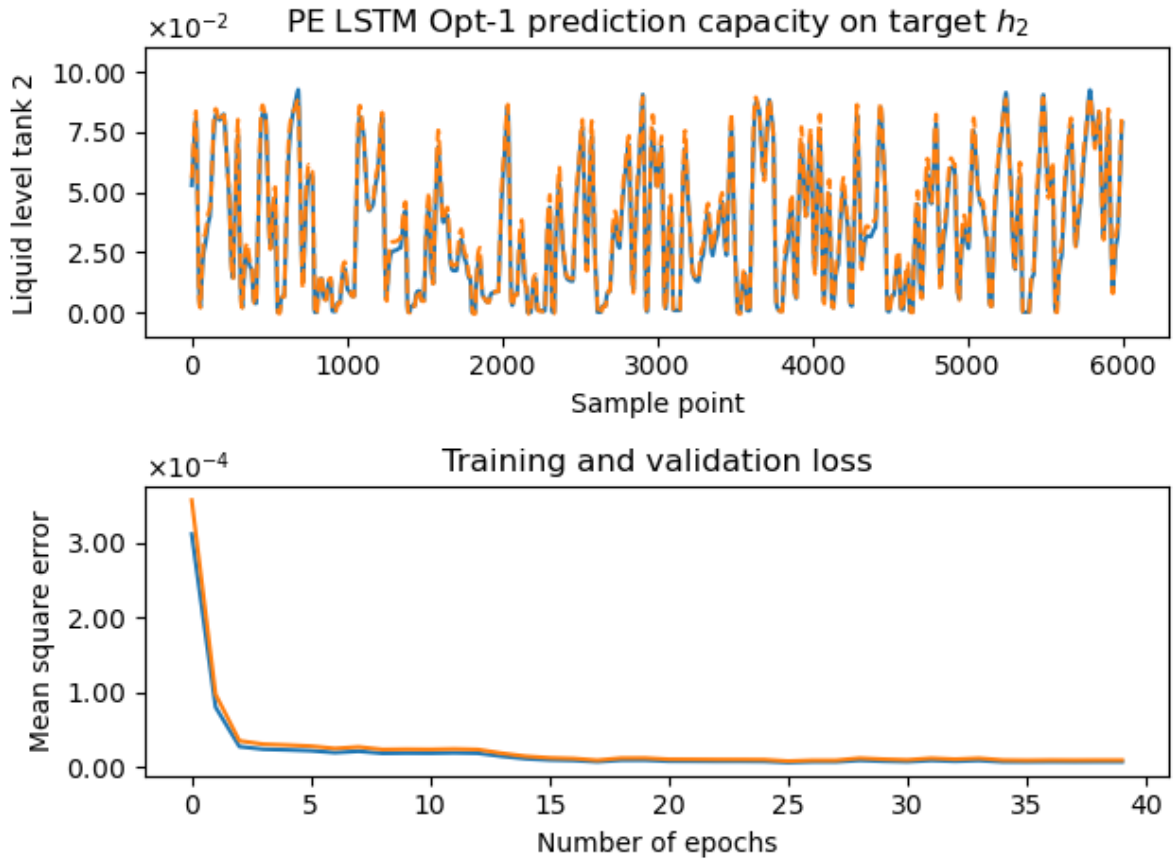

Figure 5.3: (Upper) Experiment 1 prediction capacity of the PE LSTM Opt-1 model in Table 4.3. Target value is the liquid level of tank 2 (h_2). Orange dashed lines indicate the predicted values. Blue solid line indicate target values. (Lower) Corresponding train loss (blue line) and validation loss (orange line) for each epoch.

5.1.2 Experiment 2

Table 5.3 summarise the Experiment 2 test MSE over the whole test set for the predicted liquid level of tank 2 (h_2). The model from Experiment 1, with no restriction of the model parameters, is the best performing model. This can be considered the gold standard among the models in Table 4.4 as the parameters are not restricted by any means, compared to the spectral normalised model and ISS models. The PE LSTM Opt-1 ISS-2, which is the experimental model with increased number of nodes and working on longer sequences, is the second best model, with a test MSE increase of 14.5 % in the non-perturbed case and a 13.6 % and 12.9 % increase in the FGSM perturbed case, compared to PE LSTM Opt-1. The results are almost equivalent when using the PGD method.

Table 5.3: (Experiment 2) Test error (in 1×10^{-6}) of the liquid level tank 2 (h_2) prediction of the cascaded tank system. The evaluation metric used is the mean square error (MSE). The average test MSE (outside parenthesis) and the standard deviation (inside parenthesis) stem from 10 training sessions producing in total 10 neural networks for each model type in Table 4.4. The models are tested in two scenarios. The first scenario is when the test data is not perturbed in any way. The second scenario is when the test data is perturbed. Two methods are used to perturb the test data: FGSM and PGD, with two different perturbation strengths (ϵ). The best results for the different situations are highlighted in bold.

Model	***No perturbation***	***FGSM***		***PGD***	
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.1$
PE LSTM Opt-1	**7.43(1.79)**	**10.1(2.21)**	**57.1(7.64)**	**10.1(2.21)**	**57.0(7.68)**
PE LSTM Opt-1 Spectral norm.	9.04 (3.86)	12.1 (4.85)	65.9 (18.9)	12.1 (4.85)	65.5 (19.0)
PE LSTM Opt-1 ISS-1	11.2 (2.26)	14.5 (2.71)	71.9 (12.4)	14.5 (2.70)	71.5 (12.1)
PE LSTM Opt-1 ISS-2	8.69 (1.57)	11.7(1.84)	65.6(5.74)	11.7(1.86)	65.4(5.78)

Figure 5.4 shows the prediction capacity of the ISS model, PE LSTM Opt-1 ISS-2, together with the training and validation loss. The loss curves show that there are no direct signs of overfitting, as the validation and training loss are corresponding with regard to increasing and decreasing losses. Signs of overfitting are described in detail in Section 2.3.1.

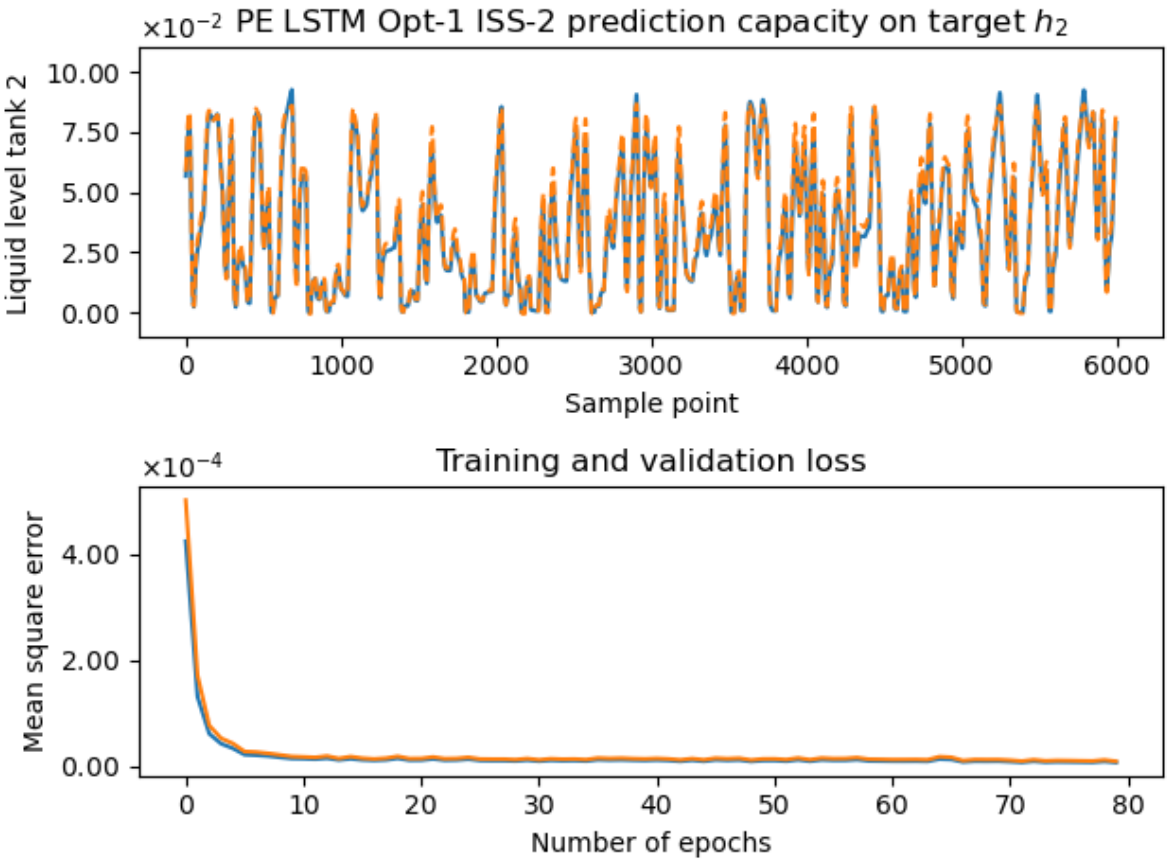

Figure 5.4: (Upper) Experiment 2 prediction capacity of the PE LSTM Opt-1 ISS-2 model in Table 4.4. Target value is the liquid level of tank 2 (h_2). Orange dashed lines indicate the predicted values. Blue solid line indicate target values. (Lower) Corresponding train loss (blue line) and validation loss (orange line) for each epoch.

5.1.3 Experiment 3

Figure 5.5 shows the estimate of the stability parameter, L, on the test dataset for the three models PE LSTM Spectral norm., PE LSTM Opt-1 ISS-1 and PE LSTM Opt-1 ISS-2. As indicated by the amplitude of the iteration samples, the two input-to-state stability models are less affected by changes (randomly) in initial condition. Moreover, PE LSTM Opt-1 ISS-2 is less affected than PE LSTM Opt-1 ISS-1.

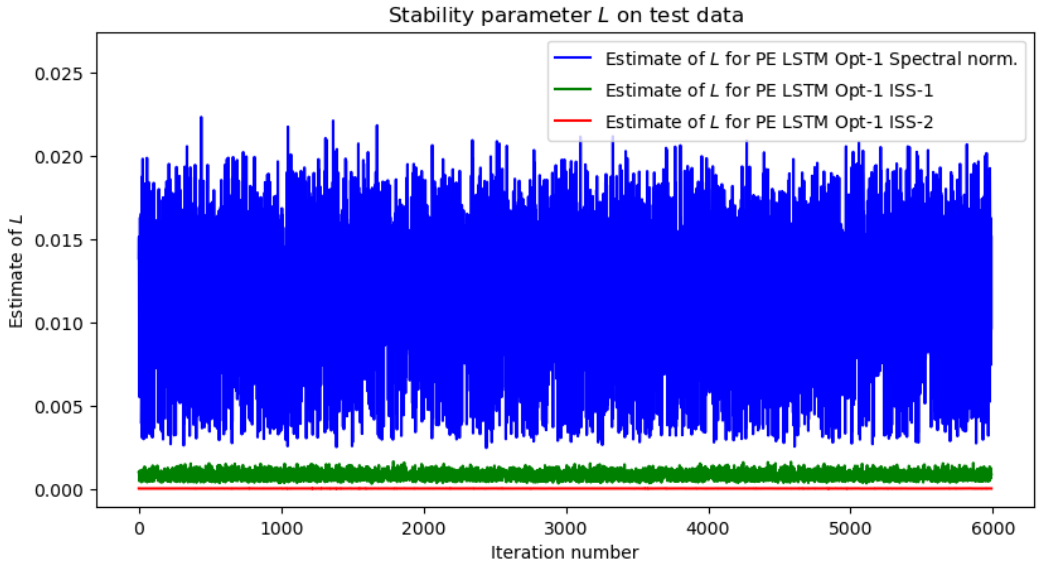

Figure 5.5: (Experiment 3) Estimate of L in eq. (4.2) for the three model types PE LSTM Spectral norm., PE LSTM ISS-1 and PE LSTM ISS-2 from Table 4.4.

Table 5.4 reports the largest occurring estimate of L, together with the mean and standard deviation over the 5990 test set samples. Both ISS models have lower maximum estimate. The maximum L estimate of PE LSTM Opt-1 Spectral norm. is 13.8 times larger than PE LSTM ISS-1 and remarkably 1629 times larger than PE LSTM Opt-1 ISS-2. The PE LSTM Opt-1 ISS-1 has larger spread in values, compared to PE LSTM Opt-1 ISS-2, as indicated by the standard deviations.

Table 5.4: (Experiment 3) Maximum L-estimate, mean and standard deviation over all 5990 test input sequences for the three models considered in experiment 3.

Model name	Max L	Mean	Standard deviation
PE LSTM Opt-1 Spectral norm.	2.20×10^{-2}	1.20×10^{-2}	4.29×10^{-3}
PE LSTM Opt-1 ISS-1	1.60×10^{-3}	8.28×10^{-4}	2.36×10^{-4}
PE LSTM Opt-1 ISS-2	1.35×10^{-5}	2.06×10^{-6}	2.06×10^{-6}

5.1.4 Experiment 4

The ISS stability definition ensures that the state trajectories of a dynamic system ought to be bounded by class $\mathcal{K}$ comparison functions for all initial conditions and inputs, as indicated by eq. (4.4). This yield that the difference between the comparison functions and the state trajectories for some initial condition and input ought to be lower-bounded by 0. Figure 5.6 and Figure 5.7 show the margins to this lower-bound (marked with red line) for all 10 model iterations. As indicated by the red cross in Figure 5.7, the worst-case margin to the above-mentioned bound is substantially larger (2.27 in amplitude) for the PE LSTM Opt-1 ISS-2 model compared to the PE LSTM Opt-1 ISS-1 model, whose worst case bound is given in Figure 5.6 (0.07 in amplitude).

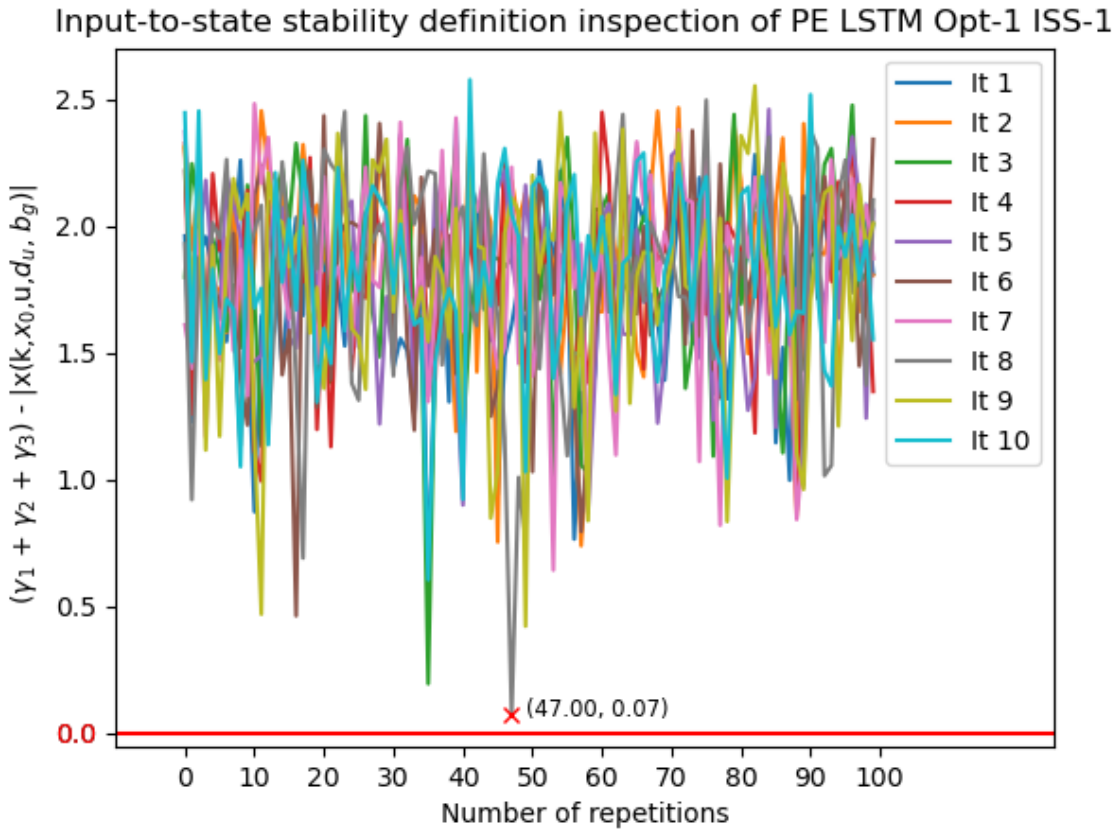

Figure 5.6: (Experiment 4) The difference between the gain functions and the norm of the state variables on the model PE LSTM ISS-1 for 100 iterations over the whole test set of 5900 input sequences. The smallest difference (in amplitude) is indicated by the red cross.

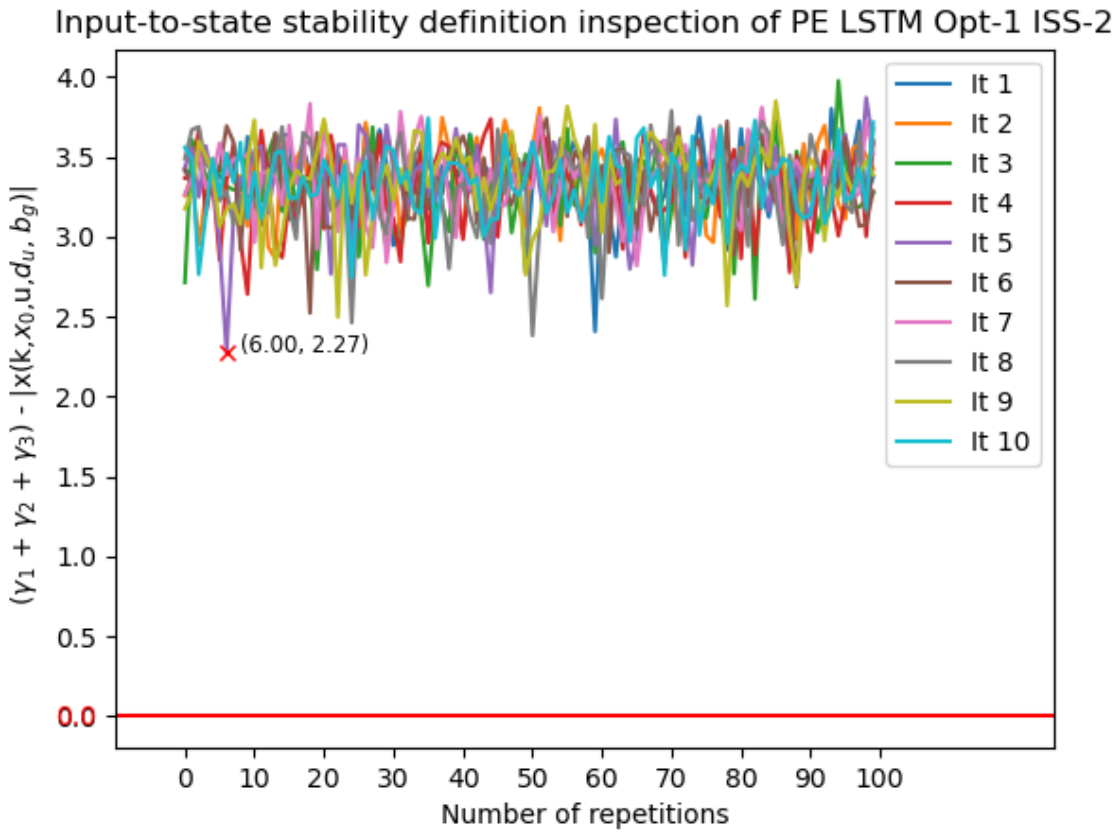

Figure 5.7: (Experiment 4) The difference between the gain functions and the norm of the state variables evaluated on the model PE LSTM ISS-2 model for 100 iterations over the whole test set of 5900 input sequences. The smallest difference (in amplitude) is indicated by the red cross.

5.2 Discussion

Experiment 1 is conducted to evaluate the two training procedures from Section 3.1.1 and Section 3.1.2, respectively, acting on the PE LSTM equations introduced in Section 3.1. The cascaded tanks dataset is used for two prediction problems: predicting the liquid level of tank 1 (h_1) and predicting the liquid level of tank 2 (h_2). As observed from Table 5.1, the two models LSTM ℓ_2 and PE LSTM Opt-1 are more or less equivalent in performance over all test cases predicting the target h_1. As indicated by Figure 5.1 and Figure 5.2, the predictions are lacking on the peaks of the graph. This may, to some extent, be mitigated by for example decreasing the "robustness parameters", i.e. using a lower ℓ_2-regularisation parameter for the LSTM ℓ_2 model, and reducing the radius of the ℓ_∞-norm ball uncertainty set for the PE LSTM models. Alternatively one can increase the number of nodes or layers in the network. The two proposed steps may, however, be problematic with regard to overfitting due to the low amount of features (1) in the input. The estimation of h_1 is as such challenging using neural networks as they may be *too* excessive for the task.

For the prediction of h_2, we see from Table 5.2 that both the training procedures attempt-

ing to persistently excite the parameters have better test set performance compared to the regularised model, both when the test set is not perturbed and when it is perturbed under all test cases. Figure 5.3 show that the prediction of the best performing model (PE LSTM Opt-1) is satisfactory. The main difference compared to the estimation of h_1 is that the input now consists of two features, and deeper layers are likely to have more purpose. This is due to how deeper layers learn more abstract features from the earlier layers (see Section 2.2.5). All the models trained on predicting the target (h_2) have an additional hidden layer compared to the models trained on the dataset with target h_1. The risk of the potential lack of persistency of excitation increases when the number of layers are increased, as expressed in Theorem 2.4.1. As such, it does appear that the persistency of excitation principle bears fruit, observing that both models attempting to persistently excite the parameters have better test accuracy compared to the norm regularised model when the network increase in depth. The parameters produced appear to be more robust, both when the test set is not perturbed, but especially in presence of adversarial examples. Comparing the two training procedures for attempting to persistently excite the parameters, the PE LSTM Opt-1 model (presented in Section 3.1.1) performs the best, overall. The PE LSTM Opt-2 model produces different quality of parameter estimates from iteration to iteration, as indicated by the standard deviation of the model in Table 5.2. As a result, the mean over all training iterations is lower. It may be beneficial to lower the learning rate additionally for this particular model as a first step attempting to stabilise the training. The two persistently exciting training procedures share similarities, such as both being bilevel optimisation problems. A substantial difference, however, is that the training procedure from Section 3.1.2 tries to minimise the loss objective under worst case disturbances, and uses these disturbances to attempt to persistently excite the parameters for each layer. The training procedure introduced in Section 3.1.1 seeks to ensure that all neighbour (within some uncertainty set) training data points are assigned values close to the corresponding target values, and generate signals (i.e. "disturbances") used to persistently excite the layer-to-layer parameters with this goal in mind.

Experiment 2 is included to evaluate the ISS constraints presented in Theorem 3.2.2, intended for persistently exciting LSTM neural networks. Only models trained with the procedure from Section 3.1.2 (option 1) are considered due to its better performance. As observed in Section 5.1.2, the ISS constraints do indeed lead to a slight worsening in performance. The constraints in the theorem may be conservative, and some relaxation of the conditions is likely advantageous, especially for the lower capacity model, PE LSTM Opt-1 ISS-1. The experimental PE LSTM, PE LSTM Opt-1 ISS-2 (increased number of nodes and trained on longer input sequences), has a 16.9% worsening of performance on the test set in case of no disturbances and approximately 13.0% worsening of performance when disturbances (adversarial examples) are present, compared to the model with no ISS constraints enforced. The prediction is lacking particularly at the higher values (peaks) of the prediction problem, as indicated by Figure 5.4. Some prediction improvement may likely be achieved by lowering the uncertainty set radius (η), at the cost of robustness. As seen from the train and validation

loss in the lower part of Figure 5.4, there are no obvious signs of overfitting as described in Section 2.3.1. Overall, this shows that although the ISS constraints might be conservative for this particular identification problem, decent performance may be achieved by increasing network capacity.

Experiment 3 and experiment 4 are conducted to analyse some stability effects of the ISS constrained models versus the spectral normalised model, all of which are presented in Section 4.4.2. The ISS property guarantees that the effect of different initial conditions on the output of a input-to-state stable system asymptotically vanishes [3]. Figure 5.5 and Table 5.4 indicate that the effect of different initial conditions is small for the models trained with the ISS constraints, especially compared to the spectral normalised model with no ISS constraints enforced. As observed from Table 5.4, there is a difference between the two ISS constrained models as well. The output of PE LSTM Opt-1 ISS-1 is affected noticeable more on different initial conditions compared to PE LSTM Opt-1 ISS-2. Experiment 4 may provide some justification. As observed from Figure 5.6, the PE LSTM Opt-1 ISS-1 is notably close to the lower bound (red line). The smallest margin is as low as 0.07 in amplitude. This may be an indication that the model trained in the regime of the constraints, i.e. in close proximity to the constraints firstly introduced in Theorem 3.2.2 and reworked in eq. (3.39) and eq. (3.40). It may thus be a sign that the constraints are *too* conservative for this particular architecture, reinforced by the observations from Experiment 2, where this particular model performed the worst. As mentioned, relaxation of the constraints could mitigate this issue for smaller architectures, and will recommended as an area for future works. Optionally, increasing the model capacity might be useful. As seen in Figure 5.7, the experimental higher-capacity model, PE LSTM Opt-1 ISS-2, has a substantially larger margin (2.27 in amplitude) to the lower-bound limit (red line). This model has an increased number of nodes in the hidden layers and is trained on longer input sequences, as reported in Table B.3.

Lastly, the results from this master thesis will be discussed in relation to the specialisation project [12], conducted by the author prior to the master thesis. Both works are concerned with the principle of persistency of excitation as means to produce more robust neural networks. The main architecture in the specialisation project is the FNN. Table D.1 in Appendix D is included from the specialisation project, and summarise the results. The datasets are organised by size. The smallest datasets are present at the top of the table, while the larger further down. We refer to [12] for specific numerical values and a more in-depth discussion regarding specific dataset results. As observed from this table summary, the FNN model (named PE) trained with the persistency of excitation training procedure introduced in Section 2.4.3, performs in general noticeably better on the larger datasets with more input features (which in turn requires deeper models). It performs slightly worse when the datasets are small and the models are of low capacity. This is similar to the observations from the LSTM neural network results in experiment 1 in the master thesis, as discussed initially in this section.

Chapter 6

Conclusions and Recommendations for Further Work

Chapter 6 presents concluding remarks and an evaluation of the progress on each objective listed in Section 1.2. Recommendations for further work will also be proposed.

6.1 Conclusions

The overall goal of this master thesis is to study the robustness and stability of long short-term memory based recurrent neural networks. The first part of the thesis is concerned with input perturbation robustness in light of persistently exciting-inspired LSTM (PE LSTM) neural networks. This work is based on Objectives 1-3 in Section 1.2. The second part of the thesis is concerned with the input-to-state stability property for the persistently exciting-inspired LSTM neural network. This work is based on Objectives 4-5 in Section 1.2.

Objective 1 is concerned with the training procedure from [2] and its adaptation and use in a recurrent neural network setting for time series data (regression). The training procedure is adapted and discussed for long short-term memory recurrent neural networks in Section 3.1.1.

Objective 2 is concerned with alternative training procedures that are inspired by persistency of excitation. A training procedure based on a loss objective from nonlinear robust optimisation is put forward in Section 3.1.2.

Objective 3 is related to the evaluation of the training procedures with regard to robustness. This is covered by experiment 1 (Section 4.4.1) and the evaluation of experiment 1 in Section 5.1.1. Both training procedures contribute to robustifying LSTM neural networks when there are multiple features in the input. The model trained with the training procedure from Section 3.1.1 (option 1) performs better in all test cases compared to the model trained with the training procedure from Section 3.1.2 (option 2), and contributes most significantly to robustifying LSTM neural networks.

Objective 4 is concerned with the feasibility of deriving sufficient conditions for ensuring

input-to-state stability for the PE LSTM neural network. In Section 3.2.1, this neural network is written on state space form, after which a Lyapunov analysis is conducted (Appendix A.2) to find constraints on the neural network parameters for ensuring the input-to-state stability property. The ISS constraints are presented in Theorem 3.2.2.

Objective 5 is concerned with studying the effect the enforcement of the conditions derived as part of objective 4 have on the PE LSTM neural network. The conditions are expressed in terms of the LSTM neural network parameters for easier enforcing them during training in eq. (3.39) and eq. (3.40), at the cost of slightly upper bounding the original conditions. The results from experiment 2 in Section 5.1.2 show that when training persistently exciting-inspired LSTM neural networks, a noticeable degradation in performance occur for the low-capacity neural networks, likely due to the conservativeness of the ISS conditions. Increasing network capacity mitigates this effect by large. The results from experiment 3 in Section 5.1.3 show that the output of the PE LSTM neural networks trained with the ISS-conditions enforced is relatively unaffected by changes in initial conditions, compared to neural networks trained without the conditions enforced. This is likely on account of the ISS property. The results from experiment 4 in Section 5.1.4 show that for both ISS constrained models, a selection of simulated state trajectories are indeed bounded by functions dependent on the inputs, according to the ISS definition (Definition 2.5.3, in which the high-capacity model is the most robust model of the two. Overall, the input-to-state stability constraints may be restricting on the performance of the network for this particular dynamic system. This can however, at large, be mitigated by increasing network capacity.

6.2 Recommendations for Further Work

There are three different main areas in which further work may be directed: the PE LSTM training procedures, the ISS constraints and the application of the combination of the two.

The persistently exciting training procedures introduced in Chapter 3 apply the same signals across all layers when training deep neural networks. It may result in an improvement of performance if the perturbation signals are found separately for each layer. This, however, is likely to cause a substantial degradation in training speed. This brings forward a second recommendation for further work. The training procedures for attempting to persistently excite the model parameters are as introduced in Section 3.1.1 and Section 3.1.2 noticeably slower than regular training procedures (for example norm regularisation). For each epoch, an inner-optimisation problem is approximated (finding the signals/perturbations), which slows down the overall training, depending on the inner-optimisation epoch hyperparameter. It can potentially speed up the training process if one skips an epoch and apply the same signals from the previous epoch for some iterations, at the cost of performance.

The stability constraints introduced in Theorem 3.2.2 are likely to be too conservative for smaller networks (with less parameters and processing shorter input sequences) on similar dynamic system identification problems, resulting in a degradation in performance, as dis-

cussed in Section 5.2. Increasing the network capacity by training on longer sequences or adding additional neurons has been proposed as a way of mitigating this issue. An area for further work, which was briefly mentioned in Section 5.2, is to relax the conditions in Theorem 3.2.2 and investigate what effect this has on the ISS property for low-capacity networks. A possible way of achieving this is to consider alternative Lyapunov function candidates in analysing the persistently exciting inspired LSTM system.

The input-to-state stability property is important as it ensures the boundedness of the output reachable set, and that the effect of different initial conditions vanish [3, 68]. In [18], an input-to-state stable regular LSTM is utilised in designing a neural network based model predictive controller. A future area of research could be to utilise the persistently exciting-inspired LSTM neural network, together with its input-to-state stability constraints, in designing *robust* neural network based controllers.

Appendix A

Proofs

A.1 Proof of Lemma 3.2.1

Proof. Define the function $h(x) = |x|$, $g(x) = \tanh\big(|x|\big)$. Observing that $h(0) = g(0) = 0$ and the derivatives of the respective functions are:

$$\begin{aligned} \frac{\partial h}{\partial x} &= \frac{x}{|x|} \\ \frac{\partial g}{\partial x} &= \frac{x \cosh^{-2}(x)}{|x|} \end{aligned} \tag{A.1}$$

we may observe that since $0 \leq \cosh^{-2}(x) \leq 1 \quad \forall x \in \mathbb{R}$, we have that $\frac{\partial h}{\partial x} \geq \frac{\partial g}{\partial x} \quad \forall x \in \mathbb{R}\backslash\{0\}$, from which, in combination with $h(0) = g(0) = 0$, eq. (3.30) follows. ■

A.2 Proof of Theorem 3.2.2

Proof. Inspired by the work of [3], we start by considering the Lyapunov function candidate in eq. (A.2). This Lyapunov candidate enjoys some useful properties, among them that $\|x_k\|_1 = \|c_k\|_1 + \|h_k\|_1$ is true, which will come in handy in the analysis.

$$V(x) = \|x\|_1 \tag{A.2}$$

In order for us to conclude on anything with regard to input-to-state stabilty, the two conditions given in Definition 2.5.4 ought to be satisfied. The proof is as such split into two parts, in which part 1 treats condition 1 and part 2 treats condition 2.

Satisfying condition 1 (eq. (2.60)) of Definition 2.5.4

The Lyapunov candidate given in eq.(A.2) must first and foremost satisfy the condition given in eq. (2.60).

Condition 1, given in eq. (2.60), is satisfied by the equivalence of the 1-norm and 2-norm: for an arbitrary vector $w \in \mathbb{R}^n$, we have $\|w\|_2 \leq \|w\|_1 \leq \sqrt{n}\|w\|_2$ [25, Chapter 2.2]. We thus

have specifically that,

$$\|r\|_2 \leq \|r\|_1 \leq \sqrt{D}\|r\|_2 \tag{A.3}$$

$\forall r \in \mathbb{R}^D$, where $D \in \mathbb{Z}_+$ is the number of features in the input.

Given the relation in eq. (A.3), define $\alpha_1(r) = \|r\|_2$ and $\alpha_2(r) = \sqrt{D}\|r\|_2 \forall \mathbb{R}^D$, where $\alpha_i : [0, a_i) \to [0, \infty)$ for $i = \{1, 2\}$. Using the definition of a class $\mathcal{K}_\infty$ function, namely Definition 2.5.1, we may show that both comparison functions belong to the class by reason of:

- $\alpha_1(0) = \alpha_2(0) = 0$
- $\frac{\partial \alpha_1}{\partial r} > 0$
- $\frac{\partial \alpha_2}{\partial r} > 0$
- For $a_1 \to \infty$, $\alpha_1(r) \to \infty$ as $r \to \infty$
- For $a_2 \to \infty$, $\alpha_2(r) \to \infty$ as $r \to \infty$

Satisfying condition 2 (eq. (2.61)) of Definition 2.5.4

We precede with investigating the condition given in eq. (2.60). Note that from now, $\|\cdot\|$ denotes the L_1 norm (1-norm). In cases of different norms, it will be specifically denoted. By definition of eq. (2.61) and common norm function properties (in particular properties in Definition (2.1.1)), we may derive the following bound,

$$\begin{aligned} V(x_{k+1}) - V(x_k) &= \left\| f(k, x_k, \bar{u}_k \right\| - \left\| x_k \right\| && \text{(A.4)} \\ &\leq \left\| f_1 \right\| + \left\| f_2 \right\| - \left\| x_k \right\| && \text{(A.5)} \\ &= \left\| \sigma(f_{f_k}) \odot c_{k_{\text{PE}}} + \sigma(f_{i_k}) \odot \tanh(f_{g_k}) \right\| && \text{(A.6)} \\ &+ \left\| \tanh\left(c_{k+1_{\text{PE}}}\right) \odot \sigma(f_{o_k}) \right\| && \text{(A.7)} \\ &- \left\| x_k \right\| && \text{(A.8)} \end{aligned}$$

where f_{f_k}, f_{i_k}, f_{g_k} and f_{o_k} are defined in eq. (3.23)-(3.26). We now utilise Lemma 3.2.1 and common norm properties, particularly that $\left\| x + y \right\| \leq \|x\| + \left\| y \right\|$ and $\left\| xy \right\| \leq \|x\| \left\| y \right\|$. Lastly, we want to get rid of the Hadamard product. In accordance with the relation given in eq. (2.12), define the diagonal matrices given in (A.9)-(A.12),

$$\begin{aligned} \bar{f}_{f_k} &= \text{diag}(f_{f_k}^{(1)}, \ldots, f_{f_k}^{(H)}) && \text{(A.9)} \\ \bar{f}_{i_k} &= \text{diag}(f_{i_k}^{(1)}, \ldots, f_{i_k}^{(H)}) && \text{(A.10)} \\ \bar{f}_{g_k} &= \text{diag}(f_{g_k}^{(1)}, \ldots, f_{g_k}^{(H)}) && \text{(A.11)} \\ \bar{f}_{o_k} &= \text{diag}(f_{o_k}^{(1)}, \ldots, f_{o_k}^{(H)}) && \text{(A.12)} \end{aligned}$$

where $f_j^{(p)}, p \in \{1, \ldots, H\}$ denotes the p-th entry of the gate vectors $f_j, j \in \{f_k, i_k, g_k, o_k\}$. Recall from Section 2.2.4 that H denotes the hid. The $diag$-operator is defined in Section 2.1.3.

With these terms defined, we may abstract away the Hadamard product and work on eq. (A.4) to achieve eq. (A.13)-(A.20).

$$
\begin{aligned}
V(x_{k+1}) - V(x_k) &\leq \left\| \sigma(\bar{f}_{f_k}) c_{k_{\text{PE}}} \right\| + \left\| \sigma(\bar{f}_{i_k}) \tanh\left(\bar{f}_{g_k}\right) \right\| && \text{(A.13)}\\
&+ \left\| \sigma(\bar{f}_{o_k}) \right\| \cdot \left\| \sigma(\bar{f}_{f_k}) c_{k_{\text{PE}}} + \sigma(\bar{f}_{i_k}) \tanh\left(\bar{f}_{g_k}\right) \right\| - \left\| x_k \right\| && \text{(A.14)}\\
&\leq \left\| \sigma(\bar{f}_{f_k}) \right\| \cdot \left\| c_{k_{\text{PE}}} \right\| + \left\| \sigma(\bar{f}_{i_k}) \right\| \cdot \left\| \bar{f}_{g_k} \right\| && \text{(A.15)}\\
&+ \left\| \sigma(\bar{f}_{o_k}) \right\| \cdot \left[\left\| \sigma(\bar{f}_{f_k}) \right\| \cdot \left\| c_{k_{\text{PE}}} \right\| + \left\| \sigma(\bar{f}_{i_k}) \right\| \cdot \left\| \bar{f}_{g_k} \right\| \right] && \text{(A.16)}\\
&- \left\| c_{k_{\text{PE}}} \right\| - \left\| h_{k_{\text{PE}}} \right\| && \text{(A.17)}\\
&= \left(\left\| \sigma(\bar{f}_{f_k}) \right\| + \left\| \sigma(\bar{f}_{o_k}) \right\| \cdot \left\| \sigma(\bar{f}_{f_k}) \right\| - 1 \right) \left\| c_{k_{\text{PE}}} \right\| && \text{(A.18)}\\
&+ \left(\left\| \sigma(\bar{f}_{i_k}) \right\| + \left\| \sigma(\bar{f}_{o_k}) \right\| \cdot \left\| \sigma(\bar{f}_{i_k}) \right\| \right) \left\| \bar{f}_{g_k} \right\| && \text{(A.19)}\\
&- \left\| h_{k_{\text{PE}}} \right\| && \text{(A.20)}
\end{aligned}
$$

where Lemma 3.2.1 is applied in the transition between ineq. (A.13)-(A.14) and ineq. (A.14)-(A.15). Moreover, the state vector components are retrived by utilising the fact that the L_1-norm is by definition the sum of the absolute value of each vector component. As such, $\left\| x_k \right\|_1 = \left\| c_{k_{\text{PE}}} \right\|_1 + \left\| h_{k_{\text{PE}}} \right\|_1$ by equality. We now simplify the notation by defining the two terms a and b in eq. (A.21).

$$
\begin{aligned}
a &= \left\| \sigma(\bar{f}_{f_k}) \right\| + \left\| \sigma(\bar{f}_{o_k}) \right\| \cdot \left\| \sigma(\bar{f}_{f_k}) \right\| - 1\\
b &= \left\| \sigma(\bar{f}_{i_k}) \right\| + \left\| \sigma(\bar{f}_{o_k}) \right\| \cdot \left\| \sigma(\bar{f}_{i_k}) \right\|
\end{aligned}
\qquad \text{(A.21)}
$$

and insert for $\bar{f}_{g_k}$ (given in eq. (3.25)) in order to recover the inputs. These changes are reflected in eq. (A.22)-(A.23).

$$
\begin{aligned}
V(x_{k+1}) - V(x_k) &\leq a \left\| c_{k_{\text{PE}}} \right\| + b \left\| \bar{f}_{g_k} \right\| - \left\| h_{k_{\text{PE}}} \right\| && \text{(A.22)}\\
&= a \left\| c_{k_{\text{PE}}} \right\| + b \left\| U_g u_k + U_g d_{u_k} + W_g h_{k_{\text{PE}}} + W_g d_{h_k} + b_g \right\| - \left\| h_{k_{\text{PE}}} \right\| && \text{(A.23)}\\
&\leq a \left\| c_{k_{\text{PE}}} \right\| - \left\| h_{k_{\text{PE}}} \right\| && \text{(A.24)}\\
&+ b \left(\left\| U_g \right\| \left\| u_k \right\| + \left\| U_g \right\| \left\| d_{u_k} \right\| + \left\| W_g \right\| \left\| h_k \right\| + \left\| W_g \right\| \left\| d_{h_k} \right\| + \left\| b_g \right\| \right) && \text{(A.25)}\\
&\leq a \left\| c_{k_{\text{PE}}} \right\| + (b \left\| W_g \right\| - 1) \left\| h_{k_{\text{PE}}} \right\| && \text{(A.26)}\\
&+ b \left[\left\| U_g \right\| \left\| u_k \right\| + \left\| U_g \right\| \left\| d_{u_k} \right\| + \left\| W_g \right\| \left\| d_{h_k} \right\| + \left\| b_g \right\| \right] && \text{(A.27)}\\
&= -(-a \left\| c_{k_{\text{PE}}} \right\| - (b \left\| W_g \right\| - 1) \left\| h_{k_{\text{PE}}} \right\|) && \text{(A.28)}\\
&+ b \left[\left\| U_g \right\| \left\| u_k \right\| + \left\| U_g \right\| \left\| d_{u_k} \right\| + \left\| W_g \right\| \left\| d_{h_k} \right\| + \left\| b_g \right\| \right] && \text{(A.29)}
\end{aligned}
$$

where the expression in (A.28) is included due to ISS-Lyapunov definition, given in Definition 2.5.4, in which the state-related comparison function is defined as $-\alpha(\|x\|_2)$. The expres-

sions appearing in eq. (A.28)-(A.29) may be lower-bounded as shown in eq. (A.30)-(A.34),

$$-a\left\|c_{k_{\text{PE}}}\right\|_1-(b\left\|W_g\right\|_1-1)\left\|h_{k_{\text{PE}}}\right\|_1\leq\left\|c_{k_{\text{PE}}}\right\|_1+\left\|h_{k_{\text{PE}}}\right\|_1=\|x\|_1\leq\sqrt{2H}\|x\|_2 \tag{A.30}$$

$$b\left\|U_g\right\|_1\left\|u_k\right\|_1\leq b\sqrt{D}\left\|U_g\right\|_1\left\|u_k\right\|_2 \tag{A.31}$$

$$b\left\|U_g\right\|_1\left\|d_{u_k}\right\|_1\leq b\sqrt{D}\left\|U_g\right\|_1\left\|d_{u_k}\right\|_2 \tag{A.32}$$

$$b\left\|W_g\right\|_1\left\|d_{h_k}\right\|_1\leq b\sqrt{H}\left\|W_g\right\|_1\left\|d_{h_k}\right\|_2 \tag{A.33}$$

$$b\left\|b_g\right\|_1\leq b\sqrt{H}\left\|b_g\right\|_2 \tag{A.34}$$

where the first inequality of (A.30) is satisfied if and only if eq. (A.35) and eq. (A.36) are satisfied,

$$\begin{aligned}a<0\Leftrightarrow&\left\|\sigma(\bar{f}_{f_k})\right\|+\left\|\sigma(\bar{f}_{o_k})\right\|\cdot\left\|\sigma(\bar{f}_{f_k})\right\|-1<0\\\Leftrightarrow&\boxed{\left\|\sigma(\bar{f}_{f_k})\right\|+\left\|\sigma(\bar{f}_{o_k})\right\|\cdot\left\|\sigma(\bar{f}_{f_k})\right\|<1}\end{aligned} \tag{A.35}$$

$$b\left\|W_g\right\|-1<0\Leftrightarrow\boxed{\left(\left\|\sigma(\bar{f}_{i_k})\right\|+\left\|\sigma(\bar{f}_{o_k})\right\|\cdot\left\|\sigma(\bar{f}_{i_k})\right\|\right)\left\|W_g\right\|<1} \tag{A.36}$$

The expressions in eq.(A.30)-(A.34) results in the definition of the comparison functions given in eq. (A.37)-(A.41),

$$\alpha_3\left(\|x\|_2\right)=\sqrt{2H}\|x\|_2 \tag{A.37}$$

$$\sigma_u\left(\|u\|_2\right)=b\sqrt{D}\left\|U_g\right\|_1\left\|u_k\right\|_2 \tag{A.38}$$

$$\sigma_{d_u}\left(\left\|d_u\right\|_2\right)=b\sqrt{D}\left\|U_g\right\|_1\left\|d_{u_k}\right\|_2 \tag{A.39}$$

$$\sigma_{d_w}\left(\left\|d_w\right\|_2\right)=b\sqrt{H}\left\|W_g\right\|_1\left\|d_{h_k}\right\|_2 \tag{A.40}$$

$$\sigma_{b_g}\left(\left\|b_g\right\|_2\right)=b\sqrt{H}\left\|b_g\right\|_2 \tag{A.41}$$

These are based on the equivalence of the 1-norm and 2 norm: for an arbitrary vector $w\in\mathbb{R}^n$, we have $\|w\|_2\leq\|w\|_1\leq\sqrt{n}\|w\|_2$ [25, Chapter 2.2]. This conversion is done to get the comparison functions in the same form as Definition 2.5.4. Lastly, the bias vector is augmented to the input space, following the work of [3]. This is done since the inputs ought to be bounded by class $\mathcal{K}$ functions according to Definition 2.5.3. For a class $\mathcal{K}$-function, f, it must be true that $f(0)=0$. As we see, this cannot be ensured for any of the defined functions in eq. (A.38)-(A.41) by adding the norm of the bias vector. Alternatively, one may restrict the bias vector of the candidate gate to 0 during training. The first mentioned option is, however, likely to be less restricting on the network during training.

Inserting these expressions into eq. (A.22)-(A.29) yields eq. (A.42)

$$V(x_{k+1})-V(x_k)\leq-\alpha_3\left(\|x\|_2\right)+\sigma_{d_u}\left(\left\|d_u\right\|_2\right)+\sigma_u\left(\|u\|_2\right)+\sigma_{d_w}\left(\left\|d_w\right\|_2\right)+\sigma_{b_g}\left(\left\|b_g\right\|_2\right) \tag{A.42}$$

The next step is to verify that α_3 is $\mathcal{K}_\infty\forall c_0,h_0$ and that σ_u, σ_{d_u},σ_{d_w} and σ_{b_g} are class $\mathcal{K}\forall u_0,d_{u_0},d_{w_0},b_{g_0}$, respectively.

For α_3, we have that $\alpha_3(0) = 0$. Moreover, we need that it is strictly increasing. This is true if the conditions given in eq. (A.35) and eq. (A.36) are satisfied. Lastly, in order to conclude that α_3 is indeed a class $\mathcal{K}_\infty$, we have to ensure that if $a \to \infty$, $\alpha(r) \to \infty$ as $r \to \infty$. We observe that this indeed is the case for α_3. By Definition 2.5.1, α_3 is a class $\mathcal{K}_\infty$ function.

We now establish conditions for the functions given in eq. (A.38)-(A.41) to be class $\mathcal{K}$ functions. The definition of this class of functions is given in the first part of Definition 2.5.1. We see by inspection that $\rho_n(0) = 0$ for $n = \{u_k, d_{u_k}, d_{w_k}, b_{g_k}\}$. The next condition that ought to be satisfied is that the functions are strictly increasing. Since $b \geq 0$, D>0 and H>0, this is achieved by the Definition of a norm function. As such, combining that all functions, $\rho_n(0) = 0$ for $n = \{u_k, d_{u_k}, d_{w_k}, b_{g_k}\}$ and the fact that the function is strictly increasing, the functions given in (A.31)-(A.33) are indeed class $\mathcal{K}$ functions by Definition 2.5.1.

Finally, by Definition 2.5.4, we may conclude that the Lyapunov function, given in eq. (A.2), for the state-space model, given in eq. (3.18), is indeed an ISS-Lyapunov function. By Theorem 2.5.1, we may conclude that the state-space model describing the persistently exciting LSTM network is input-to-state stable with respect to the defined inputs when the conditions in eq. (A.35)-(A.36) are satisfied.

■

Appendix B

Hyperparameters

B.1 Hyperparameters: Experiment 1

Table B.1: Experiment 1 hyperparameter selection for models trained at predicting the liquid height of tank 1 (h_1). Note that for the hidden layers hyperparameter, the array-like structure represents how many artificial neurons (nodes) reside in each hidden layer, starting with first hidden layer.

Hyperparameter	***Model***		
	LSTM ℓ_2	***PE LSTM Opt-1***	***PE LSTM Opt-2***
Optimiser	Adam	Adam	Adam
Learning rate	0.005	0.005	0.003
Batch size	16	24	24
Epoch	40	40	40
Hidden layers	[4, 1]	[4, 1]	[4, 1]
ℓ_2-penalty	0.01	-	-
Perturbation epoch	-	4	4
Perturbation radius	-	0.02	0.02

Table B.2: Experiment 1 hyperparameter selection for models trained at predicting the liquid height of tank 2 (h_2). Note that for the hidden layers hyperparameter, the array-like structure represents how many artificial neurons (nodes) reside in each hidden layer, starting with first hidden layer.

Hyperparameter	***Model***		
	LSTM ℓ_2	***PE LSTM Opt-2***	***PE LSTM Opt-1***
Optimiser	Adam	Adam	Adam
Learning rate	0.0007	0.0007	0.0007
Batch size	32	32	32
Epoch	40	40	40
Hidden layers	[2,2,2]	[2,2,2]	[2,2,2]
Time steps	15	15	15
ℓ_2-penalty	0.008	-	-
Perturbation epoch	-	4	4
Perturbation radius	-	0.02	0.02

B.2 Hyperparameters: Experiment 2, Experiment 3 and Experiment 4

Table B.3: Experiment 2, experiment 3 and experiment 4 hyperparameter selection for models trained at predicting the liquid height of tank 2 (h_2). Note that for the hidden layers hyperparameter, the array-like structure represents how many artificial neurons (nodes) reside in each hidden layer, starting with first hidden layer.

Hyperparameter	***Model***			
	PE LSTM Opt-1	***PE LSTM Opt-1 Spectral norm.***	***PE LSTM Opt-1 ISS-1***	***PE LSTM Opt-1 ISS-2***
Optimiser	Adam	Adam	Adam	Adam
Learning rate	0.001	0.0005	0.0005	0.0005
Batch size	16	64	64	64
Epoch	20	80	80	80
Hidden layers	[2,2,2]	[2,2,2]	[2,2,2]	[4,3,2]
Time steps	15	15	15	30
Perturbation epoch	0.02	0.02	0.02	0.02
Perturbation radius	4	4	4	4

Appendix C

Alternative min-max scaling range results

C.1 Experiment 1

Table C.1: (Experiment 1) Test error (in 1×10^{-6}) of the prediction of liquid level of tank 1 (h_1) of the cascaded tank system with min-max scaling in the range $[-1, 1]$. The evaluation metric used is the mean square error (MSE). The average test MSE (outside parenthesis) and the standard deviation (inside parenthesis) stem from 10 training sessions producing in total 10 neural networks for each model type. The models are tested in two scenarios. The first scenario is when the test data is not perturbed in any way. The second scenario is when the test data is perturbed. Two methods are used to perturb the test data: FGSM and PGD, with two different perturbation strengths (ϵ). The best results for the different situations are highlighted in bold.

Model	***No perturbation***	***FGSM***		***PGD***	
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.1$
LSTM ℓ_2	**17.4(5.63)**	**25.1(6.92)**	191 (22.6)	**25.1(6.92)**	186 (23.0)
PE LSTM-1	22.2 (12.0)	30.3 (14.3)	**189(34.6)**	30.3 (14.3)	**185(36.3)**
PE LSTM-2	19.0 (6.47)	27.0 (7.80)	196 (19.3)	27.0 (7.81)	189 (20.9)

Table C.2: (Experiment 1) Test error (in 1×10^{-6}) of the prediction of liquid level of tank 2 (h_2) of the cascaded tank system with min-max scaling in the range $[0, 1]$. The evaluation metric used is the mean square error (MSE). The average test MSE and the standard deviation (in paranthesis) stem from 10 training sessions producing in total 10 prediction models for each model type in Table 4.4. The models are tested in two scenarios. The first scenario is when the test data is not perturbed in any way. The second scenario is when the test data is perturbed. Two methods are used to perturb the test data: FGSM and PGD, with two different perturbation strengths (ϵ). The best results for the different situations are highlighted in bold.

Model	***No perturbation***	***FGSM***		***PGD***	
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.1$
LSTM ℓ_2	13.6 (6.05)	21.6 (7.98)	208 (52.2)	21.6 (7.98)	205 (50.9)
PE LSTM Opt-1	**8.16(2.48)**	**12.8(2.88)**	**116(6.51)**	**12.8(2.88)**	**116(7.06)**
PE LSTM Opt-2	13.6 (5.30)	21.2 (6.48)	196 (40.0)	21.1 (6.48)	191 (38.0)

Appendix D

Selected results from specialisation project

The results in this appendix are included as displayed in the specialisation project [12] conducted by the author autumn 2020.

Table D.1: A summary of the best-performing models on each dataset (rows) on the different perturbation bounds. Results are from the specialisation project [12].

Dataset	***No perturbation***	***FGSM***		***PGD***	
		$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.1$
Yacht	ℓ_2-MSE	ℓ_2-MSE	Huber	ℓ_2-MSE	Huber
Boston	Huber	Huber	PE	Huber	PE
TTK28	PE	PE	PE	PE	PE
Power plant	PE	PE	PE	PE	PE
CBM	PE	Huber	Huber	Huber	Huber
Slice	PE	PE	PE	PE	PE

Bibliography

[1] S. Digital, *TAPI (Towards Autonomy in Process Industries)*. Available at https://www.sintef.no/en/projects/2019/tapi-towards-autonomy-in-process-industries/. Last visited 2019-11-30.

[2] K. Nar and S. Shankar Sastry, "Persistency of Excitation for Robustness of Neural Networks," *arXiv e-prints*, p. arXiv:1911.01043, Nov. 2019.

[3] E. Terzi, F. Bonassi, M. Farina, and R. Scattolini, "Model predictive control design for dynamical systems learned by Long Short-Term Memory Networks," *arXiv e-prints*, p. arXiv:1910.04024, Oct. 2019.

[4] P. S. Foundation, *Python Language Reference, version 3.8*. Available at https://www.python.org/. Last visited 2021-04-06.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[6] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. Available at http://neuralnetworksanddeeplearning.com/. Last visited 2019-11-15.

[7] F. v. Veen, "The neural network zoo." Available at https://www.asimovinstitute.org/neural-network-zoo/. Last visited 2021-04-04.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv e-prints*, p. arXiv:1512.03385, Dec. 2015.

[9] S. Varsamopoulos, K. Bertels, and C. Almudever, "Designing neural network based decoders for surface codes," 11 2018.

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

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] E. Peci, "Robustness in neural networks," tech. rep., NTNU: Norwegian University of Science and Technology, 2020.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[14] O. Abiodun, A. Jantan, O. Omolara, K. Dada, N. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, p. e00938, 11 2018.

[15] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv e-prints*, p. arXiv:1312.6199, Dec. 2013.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, p. 84–90, May 2017.

[17] G. R. Mode and K. Anuarul Hoque, "Adversarial Examples in Deep Learning for Multivariate Time Series Regression," *arXiv e-prints*, p. arXiv:2009.11911, Sept. 2020.

[18] E. Terzi, F. Bonassi, M. Farina, and R. Scattolini, "Model predictive control design for dynamical systems learned by Long Short-Term Memory Networks," *arXiv e-prints*, p. arXiv:1910.04024, Oct. 2019.

[19] S. Qiu, Q. Liu, S. Zhou, and C. Wu, "Review of artificial intelligence adversarial attack and defense technologies," *Applied Sciences*, vol. 9, p. 909, 03 2019.

[20] A. T. Nguyen and E. Raff, "Adversarial Attacks, Regression, and Numerical Stability Regularization," *arXiv e-prints*, p. arXiv:1812.02885, Dec. 2018.

[21] L. Meng, C.-T. Lin, T.-R. Jung, and D. Wu, "White-Box Target Attack for EEG-Based BCI Regression Problems," *arXiv e-prints*, p. arXiv:1911.04606, Nov. 2019.

[22] N. Carlini, "A complete list of all (arxiv) adversarial example papers," 2019. Available at https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html. Last visited 2021-04-03.

[23] Y. Wang, "A new concept using lstm neural networks for dynamic system identification," in *2017 American Control Conference (ACC)*, pp. 5324–5329, 2017.

[24] J. Gonzalez and W. Yu, "Non-linear system modeling using lstm neural networks," *IFAC-PapersOnLine*, vol. 51, no. 13, pp. 485–489, 2018. 2nd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2018.

[25] G. H. Golub and C. F. van Loan, *Matrix Computations*. JHU Press, fourth ed., 2013.

[26] G. Antipov, S. A. Berrani, N. Ruchaud, and J.-L. Dugelay, "Learned vs. hand-crafted features for pedestrian gender recognition," 10 2015.

[27] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *arXiv e-prints*, p. arXiv:1206.5538, June 2012.

[28] M. Lefkowitz, "Professor's perceptron paved the way for ai – 60 years too soon," 2019.

[29] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec. 1989.

[30] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The Expressive Power of Neural Networks: A View from the Width," *arXiv e-prints*, p. arXiv:1709.02540, Sept. 2017.

[31] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object Recognition with Gradient-Based Learning*, pp. 319–345. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[32] F. Chollet *et al.*, "Keras," 2015. Available at https://keras.io. Last visited 2021-06-06.

[33] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 5, pp. 157–66, 02 1994.

[34] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1211.5063, Nov. 2012.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[36] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *arXiv e-prints*, p. arXiv:1402.1128, Feb. 2014.

[37] R. Kozma, R. Ilin, and H. Siegelmann, "Evolution of abstraction across layers in deep learning neural networks," *Procedia Computer Science*, vol. 144, pp. 203–213, 01 2018.

[38] A. J. R. Simpson, "Abstract Learning via Demodulation in a Deep Neural Network," *arXiv e-prints*, p. arXiv:1502.04042, Feb. 2015.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv e-prints*, p. arXiv:1512.03385, Dec. 2015.

[40] *MSELoss*, (Accessed December 4th, 2020). https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html.

[41] A. Gunes Baydin, B. A. Pearlmutter, A. Andreyevich Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *arXiv e-prints*, p. arXiv:1502.05767, Feb. 2015.

[42] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[43] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2020. Available at https://d2l.ai/. Last visited 2021-03-15.

[44] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, second ed., 2006.

[45] Y. Mu, W. Liu, and W. Fan, "Stochastic Gradient Made Stable: A Manifold Propagation Approach for Large-Scale Optimization," *arXiv e-prints*, p. arXiv:1506.08350, June 2015.

[46] B. di Chen, Y. Xu, and A. Shrivastava, "Fast and accurate stochastic gradient estimation," in *NeurIPS*, 2019.

[47] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[48] J. Chen, D. Zhou, Y. Tang, Z. Yang, Y. Cao, and Q. Gu, "Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks," *arXiv e-prints*, p. arXiv:1806.06763, June 2018.

[49] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The Marginal Value of Adaptive Gradient Methods in Machine Learning," *arXiv e-prints*, p. arXiv:1705.08292, May 2017.

[50] V. Kotu and B. Deshpande, "Chapter 13 - anomaly detection," in *Data Science (Second Edition)* (V. Kotu and B. Deshpande, eds.), pp. 447–465, Morgan Kaufmann, second edition ed., 2019.

[51] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv e-prints*, p. arXiv:1412.6572, Dec. 2014.

[52] *Adversarial Attack*, (Accessed December 4th, 2020). https://engineering.purdue.edu/ChanGroup/ECE595/files/chapter3.pdf.

[53] G. R. Mode and K. Anuarul Hoque, "Adversarial Examples in Deep Learning for Multivariate Time Series Regression," *arXiv e-prints*, p. arXiv:2009.11911, Sept. 2020.

[54] T.-J. Chang, Y. He, and P. Li, "Efficient Two-Step Adversarial Defense for Deep Neural Networks," *arXiv e-prints*, p. arXiv:1810.03739, Oct. 2018.

[55] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv e-prints*, p. arXiv:1607.02533, July 2016.

[56] H. Li, S. Shan, E. Wenger, J. Zhang, H. Zheng, and B. Y. Zhao, "Blacklight: Defending Black-Box Adversarial Attacks on Deep Neural Networks," *arXiv e-prints*, p. arXiv:2006.14042, June 2020.

[57] X. Zhang and D. Wu, "Empirical Studies on the Properties of Linear Regions in Deep Neural Networks," *arXiv e-prints*, p. arXiv:2001.01072, Jan. 2020.

[58] J. Kukačka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," *arXiv e-prints*, p. arXiv:1710.10686, Oct. 2017.

[59] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1211.5063, Nov. 2012.

[60] H. X. Constantine Caramanis, Shie Mannor, *Robust Optimization in Machine Learning*. The MIT Press, 2012.

[61] B. L. Gorissen, I. Yanıkoğlu, and D. den Hertog, "A Practical Guide to Robust Optimization," *arXiv e-prints*, p. arXiv:1501.02634, Jan. 2015.

[62] J. Duchi, *Optimization with uncertain data*. Stanford University, 2018. Available at https://web.stanford.edu/class/ee364b/lectures/robust_notes.pdf. Last visited 2021-03-18.

[63] S. Leyffer, M. Menickelly, T. Munson, C. Vanaret, and S. M. Wild, "A survey of nonlinear robust optimization," *INFOR: Information Systems and Operational Research*, vol. 58, no. 2, pp. 342–373, 2020.

[64] L. E. Ghaoui and H. Lebret, "Robust solutions to least-squares problems with uncertain data," vol. 18, no. 4, 1997.

[65] S. Sastry and M. Bodson, *Adaptive Control: Stability, Convergence, and Robustness*. USA: Prentice-Hall, Inc., 1989.

[66] C. H. Martin and M. W. Mahoney, "Traditional and Heavy-Tailed Self Regularization in Neural Network Models," *arXiv e-prints*, p. arXiv:1901.08276, Jan. 2019.

[67] H. K. Khalil, *Nonlinear systems; 3rd ed.* Upper Saddle River, NJ: Prentice-Hall, 2002. The book can be consulted by contacting: PH-AID: Wallet, Lionel.

[68] Z.-P. Jiang and Y. Wang, "Input-to-state stability for discrete-time nonlinear systems," *Automatica*, vol. 37, no. 6, pp. 857–869, 2001.

[69] G. Chen, *Stability of Nonlinear Systems*. American Cancer Society, 2005.

[70] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral Normalization for Generative Adversarial Networks," *arXiv e-prints*, p. arXiv:1802.05957, Feb. 2018.

[71] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *arXiv e-prints*, p. arXiv:1406.2661, June 2014.

[72] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural Lander: Stable Drone Landing Control using Learned Dynamics," *arXiv e-prints*, p. arXiv:1811.08027, Nov. 2018.

[73] J. Miller and M. Hardt, "Stable Recurrent Models," *arXiv e-prints*, p. arXiv:1805.10369, May 2018.

[74] E. D. Sontag, *Input to State Stability: Basic Concepts and Results*, pp. 163–220. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[75] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, pp. 9–48. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[76] T. Wigren and J. Schoukens, "Three free data sets for development and benchmarking in nonlinear system identification," in *2013 European Control Conference (ECC)*, pp. 2933–2938, 2013.

[77] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[78] H. Siddiqui, "A second look at the pareto principle," 04 2015.

[79] O. A. Akanbi, I. S. Amiri, and E. Fazeldehkordi, "Chapter 4 - feature extraction," in *A Machine-Learning Approach to Phishing Detection and Defense* (O. A. Akanbi, I. S. Amiri, and E. Fazeldehkordi, eds.), pp. 45–54, Boston: Syngress, 2015.

[80] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," *arXiv e-prints*, p. arXiv:1702.01135, Feb. 2017.